BASCOM® AVR®

BASCOM-AVR user manual

Introduction

by MCS Electronics

Dear reader.

Thank you for your interest in BASCOM.

BASCOM was "invented" in 1995. It was intended for personal usage only. I decided to make it public as I found no other tool that was so simple to use. Since that time, a lot of options and extensions were added. Without the help and patience of the many users, BASCOM would not be what it is today : "the best and most affordable tool for fast proto typing".

We hope that BASCOM will contribute in making your work with microprocessors Easy and enjoyable.

Please notice that the samples in the manual are intended as simple samples. You should have a look at the sample code provided in the SAMPLES directory.

The MCS Electronics Team

BASCOM-AVR

Printed: 20-4-2025

Publisher

MCS Electronics

Managing Editor

M.C.Alberts

Technical Editors

M.C.Alberts

Cover Designer

B.F.de Graaff

Special thanks to:

All the people who contributed to this document, all the forum members that contributed in a positive way, all beta testers , and all customers.

While there is not enough space to mention all contributors, there are a few that I feel must be mentioned :
Josef Franz Vögel. He wrote the Trig libraries, the AVR-DOS file system and the DOUBLE library.

Luciano, Ian, Adrian and Kimmi, they are very active on the user forum. They take the time to give other forum members free help and advise. They do this for free just to help other BASCOM users.

MWS for his help and debugging.

MAK for pushing XMEGA, his samples and testing.

Peter Maroudas, RIP, for his work on the FT800 and FT801 implementation.

Ben Zijlstra, RIP, for being a true ambassador for BasCom.

Table of Contents

Foreword 0

Part I Index 28

1 Keyword Reference 29
2 Changes 33
What is new in 2087 33
What is new in 2086 34
What is new in 2085 37
What is new in 2084 38
What is new in 2083 39
What is new in 2082 41
What is new in 2081 42
What is new in 2080 43
What is new in 2078-2079 44
3 About MCS Electronics 48
Custom Designs 50
Application Notes 51
About this Help 51

Part II Installation 53

1 Installation of BASCOM 53
2 Updates 61
3 Move to new PC 66
4 Installation on multiple computers 67
5 Registration 67

Part III BASCOM IDE 69

1 Running BASCOM-AVR 69
2 File New 76
3 File Open 76
4 File Close 76
5 File Save 77
6 File Save As 77
7 File Print Preview 77
8 File Print 77
9 File Project 77
10 File ZIP 80
11 File Exit 81
12 Edit Undo 81
13 Edit Redo 81
14 Edit Cut 81
15 Edit Copy 81
16 Edit Paste 81

17 Edit Find 81
18 Edit Find Next 84
19 Edit Replace 85
20 Edit Goto 86
21 Edit Toggle Bookmark 86
22 Edit Goto Bookmark 86
23 Edit Indent Block 86
24 Edit Unindent Block 86
25 Edit Remark Block 86
26 Edit Insert ASCII 87
27 Edit Fold All Subs and Functions 87
28 Edit Unfold All Code 88
29 Edit Encrypt Selected Code 88
30 Edit Proper Indent 89
31 Edit Show Excluded Code 90
32 Edit Show Dead Code 91
33 View PinOut 93
34 View PDF viewer 97
35 View Error Panel 99
36 View Show Allert Window 99
37 View Project Files 100
38 View Code Explorer 101
39 View Vertical Splitter 107
40 Program Compile 108
41 Program Syntax Check 109
42 Program Show Result 110
43 Program Simulate 111
44 Program Send to Chip 124
45 Program Reset Chip 128
46 Tools Terminal Emulator 128
47 Tools LCD Designer 131
48 Tools LIB Manager 132
49 Tools Graphic Converter 133
50 Tools Stack Analyzer 134
51 Tools Plugin Manager 135
52 Tools Batch Compile 135
53 Tools PDF Update 138
54 Tools Resource Editor 140
55 Tools Font Editor 141
56 Options Compiler 143
Options Compiler Chip 143
Options Compiler Output 145
Options Compiler Communication 146

Options Compiler I2C, SPI, 1WIRE **147**
Options Compiler LCD **148**
57 Options Communication **149**
58 Options Environment **150**
59 Options Simulator **161**
60 Options Programmer **162**
Supported Programmers **164**
ISP programmer 165
PG302 programmer 166
Sample Electronics cable programmer 166
KITSRUS Programmer 167
MCS Universal Interface Programmer 168
STK500 Programmer 170
Lawicel BootLoader 174
MyAVR/MK2/AVR910 programmer 174
AVR ISP Programmer 175
USB-ISP Programmer 175
MCS Bootloader 184
PROGGY 186
FLIP 187
USBprog Programmer / AVR ISP mkII 190
KamProg for AVR 190
USBASP 193
STK600 194
ARDUINO 197
BIPOM MINI-MAX/C 200
mySmartUSB Light 200
UPDI Programmer 202
MCS EDBG Programmer 207
LIBUSB **216**
61 Options Monitor **225**
62 Options Printer **225**
63 Options Select Settings File **226**
64 Window Cascade **227**
65 Window Tile **228**
66 Window Tile Vertically **229**
67 Window Arrange Icons **229**
68 Windows Maximize All **229**
69 Window Minimize All **229**
70 Help About **230**
71 Help Index **231**
72 Help MCS Forum **231**
73 Help MCS Shop **232**
74 Help Support **233**
75 Help Knowledge Base **233**
76 Help Credits **234**
77 Help Update **235**
78 Help Wiki **237**
79 BASCOM Editor Keys **237**

80 Program Development Order 238
81 PlugIns 238
Font Editor 238

Part IV BASCOM HARDWARE 242

1 Additional Hardware 242
2 AVR Internal Hardware 242
3 AVR Internal Registers 243
4 AVR Internal Hardware TIMER0 245
5 AVR Internal Hardware TIMER1 246
6 AVR Internal Hardware Watchdog timer 248
7 AVR Internal Hardware Port B 248
8 AVR Internal Hardware Port D 250
9 Adding XRAM with External Memory Interface 251
10 Adding XRAM to XMEGA using EBI 255
11 Adding SRAM 4-port Non Multiplexed 257
12 Attaching an LCD Display 266
13 Memory usage 267
14 Statements and Hardware Resources 278
15 Using the UART 278
16 Using a BOOTLOADER 285
17 USING RS485 295
18 Using the I2C protocol 297
19 Using the 1 WIRE protocol 310
20 Using the SPI protocol 314
21 Using USI (Universal Serial Interface) 321
22 Power Up 325
23 Reference Designs 326
EM4095 RFID Reader 326

Part V Chips 336

1 AVR 336
2 AT86RF401 336
3 AT90 336
AT90CAN32 336
AT90CAN128 337
AT90S1200 338
AT90S2313 339
AT90S2323 339
AT90S2333 340
AT90S2343 340
AT90S4414 342
AT90S4433 342
AT90S4434 344
AT90S8515 345
AT90S8535 345
AT90PWM2-3 346

AT90PWM216 ... 347
AT90US82 ... 348
AT90USB162 ... 349
AT90USB646 ... 350
AT90USB1286 ... 351
AT90USB1287 ... 352
4 ATTINY ... 352
ATTINY12 ... 352
ATTINY13 ... 353
ATTINY13A ... 353
ATTINY15 ... 353
ATTINY20 ... 353
ATTINY22 ... 354
ATTINY24 ... 354
ATTINY25 ... 355
ATTINY26 ... 355
ATTINY43U ... 356
ATTINY44 ... 356
ATTINY45 ... 356
ATTINY48 ... 357
ATTINY84 ... 358
ATTINY85 ... 358
ATTINY87 ... 358
ATTINY88 ... 359
ATTINY167 ... 359
ATTINY261 ... 360
ATTINY441 ... 360
ATTINY461 ... 361
ATTINY828 ... 361
ATTINY841 ... 362
ATTINY861 ... 363
ATTINY1634 ... 363
ATTINY2313 ... 363
ATTINY2313A ... 364
ATTINY4313 ... 365
ATTINY4313A ... 365
5 ATMEGA ... 366
ATMEGA8 ... 366
ATMEGA8A ... 366
ATMEGA8U2 ... 366
ATMEGA16 ... 368
ATMEGA16A ... 369
ATMEGA16U2 ... 369
ATMEGA16U4 ... 371
ATMEGA16M1 ... 372
ATMEGA32 ... 373
ATMEGA32A ... 374
ATMEGA32C1 ... 375
ATMEGA32M1 ... 376
ATMEGA32U2 ... 376
ATMEGA32U4 ... 378
ATMEGA48 ... 379
ATMEGA48P-ATMEGA48PA ... 379
ATMEGA48PB ... 380
ATMEGA64 ... 381
ATMEGA64C1 ... 382
ATMEGA64M1 ... 383

ATMEGA88 ... 383
ATMEGA88A ... 384
ATMEGA88P-ATMEGA88PA ... 385
ATMEGA88PB ... 386
ATMEGA103 ... 386
ATMEGA128 ... 388
ATMEGA128RFA1 ... 389
ATMEGA161 ... 390
ATMEGA162 ... 390
ATMEGA163 ... 391
ATMEGA164P ... 392
ATMEGA164PA ... 392
ATMEGA165 ... 394
ATMEGA165A ... 394
ATMEGA168 ... 396
ATMEGA168P ... 396
ATMEGA168PB ... 397
ATMEGA169 ... 397
ATMEGA169P ... 398
ATMEGA169PA ... 399
ATMEGA323 ... 400
ATMEGA324A ... 401
ATMEGA324P ... 402
ATMEGA324PA ... 402
ATMEGA324PB ... 403
ATMEGA325 ... 404
ATMEGA328 ... 405
ATMEGA328P ... 405
ATMEGA328PB ... 406
ATMEGA329 ... 407
ATMEGA406 ... 407
ATMEGA603 ... 408
ATMEGA640 ... 410
ATMEGA644 ... 411
ATMEGA644P ... 411
ATMEGA644PA ... 412
ATMEGA645 ... 413
ATMEGA649 ... 414
ATMEGA649PA ... 414
ATMEGA1280 ... 416
ATMEGA1281 ... 417
ATMEGA1284 ... 418
ATMEGA1284P ... 418
ATMEGA2560 ... 420
ATMEGA2561 ... 421
ATMEGA3250P ... 422
ATMEGA6450P ... 423
ATMEGA8515 ... 424
ATMEGA8535 ... 424
6 ATXMEGA ... 425
ATXMEGA ... 425
ATXMEGA8E5 ... 433
ATXMEGA16A4 ... 434
ATXMEGA16D4 ... 435
ATXMEGA16E5 ... 436
ATXMEGA32A4 ... 437
ATXMEGA32A4U ... 439

ATXMEGA32D4 439
ATXMEGA32E5 440
ATXMEGA64A1 441
ATXMEGA64A3 443
ATXMEGA64D3 443
ATXMEGA64D4 444
ATXMEGA128A1 445
ATXMEGA128A3 447
ATXMEGA128A4U 448
ATXMEGA128B1 448
ATXMEGA128B3 450
ATXMEGA128C3 450
ATXMEGA128D3 451
ATXMEGA128D4 452
ATXMEGA192A3 453
ATXMEGA192D3 454
ATXMEGA256A3 455
ATXMEGA256A3B 456
ATXMEGA256A3BU 458
ATXMEGA256D3 458
ATXMEGA384C3 460

7 XTINY 460

XTINY 460
ATTINY202 465
ATTINY204 466
ATTINY212 467
ATTINY214 468
ATTINY402 469
ATTINY404 470
ATTINY406 471
ATTINY412 472
ATTINY414 473
ATTINY416 474
ATTINY417 475
ATTINY804 476
ATTINY806 477
ATTINY807 478
ATTINY814 479
ATTINY816 480
ATTINY817 481
ATTINY1604 482
ATTINY1606 483
ATTINY1607 484
ATTINY1614 485
ATTINY1616 486
ATTINY1617 487
ATTINY3216 488
ATTINY3217 489

8 ATMEGAX 490

MEGAX 490
ATMEGA808 491
ATMEGA1608 493
ATMEGA3208 495
ATMEGA4808 497
ATMEGA4809 499

9 AVRX 503

AVRX 503

AVR16DD14 ... 506
AVR16DD20 ... 506
AVR16DD28 ... 508
AVR16DD32 ... 510
AVR16EA28 ... 511
AVR16EA32 ... 513
AVR16EA48 ... 514
AVR32DA28 ... 515
AVR32DA32 ... 516
AVR32DA48 ... 517
AVR32DB28 ... 518
AVR32DB32 ... 519
AVR32DB48 ... 520
AVR32DD14 ... 521
AVR32DD20 ... 522
AVR32DD28 ... 524
AVR32DD32 ... 526
AVR32EA28 ... 527
AVR32EA32 ... 529
AVR32EA48 ... 530
AVR64DA28 ... 531
AVR64DA32 ... 532
AVR64DA48 ... 533
AVR64DA64 ... 534
AVR64DB28 ... 535
AVR64DB32 ... 536
AVR64DB48 ... 537
AVR64DB64 ... 538
AVR64DD14 ... 539
AVR64DD20 ... 540
AVR64DD28 ... 542
AVR64DD32 ... 544
AVR64EA28 ... 545
AVR64EA32 ... 547
AVR64EA48 ... 548
AVR128DA28 ... 549
AVR128DA32 ... 550
AVR128DA48 ... 551
AVR128DA64 ... 552
AVR128DB28 ... 553
AVR128DB32 ... 554
AVR128DB48 ... 555
AVR128DB64 ... 556

Part VI BASCOM Language Fundamentals 559

1 Changes compared to BASCOM-8051 ... 559
2 Language Fundamentals ... 560
3 Mixing ASM and BASIC ... 573
4 Assembler mnemonics ... 578
5 Reserved Words ... 583
6 Error Codes ... 583
7 Newbie problems ... 589
8 Tips and tricks ... 595
9 ASCII chart ... 600

Part VII BASCOM Language Reference 604

1 #AUTOCODE 604
2 #IF ELSE ELSEIF ENDIF 604
3 Compiler Directives 606
$AESKEY 606
$ASM 606
$BAUD 607
$BAUD1 608
$BGF 609
$BIGSTRINGS 611
$BOOT 612
$BOOTVECTOR 613
$CRYPT 624
$CRYSTAL 625
$DATA 626
$DBG 628
$DEFAULT 630
$EEPLEAVE 630
$EEPROM 631
$EEPROMHEX 632
$EEPROMSIZE 633
$EXTERNAL 634
$FILE 635
$FORCESOFTI2C 635
$FRAMEPROTECT 637
$FRAMESIZE 641
$HWSTACK 648
$HWCHECK, $SWCHECK, $SOFTCHECK 651
$INC 653
$INCLUDE 654
$INITMICRO 656
$LCD 657
$LCDPUTCTRL 659
$LCDPUTDATA 661
$LCDRS 662
$LCDVFO 664
$LIB 665
$LOADER 667
$LOADERSIZE 683
$MAP 684
$NOCOMPILE 685
$NOFRAMEPROTECT 685
$NOINIT 686
$NORAMCLEAR 686
$NORAMPZ 686
$NOTRANSFORM 687
$NOTYPECHECK 688
$PROJECTTIME 688
$PROG 689
$PROGRAMMER 690
$REDUCEIVR 692
$REGFILE 694
$RESOURCE 694
$ROMSTART 697
$SERIALINPUT 698

$SERIALINPUT1 ... 700
$SERIALINPUT2LCD ... 701
$SERIALOUTPUT ... 701
$SERIALOUTPUT1 ... 702
$SIM ... 703
$STACKDUMP ... 703
$SWSTACK ... 705
$TIMEOUT ... 708
$TINY ... 710
$TYPECHECK ... 710
$USER ... 711
$VERSION ... 712
$WAITSTATE ... 712
$XA ... 713
$XRAMSIZE ... 713
$XRAMSTART ... 714
$XTEAKEY ... 715
4 1WIRE ... 716
1WIRECOUNT ... 716
1WRESET ... 718
1WREAD ... 720
1WSEARCHFIRST ... 723
1WSEARCHNEXT ... 725
1WVERIFY ... 727
1WWRITE ... 729
5 ADR , ADR2 ... 731
6 ALIAS ... 735
7 Math ... 736
ABS ... 736
ACOS ... 737
AND ... 738
ASIN ... 741
ATN ... 742
ATN2 ... 743
CHECKFLOAT ... 744
COS ... 746
COSH ... 747
DEG2RAD ... 747
EXP ... 748
FIX ... 749
FRAC ... 750
INT ... 751
LOG10 ... 752
LOG ... 752
NOT ... 753
OR ... 755
POWER ... 758
QSIN ... 760
QCOS ... 764
RAD2DEG ... 764
ROUND ... 765
SGN ... 766
TANH ... 767
TAN ... 767
SQR ... 768
SINH ... 769

SIN ... 769
XOR ... 770
8 AVR-DOS File I/O ... 773
BLOAD ... 773
BSAVE ... 774
CHDIR ... 775
CLEARATTR ... 776
DIR ... 777
DISKFREE ... 778
DISKSIZE ... 779
DriveCheck ... 780
DriveGetIdentity ... 780
DriveInit ... 781
DriveReset ... 782
DriveReadSector ... 782
DriveWriteSector ... 784
EOF ... 785
FILEATTR ... 786
FILEDATE ... 787
FILEDATETIME ... 787
FILELEN ... 788
FILETIME ... 789
FLUSH ... 790
FREEFILE ... 791
GETATTR ... 791
INITFILESYSTEM ... 792
KILL ... 793
LINEINPUT ... 794
LOC ... 795
LOF ... 796
MKDIR ... 796
NAME ... 797
RMDIR ... 798
SETATTR ... 798
WRITE ... 801
9 BITWAIT ... 803
10 BITS ... 804
11 BREAK ... 805
12 BYVAL ... 805
13 CALL ... 806
14 CHECKSUM ... 808
CHECKSUM CHECKSUMXOR ... 808
CRC8 ... 809
CRC8UNI ... 811
CRC16 ... 813
CRC16UNI ... 815
CRC32 ... 817
CRCMB ... 818
15 Conversion ... 819
ASC ... 819
BCD ... 822
BIN ... 824
BINVAL ... 825
BIN2GRAY ... 825
CHR ... 827

FORMAT ... 828
FUSING ... 830
GRAY2BIN ... 831
HEXVAL ... 832
HEX ... 833
MANCHESTERDEC ... 834
MANCHESTERENC ... 835
STR ... 836
STR2DIGITS ... 837
VAL ... 839
16 CAN ... 840
CANBAUD ... 840
CANGETINTS ... 841
CANID ... 842
CANCLEARALLMOBS ... 842
CANCLEARMOB ... 843
CANRECEIVE ... 843
CANRESET ... 844
CANSELPAGE ... 844
CANSEND ... 845
17 CLEAR ... 846
18 CLOCKDIVISION ... 847
19 CLOSE ... 848
20 COMPARE ... 851
21 CONFIGURATION ... 853
CONFIG ... 853
CONFIG 1WIRE ... 857
CONFIG ACI ... 861
CONFIG ACX ... 861
CONFIG ACAX|ACBX ... 862
CONFIG ADC ... 864
CONFIG ADCA|ADCB ... 867
CONFIG ADC0-ADCX ... 880
CONFIG ATEMU ... 883
CONFIG BASE ... 886
CONFIG BCCARD ... 887
CONFIG CANBUSMODE ... 889
CONFIG CANMOB ... 893
CONFIG CLOCK ... 895
CONFIG CLOCKDIV ... 909
CONFIG COM1 ... 909
CONFIG COM2 ... 911
CONFIG COMx ... 913
CONFIG DACA|DACB ... 918
CONFIG DACX ... 923
CONFIG DATE ... 925
CONFIG DCF77 ... 927
CONFIG DEBOUNCE ... 935
CONFIG DMA ... 936
CONFIG DMACHx ... 937
CONFIG EDMA ... 944
CONFIG EDMAx ... 945
CONFIG DMXSLAVE ... 949
CONFIG DP ... 951
CONFIG EEPROM ... 952
CONFIG ERROR ... 953

CONFIG EVENT_SYSTEM 953
CONFIG EVENT_SYSTEM XTINY 956
CONFIG EXTENDED_PORT 958
CONFIG FT800 959
CONFIG FUSES 962
CONFIG GRAPHLCD 964
CONFIG HITAG 970
CONFIG I2CBUS 974
CONFIG I2CDELAY 975
CONFIG I2CSLAVE 978
CONFIG INPUT 983
CONFIG INPUTBIN 984
CONFIG INTx 985
CONFIG INTVECTORSELECTION 987
CONFIG KBD 988
CONFIG KEYBOARD 989
CONFIG LCD 992
CONFIG LCDBUS 998
CONFIG LCDMODE 1001
CONFIG LCDPIN 1001
CONFIG MODBUS 1005
CONFIG OPAMP 1005
CONFIG OSC XMEGA 1008
CONFIG OSC XTINY 1009
CONFIG PORT 1011
CONFIG PORT_MUX 1014
CONFIG POWERMODE 1017
CONFIG POWER_REDUCTION 1023
CONFIG PRIORITY XMEGA 1026
CONFIG PRIORITY XTINY 1027
CONFIG PRINT 1028
CONFIG PRINTBIN 1029
CONFIG PS2EMU 1030
CONFIG RAINBOW 1033
CONFIG RC5 1037
CONFIG RC5SEND 1042
CONFIG RND 1044
CONFIG SDA 1046
CONFIG SCL 1049
CONFIG SERIALIN 1051
CONFIG SERIALOUT 1057
CONFIG SINGLE 1059
CONFIG SHIFTIN 1060
CONFIG SPI 1061
CONFIG SPIx XTINY 1065
CONFIG SPIx XMEGA 1068
CONFIG SERVOS 1071
CONFIG STRCHECK 1075
CONFIG SUBMODE 1079
CONFIG SYSCLOCK XMEGA 1081
CONFIG SYSCLOCK XTINY 1082
CONFIG TCA0 1084
CONFIG TCB0-TCB1 1087
CONFIG TCD0 1090
CONFIG TCXX 1094
CONFIG TCPIP 1098
CONFIG TIMER0 1109
CONFIG TIMER1 1112

CONFIG TIMER2 ... 1115
CONFIG TWI, TWIx ... 1116
CONFIG TWISLAVE ... 1123
CONFIG TWIxSLAVE ... 1128
CONFIG USB ... 1132
CONFIG USI ... 1138
CONFIG VARPTRMODE ... 1142
CONFIG VPORT ... 1144
CONFIG VREF ... 1147
CONFIG VREGPWR ... 1148
CONFIG WAITSUART ... 1149
CONFIG WATCHDOG ... 1149
CONFIG X10 ... 1156
CONFIG XPIN ... 1158
CONFIG XRAM ... 1161
CONFIG ZCDx ... 1167

22 CONTINUE ... 1168
23 CONST ... 1170
24 COUNTER0 and COUNTER1 ... 1172
25 CPEEK ... 1173
26 CPEEKH ... 1174
27 CRYSTAL ... 1176
28 DATA ... 1177
29 Date and Time ... 1180

DAYOFWEEK ... 1181
DAYOFYEAR ... 1190
DATE$ ... 1191
DATE ... 1193
SECELAPSED ... 1202
SECOFDAY ... 1203
SYSSEC ... 1204
SYSSECELAPSED ... 1206
SYSDAY ... 1206
TIME$ ... 1208
TIME ... 1209

30 DBG ... 1210
31 DCF77TIMEZONE ... 1210
32 DEBUG ... 1211
33 DEBOUNCE ... 1212
34 DECR ... 1214
35 DECLARE FUNCTION ... 1215
36 DECLARE SUB ... 1221
37 DEFxxx ... 1227
38 DELAY ... 1227
39 DIM ... 1228
40 DISABLE ... 1240
41 DO-LOOP ... 1243
42 DTMFOUT ... 1244
43 ECHO ... 1247

44 ELSE 1248
45 ENABLE 1250
46 ENCODER 1254
47 END 1256
48 EXIT 1257
49 FLIP 1259
50 FOR-NEXT 1260
51 GET 1262
52 GETADC 1265
53 GETATKBD 1270
54 GETATKBDRAW 1274
55 GETKBD 1275
56 GETRC 1277
57 GETRC5 1278
58 GETREG 1284
59 GOSUB 1284
60 GOTO 1286
61 HIGH 1287
62 HIGHW 1288
63 Encryption-Decryption 1288
AESENCRYPT 1288
AESDECRYPT 1290
DESENCRYPT 1292
DESDECRYPT 1294
XTEAENCODE 1297
XTEADECODE 1298
64 I2C-TWI 1300
I2CINIT 1300
I2CRECEIVE 1303
I2CSEND 1305
I2START,I2CSTOP, I2CRBYTE, I2CWBYTE, I2CREPSTART 1306
65 IDLE 1313
66 IF-THEN-ELSE-END IF 1313
67 INCR 1314
68 INP 1315
69 LCD Commands 1317
BOX 1317
BOXFILL 1319
CIRCLE 1319
CLS 1322
CURSOR 1325
DEFLCDCHAR 1328
DISPLAY 1329
FOURTHLINE 1332
GLCDCMD 1333
GLCDDATA 1333
HOME 1334
INITLCD 1334

LCD ... 1335
LCDAUTODIM ... 1338
LCDAT ... 1339
LCDCMD ... 1341
LCDDATA ... 1341
LCDCONTRAST ... 1342
LCDFONT ... 1342
LINE ... 1344
LOCATE ... 1347
LOWERLINE ... 1347
PSET ... 1348
RGB8TO16 ... 1350
SETFONT ... 1351
SHIFTCURSOR ... 1353
SHIFTLCD ... 1354
SHOWPIC ... 1355
SHOWPICE ... 1355
THIRDLINE ... 1357
UPPERLINE ... 1357
70 LOAD ... 1358
71 LOADADR ... 1358
72 LOADLABEL ... 1359
73 LOADWORDADR ... 1359
74 LOCAL ... 1360
75 LOOKDOWN ... 1363
76 LOOKUP ... 1365
77 LOOKUPSTR ... 1366
78 LOW ... 1367
79 MACRO ... 1368
80 MAKEBCD ... 1369
81 MAKEDEC ... 1369
82 MAKEINT ... 1370
83 MAX ... 1370
84 MEMCOPY ... 1372
85 MEMFILL ... 1374
86 MIN ... 1375
87 MOD ... 1376
88 NBITS ... 1377
89 NOP ... 1378
90 ON INTERRUPT ... 1379
91 ON VALUE ... 1383
92 OPEN ... 1386
93 OUT ... 1394
94 PEEK ... 1395
95 POKE ... 1396
96 POPALL ... 1397
97 POWER MODE ... 1397

98 POWERDOWN 1398
99 POWERSAVE 1399
100 PS2MOUSEXY 1399
101 PULSEIN 1400
102 PULSEOUT 1401
103 PUSHALL 1402
104 PUT 1402
105 RC5SEND 1405
106 RC5SENDEXT 1409
107 RC6SEND 1411
108 READ 1413
109 READEEPROM 1415
110 READHITAG 1417
111 READMAGCARD 1420
112 REDO 1422
113 READSIG 1424
114 READUSERSIG 1427
115 REM 1427
116 RESET 1428
117 RESTORE 1429
118 RETURN 1430
119 RND 1431
120 ROTATE 1432
121 SEEK 1434
122 SELECT-CASE-END SELECT 1437
123 SET 1438
124 SETREG 1441
125 SENDSCAN 1441
126 SENDSCANKBD 1443
127 SHIFT 1447
128 SHIFTIN 1449
129 SHIFTOUT 1453
130 SONYSEND 1454
131 SIZEOF 1457
132 SORT 1458
133 SOUND 1460
134 RAINBOW 1462
RB_SELECTCHANNEL 1462
RB_SETCOLOR 1463
RB_SEND 1464
RB_CHANGEPIN 1465
RB_ADDCOLOR 1468
RB_ANDCOLOR 1468
RB_ORCOLOR 1470

RB_SUBCOLOR ... 1470
RB_CLEARSTRIPE ... 1471
RB_CLEARCOLORS ... 1472
RB_FILL ... 1472
RB_FILLCOLORS ... 1473
RB_FILLSTRIPE ... 1475
RB_SWAPCOLOR ... 1475
RB_ROTATELEFT ... 1476
RB_ROTATERIGHT ... 1477
RB_SHIFTLEFT ... 1478
RB_SHIFTRIGHT ... 1479
RB_SETTABLECOLOR ... 1479
RB_GETCOLOR ... 1483
RB_LOOKUPCOLOR ... 1484
RB_COLOR ... 1484
RB_COPY ... 1486

135 Serial Data RS232-RS485 ... 1488

BAUD ... 1488
BAUD1-BAUDx ... 1489
BUFSPACE ... 1491
INKEY ... 1492
INPUT ... 1493
INPUTHEX ... 1496
INPUTBIN ... 1497
ISCHARWAITING ... 1498
MAKEMODBUS ... 1499
PRINT ... 1501
PRINTBIN ... 1504
SERIN ... 1506
SPC ... 1508
SEROUT ... 1509
WAITKEY ... 1511

136 SPI ... 1513

SPIIN ... 1513
SPIINIT ... 1514
SPIMOVE ... 1515
SPIOUT ... 1518
SPI1INIT, SPI1IN, SPI1OUT, SPI1MOVE ... 1519

137 STRINGS ... 1519

CHARPOS ... 1519
DELCHAR ... 1520
DELCHARS ... 1521
INSERTCHAR ... 1522
JOIN ... 1524
INSTR ... 1526
LCASE ... 1527
LEFT ... 1528
LEN ... 1529
LTRIM ... 1529
MID ... 1530
REPLACECHARS ... 1530
RIGHT ... 1531
QUOTE ... 1532
RTRIM ... 1532
SPACE ... 1533
SPLIT ... 1534
STRING ... 1536

TRIM 1536
UCASE 1537
138 START 1538
139 STCHECK 1540
140 STOP 1544
141 SUB 1545
142 SWAP 1545
143 TCP/IP 1547
BASE64DEC 1547
BASE64ENC 1549
GETDSTIP 1551
GETDSTPORT 1551
GETSOCKET 1553
GETTCPREGS 1555
IP2STR 1555
MAKETCP 1556
SETIPPROTOCOL 1556
SETTCP 1559
SETTCPREGS 1560
SNTP 1562
SOCKETCLOSE 1564
SOCKETCONNECT 1567
SOCKETDISCONNECT 1570
SOCKETLISTEN 1571
SOCKETSTAT 1571
TCPCHECKSUM 1574
TCPREAD 1576
TCPREADHEADER 1577
TCPWRITE 1578
TCPWRITESTR 1579
UDPREAD 1582
UDPREADHEADER 1585
UDPWRITE 1588
UDPWRITESTR 1589
URL2IP 1592
144 TOGGLE 1595
145 TYPE 1597
146 VARPTR 1604
147 VER 1605
148 VERSION 1606
149 WAIT 1607
150 WAITMS 1607
151 WAITUS 1608
152 WHILE-WEND 1609
153 WRITEDAC 1610
154 WRITEEEPROM 1611
155 X10DETECT 1613
156 X10SEND 1615
Part VIII ASM Libraries and Add-Ons 1618

1 FT800 1619

Commands **1621**

AlphaFunc 1625
Begin_G 1625
BitmapHandle 1627
BitmapLayout 1627
BitmapSize 1628
BitmapSource 1630
BitmapTransform 1631
BlendFunc 1633
Call_C 1634
Cell 1635
Clear_B 1635
ClearColorA 1636
ClearColorRGB 1637
ClearColorRGBdw 1638
ClearStencil 1639
ClearTag 1639
ClearScreen 1640
CMD8 1640
CMD16 1641
CMD32 1641
CmdAppend 1642
CmdBgColor 1642
CmdButton 1643
CmdCalibrate 1644
CmdCalibratex 1645
CmdClock 1645
CmdColdStart 1648
CmdDial 1649
CmdDlStart 1650
CmdFgColor 1650
CMDFTSTACK 1651
CmdGauge 1652
CmdGetMatrix 1655
CmdGetPtr 1655
CmdGradColor 1656
CmdGradient 1657
CmdInflate 1658
CmdInterrupt 1659
CmdKeys 1659
CmdLoadIdentity 1661
CmdLoadImage 1662
CmdLogo 1662
CmdMemCpy 1663
CmdMemCrc 1663
CmdMemSet 1664
CmdMemWrite 1664
CmdMemZero 1665
CmdNumber 1665
CmdProgress 1667
CmdRegRead 1668
CmdRotate 1669
CmdRotateA 1670
CmdScale 1671
CmdScreenSaver 1672
CmdScrollBar 1672

CmdSetFont ... 1674
CmdSetMatrix ... 1674
CmdSketch ... 1674
CmdSlider ... 1675
CmdSnapShot ... 1677
CmdSpinner ... 1677
CmdStop ... 1680
CmdSwap ... 1680
CmdText ... 1680
CmdToggle ... 1682
CmdTrack ... 1684
CmdTranslate ... 1686
CmdTranslateP ... 1687
Color_A ... 1688
ColorMask ... 1688
ColorRGB ... 1689
ColorRGBdw ... 1690
Display_E ... 1690
End_G ... 1691
Jump ... 1691
LineWidth ... 1692
Macro_R ... 1693
PointSize ... 1693
RD8 ... 1694
RD16 ... 1694
RD32 ... 1695
RestoreContext ... 1696
Return_C ... 1696
SaveContext ... 1697
ScissorSize ... 1697
ScissorXY ... 1698
StencilFunc ... 1699
StencilMask ... 1700
StencilOp ... 1700
Tag ... 1701
TagMask ... 1702
Vertex2f ... 1702
Vertex2ii ... 1703
UpdateScreen ... 1704
WaitCmdFifoEmpty ... 1705
WR8 ... 1705
WR16 ... 1706
WR32 ... 1706

Getting Started ... 1707
How to add another SPI device with the FT800 ... 1708
How to Screen Capture ... 1709
Demos ... 1710

2 EXTENDED I2C ... 1741
3 FM24C16 ... 1743
4 FM24C64_256 ... 1745
5 FM24C64_256-XMEGA ... 1746
6 FM25C256 ... 1748
7 HEXVAL ... 1753
8 I2C_MULTIBUS ... 1753
9 I2C_TWI ... 1753

10 I2C_TWI-MULTI 1754
11 I2C_USI 1756
12 I2C_USI_SLAVE 1757
13 I2CV2 1763
14 MCSBYTE 1764
15 MCSBYTEINT 1764
16 PULSEIN 1764
17 SERIN 1765
18 TCPIP 1765
19 M128-1wire-PortF 1766
20 TVOUT 1766
21 RAINBOWBSC 1774
22 LCD 1774
LCD4BUSY 1774
LCD4_anypin_oled_RS0010 1775
LCD_RX1602A5 1776
LCD4.LIB 1776
LCD4E2 1777
GLCD 1778
GLCDSED 1778
PCF8533 1778
LCD-EPSON 1780
LCD_DOGS104a_I2C 1780
glcdR7565R 1784
glcdSSD1325_96x64 1785
GLCDEADOGMXL240-7-I2C 1786
GLCDdSSD1306-I2C 1788
LCD_I2C_PCF8574 1790
23 AVR-DOS 1794
AVR-DOS File System 1794
MMCSD_HC.LIB 1823
24 CF Card 1823
Compact FlashCard Driver 1823
Elektor CF-Interface 1825
XRAM CF-Interface for simulation 1826
New CF-Card Drivers 1827
25 Floating Point 1827
FP_TRIG 1827
DOUBLE 1830
26 I2C SLAVE 1830
CONFIG I2CSLAVE 1830
I2C TWI Slave 1831
27 SPI 1831
SPISLAVE 1831
28 DATE TIME 1834
EUROTIMEDATE 1834
DATETIME 1835
29 PS2-AT Mouse and Keyboard Emulation 1836
AT_EMULATOR 1836
PS2MOUSE_EMULATOR 1836

30 BCCARD 1836
BCCARD 1836
BCDEF 1837
BCCALL 1838
BCRESET 1844
31 USB 1845
USB Add On 1845
32 MODBUS Slave/Server 1858

Part IX Tools 1862

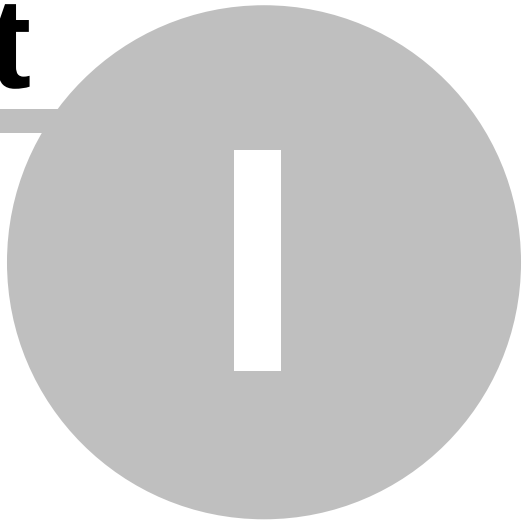
1 LCD RGB-8 Converter 1862
2 BASCOMP 1863

Part X International Resellers 1869

1 International Resellers 1869

Index 1870

Part



1 Index

Version 2.0.8.7 document build 102

1.1 Keyword Reference

1WIRE

1Wire routines allow you to communicate with Dallas 1wire chips.
1WRESET 718 , 1WREAD 720 , 1WWRITE 729 , 1WSEARCHFIRST 723 , 1WSEARCHNEXT 725 ,1WVERIFY 727 , 1WIRECOUNT 716

CAN

CONFIG CANBUSMODE 889, CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843 , CANID 842, CANSELPAGE 844, CANGETINTS 841

Conditions and Loops

Conditions execute a part of the program depending on a condition being True or False
IF-THEN-ELSE-END IF 1313 , WHILE-WEND 1609 , ELSE 1248 , DO-LOOP 1243 , SELECT CASE - END SELECT 1437 , FOR-NEXT 1260 , CONTINUE 1168, REDO 1422

Conditional Compilation

#IF #ELSE #ELSEIF #ENDIF , VAREXIST 604

Configuration

Configuration commands initialize the hardware to the desired state.
CONFIG 853 , CONFIG ACI 861 , CONFIG ADC 864 , CONFIG ADCx 867 , CONFIG BCCARD 887 , CONFIG CLOCK 895 , CONFIG COM1 909 , CONFIG COM2 911 , CONFIG DAC 918 , CONFIG DATE 925 , CONFIG DMXSLAVE 949, CONFIG EEPROM 952 ,CONFIG EXTENDED_PORT 958 , CONFIG PS2EMU 1030 , CONFIG ATEMU 883 , CONFIG I2CSLAVE 978 , CONFIG INPUT 983, CONFIG GRAPHLCD 964 , CONFIG KEYBOARD 989 , CONFIG TIMER0 1109 , CONFIG TIMER1 1112 , CONFIG LCDBUS 998 , CONFIG LCDMODE 1001 , CONFIG 1WIRE 857 , CONFIG LCD 992 , CONFIG OSC 1008, CONFIG SERIALOUT 1057 , CONFIG SERIALIN 1051 , CONFIG SPI 1061 , CONFIG SPIx 1068, CONFIG SYSCLOCK 1081 , CONFIG LCDPIN 1001 , CONFIG PRIORITY 1026 , CONFIG SDA 1046 , CONFIG SCL 1049 , CONFIG DEBOUNCE 935 , CONFIG WATCHDOG 1149 , CONFIG PORT , 1011COUNTER0 AND COUNTER1 1172 , CONFIG TCPIP 1098 , CONFIG TWISLAVE 1123 , CONFIG SINGLE 1059 , CONFIG X10 1156 , CONFIG XRAM 1161 , CONFIG USB 1132 , CONFIG DP 951 , CONFIG TCXX 1094 , CONFIG VPORT 1144 CONFIG ERROR 953 , CONFIG POWER REDUCTION 1023, CONFIG EVENT_SYSTEM 953 , CONFIG DMA 936, CONFIG DMACHx 937 , CONFIG SUBMODE 1079 , CONFIG POWERMODE 1017 , CONFIG XPIN 1158 , CONFIG FT800 959 , CONFIG I2CBUS 974 , CONFIG EDMA 944 , CONFIG EDMAx 945 , CONFIG INPUTBIN 984 , CONFIG MODBUS 1005 , CONFIG PORT_MUX 1014 , CONFIG VREF 1147 , , CONFIG TCA 1084, CONFIG TCB 1087, CONFIG TCD 1090 , CONFIG RC5 1037, CONFIG RC5SEND 1042 , CONFIG VARPTRMODE 1142

Conversion

A conversion routine is a function that converts a number or string from one form to another.
BCD 822 , GRAY2BIN 831 , BIN2GRAY 825 , BIN 824 , MAKEBCD 1369 , MAKEDEC 1369 , MAKEINT 1370 , FORMAT 828 , FUSING 830 , BINVAL 825 , CRC8 809 , CRC16 813 , CRC16UNI 815 , CRC32 817 , HIGH 1287 , HIGHW 1288 , LOW 1367 , AESENCRYPT 1288 , AESDECRYPT 1290 , FLIP 1259 , CRCMB 818 , CRC8UNI 811 , MANCHESTERDEC 834, MANCHESTERENC 835 , DESENCRYPT 1292 , DESDECRYPT 1294

DateTime
Date Time routines can be used to calculate with date and/or times.
DATE 1193 , TIME 1209 , DATE$ 1191 , TIME$ 1208 , DAYOFWEEK 1181 , DAYOFYEAR 1190 , SECOFDAY 1203 , SECELAPSED 1202 , SYSDAY 1206 , SYSSEC 1204 , SYSSECELAPSED 1206

Delay
Delay routines delay the program for the specified time.
WAIT 1607 , WAITMS 1607 , WAITUS 1608 , DELAY 1227

Directives
Directives are special instructions for the compiler. They can override a setting from the IDE.
$ASM 606 , $BAUD 607 , $BAUD1 608 , $BIGSTRINGS 611 , $BGF 609 , $BOOT 612 , $CRYSTAL 625 , $DATA 626 , $DBG 628 , $DEFAULT 630 , $EEPLEAVE 630 , $EEPROM 631 , $EEPROMHEX 632 , $EEPROMSIZE 633 , $EXTERNAL 634 , $HWSTACK 648 , $INC 653 , $INCLUDE 654 , $INITMICRO 656 , $LCD 657 , $LCDRS 662 , $LCDPUTCTRL 659 , $LCDPUTDATA 661 , $LCDVFO 664 , $LIB 665 , $LOADER 667 , $LOADERSIZE 683 , $MAP 684 , $NOCOMPILE 685 , $NOINIT 686 , $NORAMCLEAR 686 , $NORAMPZ 686 , $PROJECTTIME 688 , $PROG 689 , $PROGRAMMER 690 , $REGFILE 694 , $RESOURCE 694 , $ROMSTART 697 $SERIALINPUT 698 , $SERIALINPUT1 700 , $SERIALINPUT2LCD 701 , $SERIALOUTPUT 701 , $SERIALOUTPUT1 702 , $SIM 703 , $SWSTACK 705 , $TIMEOUT 708 , $TINY 710 , $WAITSTATE 712 , $XRAMSIZE 713 , $XRAMSTART 714 , $XA 713 , $CRYPT 624 , $NOTRANSFORM 687 , $FILE 635 , $AESKEY 606 , $XTEAKEY 715 , $STACKDUMP 703 , $NOFRAMEPROTECT 685 , $FRAMEPROTECT 637 , $FORCESOFTI2C 635 , $BOOTVECTOR 613 , $REDUCEIVR 606 , $TYPECHECK 710 , $NOTYPECHECK 688

File
File commands can be used with AVR-DOS, the Disk Operating System for AVR.
BSAVE 774 , BLOAD 773 , GET 1262 , VER 1605 , DISKFREE 778 , DIR 777 , DriveReset 782 , DriveInit 781 , LINE INPUT 794 , INITFILESYSTEM 792 , EOF 785 , WRITE 801 , FLUSH 790 , FREEFILE 791 , FILEATTR 786 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , FILELEN 788 , SEEK 1434 , KILL 793 , DriveGetIdentity 780 , DriveWriteSector 784 , DriveReadSector 782 , LOC 795 , LOF 796 , PUT 1402 , OPEN 1386 , CLOSE 848 , CHDIR 775 , MKDIR 796 , RMDIR 798 , NAME 797 , GETATTR 791 , SETATTR 798 , CLEARATTR 776

Graphical LCD
Graphical LCD commands extend the normal text LCD commands.
GLCDCMD 1333 , GLCDDATA 1333 , SETFONT 1351 , LINE 1344 , PSET 1348 , SHOWPIC 1355 , SHOWPICE 1355 , CIRCLE 1319 , BOX 1317 , RGB8TO16 1350

I2C
I2C commands allow you to communicate with I2C chips with the TWI hardware or with emulated I2C hardware.
I2CINIT 1300 , I2CRECEIVE 1303 , I2CSEND 1305 , I2CSTART, I2CREPSTART, I2CSTOP, I2CRBYTE,I2CWBYTE 1306

IO
I/O commands are related to the I/O pins and ports of the processor chip.
ALIAS 735 , BITWAIT 803 , TOGGLE 1595 , RESET 1428 , SET 1438 , SHIFTIN 1449 , SHIFTOUT

1453 , DEBOUNCE 1212 , PULSEIN 1400 , PULSEOUT 1401

Micro

Micro statements are specific to the micro processor chip.

IDLE 1313 , POWER mode 1397 , POWERDOWN 1398 , POWERSAVE 1399 , ON INTERRUPT 1379 , ENABLE 1250 , DISABLE 1240 , START 1538 , END 1256 , VERSION 1606 , CLOCKDIVISION 847 , CRYSTAL 1176 , STOP 1544

Memory

Memory functions set or read RAM , EEPROM or flash memory.

ADR 731 , ADR2 731 , WRITEEEPROM 1611 , CPEEK 1173 , CPEEKH 1174 , PEEK 1395 , POKE 1396 , OUT 1394 , READEEPROM 1415 , DATA 1177 , INP 1315 , READ 1413 , RESTORE 1429 , LOOKDOWN 1363 , LOOKUP 1365 , LOOKUPSTR 1366 , LOADADR 1358 , LOADLABEL 1359 , LOADWORDADR 1359 , MEMCOPY 1372 , GETREG 1284 , SETREG 1441 , VARPTR 1604 , MEMFILL 1374

Remote Control

Remote control statements send or receive IR commands for remote control.

RC5SEND 1405 , RC6SEND 1411 , GETRC5 1278 , SONYSEND 1454

RS-232

RS-232 are serial routines that use the UART or emulate a UART.

BAUD 1488 , BAUD1 1489 , BUFSPACE 1491 , CLEAR 846 , ECHO 1247 , WAITKEY 1511 , ISCHARWAITING 1498 , INKEY 1492 , INPUTBIN 1497 , INPUTHEX 1496 , INPUT 1493 , PRINT 1501 , PRINTBIN 1504 , SERIN 1506 , SEROUT 1509 , SPC 1508 , MAKEMODBUS 1499

SPI

SPI routines communicate according to the SPI protocol with either hardware SPI or software emulated SPI.

SPIIN 1513 , SPIINIT 1514 , SPIMOVE 1515 , SPIOUT 1518 , SPI1IN 1513 , SPI1INIT 1514 , SPI1MOVE 1515 , SPI1OUT 1518

String

String routines are used to manipulate strings.

ASC 819 , CHARPOS 1519 , UCASE 1537 , LCASE 1527 , TRIM 1536 , SPLIT 1534 , LTRIM 1529 , INSTR 1526 , SPACE 1533 , STRING 1536 , RTRIM 1532 , LEFT 1528 , LEN 1529 , MID 1530 , RIGHT 1531 , VAL 839 , STR 836 , CHR 827 , CHECKSUM 808 , CHECKSUMXOR 808 , HEX 833 , HEXVAL 832 , QUOTE 1532 , REPLACECHARS 1530 , STR2DIGITS 837 , DELCHAR 1520 , DELCHARS 1521 , INSERTCHAR 1522 , JOIN 1524

TCP/IP

TCP/IP routines can be used with the W3100/IIM7000/IIM7010/W5100/W5200/ W5300 modules.

BASE64DEC 1547 , BASE64ENC 1549 , IP2STR 1555 , UDPREAD 1582 , UDPWRITE 1588 , UDPWRITESTR 1589 , TCPWRITE 1578 , TCPWRITESTR 1579 , TCPREAD 1576 , GETDSTIP 1551 , GETDSTPORT 1551 , SOCKETSTAT 1571 , SOCKETCONNECT 1567 , SOCKETLISTEN 1571 , GETSOCKET 1553 , SOCKETCLOSE 1564 , SETTCP 1559 , GETTCPREGS 1555 , SETTCPREGS 1560 , SETIPPROTOCOL 1556 , TCPCHECKSUM 1574 , SOCKETDISCONNECT 1570 , SNTP 1562 , TCPREADHEADER 1577 , UDPREADHEADER 1585 , URL2IP 1592

Text LCD
Text LCD routines work with normal text based LCD displays.
HOME 1334 , CURSOR 1325 , UPPERLINE 1357 , THIRDLINE 1357 , INITLCD 1334 , LOWERLINE 1347 , LCD 1335 , LCDAT 1339 , FOURTHLINE 1332 , DISPLAY 1329 , LCDCONTRAST 1342 , LOCATE 1347 , SHIFTCURSOR 1353 , DEFLCDCHAR 1328 , SHIFTLCD 1354 , CLS 1322 , LCDAUTODIM 1338 , LCDCMD 1341, LCDDATA 1341 , LCDFONT 1342

Trig & Math
Trig and Math routines work with numeric variables.
ACOS 737 , ASIN 741 , ATN 742 , ATN2 743 , EXP 748 , RAD2DEG 764 , FRAC 750 , TAN 767 , TANH 767 , COS 746 , COSH 747 , LOG 752 , LOG10 752 , ROUND 765 , ABS 736 , INT 751 , MAX 1370 , MIN 1375 , SQR 768 , SGN 766, POWER 758 , SIN 769 , SINH 769 , FIX 749 , INCR 1314 , DECR 1214 , DEG2RAD 747 , CHECKFLOAT 744 , MOD 1376 , QSIN 760 , QCOS 764 , AND 738, OR 755 , XOR 770 , NOT 753

Various
This section contains all statements that were hard to put into another group
CONST 1170 , DBG 1210 , DECLARE FUNCTION 1215 , DEBUG 1211, DECLARE SUB 1221 , DEFXXX 1227 , DIM 1228 , DTMFOUT 1244 , EXIT 1257 , ENCODER 1254 , GETADC 1265 , GETKBD 1275 , GETATKBD 1270 , GETRC 1277 , GOSUB 1284 , GOTO 1286 , LOCAL 1360 ,ON VALUE 1383 , POPALL 1397 , PS2MOUSEXY 1399 , PUSHALL 1402 , RETURN 1430 , RND 1431 , ROTATE 1432 , SENDSCAN 1441 , SENDSCANKBD 1443 , SHIFT 1447 , SOUND 1460 , STCHECK 1540 , SUB 1545 , SWAP 1545 , VARPTR 1604 , X10DETECT 1613 , X10SEND 1615 , READMAGCARD 1420 , REM 1427 , BITS 804 , BYVAL 805 , CALL 806 , #IF 604 , #ELSE 604 , #ENDIF 604 , READHITAG 1417 , SORT 1458 , XTEADECODE 1298 , XTEAENCODE 1297 , BREAK 805 , COMPARE 851 , NOP 1378 , SIZEOF 1457 , WRITEDAC 1610 , TYPE 1597

RAINBOW WS2812
Rainbow or WS2812 LED statements and functions.
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COLOR 1484 , RB_COPY 1486

FT800-FT801-FT810
CMD8 1640 , CMD16 1641 , CMD32 1641 , RD8 1694 , RD16 1694 , RD32 1695 , WR8 1705 , WR16 1706 , WR32 1706

XMEGA
READSIG 1424 , ATXMEGA 425

XTINY
XTINY 460

MEGAX
MEGAX 490

AVRX
AVRX 503

1.2 Changes

1.2.1 What is new in 2087

Modifications in 2087

- options, programmer shows an image (when available) of the selected programmer
- config varptrMode 1142 added to change behavior of Varptr()
- TYPE 1597 support added.
- xtiny low IO registers would not paint red/blue in the simulator.
- print fusing() with a single array would fail because of a bug that trashed a register.
- new $USER 711 directive added for creating **.usr** files for the Xtiny platform. This is a user signature data. Using $USER and DATA you can create the .usr file
- config TCDx changed so CTRLA is written last and the the status.0 bit is checked. without this the timer would not initialize properly
- readsig() for xtiny uses a named constant since the address might be different for other xtiny platforms.
- $PROGRAMMER 690 supports serial number for MCS EDBG programmer
- 1wread 720(bts , PinE , 3) failed when bts was a variable.
- Config printX 1028 has an additional parameter : DELAY=time. This is an optional parameter that can be used to delay for the specified time before the data direction pin is switched
- Options, Environment, IDE, new parser has an updated description.
- When using new parser and wrong syntax for $regfile, the setting from the config file was used. Now you will get an error so it is clear you use the wrong DAT file.
- When storing project .prj file in a folder other than the source, source code errors reported by the compiler were not jumped to when double clicked in the Error window.
- Alert window position is saved.
- STOP WATCHDOG bug fixed for Xtiny platform.
- Simulator IO grid improved for speed. Other grids in simulator are improved as well since they use the same improved code.
- Simulator SRAM grid improved for speed and displaying options. See help.
- I2CINIT bug fixed for xtiny 8 pin devices which have a different port for I2C
- When updating from an old version and keeping settings files, the options, Compiler , LCD could have a value of nothing(not selected). This could cause the compilation to end without creating bin/hex files.
- Alternative rename added using References, right click menu : RENAME 101. This will rename the selected item in the whole project, including the files on disk that are not loaded.
- Microchip CMIS-DAP programmers 207 support added. The programmer is named **MCS EDBG** programmer.
- Config DMXSLAVE 949 bug fixed for COM4. Also COM5 and COM6 added.
- Search and Replace 85 in files added. this is similar to Search and Find in Files but you can replace text.
- Find, search and replace button added to clear the history. this replaces the right click menu.
- readSig 1424() support for Xtiny platform added.
- MCS UPDI programmer 202 info added for USB virtual COM drivers/chips.

- _XTINY 460 constant in the DAT files set to 4 for the **EA** series since they differ significantly.
- Xtiny EEPROM writing updated and modified since all series handle this different in the NVM.
_EPROM_PAGESIZE is now an internal constant since it is used in the xtiny.lib. This does also fix a potential EEPROM write problem for some UPDI processors. This constant is auto generated by the compiler.
- Pulseout did not work on normal AVR extended ports. An extended port has a memory address > &H60
- Using INPUT on XMEGA with a software UART would result in stack loss caused by code intended for the HW UART.
- Using .\ for include files could load the file twice in the IDE when clicking the Error window and the code explorer. While this is fixed now, to include files that reside in the same folder as the main application, do not prefix the file.
- $programmer 690 additional COMPORT option did not work for MCS Bootloader. Also notice that New Parser must be selected in Options, Environment, IDE for this option.
- GETRC fix for xmega and Xtiny. While XMega would work with passing DDR register, the Xtiny passed the wrong register. Now all platforms can pass PINx register.
- CONFIG XPIN accepts both OUTPULL and PULLUP as a parameter to activate pullup. While normal AVR only supports pullup for input mode and the Xtiny till today also support pullup for input mode only, the Xmega supports various
modes in output mode as well. Since XMEGA was the first chip with pullup and atmel named it outpull we used this term too. But as it will be confusing for Xtiny/ normal AVR, PULLUP is added.
- MCS UPDI programmer supports P3 protocol too as found in the EA series.
- Simulation of xmega eeprom did not work.
- READ statement did not support multi dim arrays. The optional number of bytes parameter was not described in the help.
- Room for labels and other data increased in order to fix error 337.
- MCS UPDI programmer write counter added. It keeps track how many times a processor is programmed based on its serial number.
- SAFE option for DIM had a bug for one usage case. Also optimized the code for multiple safe access.
- The simulator memory handling had a bug for LD reg,X so depending on used platform this could result in wrong simulation.
- HW register color could not be changed anymore.
- Bigbuffers added for COM1/USART0. It uses bigbuf.lib which need to be included using $LIB. Set this option using CONFIG COM1 BIGSIZE instead of SIZE.
- Config-kbd extended in DAT file with possible options
- CANSEND 845 statement added. this statement does not wait/blocks the code. see help.
- Watchdog documentation improved for xtiny. also added example watchdog-avrx128da28.bas
- Splash window adjusted to show new xtiny processors
- Some DAT files had the wrong flash_size value which results in programming problems
- Find window lost right click copy,past,cut options when 'clear history' was added.
- Config comx new options RX and TX enabled/disabled were reversed. (tx would disable rx and vice versa)

1.2.2 What is new in 2086

- config sections are also grouped for code collapse
- toggle code improved for word,int,long. Also bug fixed when toggle as used on a port with constant like porta.pd3
- spi1move and spimove for xtiny using manual SS setting would set DDR instead of PORT register

- SW UART Inkey, Input, Waitkey revised for $timeout. $timeout maximum value is **&H**FF_FF_FF. Input will end when one of the incoming characters times out.
- attiny861 , START TIMER did actual stop the timer since the wrong value was written.
- buffered serial port changed. since a global variable is used for the buffer count, the interrupts are disabled and reenabled during updating this variable. but users that use the BYTEMATCH option in combination with serial input code
like input, inputbin, etc. autmatic enable the interrupts since reading disable/ enables global interrupts. this can lead to problems. it is not good practice to read data from the interrupt but since many users seems to use this
we changed the CLI/SEI so that the I-flag is restored and thus global ints are not enabled by reading inside the ISR.
- using rnd() with config rnd=32 on a word/integer result in an internal variable error

CODE BREAKING CHANGE
- config comx 913 **TXPIN** becomes **TX_RX_XC_XD_PIN.** This better reflects that all pins belonging to the USART will have an alternative pin value
This only breaks old code when the TXPIN option was used.
##########################

- config comx for xtiny platform has a new option : TX=DISABLED|ENABLED and RX=ENABLED|DISABLED by default both TX and RX are enabled. but you can disable the Transmitter or Receiver
- M324PBdef.dat updated to support second TWI channel. See also the M324PB sample.
- included fonts for graphical LCD are now word aligned. this in case the user uses multiple END statements.
- drivers(libs) updated that were unsafe when using port pins on an extended address.
- I2C_TWI-MULTI.lib : i2cstop missed setup call to _i2c_chan_setup when CONST _TWI_STOP_TIMEOUT was not defined resulting in a hang up
- AVR-DOS updated for xtiny and added sample files : FlashCard-demo-XTINY.bas , Config_MMCSD_HC_XTINY.inc , CONFIG_AVR-DOS.inc. See also the AVR-DOS topic description for a sample.
- bootloader added for megax : BootLoader-MegaX.bas the file m4808-tca0-BOOT.bas can be used to load the demo which uses $romstart 697. See also Using a BOOTLOADER 285
- dim SAFE changed see help
- val() to a double for a string with leading + would result in NAN.
- decr/incr did not protect variables dimmed with SAFE
- editor multi search highlight and multi selection highlight added. See Options Environment 150
- com5 and com5 buffered output for xtiny bug fixed where there was a double label used.
- RTF export now capitalizes IO registers, just as they appear on screen.
- simulator now uses a separate register space. old AVR, xmega and xtiny have different memory maps. while older AVR can reach the registers by a pointer, the xmega and xtiny can not do so.
while desgning the simulator there was one memory area. but it turned out that some instructions writing to registers, actually wrote to lower IO space. So this has been rewritten.
This means that a lot of the simulation code has been changed. Please report any simulation bug to support.
- config dmxslave support added for xtiny platform
- xtiny buffered serial input for com5 and com6 gave a duplicate label error
- some programmers like stk200,stk300, kamprog had no icon any longer in the toolbar. and the chosen size was not working either.
- clearing history in search box did not work properly(right mouse button)

- config tcpip for w5500 supports xtiny
- CLS in $ASM block mistaken as BASIC CLS
- shiftout number_of_bits would not accept a local/param
- buffered serial output com5 for xmega caused an error
- dcf77 927 xtiny support added for timer TCA0
- user EDC had a great idea about notification of usb-serial ports. you are notified with an alert window when a CDC is added or removed
- $programmer extended with constants for programmers and additional COM parameter. prog.inc contains the constants for the programmers
- writedac 1610 statement added for xtiny platform
- simulator usart emulation added for xtiny platform
- baud statement implemented for Xtiny
- getrc5 1278 xtiny support, also background mode added for TCBx. check config-rc5 in the help and the avrx128da28-RC5-Background-send-receive.bas example
- rc5send 1405 xtiny support, check config rc5send
- config tca0 splitmode fixed. see also the example. this also requires an update of all DAT files.
- $romstart number of bugs in simulator fixed (elpm, lpm)
- Baud statement improved. it did not support channels for a HW UART. it will also raise an error when you try to use non existing channels.
- FM24C64_256-XMEGA 1746.lib support for strings added.
- UPDI addon support for more processors : mega808,1608 and 3208. also added 64DB32,128DB32,128DA48,128DA32,32DA48,64DA64,128DA28
- updi programmer 202 enhanced, it works up to 1.6 Mbaud now. not all processors support this. See help. Also added unlock chip function.
  also added lockbit table to DA/DB series. these platforms use a 4 byte ID to lock.
  updi can use DTR/RTS for switching data and/or a 12V pulse. 12V pulse is used to access UPDI when the updi/reset are shared and the pin is programmed for reset function.
- tcpip w5500 corrected a bug for xmega with > 64KB processors. mixing tcpwrite and tcpwritestr could result in sram accessing a wrong page.
- some checks added for string assignment/passing byval. when constants are used that are too big you get an error. you also get an error when you claim more temp memory than specified with $framesize.
  see also the help for error 406. The option need to be turned on using CONFIG STRCHECK=ON
- string to double conversion with string in scientific notation conversion bug fixed when there was a leading minus. also added support for the bigger xtinies.
- some scaling problems fixed.
- DB/DA series use a different method for the UPDI fuse/eeprom/signature writing. This is corrected.
- xtiny with 128 SRAM did not simulate correct. Also the MSB of pointers were not set since normal AVR do not need this for 128 SRAM chips.
  since xtiny has a different memory model this lead to memory bugs
- optimization did not recognize flag registers. code like : portf_flags = portf_flags would not be executed.
- kamprog icons were not visible due to a change in icons
- writing a constant to an eeprom string on a normal AVR would fail
- closing bascom with programmer window open produced an access error
- lib manager can add a routine from the clipboard
- int_trig.lib 64K boundary bug fixed (qsin/qcos)
- xtiny/megaX dat files had ADC value exchanged for 8bit/10bit resolution selection
- mcusr register was not properly cleared at start up for old AVR processors. only the WD flag was reset, notice you should not write a 0, since some AVR have other unrelated bit flags in the MCUSR register!
  processors like that will have an additional mask in the dat file : WD=MCUCSR.WDRF,$E0
- assigning a hex number to a double did not work for all numbers

- waitkey() for the software uart did not support timeout.
- View Show Alert Window 99

1.2.3 What is new in 2085

version 2085.003(beta 3)
- getadc() checks data type for xmega/xtiny. data type should be integer/word
- $SIM -> _SIM was not properly colored when using 'show excluded code' option
- simulator gave overflow error when kept running for a long time.
- simulator dual port registers support added. this means that using virtual ports will update correctly the normal registers
- join 1524() function added as counter part of Split 1534() function
- lookdown 1363() support added for dword/long

version 2085.002(beta2)
- movw code replaced by mov when the processor does not have the movw instruction. only applies to old processors
- xtiny platform config SPIx has an additional option SPIPIN to specify which port pins must be used for the SPI bus
- since some xtiny(megaX,AVRX) have multiple SPI, config SPI1 is added to configure the second SPI bus. This works for SPI1IN, SPI1OUT, SPI1MOVE.
- mid function and mid statement rewritten. the start position is not simply added but checked so it can not be placed beyond the end of the string
  use the byte variant to terminate the string like : mid(somestring,index)=0.
- Both MID 1530 statement and function support $bigstrings
- instr() function updated to be more safe. since you can specify an offset, this offset is checked so it will not read beyond the string.
- space 1533() and string 1536() functions moved to mcs.lib and also added support for $bigstrings
- delchar 1520/insertchar 1522 support $bigstrings 611
- bascomp 1863 utility updated

version 2085.001 (beta1)
This is an updated to support the DB series.

- make sure to read about PRESERVE and OVERWRITE in the CONFIG 853 options.
- lcd_i2c_PCF8574.LIB updated version included.
- mega16M1,32M1 and 64M1 corrected for LIN/USART
- rainbow libs rolled back. the automatic platform code had the disadvantage of requiring a call instead of rcall
- font editor fixed. The width of the font was not properly saved when it wasn't a multiple of 8.
- various fonts changed in IDE. There is also a new option to override the windows system settings.
- m48pb dat file modified. DDRE entry was invalid
- a number of icons are changed. Also new bigger icons included which can be selected in the options. This is intended for high resolution DPI. The icons are still in the ICO format however which mean that they do not scale perfect as SVG would do. This is in the works as well.
- DB/AVRX 503 support. The xtiny add on is required. You need to update the add on lib.

1.2.4 What is new in 2084

version 2084.001

- mega4809 added to xtiny platform. See also MEGAX 490
- xtiny support added to i2cslave add on.
- LCD I2C 1790 driver from O-Family included that supports up to 8 LCD.
- xtiny alias portx,ddrx and pinx have been changed from the port_out to the virtual address. this also required the following :
  - 1wire,i2c,getrc,pulsein,pulseout,serout,serin,i2cbus and rainbow changed for new port mapping
- config COM 913 for xtiny has a new option to chose the alternative pin. TXPIN=option
- xtiny TCB0: CCMP_OTP renamed into CCMP_OUTPUT. Also reversed enable/disable. And ASYNC enabled/disabled were reversed too.
- xmega dat files corrected for DACA/DACB.
- xmega config eeprom=quick|mapped did not simulate properly
- xtiny config port_mux did not set the proper register value for TCAx and TCBx
- portmux support complete rewritten. data is stored in the dat files. see also config_portmux 1014 for important information.
  most choices list the pin number name now.
- sizeof 1457() function added. it returns the size of a variable in memory.
- xtiny config sysclock prescaler value 6 was missing.
- simulator fix for xtiny (register offset). Also register name length extended to 32 characters
- htrc110.LBX added * for used equ so they can be adjusted by the user
- DTR option for terminal emulator. you can set the DTR pin level for the terminal emulator just like you can for the RTS pin.
- mysmartusb light programmer problem with EEPROM programming solved
- const _TEXTLCDKIND 992 added which contains the text LCD kind like : 162 for 16x2
- the tool tip info (SHIFT key) shows the length of a string constant when moving over a string constant.
- xtiny support added for AVR-DOS
- serin/serout implemented for xmega and xtiny
- SWAP 1545 can swap a long/dword too
- glcdST7565R.lib adjusted for RAMPX boundary in showpic
- xtiny enable/disable set wrong bits for the timers
- xtiny start/stop switch the enable bit for timers
- datetime.lib modified for xtiny
- split() did not raise an error when using non-strings. The result array must be a string array.
- syntax check/compile did reset the stk200 programmer reset pin
- xtiny tcb1 added which was missing.
- UPDI programmer speed increased. baud is selectable. 225000 is the maximum for the default updi clock.
- IDE did not compile for the right processor when multiple $regfile directives were used with #IF#ENDIF.
- Font size increased in IDE. Default is now SEGUI 12. When you use a different language in options, this might not be visible.
  Also changed IDE so that high resolution monitor should show better font when bigger fonts are chosen. The icons/images still need to be changed to vector drawn images so they can scale better.
- more xtiny samples
- IDE can update 61 a number of add ons

1.2.5 What is new in 2083

For the Xtiny the following changes were made to 2083.008:

- xtiny 8 pin chips set the wrong tx pin for config com. as a work around, you can define a portb alias that points to portA
- printing a constant using the str() function : print str(number_constant) would include a leading space for positive numbers. Variables do not have this (VB behavior).
- xtiny config xpin was missing the INVERTIO option in the DAT files.
- passing a big string for which the size could not be determined, would not release space which would result in a quick crash. to determine the size of passed strings, you best declare with the size indicator like : someString as String * cSize
- config spi0 for xtiny 202,402,212,412(8 pins) did not set the SS pin in the proper state.
- config vref was not described in the help
- adc.bas sample added for xtiny
- xtiny dat files updated for config adc and signature row offset adjusted.
- xtiny adc1 can also be configured now
- xtiny renamed ACI to AC : config ac0. Also added ac1 and ac2
- xtiny config dac1 and dac2 added.
- xtiny config port_mux added.
- xtiny SPI sample added
- showing report in project mode inside the editor as a TAB did not set focus to the report.
- xtiny dac.bas sample added
- xtiny portmux.bas sample added
- optional custom defined menu shortcuts added. See help : Options Environment
- RESET MICRO added which will soft reset the micro. for xtiny/xmega a hardware soft reset is performed. For normal AVR a jump to the start address is performed.
- getadc can also read and process the internal temp sensor when you define a const named _adc_kelvin. The value is unimportant. see adc.bas example
- multiple asm .def with different register but same name will give an error now since .defs are global.
- num2str for xtiny/xmega offset added to avoid str() causing problems when passed to a function

Public release

- new option SAFE for variables. Dim 1228 b as bit SAFE , see help.
- added BOOTONLY option to $LOADER 667 directive. $loader bootaddress[, BOOTONLY] this will write just the boot loader code to the BIN file. The HEX remains as is.
- you can select the Options_Select_Settings_File 226 now. This setting is stored in the registry.
- project files are stored with absolute files names inside the prj file. An absolute file name is relative to the location.
- simulator bug fixed where SI file simulation data was not processed properly.
- MemFill 1374 added.
- using instr() with {xxx} for the search string does not work : pos=instr (someAString,"{065}")

- using compare_a/compare_b=clear for timer0 resulted in SET instead of CLEAR.
- bascomp command line utility updated to support new file structure
- multiple instances bug fixed.
- mcs.lbx was not in sync with mcs.lib (it was not compiled when a last minute change was made)
- when using channel specifier without # you will get an error
- stk500v2 based programmers like mkII and stk500v2 could give a program error when your code contains empty blocks And the processor has multiple 64KB segments. applies to normal mega only.
- crc8 809 overloaded version added for big strings.
- 2082 broke the default printing function
- added DES asm instruction.
- added DesEncrypt 1292 and DesDecrypt 1294 which are also supported by the simulator
- inputbin 1497 accepts an optional variable for the number of bytes to receive. delimited by a **;**
- simulator 111 update.
- simulator double click cycles, will reset cycles
- simulator allows to load a custom serial data file from file
- Xtiny 460 support, requires a **commercial** add on
- CTS/RTS bug fixed : only part of the buffer was used
- added PA version of dat files M88PAdef,M644PAdef,M48PAdef,M168PA. These are almost the same as the P versions. They are binary compatible and have the same ID
- searching in files would not search in the specified folder when the folder name contained a space. Instead the root folder was used.
- $programmer option did not support conditional compilation. It was global. Now supports #IF/#ENDIF. but only when 'Use new method' is used in environment IDE options.
- CLEAR serialinx buffer did not clear the RTS pin when cts/rts was used for xmega uarts 4-7.
- xmega high baud calculation > 2MB and higher did not support double rate flag resulting in a wrong baud rate
- for next using a step for bytes could fail when the byte boundary was crossed.
- xmega num2str code rewritten and xmega routine rewritten that used _XmegaFix_Rol_R1014 and _XMEGAFIX_CLEAR. These routines are not used anymore!
- using a string function with select case, could result in improper branching, depending on the user function.
  select case mid(someString,start,len) for example.

- bascom-AVR and the SETUP are now code signed.
- CONFIG XPIN for the E-series : slewrate will be set the whole port, not for an individual pin
- crc16uni 815 can handle 65535 bytes
- low/high can be used as a procedure too for BASCOM-8051 compatibility.
- UPDI programmer can write fuses

Please notice that this version has significant changes in order to support the Xtiny platform.
While everything was extensively tested, it is still possible you encounter a bug.
When you encounter a problem you did not had with 2082 you best contact support.

1.2.6 What is new in 2082

- rearranging memory order for usb support caused a bug in config clock : the date time bytes are not mapped after each other. Fixed.
- START/STOP statements worked on the wrong register for the TINY1634
- i2c_twi_multi lib had a problem in the readbyte function.
- SPIMOVE 1515 added for Xmega
- split screen did not allow copy & paste
- USI slave support added for tiny1634
- **All DAT files are now stored in a sub folder named DAT. This means you need to copy your DAT file to this folder if you made custom DAT files.**
- some registers of tcc1 were missing in xmega D3 series
- read only files could not be opened anymore. fixed.
- On win10 you could get a HID error message.
- w5500 tcp lbx : removed RST status bit check since the bit never becomes 0 and hangs the code
- url2ip 1592 bug fix. one byte of the IP address could get trashed
- url2ip added to w5500
- new samples for w5500 wiznet chip
- accessing a zero based array inside a sub coulld result in an index error.
- simulator did not support writing to xmega portx_CLR _SET and TGL registers
- using search in files function could result in 'out of bound' error.
- manchesterEnc 835 and manchesterDec 834 functions added for manchester coding/ decoding
- assigning a byte with a string constant with spaces, resulted in 0, not 32.
- VARPTR 1604() function returned &H1000 too much for Xmega ERAM data type.
- using getadc() with 2 numeric parameters or constants like : getadc(4,&H20) would not set the right bits.
- i2csend and i2creceive updated for xmega. after the start/slave address, the status is now checked and does not send data in case of a bus problem. this to prevent a hangup in the twi logic.
- printing supports selection of text and page range now. you need to use print preview for this.
- OUT instruction did not clear RAMPZ for Xmega with ROM > 64KB and normal SRAM.
- PS2MOUSEXY 1399 accepts an additional optional parameter for mouse wheel support. **Notice that you MUST download an update of the ps2 lib add on**
- config spi on non xmega did not support the extended mode for HW SPI
- config tcpip 1098 now supports SPI1 for the SPI bus
- windows 10 DEP and ASLR support added.
- printbin: when using automatic rs485 and printing a long/dword constant on a chip with extended port register, R23 was trashed. Example : printbin &HABCDEF00
- FLIP() function resulted in an error about $REGS
- terminal emulator component is replaced in order to support windows DEP/ASLR. This means that some features from the terminal emulator have changed.
- printbin 1504 can print a variable amount of bytes now. while ; is used to separate multiple variables, the comma can be used to specify the amount of bytes
  like : printbin ar(1) , numbytes ; othervar
  this makes the syntax compatible with the old syntax. We recommend to use the new syntax
- terminal emulator custom messages extended to 16
- UPDI programmer 202 added for new AVR processors with UPDI interface.
- A table is added to $LOADER 667 with the size of maxwordbit. This constant depends on the number of flash pages.

FILE LOCATION
With DOS things were simple : all files could go in a folder and sub folder. To make a backup all you had to do was using XCOPY.

With Windows things are not so simple : files are located all over the PC. Some folders are write protected and to make a backup is not so simple.
A lot of customers are looking for the SAMPLE files. These are put in the documents folder and can be accessed using the File, Open Sample option.
In 2082 the preferred folder for installation is C:\MCS\BASCAVR2082
But of course you are free to install in any other folder of your choice.
The samples are installed in a sub folder of the application folder too.
In the Environment Options of the IDE you can specify which folder you want to use for the sample files.

About UPDI
The new UPDI processors have a total different architecture compared to normal AVR. In fact the differences are similar to XMEGA. For this reason we refer to these processors as XTINY since they are tiny Xmega processors.
Because of the work and support for XMEGA fresh in mind, the actual UPDI compiler/ DAT support will be available very soon in a next update as an add on.
The TINY816/817 will be the first processor to be supported.

1.2.7 What is new in 2081

- CONTINUE 1168 statement added
- REDO 1422 statement added
- NOP 1378 is now also a BASCOM BASIC statement
- The editor supports jump to implementation : hold CTRL key and hover the mouse over an identifier. When it becomes underlined and blue you can click it with the left mouse key.
  Use CTRL+BACKSPACE to jump back
- when defining a constant named Updateeprom 1228 , the eeprom will be updated. which means that the value will only be written when it differs. The value of the constant does not matter.
- config timer1 for tiny 25/45/85 set the wrong register bits.
- the watchdog is disabled as part of the init procedure. it is now disabled BEFORE the optional call to init_micro 656 and not after as in 2079.
- passing string constants with embedded {034} resulted in an extra (unwanted) space.
- accessing passed string array in sub without length info, but with constant index failed.
- crcmb 818 funtion added to help. (checksum for modbus)
- for xmega i2cstop 1306 you can define a constant named _TWI_STOP_1 or _TWI_STOP_2 to change the behavior.
- makemodbus 1499() function 1, 2 and 4 added to modbus.lib
- support for xmega added to getrc 1277
- PDF download now also checks/download the BASCOM-AVR manual
- PRINTBIN did not accept a constant for the optional channel : printbin #someconst. Fixed.
- update 235 from within the application simplified. see help.

- printbin raised error while printing multiple variables
- simulator did not show proper hex value for single variables.
- fusing which uses ftoa uses a table which could be loaded on a page boundary. this could lead to rampz problems. fixed.
- crc8UNI 811 added for normal crc8 CCITT
- config clock 895 additional option : highESR=1 to enable high ESR mode in xmega with 32 bit RTC

- FM24C64_256 1746-XMEGA.lib added for xmega. read the lib notes.
- tcpip-w5500.lbx has been updated to support usage from boot space. See $loader 667
- stk500 board. osc can be set from menu
- using spimove 1515() inside a sub with a parameter for the count, would load the wrong data.
- i2cwbyte 1306 would raise an error if a multi dim array was used.
- get/put/seek in AVR-DOS used in combination with $bigstrings would fail for numeric data
- xmega i2cstop 1306 has two new optional mode. See help.
- CONFIG SPI 1061 has a new option : EXTENDED=1 to have extended data size reading/writing.
- support for rgbW 1033 leds added (ws2812 with extra white led)
- CONFIG USI 1138 has a new option to support optional pins.

1.2.8 What is new in 2080

- tiny461 361 and tiny861 363 only did set pcie0 when you enable the PCINT because there is just one interrupt in the chip. In 2080, both PCIE0 and PCIE1 are enabled/ disabled.
- added m48PB, m88PB, 168PB and m328PB dat files.
- new Rainbow functions : RB_Color 1484 and RB_Copy 1486 added by Galahat
- simulator did not show maximum values of DWORD correct.
- RB_GETCOLOR 1483 and RB_LOOKUPCOLOR 1484 functions did return false result when index was a variable.
- some font problems solved.
- simulator could crash for xmega processors.
- when using non-mono font like Arial, text selection does not work properly. Use a font like CONSOLAS.
- Added option 'Use Monofont' for backwards compatibility
- Some new atmel PDF files could not be loaded with the PDF viewer. Viewer is rewritten and requires a new DLL named BASPDF.DLL
- getadc 1265() on m640.m1280/m2560 or any other processor with 6 mux bits did not set mux5 bit for getadc(32) and higher.
- generic byte compare 851() function added, based on code and idea from MWS. (Magic White Smoke)
- varexist() did not support ALIAS.
- XMega64A1-SRAM 4-Port-Sample.bas sample added for setup EBI 4 port on XMega. See also Adding XRAM to XMEGA using EBI 257
- when bascom-avr.xml options file exists in the bascom application folder, that option file will be used.
- format 828 is extended to use a variable for the mask.
- config xpin 1158 did not support alias for the pin.
- bufspace 1491() did not support UART 5-8
- INSERTCHAR 1522 and DELCHAR 1520 use Z pointer which must be cleared for XMEGA. fixed in mcs.lib
- programmer did not fetch correct chip from editor when code was not saved. this would give a chip mismatch.
- assigning a negative value to a dword did not throw an error.
- code explorer 101 can show estimated stack usage.
- higher standard baud rates added to terminal emulator
- added support for EDMA in xmega8/16/32 E5. See config EDMA 944
- version 1606() function did not append to string but would overwrite existing string data.
- right 1531() adds an additional null byte when a numeric constant is used for the number of characters to copy.
- new dim 1228 option to specify multiple items : dim a,b,c,d as byte failed when using multiple indexed items.

- all dat files updated with CONFIG information.
- printing values from multi index variables failed : print index(index1,index2)
- m1284pdef.dat updated with missing TIFR3 register.
- more fonts in various sizes from Adam Siwek.
- power() 758 function for doubles did not work correct when assigned to a function
- some new atmel PDF files can not be loaded with the PDF viewer. Viewer is rewritten.
- SSD1306 i2c oled driver updated for Xmega.
- m649A and m649P dat files added.
- LCDFONT 1342 prm, added. prm selects the font table (0-3) of a text LCD.
- CONFIG POWER_REDUCTION 1023 set register to 0 in some conditions. Also added LCD and other new Xmega power reduction options.
- CONFIG OSC extended with calibration register settings and DFLL.
- val() for doubles has a bug for XMega >64KB chips
- added flip 1259(byte) function to mirror bits in a byte
- xmega128B3 dat file added
- readsig 1424 also works for normal AVR processors.
- inputbin and printbin load 1 element too many with arrays using base 0.
- config_inputbin 984 added to allow reading packets of up to 64 KB
- added support for LCD text OLED RS0010 lcd4_anypin_oled_RS0010.lib
- FT81x 1619 support added
- M324PB 403 dat file added.
- I2CINIT 1300 enhanced for multiple TWI
- I2C_TWI-MULTI.lib 1754 added to support multiple TWI busses.
- second SPI on m328PB added : INIT1SPI 1519, SPI1OUT, SPI1MOVE, SPI1IN
- user donated library I2C DOGS104 1780 driver, SSD1803A included.
- URL2IP 1592(url) function added to W5100 to do DNS lookup using google DNS server
- when defining a const Updateeprom 1611 , the eeprom will be updated. which means that the value will only be written when it differs
- BASE64ENC 1549 and BASE64DEC 1547 can work on byte arrays too.

2017, 2080 release
- SGN 1560 extended to byte, integer, word, dword and long
- LOADLABEL 1359 assigns a 24 bit address when used with a DWORD
- CTRL+SPACE for code help.

1.2.9 What is new in 2078-2079

Beta version 2079

- Support for WS2812 RGB led : CONFIG RAINBOW 1033. This is the rainbow lib from Galahat, see : http://bascom-forum.de/mediawiki/index.php/Rainbow_Lib
- SETATTR 798 and CLEARATTR 776 added to AVR-DOS, by Josef.
- shift & rotate left/right did not work for xmega port registers
- IDE : improved speed for showing deadcode/unused variables
- IDE : stacktrace speed up. big projects made the stacktrace slow.
- included FT801 support. See CONFIG FT800 959. Notice that the INC files have been renamed into FT80x
- fixed attiny261,461 and 861 interrupt entries. this chip has only 1 pcint.
- added check when $loadersize 683 and $boot 612 are combined.
- Dim 1228 supports a list ; dim a,b,c,d as byte. It also supports identifiers like %,#,& and !
- Font Editor plugin is replaced by integrated Font Editor: Tools, Font Editor 141
- Sample added for USI Slave lib 1757
- fonts contributed by Adam Siwek included. You can find them in the Samples\LCDgraph\Fonts folder.
- report 110 can be opened in IDE as text file.

- mySmartUSB light 200 programmer support added.
- support added for W5500 1098 tcp/ip chip
- W5500 socketconnect 1567 has a 4-th parameter : nowait. When you make it 1, there is no wait for connection.
- $ROMSTART 697 added : $romstart = &H8000 , will let the code start at &H8000. Default is 0.
- jtag ice mkII programmer new firmware 7.26 from AVR studio resulted in signoff problem. Workaround implemented.
- editor can show unused code in conditional compilation. Edit, Show Excluded Code 90 menu option.
- usbasp programmer updated. chosen clock frequency will work.
- makemodbus() did not support locals/passed parameters properly.
- crc16 813 can now directly read a range from eeprom memory to calculate a checksum for you. To enable it, just add const CRC16_EEPROM=1 to the beginning of your code.
- simulator fix for xmega low IO registers. registers were simulated with a 32 byte offset as in plain AVR.
- config lcd 992 has 2 new options : BEFORE and AFTER. with a parameter value of 1 a sub will be called _lcdBefore and _lcdAfter
  just before the LCD is used. This allows for example to turn off interrupts when executing LCD code.
  Only text LCD is supported.
- getadc 1265() when used on normal AVR with offset parameter, and both parameters numeric will give an error when MUX5 bit must be set.
  Use getadc() with just the channel parameter.
- multi dim arrays, added ERAM byte support, and used registers are saved now.
- saving programming buffer as HEX file created wrong HEX files which would not load in AVR Studio. This would occur for chips with multiple segments like xmega128

- Full Kamprog 190 support added.
- multi dim arrays had no check on invalid index value (non dimmed)
- using a constant float without leading 0 resulted in an error message : var= var + .12344
- INPUT did not support DWORD.
- added user definable command buttons to terminal emulator 128.
- using {} in constants was not working as expected : Const Cmd_suffix_ver1 = Asc ("{013}") was not interpreted as 13 but 123 (the { sign)
- changed PDF download 138 from HTTP to FTP. This is quicker and better for the load of the server. PORT 211 is used for FTP. So you need to have port 211 open on your firewall.
- atxmega128c3 added.
- FT800 1619, vertex2ii , the X is clipped. Change call in sub vertex2II into Cmd32 _vertex2ii(___wtmpb , R18 , R17 , R16)
- support for EADOGXL240-7 I2C 1786 added, see eadogxl240-7.bas. This is a customer sponsored lib.
- added support for SSD1306 I2C 1788 OLED, see SSD1306-I2C.BAS.
- i2c multi bus lib did not clear ERR bit correctly.
- when a multi dim array is only used within sub/functions and submode=new is used, an error was raised since the index table was not written at that stage.

- multi dim arrays can only be used to read/assign variables. Using them in functions and statements will not work.
- str 836() can have an optional parameter to specify the amount of digits. This works for double, but now also for singles.
- MOD 1376 for singles changed in fp_trig.lib so it uses the same algorithm as excel/ VBA.
- FOR..NEXT with WORD data type and STEP with values other than 1 failed : for w=1 to 10 step 2

- when opening a single file in non-project mode, the code explorer does not get updated until you set the cursor on the code.
  This also resulted in not updating the pinout viewer.
- R0-R31 internal variables are now exposed as byte variables. This is simpler than using getreg/setreg.
- added option to skip eeprom cell test. This allows to write all FF to the EEPROM whithout erasing the chip.
- terminal emulator font color could not be selected from the font dialog.

- various programmers : added chip name to info panel when chip does not match. no match will result in a red font, a match will show in green.
- added an error message when $hwstack,$swstack and $framesize are missing from the source. Also put back compatibility to 2077 when these directives are not specified.
- hovering the indention line will show the begin of the structure in the tool tip (just try it).
- Terminal emulator 128 has 8 user definable buttons
- SEROUT 1509 defaults to CONST SEROUT_EXTPULL=1 to be in Hi-Z mode. In this mode a pull up resistor is required. To use PORT output mode, set the constant to 0 : CONST SEROUT_EXTPULL=0

2.0.7.8.001 public release

- changing a bit on a passed array inside a sub/function gave a bit index error.

- while moving all single FP code to fp_trig.lib, some double (but WRONG) functions were moved to the top. It causes various problems.
- clear buffer did not reset the RST pin in case CTS/RTS was used.
- val()/asc2float contained a bug for converting big values not fitting into the mantissa. The exponent was not increased.
- asc 819() can have an additional index parameter : byte=Asc(string|string constant[, index]). Use this instead of asc(mid(
- user functions/subs can have a custom color
- added support for i2c lcd display RX1602A5. Use : config lcd = 16x2 , chipset = st7032. See sample LCD-RX1602A5.bas
- using overlay pointing to a string array resulted in a wrong overlay address.

- additional XTEA2.LIB added. This lib complies with the original standard.

- Tab order can be changed with drag and drop.
- USI master TWI mode added.
- when config submode=new was used, the syntax check could give false errors.

- mkII programmer would give a warning about chip mismatch when atmel chip ID was the same.
- pulsein.lib was missing from distribution.
- documented beta switches $NOTYPECHECK 688, $TYPECHECK 710 and $REDUCEIVR 692

- when using a serial boot loader compiled with an older version, and when calling it from code (not after a reset) you need to reset the u2x flag in ucsrxA.
  Or you can compile both the bootloader and main code with the new version. When you want the old behaviour, you can remark the u2x constant in the dat file.

- FLIP programmer will not erase EEPROM anymore.
- use ALT key to select blocks of text in the editor.
- hint window location fixed for multi monitors systems.
- rgb8to16 1350() function added to convert RGB8 to RGB16.

- xmega, config OSC : when using external osc, the oscillator ready test was not working properly. enable the internal osc as a workaround in 2077. fixed in this release.
- arduino leonardo can be programmed with myAVR MK2/AVR910. You need to give a manual reset before pressing F4.
- attiny87 dat file contained an error : INTADR = 1 ; it was 2 but must be 1

- xmega spi length parameter supported globals only. now it supports locals and parameters as well.
- syntax check gave errors when config submode=new was used in some cases.

- 1wirecount returned with ERR set, even when sensors were found.

- simulator 111 has trace log option to dump all executed lines to a file.

- error list content can be copied to clipboard with right mouse popup menu

- xmega uarts 5-8 serial buffered output enable the wrong uart.
- indention line colors can be customized.
- proper indent 89 will not indent comment
- getkbd 1275() required change for xmega. (xmega does not use port register for pull up)
- added dword support to lookup()
- config rnd 1044=16|32 added to support bigger random numbers.

- attiny441 stk500 settings changed. attiny441 and attiny841 verified with real chips. some mods made to the dat files.
- increased internal constant string storage length to 1024. for cases like : s="some very long constant". previously the max size was 256.
- xmegaE5 timers 4/5 support added.
- xmegaE timer4/5 OVF bit need a manual reset, writing a 1 to intflags register. it is not cleared automatically.
- xm128a1U dat file added
- code folding added to editor. Press F11 to fold a sub/function
- all project files can be placed in an archive 80 (zip) file.
- AVR-DOS, GET and PUT support $bigstrings 611
- $boot 612 extended to support >64KB processors. $boot can be used together with $inc to include a boot loader in your code.
- FM25C256 example with BMA.bas sample added which demons xmega ramtron lib with shared bus.
- baudX=value added for Xmega. This will change the baud rate on an xmega at run time.
- special multi bus i2c added for normal AVR processors. See config i2cbus 974.

- muli dimensional array support added like : dim ar(10,50,3,5,2).

- long/dword data types added to SORT statement.
- report extended with bit position in memory and length of dimensioned strings

- Lookup 1365 supports a numeric variable too for the label : novar = LOOKUP(value, label|address).
- soft spi supports DATA ORDER LSB and MSB
- str 836() second optional parameter added to help. It specifies number of digits after the DP. Only for doubles.
- m324/m164/m644/m1280 config timer0, disconnect option fixed.

- settings xml file can be passed as parameter to allow different settings files and versions.

- added option to show invisible characters.
- added support for DWORD to SWAP 1545
- added insertionsort.bas sample
- m64C1 and M32C1 dat files added.
- History Backup option added : it will create a unique copy of the source file each time you save a file.
- code explorer can show INC files under their own branch (options, environment, IDE 150)
- Qsin 760 and Qcos 764 integer trig added.

1.3 About MCS Electronics

About the Founder

Since I was young, I was intrigued by remote control, robots, transmitters, in short, all kinds of electronics. I created countless electronic devices. I designed a lot of PCB's by hand using ink and later using tape.
At the ETS(electronic technical school in Amsterdam) we had a Philips main frame with terminals which could be programmed in a simple form of BASIC.
When working at Philips in Hilversum i also worked with an industrial computer that could be programmed in BASIC.
The Apple II we got later on at the ETS could also be programmed in BASIC.

When the ATARI came with the 1040ST and an affordable PCB design tool, I bought my first real computer. I bought the ATARI just for the purpose of PCB design. The netlists had to be manually entered.

Only Dot matrix printers where available at that time. And the prints were not really usable. That only changed when laser printers became available.
I found out that a nice BASIC interpreter, which was similar to GW-BASIC, was included in the OS(TOS). For some reason, I liked this language. It was easy to master and very intuitive.
I made some programs for the PTT(now KPN) that were revolutionary at that time.

For hobby purpose i used the 8052AH BASIC programmable processor from Intel. I made a lot of interfaces using PIO, relay, etc. My home was automated in 1986.
Because of my work for the PTT i was also able to get caller info, something not available as a service yet. I used the 8052AH to show the caller info on an LCD.
The 8052 was great but the UV eeproms had to be erased using UV light. It was slow.

I found out, that Atmel made the 89C2051, which was a 20 pin chip with flash memory. I was excited to find out that there was a small micro processor that could be erased/reprogrammed without the need to UV erase the EPROM.

In those days, electronic circuits consisted of numerous CMOS and TTL chips. I saw the 89C2051 as an ideal replacement for a lot of CMOS/TTL chips. It would make PCB design much simpler. So the 2051 became a replacement chip. Like a small black box chip. Now one was able to design his own chips!

The idea to be able to change the behaviour of an electronic circuit, just by reprogramming it without using a solder iron, intrigued me. Today, it is a common practice, to update firmware, to fix bugs or add features. In 1993, it was not so common, at least not to my knowledge.

I initially wrote a complete tool for DOS. I rewrote the tool, when I was reasonably satisfied that Windows 3.1 was stable. The tool was for my own usage. When I

discovered that it would be usable to others, I decided to add Help files and a simulator and to sell it for a small fee to support my hobby. Today you can get electronic devices for little money. But a resistor used to cost 5 cents !

In 1995, MCS started to sell BASCOM-LT, a BASIC compiler for Windows 3.1. It was the first Windows application that offered a complete and affordable solution, editor, compiler, simulator and programmer. BASCOM-LT was a 8051 BASIC compiler. The reason it became popular was that it included a lot of functionality that was easy to use from BASIC. Using an LCD display was simple, just a configuration line to define the used pins and voila, a working application in minutes. When you needed a different LCD display, you could simply change the CONFIG line.

When a different processor was needed, you only had to change the name of the definition file. No need for a lot of .h files.

Another reason for its success, was that we hide much of the complexity for the user. No ASM to deal with, simple statements. Of course free updates and support.

Small companies that used the BASIC Stamp also recognized another advantage : There was no need for expensive modules and the code ran much quicker.

When Windows 95 became an industry standard, users also wanted a 32 bit version. A big part of BASCOM-LT was rewritten with the additional support for arrays and floating point (single).

With the many different 8051 variants, it was impossible to support all the chips. Having device definition “DAT” files, made it easy for the user to configure the 8051 variants.

When Atmel launched the AVR chip, the 8051 compiler was rewritten, once again, to support the powerful AVR chips. The result was BASCOM-AVR.

The AVR chip has a lot of internal memory. It uses simple linear memory addressing. The best part, is that you can make the chip program itself. No wonder this chip family became so popular.

Since the AVR chip is so powerful, we could extend the compiler as well. We could add features, which are almost impossible to add to the 8051.

With more and more users, there was no way I could manage everything in my spare time. So in order to guarantee the future of BASCOM, I decided to work full time for MCS.

Today, MCS is still a small company, with only 3 employees and a few contract workers.

We believe in free updates and support. With the number of (demo) users, it is however not possible to support everybody. You need to realize that reading and answering emails is time consuming.

Not to mention to duplicate used hardware. We are unique, in that we even support hardware!

In order to migrate to a new version, it is important that you keep your software up to date. This will make migration more simple.

Things we find important :

- The environment. We reuse all usable packing material like foam, plastic bubbles, when we ship your order.
- That everybody can use micro processors. They are like all other chips but you can define their behaviour.
- Customer privacy: We keep your name, details and code confidential. We do not sell or share any of your details.
- Free updates. They have been free since 1995 but there is no guarantee that they will remain free for ever. The intention is to keep them free. In order to apply for free updates you MUST register your software within 1 year.
- Free, but limited, support. Limited only, because we do not have the resources to read/answer all emails. Professional users can get an SLA with guaranteed response time. This is a paid option/service.
- Support for new chips. It is important to be able to support newly released chips.
- The customer : We simply add what is requested most. It does not matter what, as long as it is requested a lot and it does makes sense and doesn't conflict with other features.
- That you have fun with electronics, no matter where you live, no matter which religion you have, no matter how old you are, if you are male or female, purple or white.
- That you can use the demo for free. The DEMO has no nag screens. You should purchase the full version if you use it commercial. Please do not use cracked software. Only download from the **www.mcselec.com** domain. Copies from other sites may contain spy ware, virus or other malware. When we detect a cracked version the compiler generates tiny bugs at random which are hard to detect. We ban all IP numbers of users with a cracked version.

Mark Alberts
Managing Dire
MCS Electronics

1.3.1 Custom Designs

MCS does produce hardware to support special options. Like the EM4095 Reference Design 326 or the TCP TWI motherboard and adapter boards. We try to avoid SMD parts. In some cases this is not possible however.
For a prototype or small series, through hole components are simple to use. We do this with the hobbyist in mind. So our reference designs use little SMD parts too.

You can contact us also for :
- custom bascom software
- customer ASM drivers
- windows software development
- electronic or software projects
- code review
- SMD/TH electronic design

1.3.2 Application Notes

When you want to show your application at our web as a reference example on what you can achieve with BASCOM, we like to show it at our web, but of course with your permission.
We never publish anything without your explicit permission.

AN's are also welcome. When you developed a great AN you want to share with other BASCOM users, just send it and we will make an AN out of it. It is important that the comment in the source is in English.
You can also share your code at the MCS Electronics user forum.

1.3.3 About this Help

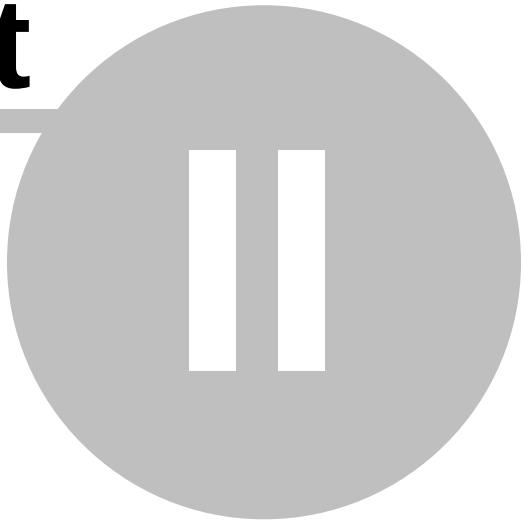
This help manual is available in CHM format for the IDE. And also in HTML on line.
A PDF version can also be downloaded. You can do that using the Tools, PDF function of the IDE.

Some topics show examples. Some examples are partial samples. Some examples demonstrate a number of statements.
And some topics contain full examples.
When you want to use examples you can best load the examples that come with the IDE.
By default they are installed in the SAMPLES folder.

The samples from the distribution are checked with each release since they are intended to be compiled.
So they contain a $regfile directive, and default stack settings. And when print is used, communication set up.

The samples from the help are only intended to demonstrate a statement or function and are extremely simple. Often $regfile and stack is omitted.
When you want to use the sample from the help you can copy & paste it.
Do not forget to include $regfile, $hwstack, $swstack and $framesize when it is not included in the sample. The sample will compile without this info too but for later reference it is best to include this info.
Some examples from the help are from 1995 and the compiler has been extended.
When you encounter a sample from the help that does not work, send an email to support so it can be corrected.

Part



2 Installation

2.1 Installation of BASCOM

After you have downloaded the ZIP file you need to UNZIP the file.
On Windows XP, for the DEMO version, you may run the setupdemo.exe file from within the Zipped file. For the full version you should unzip the ZIP file.

The commercial version comes with a license file in the form of a DLL. This file is always on the disk where the file SETUP.EXE is located. When explorer does not show this file, you must set the option in explorer to view system files (because a DLL is a system file).
For the commercial version the setup file is named SETUP.EXE

Some resellers might distribute the DLL file in a zipped file. Or the file might have the extension of a number like "123". In this case you must rename the extension to DLL.

 Make sure the DLL is in the same directory as the SETUP.EXE file.

When you are using the DEMO version you don't need to worry about the license file.

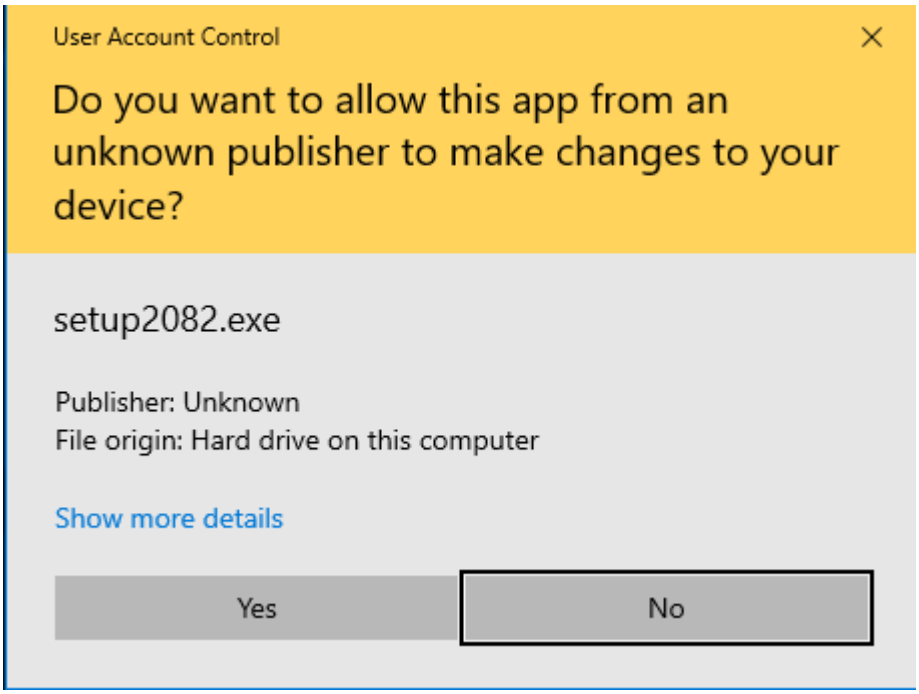
When you are installing on a NT machine like NT4 , W2000, XP, Vista, Win7, Win8 or Win10, you need to have Administrator rights.
After installing BASCOM you must reboot the computer before you run BASCOM.

The installation example will describe how the FULL version installs. This is almost identical to the installation of the DEMO version.

Before installing the software : make sure you downloaded from mcselec.com domain. Or that you purchased from an authorized reseller.
When in doubt you can always check the executable on your PC using your browser at virustotal.com. In fact it is good practice to check files before you install them. virustotal.com will use 50 or more virus scanners.
This will give a good idea about the safety of a file.

Run the SETUPDEMO.EXE (or SETUP.EXE) by double clicking on it in explorer.

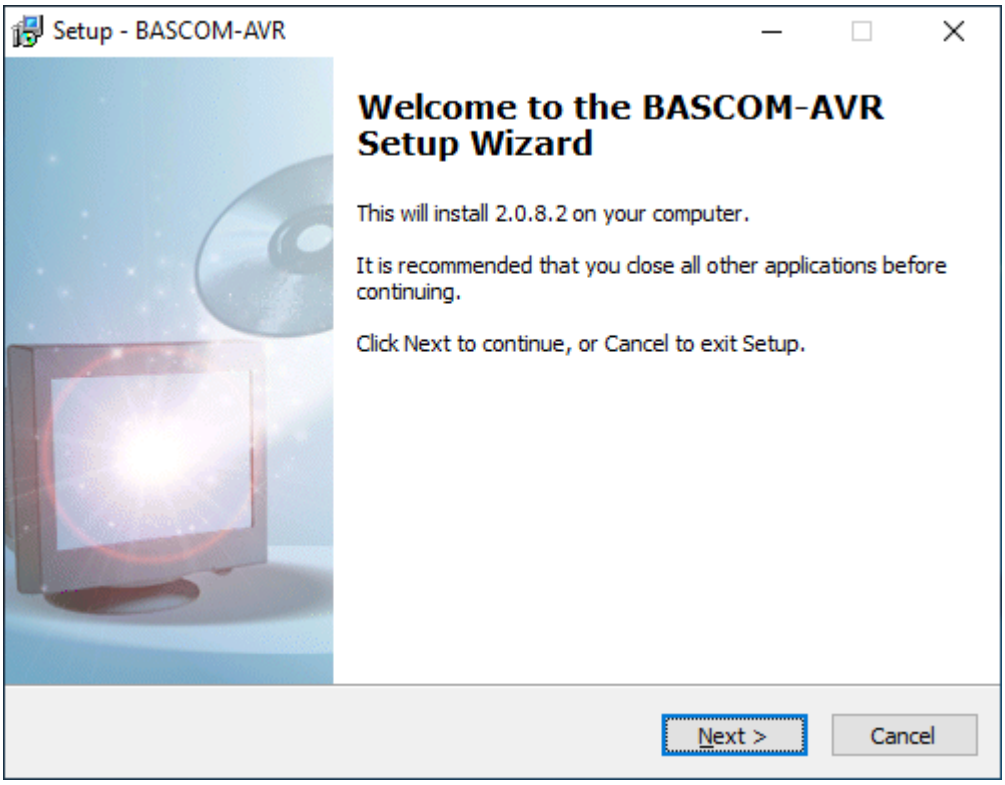
Depending on the windows version and your user rights, windows might give the following message :



You need to click the YES button.

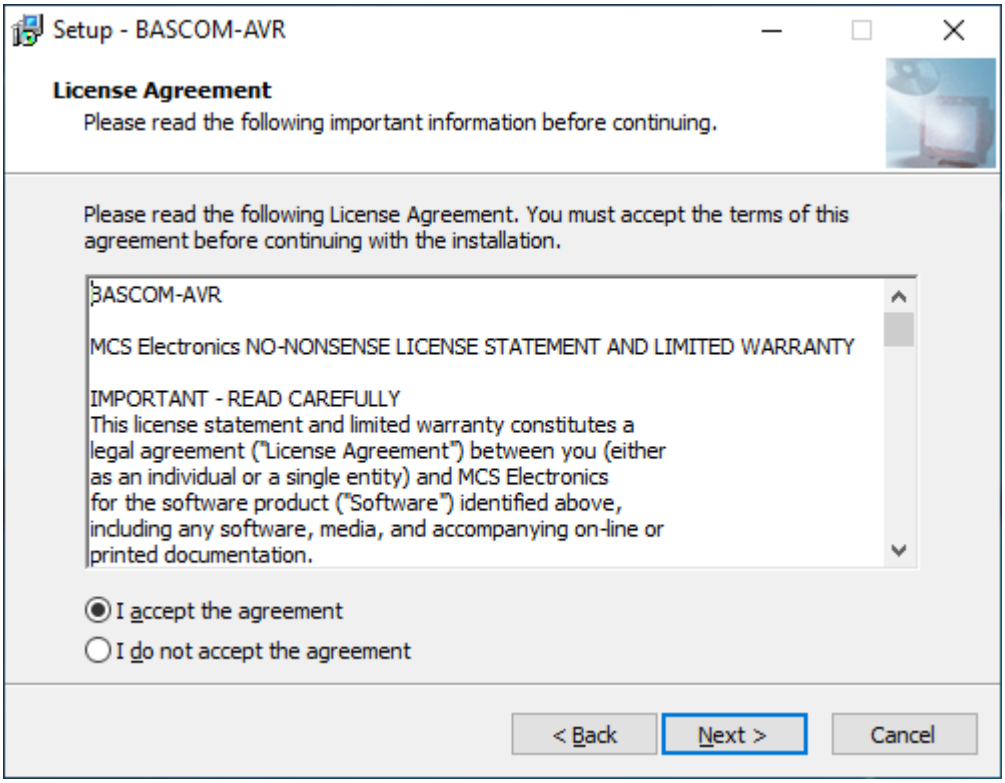
The following window will appear:

(screen shots may differ a bit)



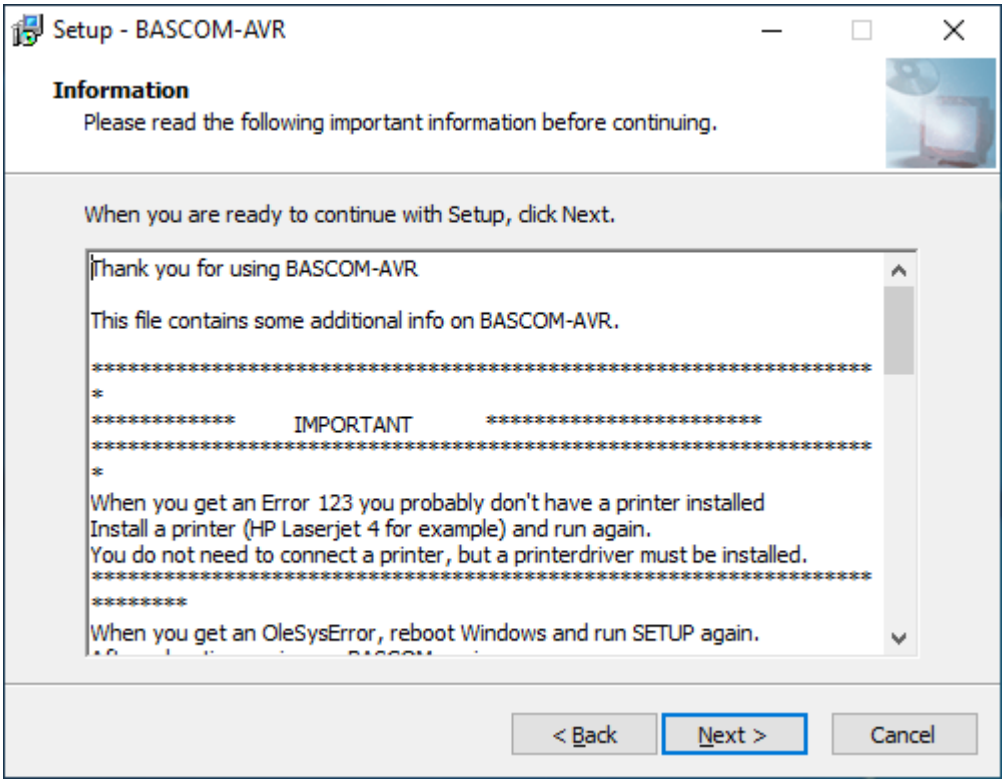
Click on the **Next button** to continue installation.

The following license info window will appear:



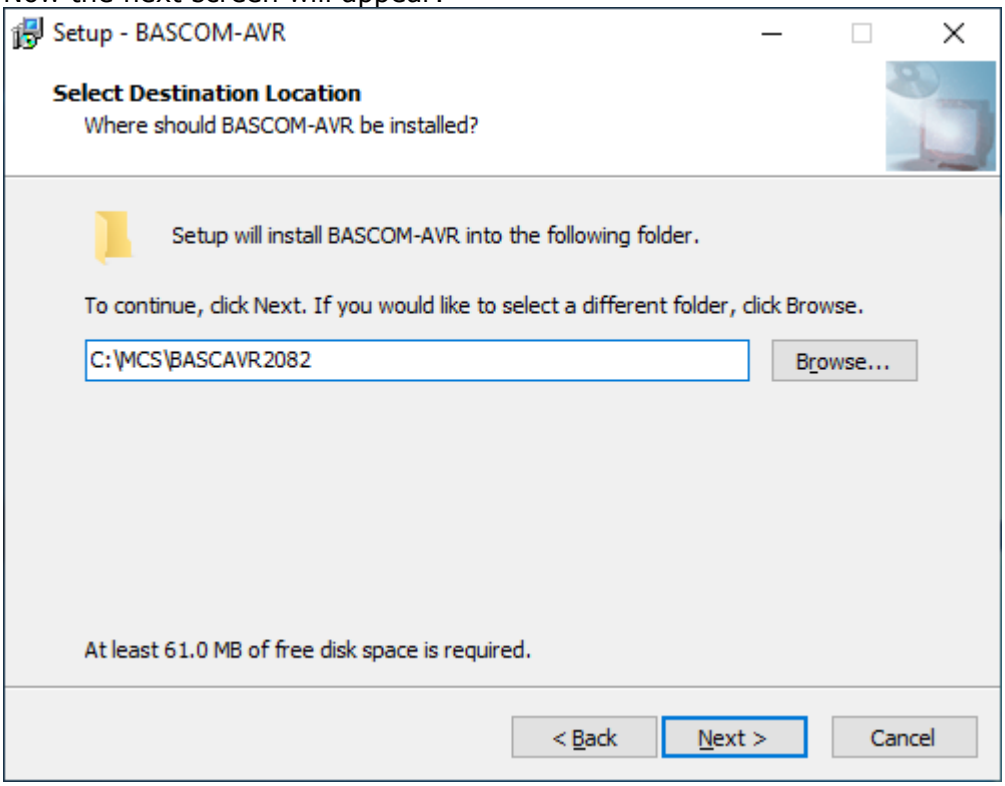
Read the instructions , select **'I accept the agreement'** and press the **Next button**.

The following window will be shown :



Read the additional information and click the **Next button** to continue.

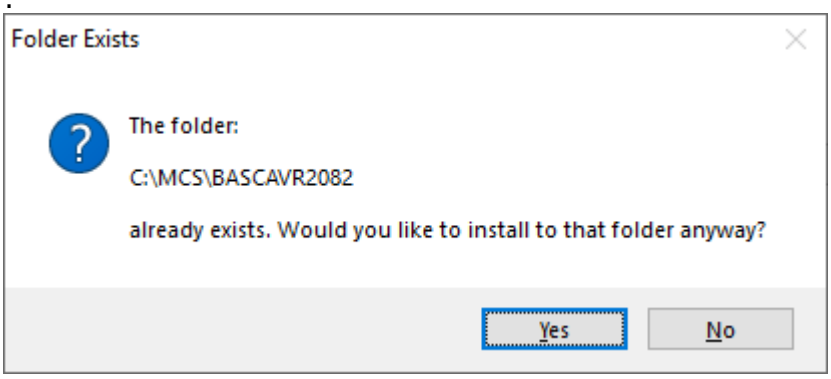
Now the next screen will appear:



You can select the drive and path where you like BASCOM to be installed. You can also accept the default value which is :
C:\MCS\BASCAVR2082
or you can install into a folder like :
C:\Program Files\MCS Electronics\BASCOM-AVR

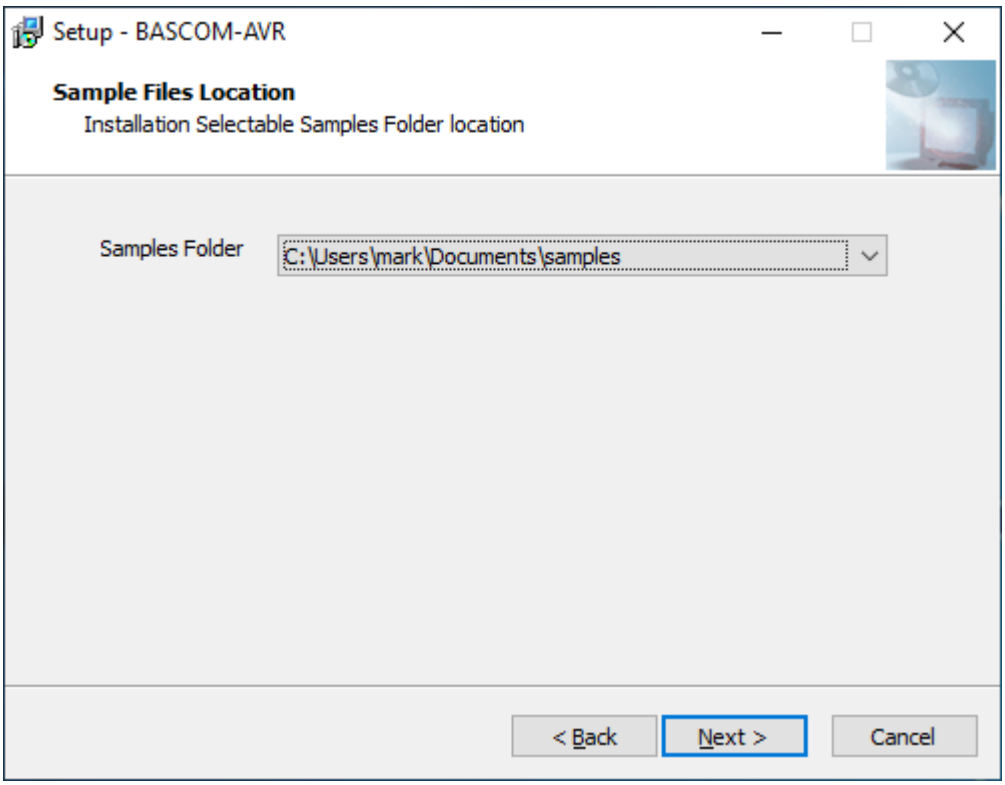
Microsoft likes software to be installed into the Program Files folder. But this also means that all sub folders must be stored elsewhere since all folders under Program Files are write protected by Windows.
Using a user writable folder, all the files can be stored in one location.
It is a good idea to install each new version into its own folder. This way, you can use multiple versions at the same time. As of version 2082, the settings file is stored in the application folder too.

When you are finished click the **Next Button** to continue.
When the directory exists, because you install a newer version, you will get a warning :



In case of this warning, select **Yes**. Or select NO and select a different folder.

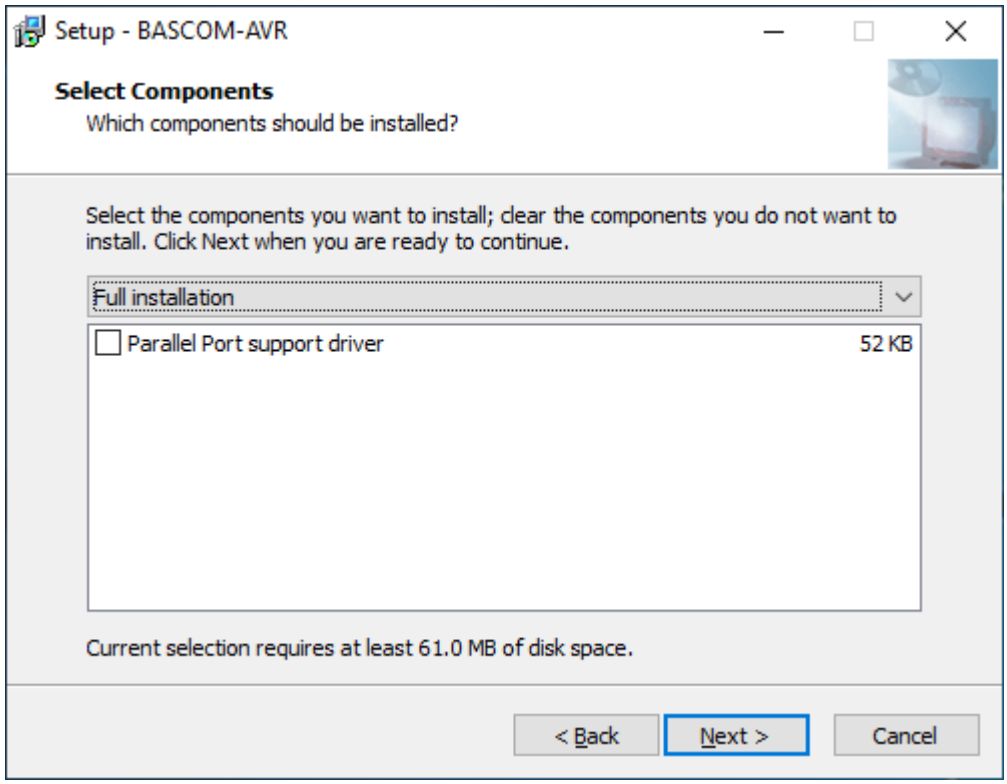
You will now see the following window:



You can select the folder where the sample files are installed. This can be :
c:\users<USER>Documents\samples
or c:\MCS\BASCAVR2082\Samples

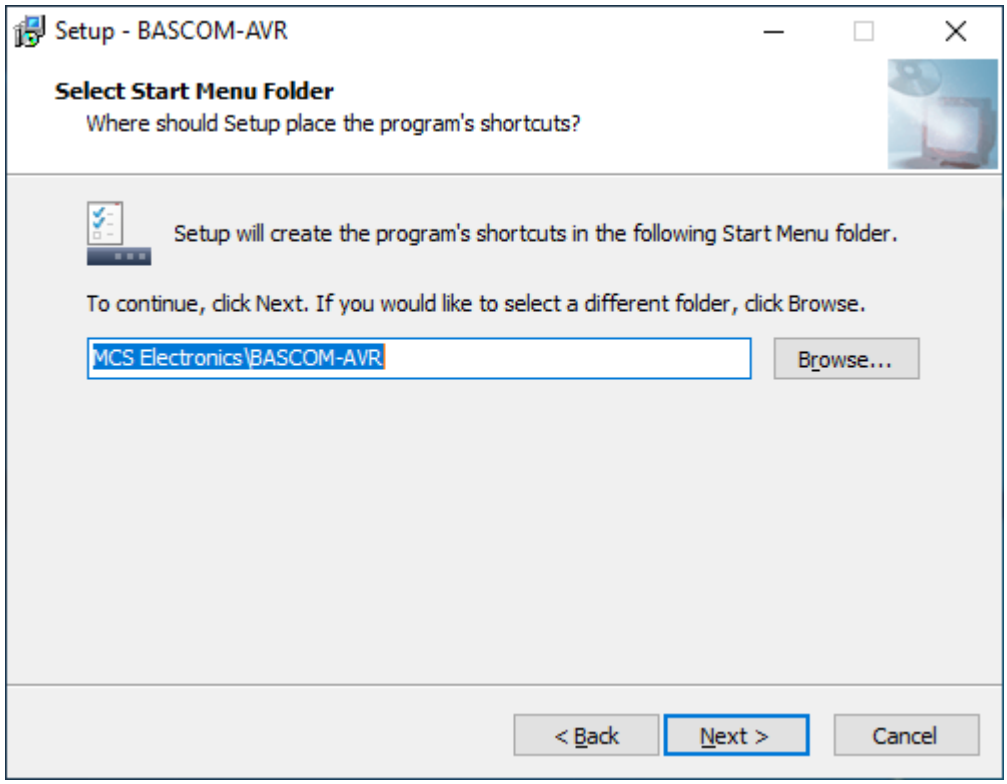
We recommend to use the second option so all files are placed under the application folder.
After you made your choice, click the **Next** button.

You are now presented with an optional component : parallel printer programming support.
Nowadays there are plenty serial and USB programmers available. Only select this option when you still use the LPT port for ISP programming.



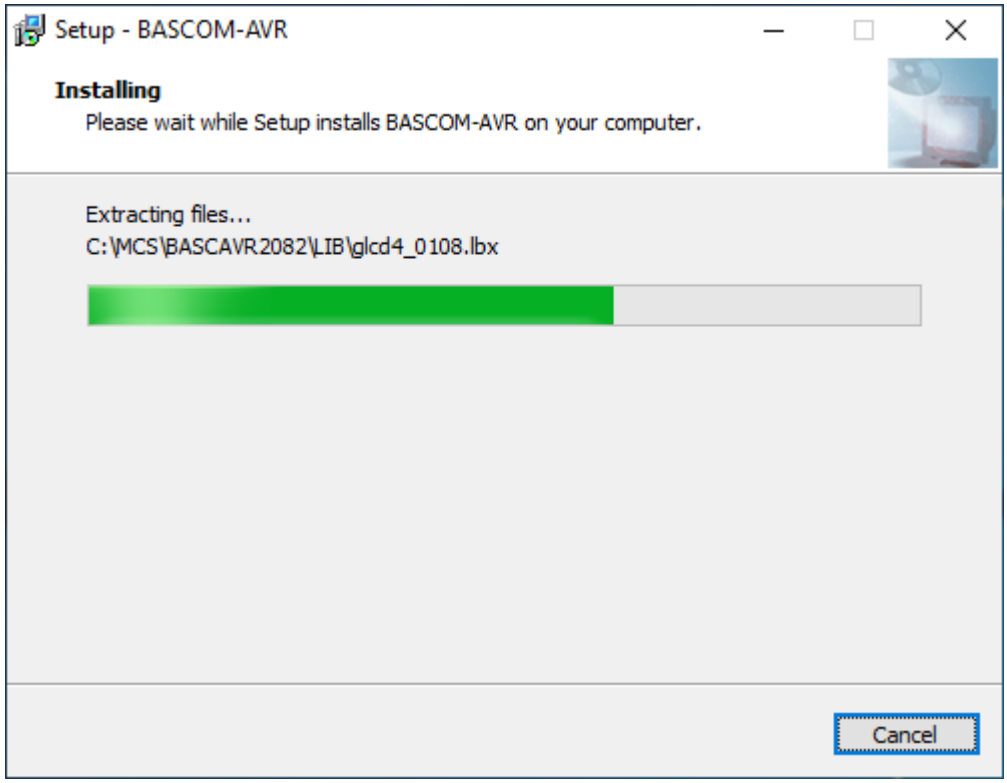
Click the **Next** button to continue.

You will now be presented a choice for the program group name and location.

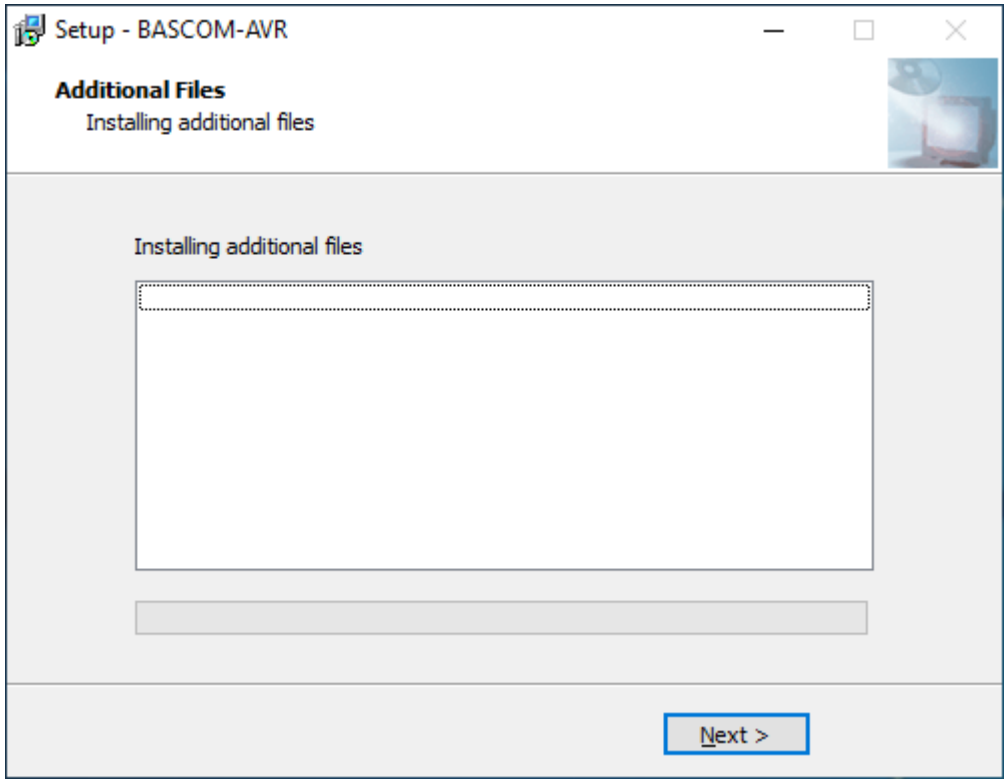


You can choose to create into a new Program Group named 'BASCOM-AVR' , or you can modify the name, or install into an existing Program Group. Press the **Next-button** after you have made your choice.

Now the files will be installed.

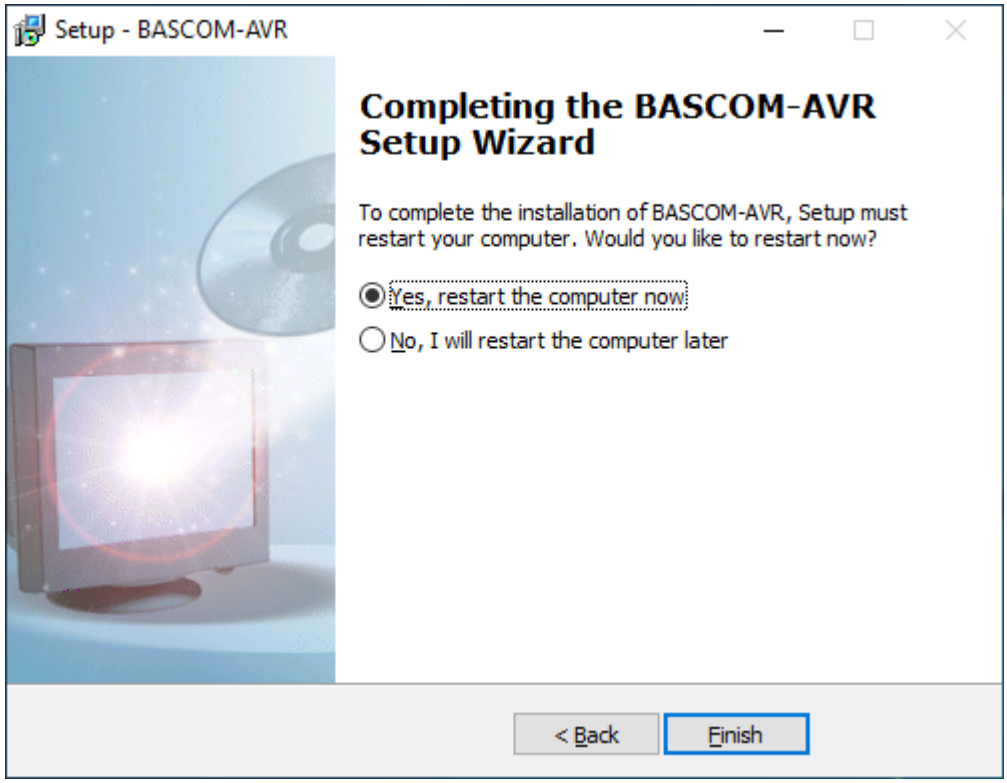


After the main files are installed, some additional files will be installed. This depends on the distribution.



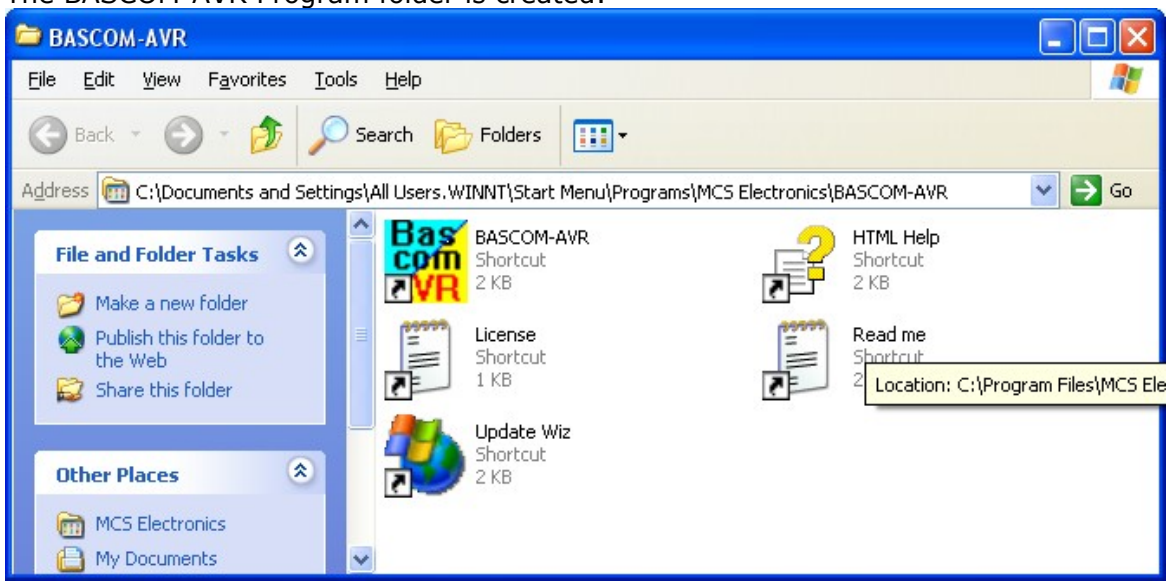
These additional files can be PDF files when the program is distributed on a CD-ROM.

When the installation is ready you will see the last screen :



You have to reboot your computer when you want to make advantage of the programmers that BASCOM supports. You can also do this at a later stage.

The BASCOM-AVR Program folder is created:



You can view the "Read me" and "License" files content and you can start BASCOM-AVR.
BASCOM supports both HTML Help and old Win help(HLP). The HLP file is not distributed in the setup. You need to use the Update Wiz to download it. But it is advised to use the HTML-Help file.

When you used to use the HLP file, and find it missing now, turn on 'Use HTML Help' in Options, Environment, IDE. 150

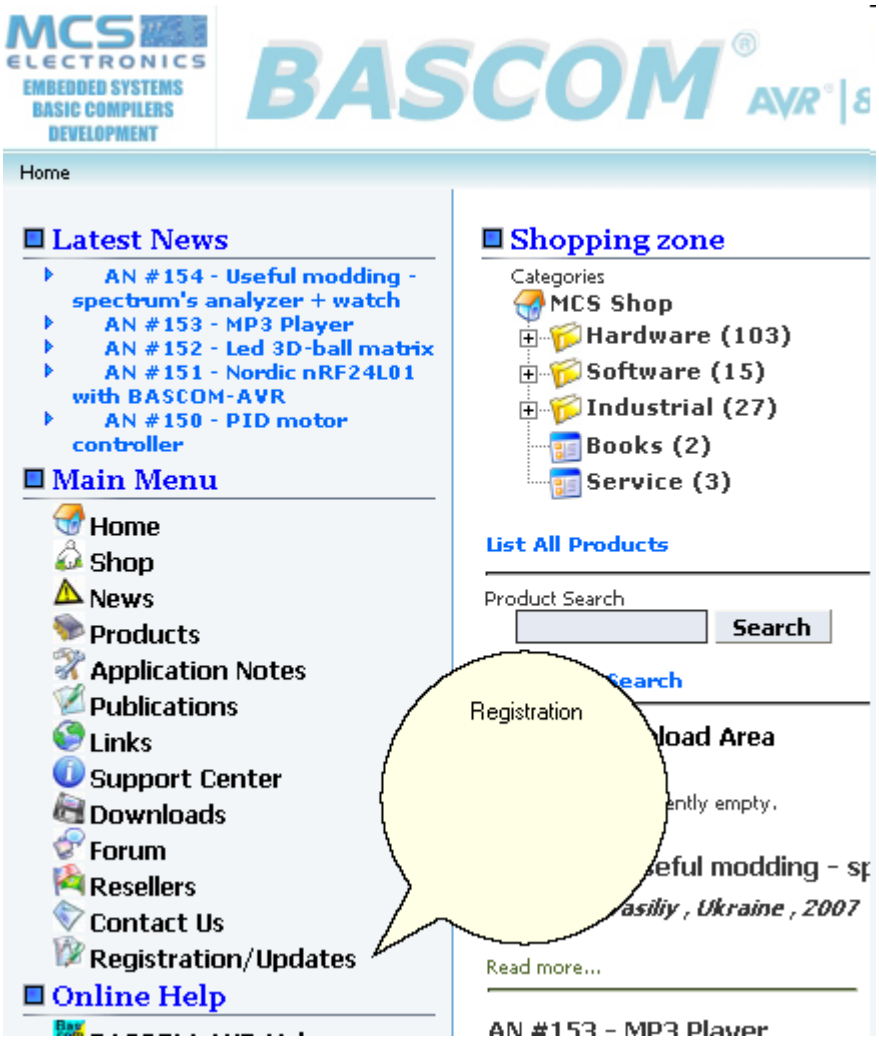
When the UpdateWiz is not installed, you can download it from the register 61.
The option Help, Update 235 will also download the wiz.

Till version 2074 all sample files were placed under the MCS Electronics\BASCOM-AVR folder.
Version 2075 places the sample files under the user Documents\MCS Electronics\BASCOM-AVR\Samples folder.
While we prefer to keep all files at one location and sub folders, this is not allowed in Windows 7 where the **Program Files** folder and all it's sub folders are write protected.
In version 2082 you can decide where the samples must be installed

The BASCOM-AVR application contains a number of folders.
\DAT : processor data files. These files contain processor info. When you use $REGFILE, the value should match with one of the files.
\LIB : library files. They have the extension LIB or LBX. LBX is a compiled LIB file. A library files contains ASM sub routines.
\INC : include files. Notice that these server only the compiler. Do not change or store include files here. Normal include files are stored along with the samples.
\PDF : PDF files with the bascom-avr manual and processor files from microchip/ atmel.
\PINOUT : processor pinout and XML description files
\SAMPLES : this depends on the user choice during installation

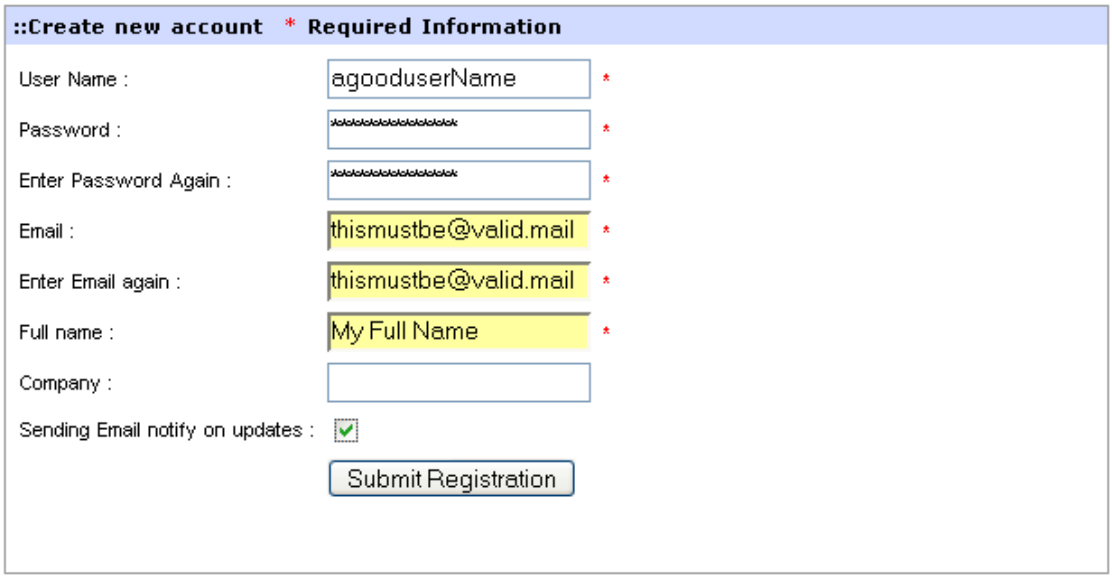
2.2 Updates

The update process is simple if you follow **all** steps.
- Go to the main MCS website at https://www.mcselec.com
- In the left pane under 'Main Menu' you will find a link named 'Registration/Updates'
- Optional you can enter the address yourself : https://register.mcselec.com



Notice that the website uses **two different** accounts : one for the forum/shop and one for the registration/updates. You will see the following screen:

- When you don't have an account yet, Click the link and select 'Create new account'



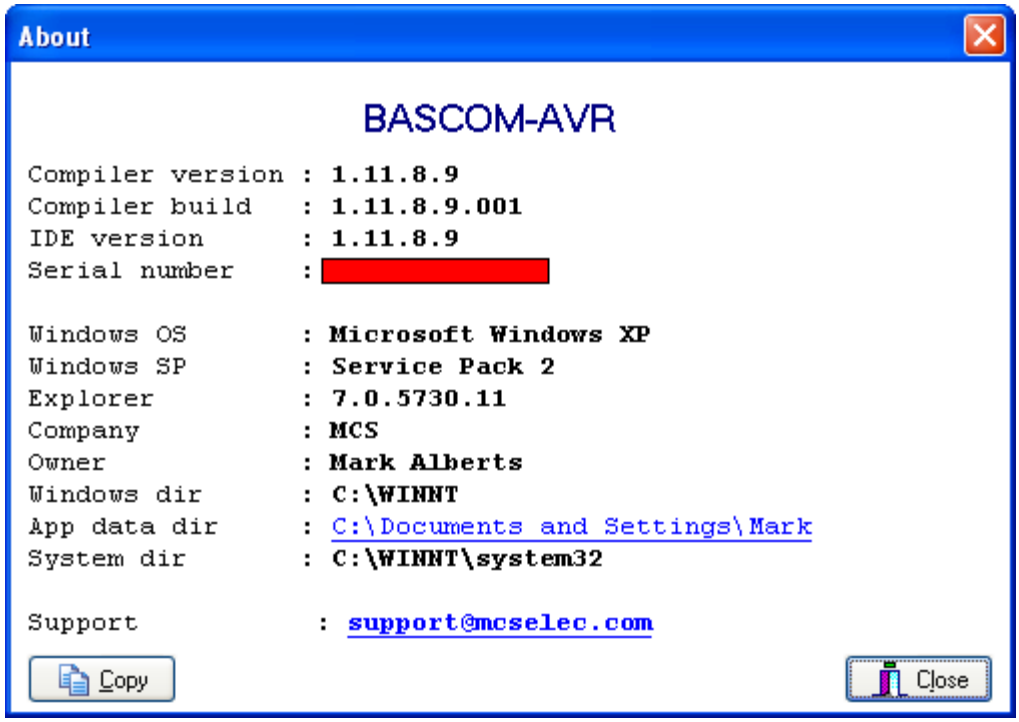
You need to provide a username, password, email and full name. Company name is optional.
When you filled in the information, click 'Submit Registration'.

- After you click submit, you can get various error messages. For example that a username already exists. Press the Back-button in your browser, and correct the problem, then try again
- If the registration is successful you will get a message that the registration succeeded.
- Watch your mail : you will receive an email with a confirmation link. You need to click this link in order to finalize your account
- Now you can login. You will see the following or similar screen :

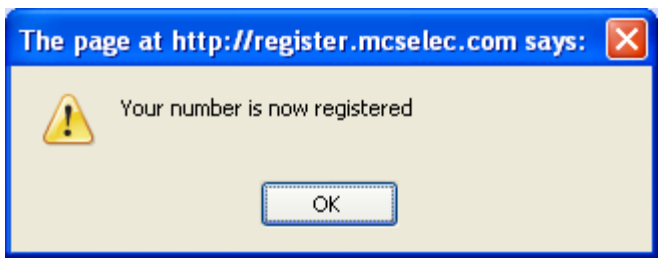
- You need to chose 'Product registration'.
- The following screen will be shown:

- Select a product from the list.
- Enter the serial number

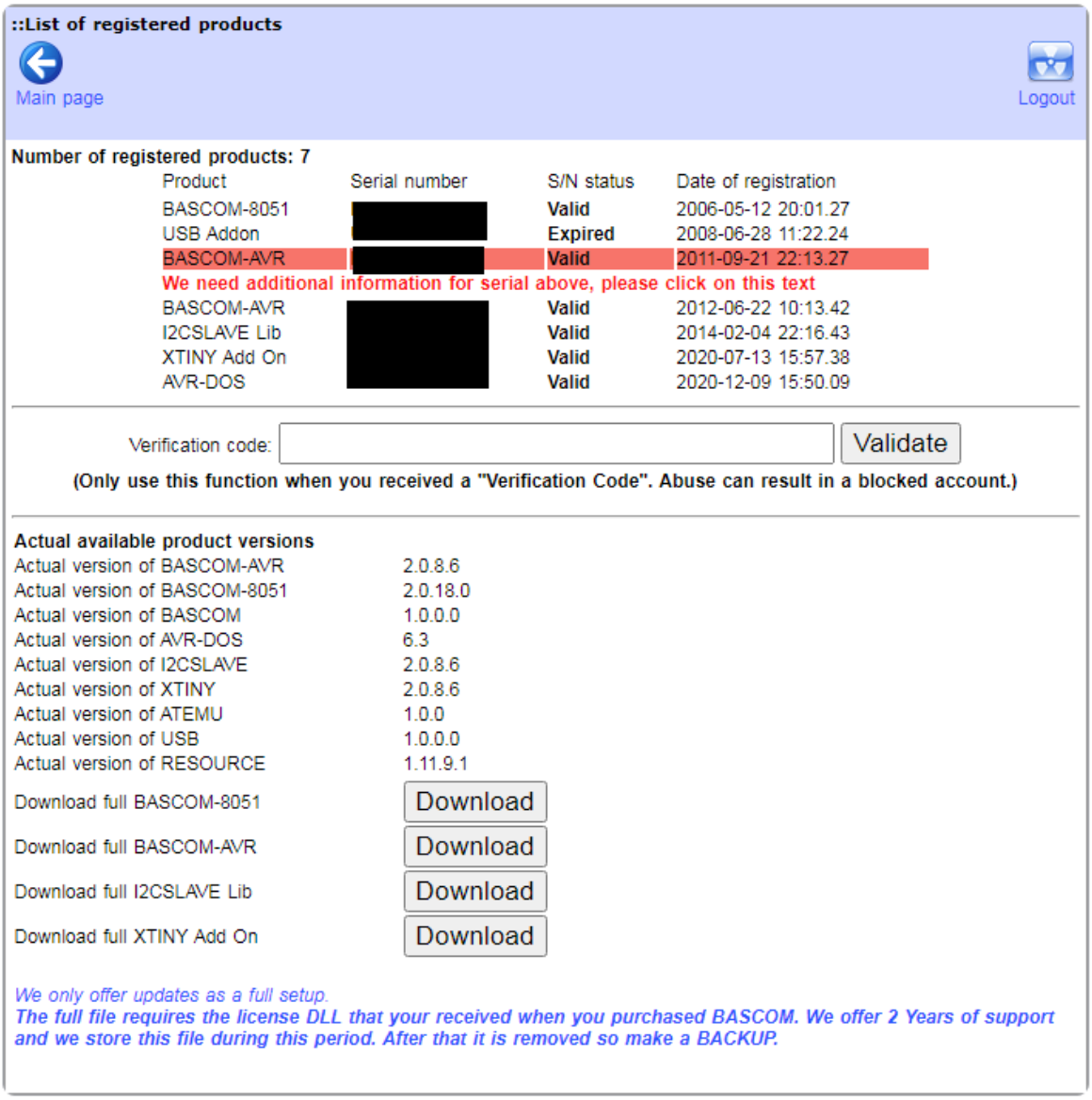
It is important that you enter a **valid** serial number. Do not try to enter serial numbers from cracked versions. When you enter invalid serial numbers, you will loose support and the ability to update. We will also ban your IP number from our web. The valid serial number is shown in the Help, About box.



When the product is selected, the serial number is entered and you press 'Register product' you will see the following message :



- This does mean that you registered successfully.
- MCS Electronics will validate all registrations once in a while. When the product is validated you will receive an email. After you receive the email, you can login to the register again.
- Note that you need to register within 2 years after purchase. Only the original buyer can register and is qualified to get updates and support.
- Once registered you will see the following or similar screen :



At the top you can see which products are registered and which status they have.
When you want to do an update you need to do a FULL SETUP, you need to download the full version. LIC files for partial updates are no longer offered.
You **do not** need to **uninstall a previous version**. You can install an update into the same directory or a new directory.
We recommend to install into a new folder.
When you uninstall a previous version, it will remove the license file which is not part of the setup.exe

 So in the event that you do run uninstall first, make a backup of the license dll named **bscavrL.DLL**

For BASCOM, the ZIP file you download contains only one setup.exe. You need to run this executable.
It is also important that you put the license DLL into the same directory as setup.exe Setup will copy this file to the BASCOM application directory. You can also manual copy this file.
The license file is on CD-ROM, diskette, or the media (email) you received it on. **It is only supplied once.**
Without the file, BASCOM will not run.

The file is named bscavrL.DLL for BASCOM-AVR
When you got the license by email, it was zipped and probably had a different extension. Consult the original installation instructions.
The file is only provided once, we can not, and do not provide it again.

Add-on products can contain multiple files like lib, bas, pdf, etc.

See Installing BASCOM 53 on how to do a full install.

IMPORTANT

 As of version 2080, the Update Wiz is phased out. This means that you need to download and install the full setup.exe
The BASCOM-IDE has a new simplified update method. See also Help, UPDATE 235

2.3 Move to new PC

When you want to move BASCOM to a new PC. You have a number of options.

1 - Run the installer with admin rights from CD-ROM on your new PC. The setup will copy the license file automatically.
2 - Download the latest version of the setup.exe from https://register.mcselec.com , extract the setup.exe , and run setup.exe with admin rights

For the register link above, you need access(an account). This account is not the same as for the shop/forum.
You need to create an account if you don't have one.
This procedure is explained in the help topic 'Updates 61'

After the installation, copy the license file **bscavrL.DLL** to the bascom-avr application directory of the new PC.
Or let setup.exe do this for you. When you put the license file in the same directory as setup.exe, setup will copy/install the file for you.

In general it is always better to install the latest version.
Support is only offered on the latest available version.

2.4 Installation on multiple computers

The following applies to the licensed version and the license key.

You may install BASCOM on multiple computers. For example on your laptop and your desk PC. There is no limit to the number of PC's you install the software on.
But you may only use one PC at the same time. Since you can only operate one PC at the same time, this is not a real restriction.
When you install on multiple PC's and others work on these PC's at the same time as you, you need multiple licenses!

The same applies for all the add-on libraries/products you purchase.

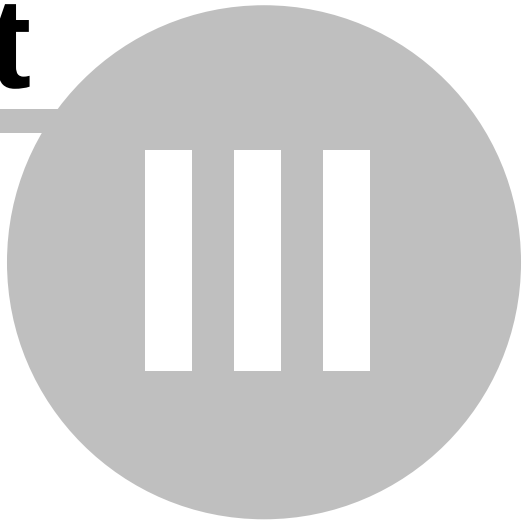
We do not want to bother customers with anti piracy dongle and internet controlled licenses.

2.5 Registration

The software must be registered in order to get updates. This is explained in the Updates 61 topic.

We want to emphasize that it is important that you register soon as possible, at least within 1 year after purchase.
When you do not register timely, the license will not be approved. This means that you can no longer update the software.
Of course the software keeps working as is. It does not depend on the registration.
But when you need support, we check if you use an actual version.

Part



3 BASCOM IDE

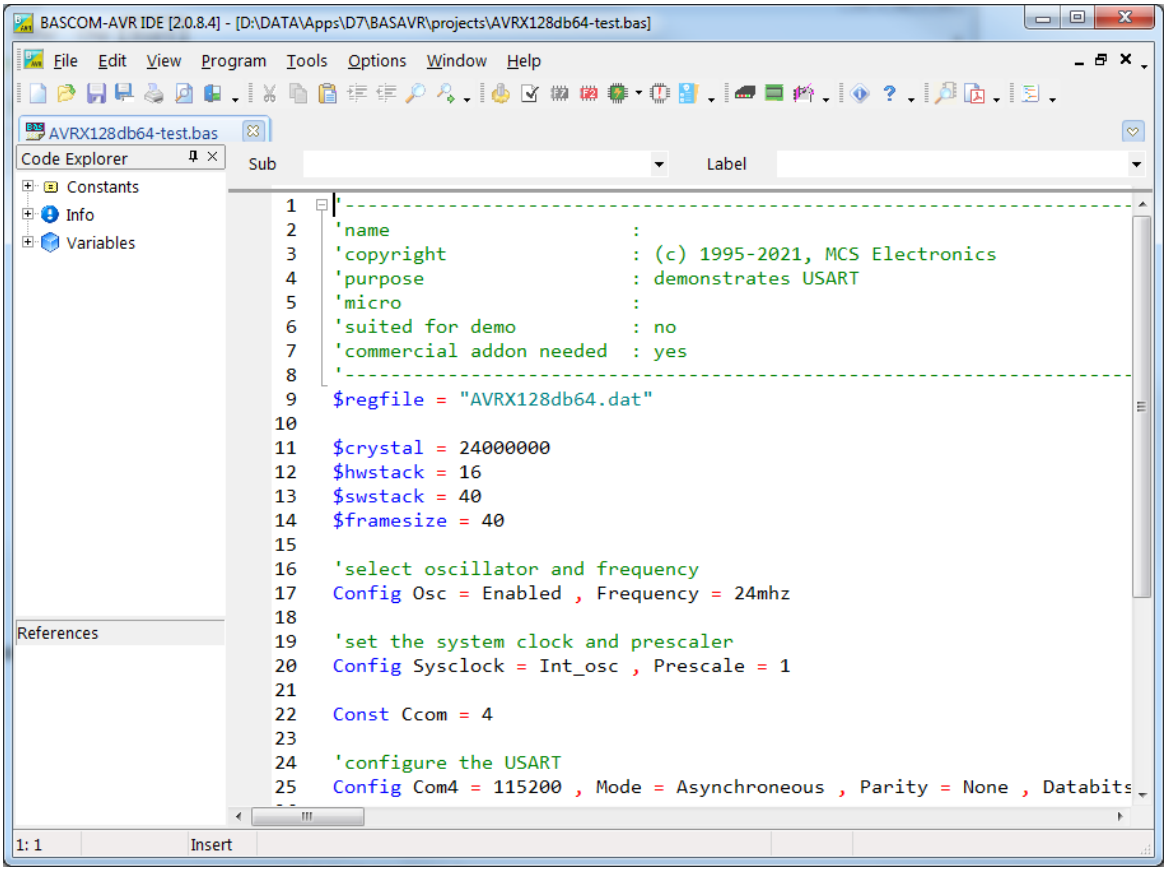
3.1 Running BASCOM-AVR

After you have installed BASCOM, you will find a program entry under *MCS Electronics\BASCOM-AVR*

Double-click the BASCOM-AVR icon 

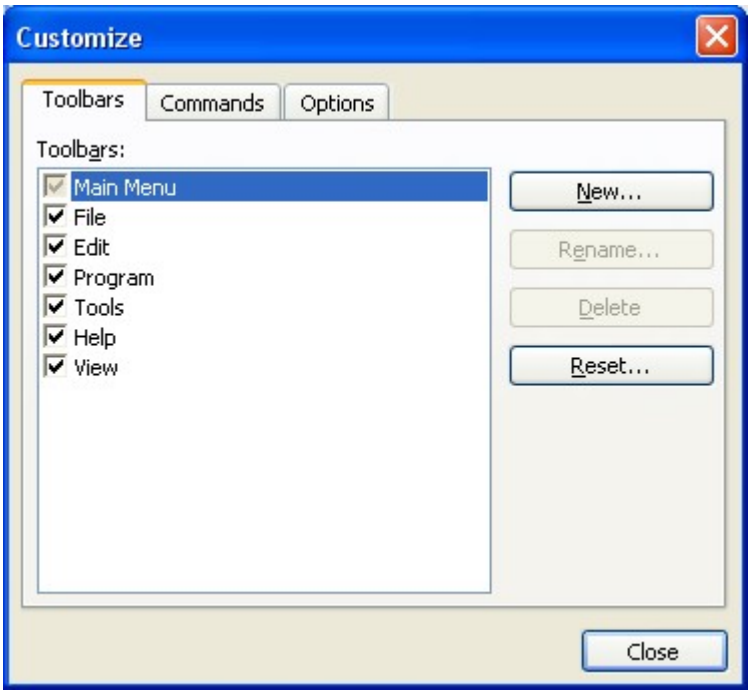
to run BASCOM.

The following window will appear. (If this is your first run, the edit window will be empty.)



The most-recently opened file will be loaded automatically. Like most Windows programs, there is a menu and a toolbar. The toolbar can be customized. To do this, place the mouse cursor right beside the 'Help' menu.
Then right-click. You can turn on/off the toolbars or you can choose 'Customize'.

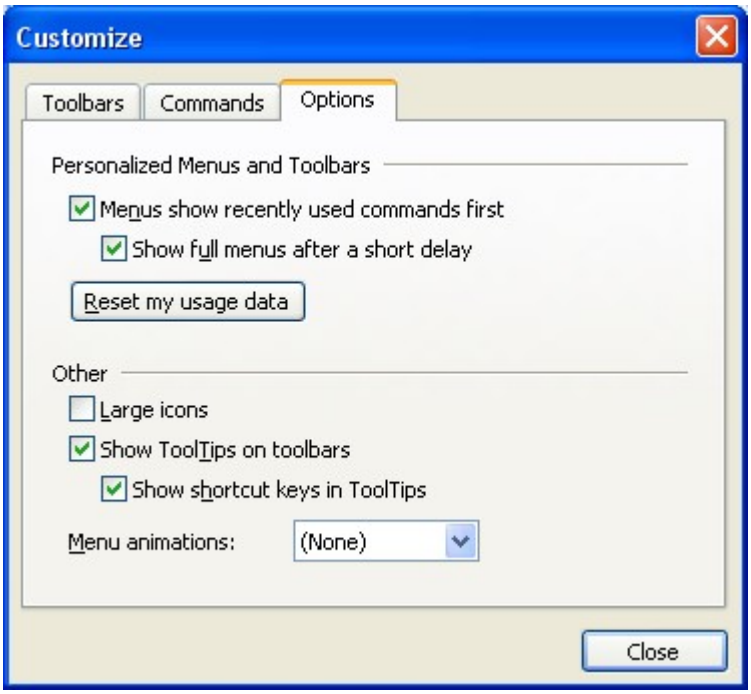
This will show the following window:



You have the option to create new Toolbars or the reset the toolbars to the default.
To place a new button on a menu bar, select the 'Commands' TAB.

In the example above, the Program Category has been selected and at the right pane, all buttons that belong to the Program-category are shown.
You can now select a button and drag & drop it to the Toolbar. To remove a button from the Toolbar, you drag it out of the Toolbar and release the left mouse button.

On the Options-TAB you can further customize the Toolbar:

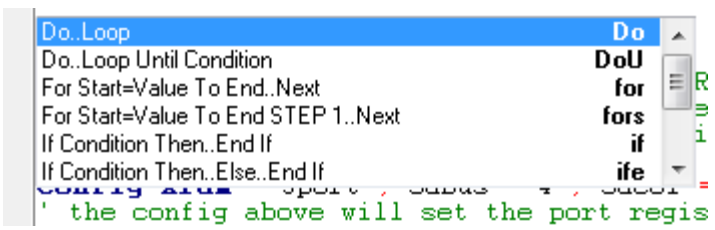


To preserve screen space there are no large icons available.

Option	Description
Menus show recent used commands first	With this option the IDE will learn the menu options you use. It will show only the most used menu options. The idea is that you can find your option quicker this way.
Show full menus after a short delay	This option will show the remaining menu options after short delay so you do not need to click another menu option to show all menu options.
Reset my usage data	This option will reset the data the IDE has collected about your menu choices.
Show Tool tips on toolbars	This option is on by default and it will show a tool tip when you hold the mouse cursor above a toolbar button
Show shortcut keys in Tool tips	This option is on by default and it will show the shortcut in the tool tip. For example CTRL+C for the Copy button.

The Editor

The editor supports syntax highlighting. Code you enter can be reformatted automatically.
When you press CTRL+J you can select a template. A template is a small piece of code that can be inserted automatically.
When you press CTRL+J you can select a template or you can type the template name and press CTRL+J. If there is only one template starting with that name, the template will be inserted. Otherwise the options are shown.



Templates are stored in the file **bascavr.tpl**

When you press SHIFT and move the mouse cursor over a variable, constant or other element you will get a tool tip with info.

```
Dim Ar(10) As Byte
Dim J As Byte
Dim W As Word
Dim L As Long
Dim S As String * 16
```

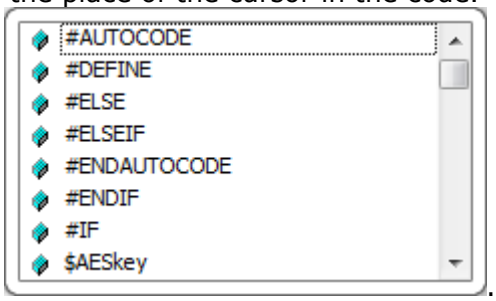
S As String * 16 - crc8-16-32.bas

In the sample above the variable 's' was selected and the tool tip shows that it is a string with a length of 16 bytes in the modules crc8-16-32.bas

Intellisense

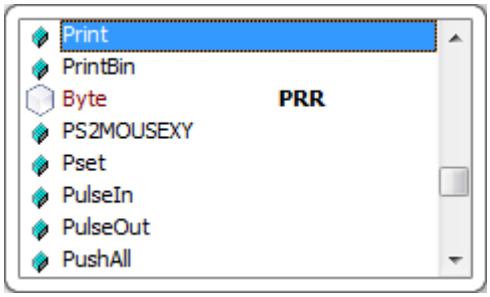
The editor has built in intellisense.
It is important that your code contains the $REGFILE directive like : $REGFILE = "M88def.dat".
When you press CTRL+SPACE you get a list of statements, sub routines, functions, labels, asm registers, etc. This list depends on the place of the cursor in the code.

- At the start of a line you will get a list like :



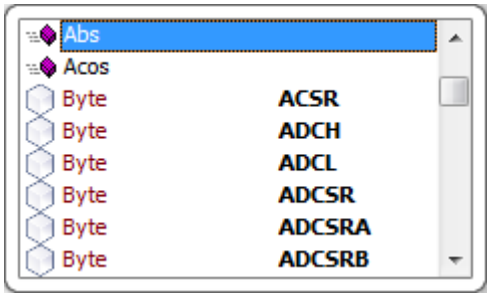
.
You can select a value from the list and press enter to insert it into the code.

- When you type a letter of some letters like pr

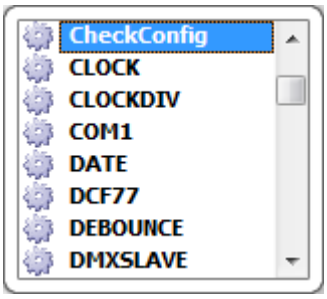


Here you can see the position is set to the first item that starts with PR : PRINT

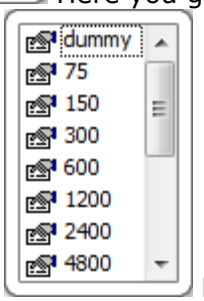
- After PRINT when a variable is expected :



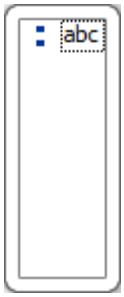
Here you get functions, variables and constants

- After CONFIG 

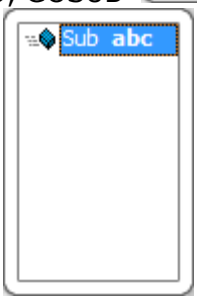
Here you get a list of all CONFIG statements.

- After CONFIG param (the = sign). 

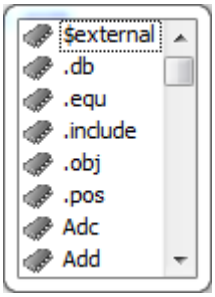
Here you get a lost of parameter values.

- After GOTO, GOSUB 

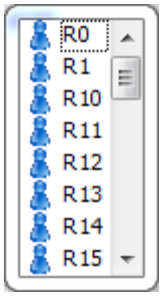
Here you get a list with labels.

- After CALL 

Here you get a lost with sub routines.

- Inside $ASM-$END ASM 

Here you get a list of ASM mnemonics.

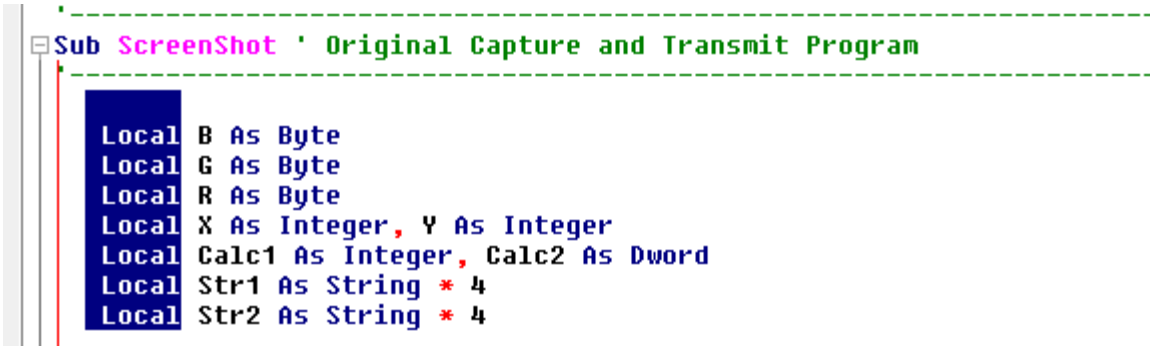
- After ASM mnemonic 

Here you get a list of registers.

PLEASE NOTICE :
- intellisense is considered a beta function. It is subject to change. It will only work when there are no syntax errors.
- values for CONFIG might not be shown. This is because all these values need to be present in the DAT files. And each processor has specific options.

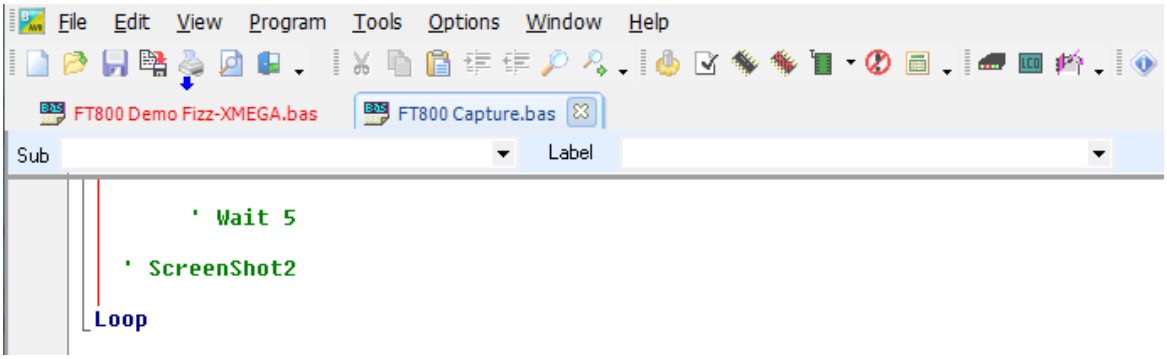
Select Text

Selection of text can be done by double clicking the text, by holding SHIFT down and moving the cursor or you can select a block of text by pressing the ALT key and dragging the mouse cursor.



TABS

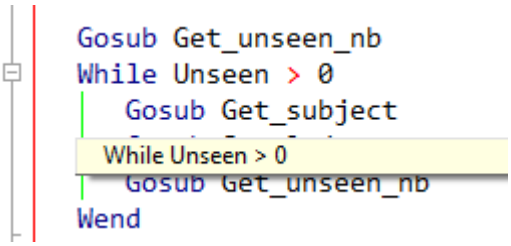
When you have loaded multiple files, each file will be shown in a TAB. The active TAB can be closed or dragged to a new position. When a file is modified the TAB caption will be shown in red.



SHIFT + MOUSE

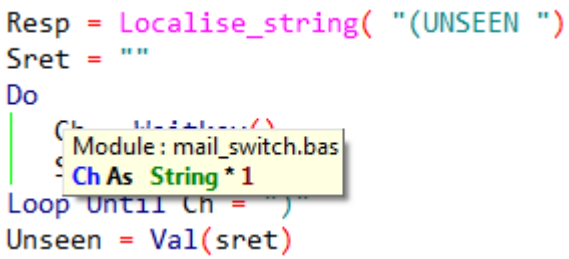
When you move the mouse cursor to the TAB caption you will see the full path of the loaded file.

When you press the SHIFT key and move the mouse cursor you can get information in a tool tip.
For example when you hover over an indention line :



The tooltip shows info about the structure. So you know that the green line belongs to While Unseen > 0

When we hover over a code element like CH :



This time since CH is a variable. the data type is shown.

In 2084 String constants will be shown with their length.

The Reference window will list all referenced variables :

When you click an item, the cursor will be changed automatically.

Custom Configuration

You can load a custom configuration file by specifying the filename as a parameter. This allows you to run different versions of the software with different setting/option files.
The configuration file has the XML extension. It can be found by clicking the XML data folder link in the Help, About window.
By default bascom uses the file : \Users\<USER>\AppData\Roaming\MCS Electronics\bascom-avr**XXXX**.xml
The **XXXX** is the version. For example 2082.

When you want to use a custom file we would recommend to store it in the bascom-avr application folder. This way you can run multiple versions of bascom, all with their own settings.

The name of the settings file must be provided as a parameter to BasCom. For example to use a settings file named **mysettings.xml** :
bascavr.exe **mysettings.xml**

When BasCom is started it will check if the provided file exists and will load the settings of that file.
If the files does not exist, the normal setting file ise used.

The reason for unique file names is that once in a while a menu option is added. That is no problem when you update, but when you want to use the xml with an older file you could get errors because of non existing menus.

3.2 File New

This option creates a new window in which you will write your program.
The focus is set to the new window.
You can have multiple windows open at the same time.
Only one window can have the focus. When you execute other functions such as Simulate 111 or Program Chip 124, BASCOM will use the files that belong to the current active program. This is in most cases the program which has the focus.

File new shortcut:  , CTRL + N

3.3 File Open

With this option you can load an existing program from disk.

BASCOM saves files in standard ASCII format. Therefore, if you want to load a file that was made with another editor be sure that it is saved as an ASCII file. Most programs allow you to export the file as a DOS or ASCII file.

Note that you can specify that BASCOM must reformat the file when it opens it with the Options Environment 150 option. This should only be necessary when loading files made with another editor.

File open shortcut :  , CTRL+O

3.4 File Close

Close the current program.

The current editor window will be closed. When you have made changes to the program, you will be asked to save the program first. You can then decide to save, cancel, or not to save the changes you have made.

File close shortcut : 

3.5 File Save

With this option, you save your current program to disk under the same file name. The file name is visible in the Windows caption of the edit window.

If the program was created with the File New 76 option, you will be asked to name the file first. Use the File Save As 77 option to give the file another name.

Note that the file is saved as an ASCII file.

File save shortcut :  , CTRL+S

3.6 File Save As

With this option, you can save your current program to disk under a different file name.
When you want to make some changes to your program, but you do not want to make changes to the current version you can use the "Save As" option. It will leave your program as it was saved, and will create a new file with a new name so you end up with two copies. You then make changes to the new created file.

Note that the file is saved as an ASCII file.

File save as shortcut : 

3.7 File Print Preview

With this option, you can preview the current program before it is printed.
Note that the current program is the program that has the focus.

File print preview shortcut : 

3.8 File Print

With this option, you can print the current program.
Note that the current program is the program that has the focus.

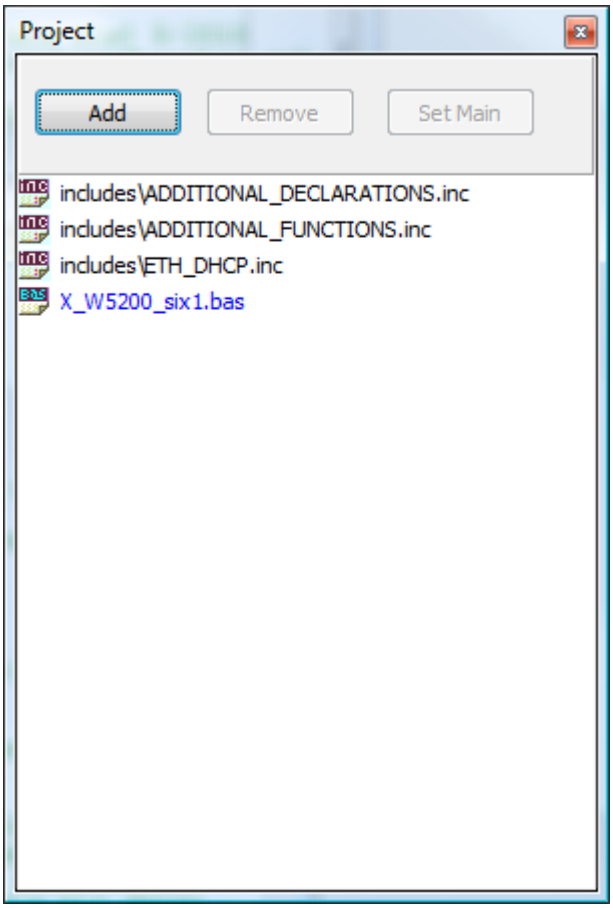
File print shortcut :  , CTRL+P

3.9 File Project

Originally the IDE was not designed to support projects. Each file you open is a project.
Most chips were not even suited for big projects.
Some projects use a lot of include files. It is a good idea to break up your code in modular tested modules.
You can simply include the modules with $include 654.

In order to make working with a project more convenient, a number of Project options have been added. The Project menu can be found under the File menu. The Project menu has 4 sub menu items and a MRU list(most recent used projects).
When in project mode, the main project file will be compiled. In normal mode, the active window is considered the project and will be compiled. The same is true for the simulator and programmer.

A simple project explorer has been added that will list all project files. The active project will be shown in blue. The relative path is shown.



You can add a new file to the active project. By default the INC extension will be selected. It will be good practice to give included files the INC extension. The main project should have the BAS extension. When you click the ADD button, a file selection dialog will appear. You can select multiple files by using the SHIFT and/or CTRL keys.

When you add a file to a project, it will be added to the project list. When you double click the file in the list it will be selected. Or when it was not loaded before, it will be loaded from disk.
That a file is part of a project collection does not mean that the file will be used or included : you still need to $INCLUDE 654 a file that you want to use in your project.

You can also remove a file from the project. This will not remove or delete the file from disk. The file will only be removed from the project collection.

Only one file can be the main project. This is the file that will be compiled. The main file is colored in blue.

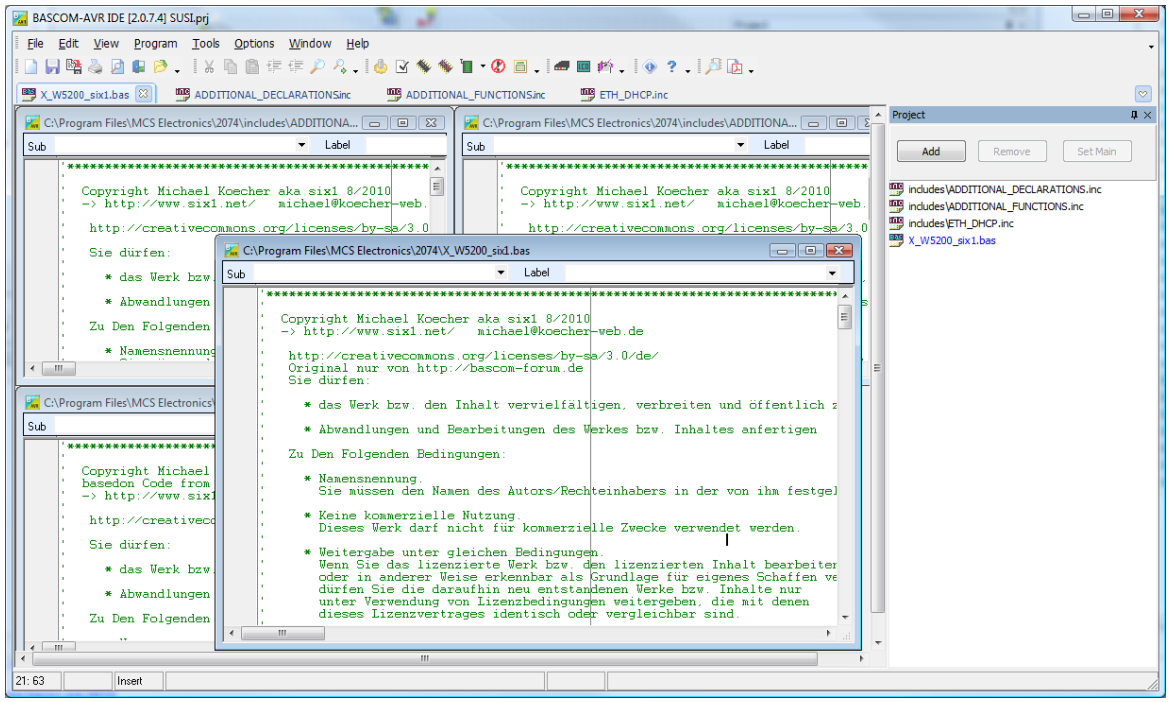
When you updated from a previous version, you need to reset the docking in order to make the Project List window visible. This option you can find under Options, Environment, IDE 150

Project New

This option will close all files and the current project and will query for a project file name. The file will have the PRJ extension.

Project Open

This option will close all open files and let you select an existing project file. A project file has the PRJ extension.
The PRJ file contains no code, it only contains data about the project files.
All files from the project will be loaded when they were loaded when you closed the project.
The position and size will be set exactly as when you closed.



Project Save

This option will save all project files. It will also save other opened non-project files.

Project Close

This option will close the active project. This will end the project mode. The project mode is started when you open a PRJ file either with OPEN or by clicking a PRJ file from the MRU menu.

When you close bascom and you have the Option 'Auto Load All Files' checked, then like usual, all open files will be saved and when you run bascom again, they will all be opened. This might be confusing since you work in normal mode by default. It is recommended to deactivate the 'Auto Load All Files' when working with projects.

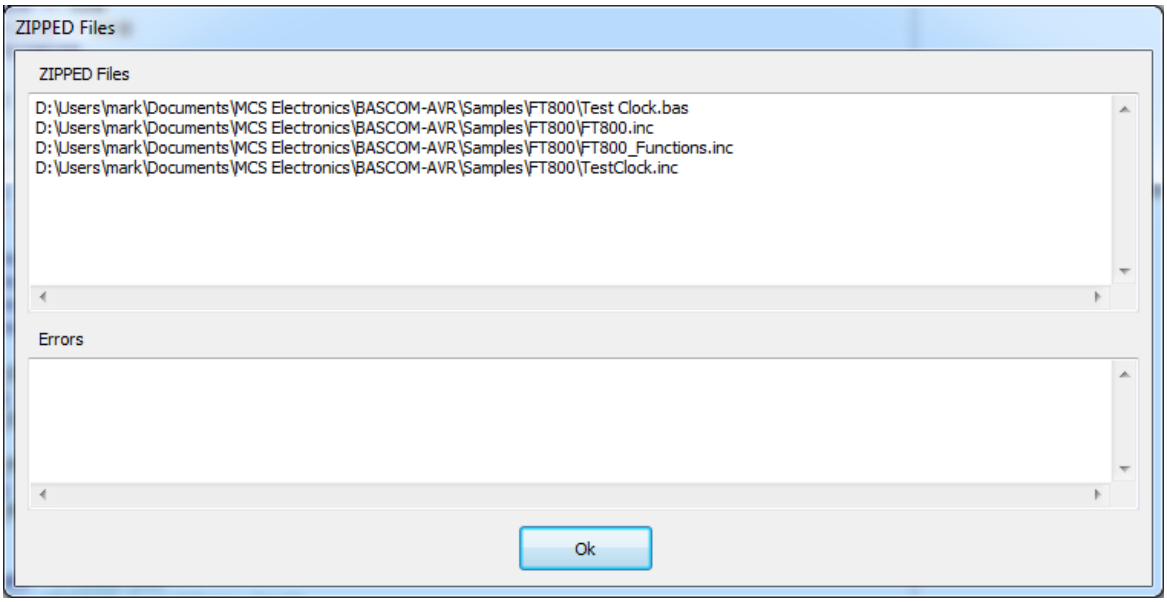
In project mode, you can also drag and drop files to the IDE. If they have the BAS or INC extension, they will be added to the project. In normal mode, the file will be opened.

3.10 File ZIP

This option will put all project files into a ZIP file.
The file will be given the ZIP extension and is saved into the same folder as the main file.
When your file is named main.bas, the file main.zip will be created.

The following files will be included :
- all files which are included with $INCLUDE
- all files which are included with $INC
- all files which are included with $BGF

If a file is included in the code but can not be found you will get a warning.



This option does take conditional compilation into account.
Meaning that :

```
#const a=1
#if a=2
   $Include "FT800.inc"
   $Include "FT800_Functions.inc"
#endif
```

The files ft800.inc and ft800_functions.inc are not included since the condition does not match.

3.11 File Exit

With this option, you can leave BASCOM.
If you have made changes to your program, you can save them upon leaving BASCOM.

All of the files you have open, at the moment you choose exit, will be remembered. The next time you run BASCOM, they will be opened automatically.

File exit shortcut : 

3.12 Edit Undo

With this option, you can undo the last text manipulation.

Edit Undo shortcut :  , CTRL+Z

3.13 Edit Redo

With this option, you can redo the last undo.

Edit Redo shortcut :  , CTRL+SHIFT+Z

3.14 Edit Cut

With this option, you can cut selected text into the clipboard.

Edit cut shortcut :  , CTRL+X

3.15 Edit Copy

With this option, you can copy selected text into the clipboard.

Edit copy shortcut :  , CTRL+C

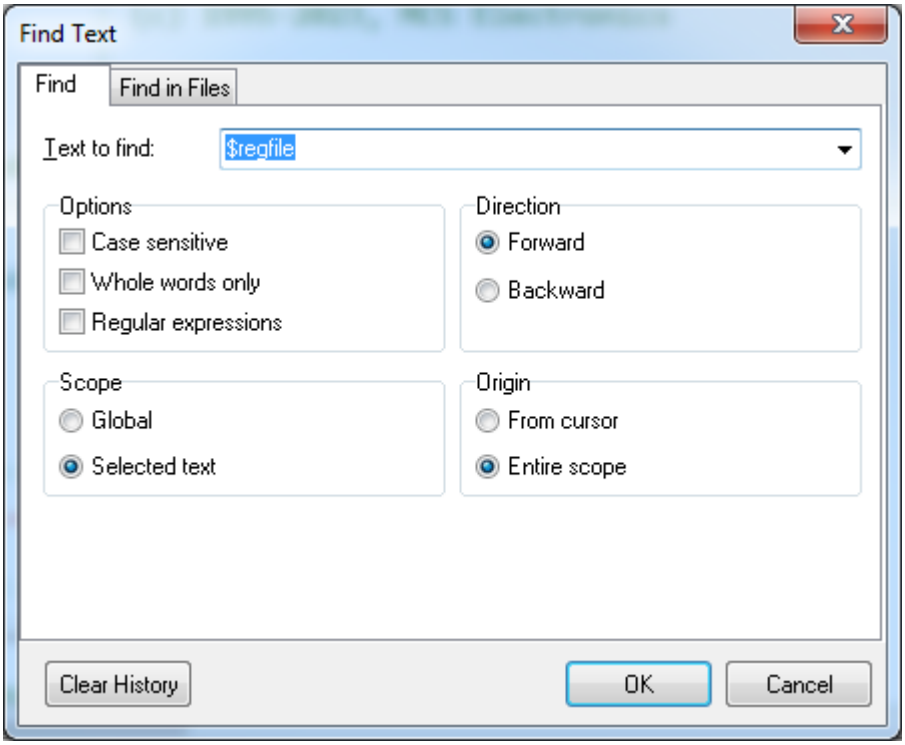
3.16 Edit Paste

With this option, you can paste text from the clipboard starting at the current cursor position.

Edit paste shortcut :  , CTRL+V

3.17 Edit Find

With this option, you can search for text in your program.
Text at the current cursor position will automatically be placed in the find dialog box. All text you search for is saved so the next time you search, you can retrieve the search phrase from a list.
Click the 'Clear History' button to clear the history. This will clear ALL find history from all pull down boxes in both the Find and Replace windows.



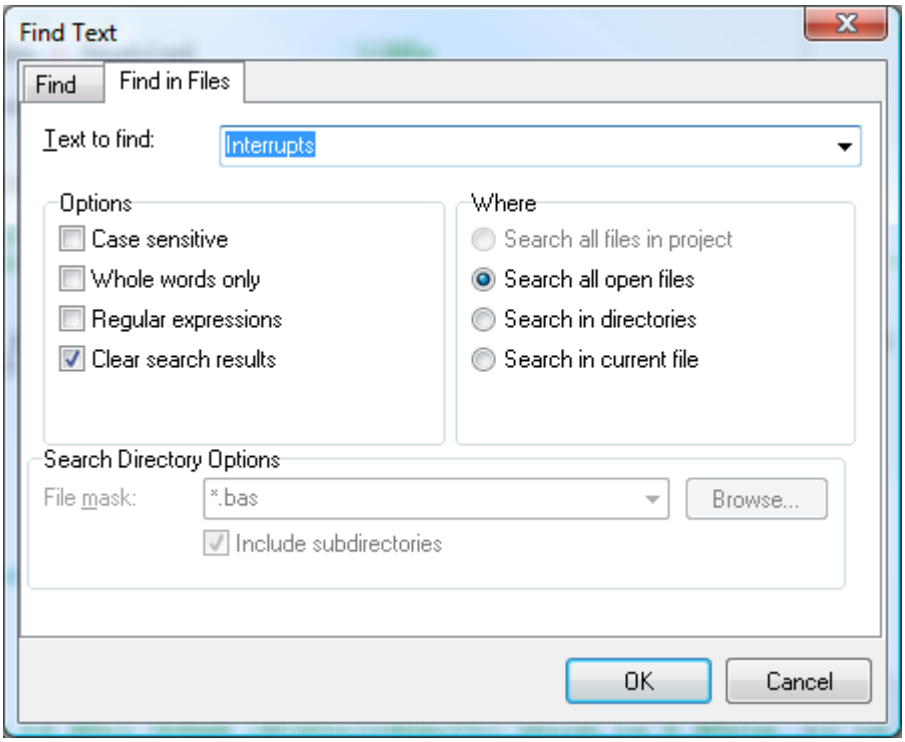
The following options available:

Option	Description
Case Sensitive	When selected, the case must match. Searching for PRINT will not find pRint. With this option turned off, Print will find print, PRINT, PRinT, etc.
Whole words only	When selected, only whole words are considered. A whole word is a word that is surrounded by spaces, or that is at the start of a line. Looking for PRINT will find : "Print test" and "print" and "print print". But not "printer"
Regular expressions	You can use a regular expression to find a match. ^ A circumflex at the start of the string matches the start of a line. $ A dollar sign at the end of the expression matches the end of a line. . A period matches any character. * An asterisk after a string matches any number of occurrences of that string followed by any characters, including zero characters. For example, bo* matches bot, bo and boo but not b. + A plus sign after a string matches any number of occurrences of that string followed by any characters except zero characters. For example, bo+ matches boo, and booo, but not bo or be. [ ] Characters in brackets match any one character that appears in the brackets, but no others. For example [bot] matches b, o, or t. [^] A circumflex at the start of the string in brackets means NOT. Hence, [^bot] matches any characters except b, o, or t. [-] A hyphen within the brackets signifies a range of

	characters. For example, [b-o] matches any character from b through o. \ A backslash before a wildcard character tells the Code editor to treat that character literally, not as a wildcard. For example, \^ matches ^ and does not look for the start of a line.
Forward	This is the search direction. By default it will search forward. Forward means down in this context.
Backward	This is the search direction. You can use backwards in case you pressed **F3** too many times and want to go back to the previous found text.
Global	All the text of the current editor will be searched.
Selected text	Only the selected text will be searched. So before you press CTRL+F to search for text you can select text and this option will be selected automatic. Otherwise global is selected.
From cursor	Search from the current cursor position to the end of the code.
Entire scope	Search from the current cursor position to the end, then search till the start of the cursor position. This will search the entire text.

Find in Files

The **Find in Files** option can be used to search for text in files.



Option	Description
Case Sensitive	When selected, the case must match. Searching for PRINT will not find pRint. With this option turned off, Print will find print, PRINT, PRinT, etc.

Whole words only	When selected, only whole words are considered. A whole word is a word that is surrounded by spaces, or that is at the start of a line. Looking for PRINT will find : "Print test" and "print" and "print print". But not "printer"
Regular expressions	You can use a regular expression to find a match. ^ A circumflex at the start of the string matches the start of a line. $ A dollar sign at the end of the expression matches the end of a line. . A period matches any character. * An asterisk after a string matches any number of occurrences of that string followed by any characters, including zero characters. For example, bo* matches bot, bo and boo but not b. + A plus sign after a string matches any number of occurrences of that string followed by any characters except zero characters. For example, bo+ matches boo, and booo, but not bo or be. [ ] Characters in brackets match any one character that appears in the brackets, but no others. For example [bot] matches b, o, or t. [^] A circumflex at the start of the string in brackets means NOT. Hence, [^bot] matches any characters except b, o, or t. [-] A hyphen within the brackets signifies a range of characters. For example, [b-o] matches any character from b through o. \ A backslash before a wildcard character tells the Code editor to treat that character literally, not as a wildcard. For example, \^ matches ^ and does not look for the start of a line.
Search all project files	This option will search through all project files. Files considered are $INCLUDE files. Nested $include files are not considered.
Search all open files	This option will search though all open files. This are loaded files visible in the TABS
Search in directories	You can specify a custom folder to search for the text.
Search in current file	This option will restrict the search to the current file.

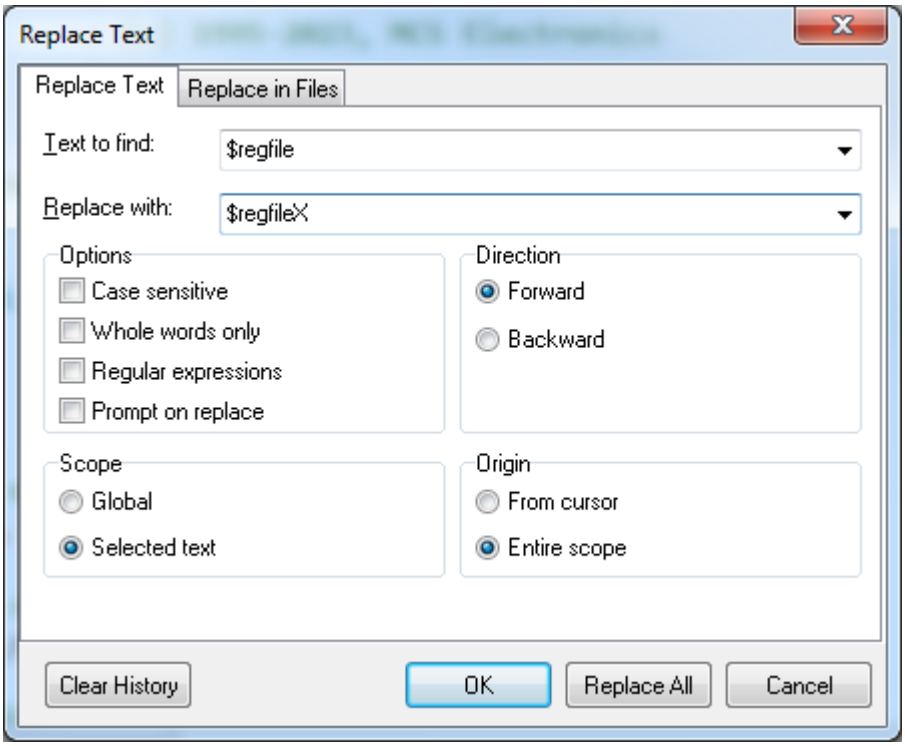
Edit Find shortcut :  , CTRL+F

3.18 Edit Find Next

With this option, you can search again for the last specified search item.
Edit Find Next shortcut :  , F3

3.19 Edit Replace

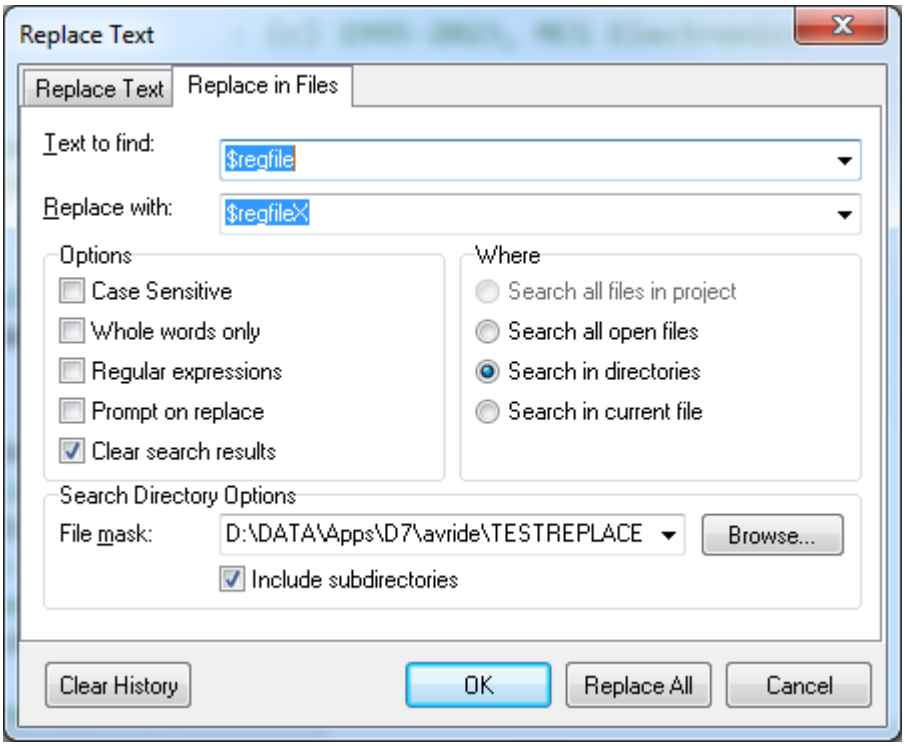
With this option, you can replace selected text in your program.



All options except 'Replace With' are described under Edit Find 81.

Replace With : this is the new text that will replace the old text.

In version 2087 you can also replace text in files.



The search and replace works similar to Search And Find in Files.
We recommend to make a backup of your project before you use 'Replace All'.

Edit Replace shortcut :  , CTRL+R

3.20 Edit Goto

With this option, you can immediately go to a specified line number.

Edit go to line shortcut :  ,CTRL+G

3.21 Edit Toggle Bookmark

With this option, you can set/reset a bookmark, so you can jump in your code with the Edit Go to Bookmark option. Shortcut : CTRL+K + x where x can be 1-8

Bookmarks are stored in a file named <project>.BM

3.22 Edit Goto Bookmark

With this option, you can jump to a bookmark.

There can be up to 8 bookmarks. Shortcut : CTRL+Q+ x where x can be 1-8

Bookmarks are stored in a file named <project>.BM

3.23 Edit Indent Block

With this option, you can indent a selected block of text.

Edit Indent Block shortcut :  , CTRL+SHIFT+I

3.24 Edit Unindent Block

With this option, you can unindent a block.

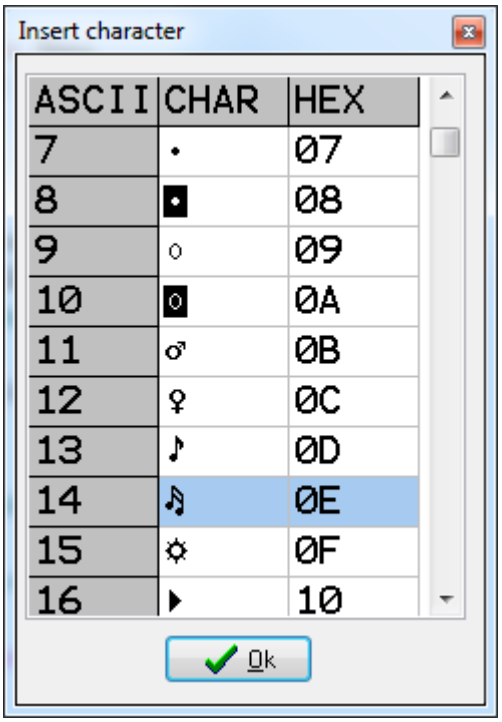
Edit Unindent Block shortcut :  , CTRL+SHIFT+U

3.25 Edit Remark Block

With this option, you can Remark or Unremark a selected block of text.
While you can use **'(** and **')** to remark a block of code, you might prefer the old BASIC way using just one **'** .
When a remark is found, it will be removed. When there is no remark, it will insert a remark.

3.26 Edit Insert ASCII

This option will show a pop up window from which you can select an ASCII character.



In BASCOM you can embed ASCII characters by using brackets embedded with the ASCII code like : {065}
For example : Dim S As String : S="AB{067}"
This is the same as S="AAA"

The pop up lists shows all ASCII values and when you click the OK-button, the brackets are added.

3.27 Edit Fold All Subs and Functions

When Code folding is enabled in Options, Environment, IDE, Editor 150, this options will fold/collapse all sub procedures and functions.
Other structures that can be folded with F11 remain unaltered.
Using SHIFT+F11 or CTRL+ENTER, you can fold/unfold the current block.

Consider this example :

```
Sub Random_color()
    R = Rnd(255)
    G = Rnd(255)
    B = Rnd(255)
    Color = Color24to16(r , G , B)
End Sub

For PORTB=1 to 10
    Waitms 1000
Next
```

Both the Sub and For/Next can be folded but the *Fold All Subs and Functions option*, will only fold the sub :

```
Sub Random_color()...

For PORTB=1 to 10
    Waitms 1000
Next
```

See Also

Edit Unfold All Code 88

3.28 Edit Unfold All Code

This option will unfold all folded code so all code becomes visible.

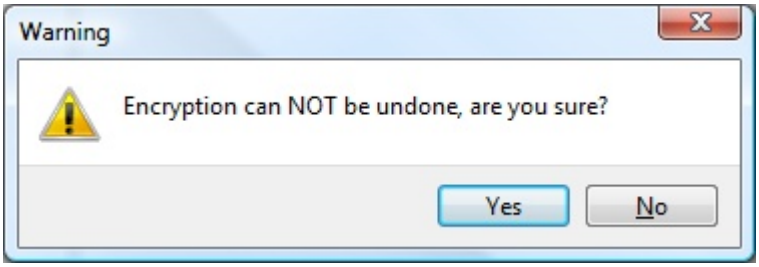
See Also

Edit Fold All Subs and Functions 87

3.29 Edit Encrypt Selected Code

This add on option allows you to encrypt portions of your code.

Because the encryption can not be undone, you will get this warning:



If you chose YES, the selected code will be encrypted and will result in lines like :

```
$CRYPT 6288E522B4A1429A6F16D639BFB7405B
$CRYPT 7ABCF89E7F817EB166E03AFF2EB64C4B
$CRYPT 645C88E996A87BF94D34726AA1B1BCCC
```

$CRYPT 9405555D91FA3B51DEEC4C2186F09ED1
$CRYPT 6D4790DA2ADFF09DE0DA97C594C1B074

Only the compiler can decrypt and process these lines. There is no way you can change the $CRYPT lines back into source code !
So make a backup of your code before you use this option. Typically, it will only be used on finished projects.
If the encrypted code contains errors, you will get error messages pointing to the $CRYPT 624 lines.

 This option is not available/enabled by default. You need to buy a license that will unlock this option. Our sales requires your BASCOM serial number too.

3.30 Edit Proper Indent

This option will properly indent your code.

Indention is used to make code better readable.
Every structure will be indented. And nested will increase indenting.

This code :

```
  For C = 0 To 100
   B = A(c)
   Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
   Waitms 4
 Next
```

Will be transformed into :

```
  For C = 0 To 100
    B = A(c)
    Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
    Waitms 4
  Next
```

And this is a sample with nesting :

```
Do
  Input "Data to write ? (0-255)" , D

  Print "Reading content of EEPROM (via ERAM Byte)"
  For C = 0 To 100
    B = A(c)
    Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
    Waitms 4
  Next
Loop
```

When indenting does not work you need to check your code for mistakes. For example for endif instead of End If.

Proper indenting is also required for proper drawing of indention 90.

3.31 Edit Show Excluded Code

This option turns on/off marking of excluded code.
Excluded code is code that is not compiled as part of the project because conditional compilation parameters exclude the code.
Excluded code is shown in Italic and gray but you can change the default colors.

For example when using an XMEGA processor, the _XMEGA constant will be set to 1.
When the option is turned off, it will show normal like :

```
#if _xmega
   print "XMEGA"
#else
   print "NORMAL"
#endif
```

When then option is turned on, the editor will show it like :

```
#if _xmega
   print "XMEGA"
#else
   print "NORMAL"
#endif
```

When you have a lot of conditional code it is hard to see which code is executed.
When you turn the option on, it is much easier to see.
Check out this example:

```
#if Lcd_enable_backlight = True Or Lcd_enable_backlight_pwm = True

  ' Sets the backlight value (0: Off, Standby On; 1-254: PWM, 255: On,
  Sub Lcd_backlight(ByVal Value As Byte)
    Local Lcd_off As Byte
    Lcd_off = 2
    #if Lcd_enable_backlight_pwm = True
      #if Lcd_invert_backlight = False
        Lcd_set_pwm Value
        If Value = 0 Then
          If Lcd_sleepmode = False Then Lcd_off = True
          Else
          If Lcd_sleepmode = True Then Lcd_off = False
        End If
      #else
        Value = &HFF -Value
        Lcd_set_pwm Value
        If Value = &HFF Then
          If Lcd_sleepmode = False Then Lcd_off = True
          Else
          If Lcd_sleepmode = True Then Lcd_off = False
        End If
      #endif
    #else
      #if Lcd_invert_backlight = False
        If Value = 255 Then
          'Lcd_ctrl_port.lcd_pin_backlight = True
          IN R16, Lcd_backlight_port
          ORI R16, Lcd_backlight_high
          !OUT Lcd_backlight_port, R16
          If Lcd_sleepmode = True Then Lcd_off = False
          Else
          'Lcd_ctrl_port.lcd_pin_backlight = False
          IN R16, Lcd_backlight_port
          ANDI R16, Lcd_backlight_low
          !OUT Lcd_backlight_port, R16
          If Lcd_sleepmode = False Then Lcd_off = True
        End If
```

In order for this option to work correct, your code should not contain syntax errors.

See Also

#IF, #ELSEIF . #ELSE 604 , Show Dead Code 91

3.32 Edit Show Dead Code

This option turns on/off marking of 'dead' code.
Dead code is code that does not do a thing and could be removed.
Dead code is shown in Italic and gray but you can change the color and italic.

Dead code is similar to Excluded code with the difference that excluded code is not compiled while dead code is compiled.

Dead code is a new feature in 2080 and intended to show you which variables or code are not used.
You can decide if the code is really dead, and need to be removed, or not.

Since this is a new feature, you should take care before deleting 'dead code'

```
Function Dhcp_ok() As Byte
   Local Btmp As Byte
   Local Dead As Byte
   Dhcp_ok = 2
   Dhcp_ok = 0

   Goto Test
   Print
   Test:
   If Getsocket(0 , Sock_dgram , 68 , 0) = 0 Then
```

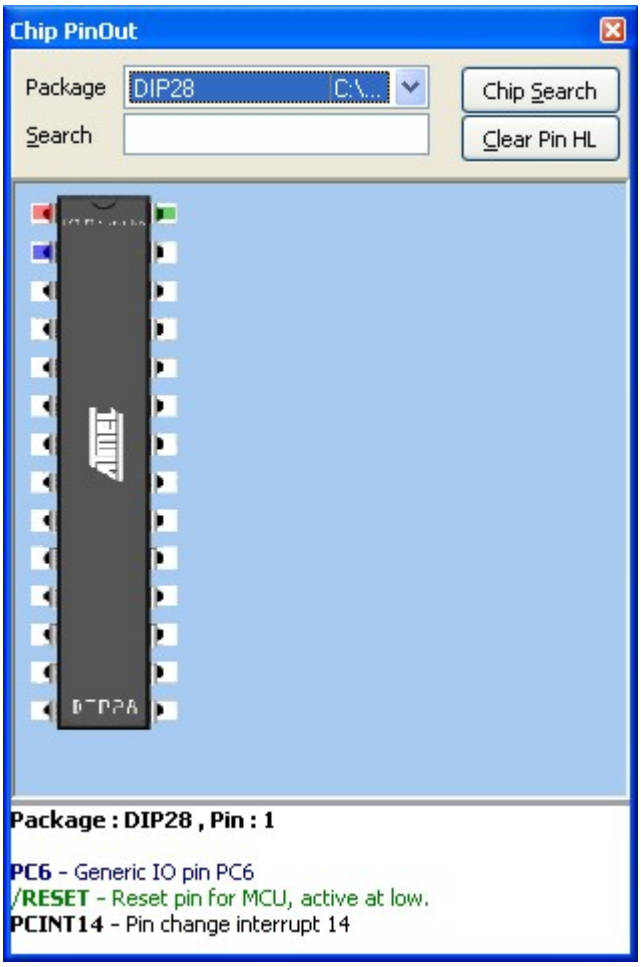
The example above demonstrates a few dead code elements:
- the local dead as byte, is not used in the code
- the function result is assigned twice without that the result is used, this does not make sense
- the GOTO skips over some code which is never used (print)

See Also

Edit Show Excluded Code 90

3.33 View PinOut

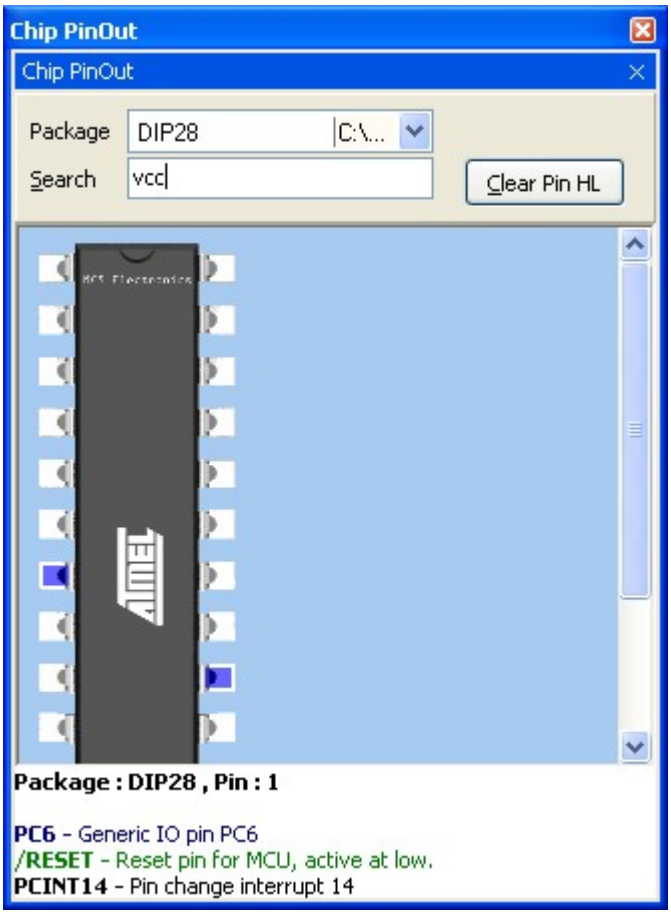
The Pin Out viewer is a dock able window that shows the case of the active chip.
The active chip is determined by the value of $REGFILE 694.



When you move the mouse cursor over a pin, you will see that the pin will be colored red. At the bottom of the window, a pin description is show. In the sample above you will see that each line has a different color. This means that the pin has multiple alternative functions.
The first blue colored function is as generic IO pin.
The second green colored function is RESET pin.
The third black colored function is PIN change interrupt.

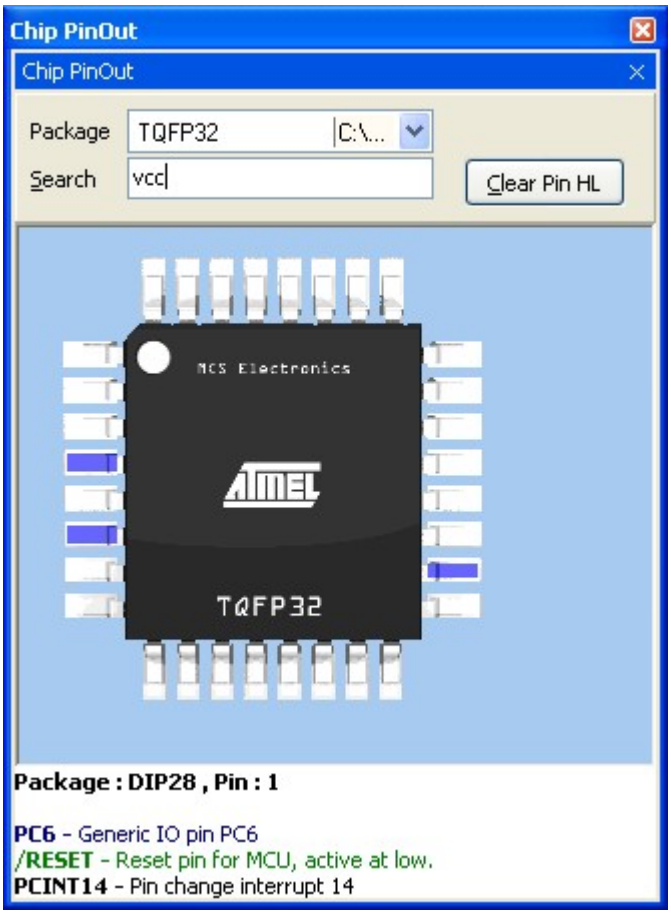
A pin can have one or more functions. Some functions can be used together.
When you move the mouse cursor away, the pin will be colored blue to indicate that you viewed this pin. For example, when you need to look at it again.

You can also search for a pin description. Enter some text and return.
Here is an example when you search the VCC pin :



When pins are found that have the search phrase in the description, the pin will be colored blue.
By clicking 'Clear Pin HL' you can clear all colored pins.

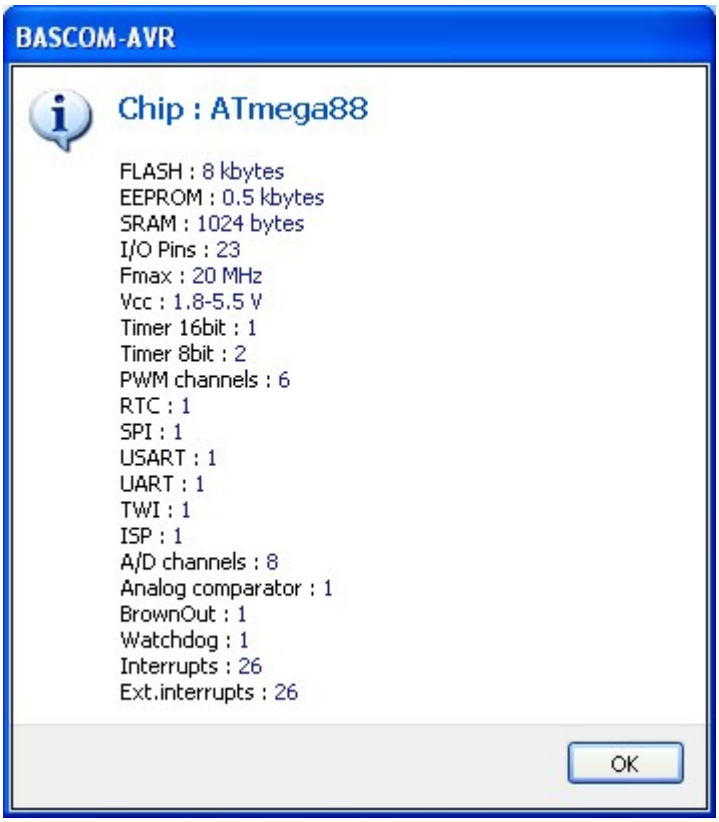
Some chips might have multiple cases. You can select the case from the package list.



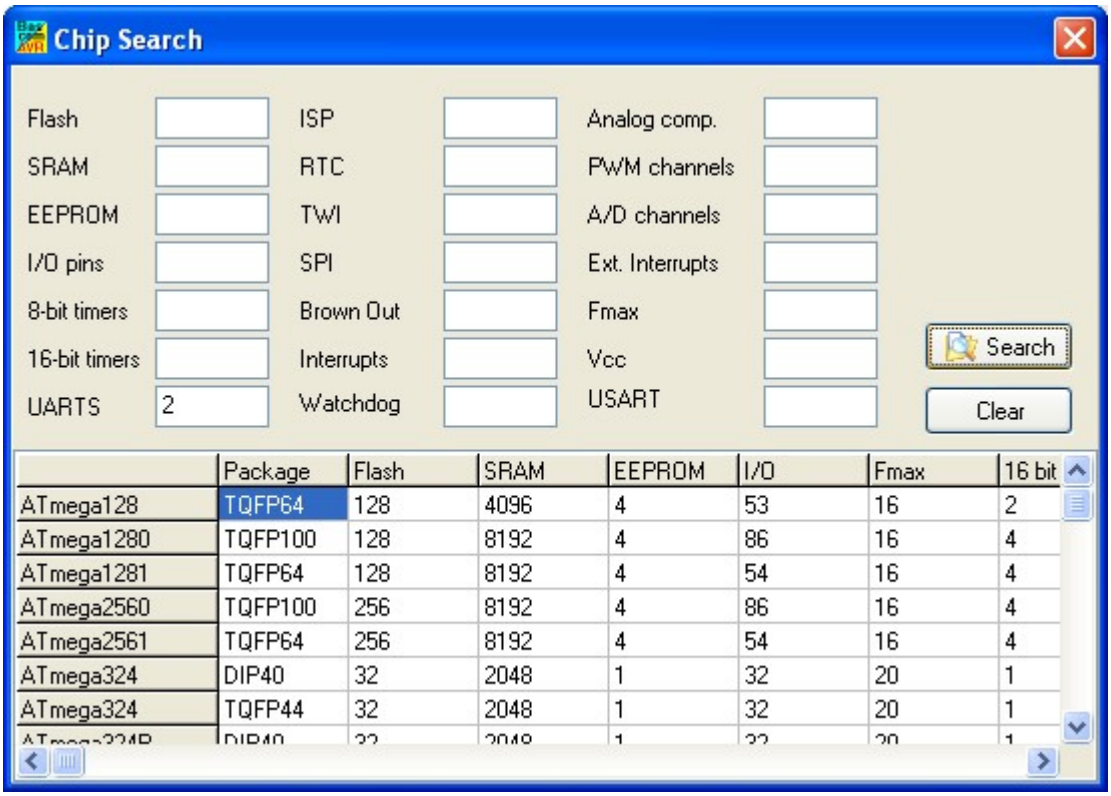
When you change from package, all pin colors will be cleared.
When you double click a pin, the pin will be colored green. Another double click will color it red/blue.
When a pin is green, it will not be colored red/blue. The green color serves as a kind of bookmark.
The only exception is the search function. It will make bookmarked green pins, blue too.

Use the right mouse to access a popup menu. This menu allows you to zoom the image to a bigger or smaller size.

Double click the chip to show the chip data.



When you want to search for a chip, click the 'Chip Search' button.
It will show the following window:



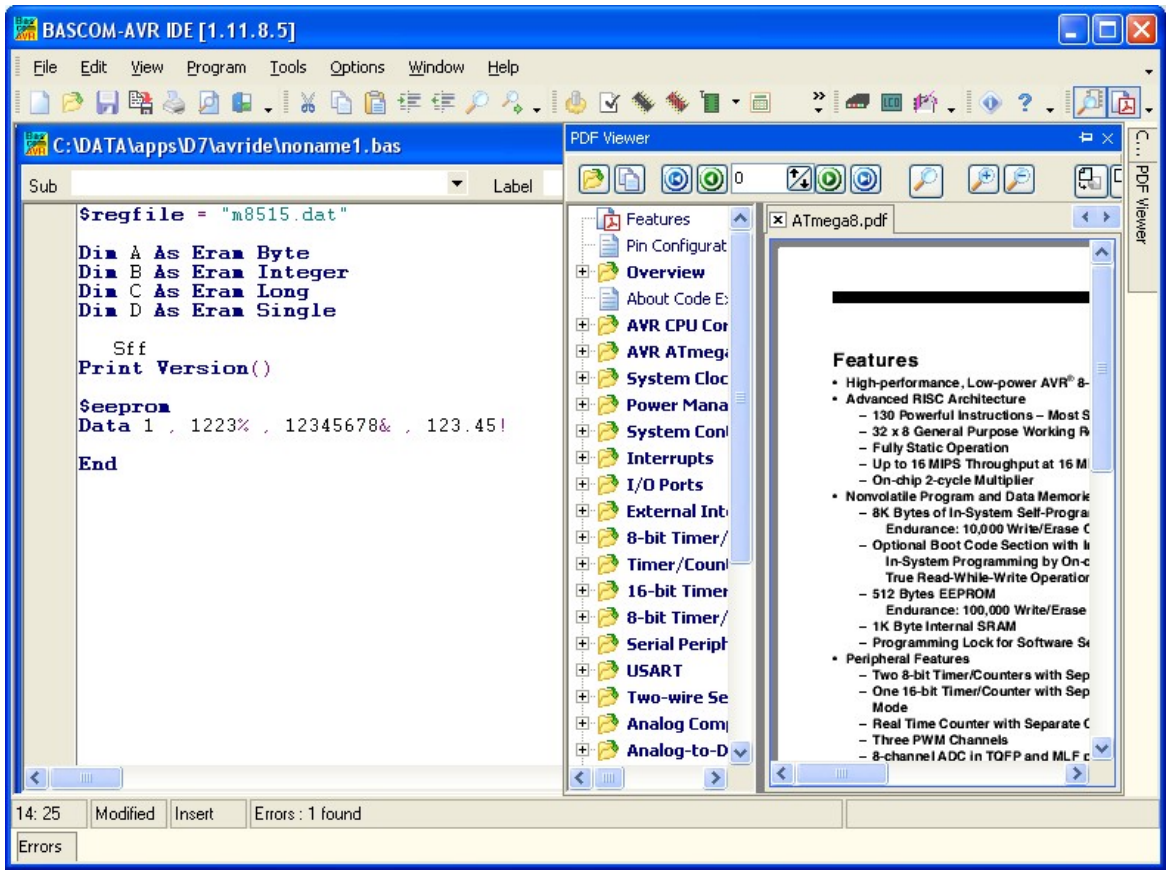
You can provide criteria such as 2 UARTS. All criteria are OR-ed together. This means that when one of the criteria is met, the chip will be included in the list.

Only chips supported by BASCOM will be listed. When a chip has SRAM, and is not supported yet, it will be in the near future since the goal is to support all chips.

When you find an error in the pin description, please send an email to support so it can be corrected.

3.34 View PDF viewer

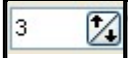
The PDF viewer is dock able panel which is located by default on the right side of the IDE.



The viewer itself contains a tree with the topics and the actual PDF viewer.
The tree topics can be searched by right clicking on the tree. Choose 'Search' and enter a search text.
When a topic has sub topics, the topic is **bold**.

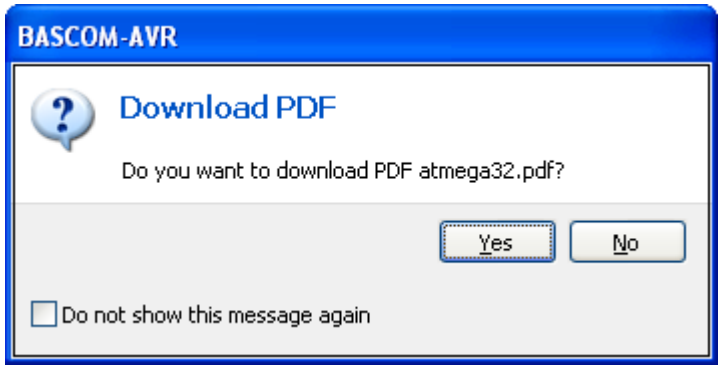
When you have enabled 'Auto open Processor PDF' in Options, Environment, PDF, the data sheet will be automatically loaded when you change the $REGFILE value.
It can be shown in a new sheet or it can replace the current PDF.

	Open a PDF.
	Copy selected text to the clipboard. You can not copy from protected PDF documents.
	First page.

	Previous page.
	Current page indicator. You can enter a page number to jump to a different page.
	Next page.
	Last page.
	Find text in PDF.
	Zoom in.
	Zoom out.
	Rotate page to the left and right.
	Print page(s).

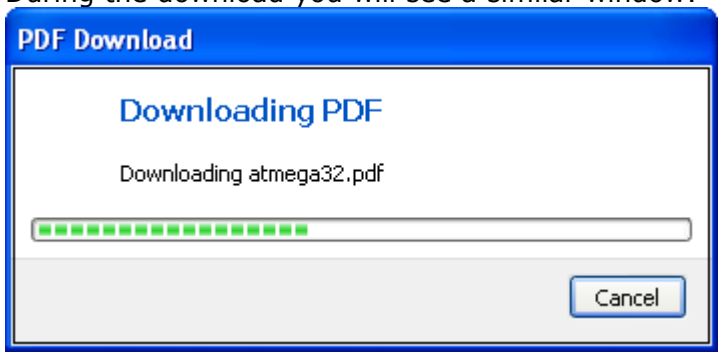
When you right click in the PDF, a pop up menu with the most common options will appear.
In Options, Environment, PDF 150 you can specify how data sheets must be downloaded.

Data sheets can be downloaded automatic. When the $REGFILE is changed and the PDF is not present, you will be asked if the PDF must be downloaded.
If you choose to download, it will be downloaded from the Atmel website.



When you click 'Do not show this message again' , you will not be asked anymore if you want to download the Mega32.PDF. You will be asked to download other PDF documents when they do not exist.

During the download you will see a similar window:



You can also download all newer PDF's from the Atmel website with the option : Tools, PDF Update 138

When PDF's are downloaded with the UpdateWiz, they are downloaded from the MCS Electronics website.

3.35 View Error Panel

This option will show the Error panel.

When there are no errors, the list will be empty. You will also be able to close the window.
When there are errors :

You will not be able to close the window until the error is solved and the program is checked/compiled.
The panel is dockable and by default docked to the bottom of the IDE.

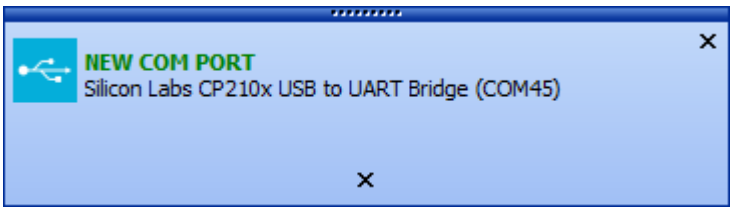
When you right click the mouse inside the error panel, a menu will popup with one option : Copy to Clipboard. All data from the error window will be copied to the windows clipboard if you select this option.

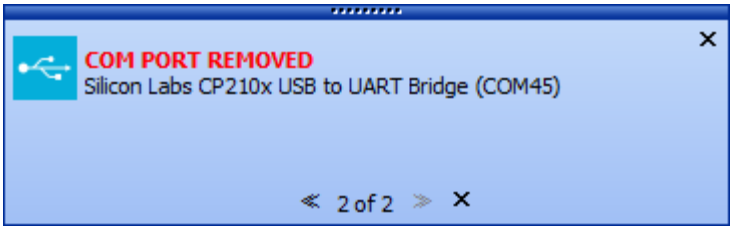
3.36 View Show Allert Window

Action

Shows the Alert Windows when available.

An alert window can contain various info. In version 2086 it is limited to show when COM ports are added or removed.
When you use a CDC device (virtual COM port) and you plug the device, depending on settings and hub/usb port a new COM number will be assigned.
In the lower right of the screen an alert window/message will appear. It will not have focus and it will fade away after some time.
Below are 2 examples. One when a new COM port is found. And one when that same COM port is removed.





The X on the alert window can be used to close the window.
The X on the bottom can be used to Delete the window.

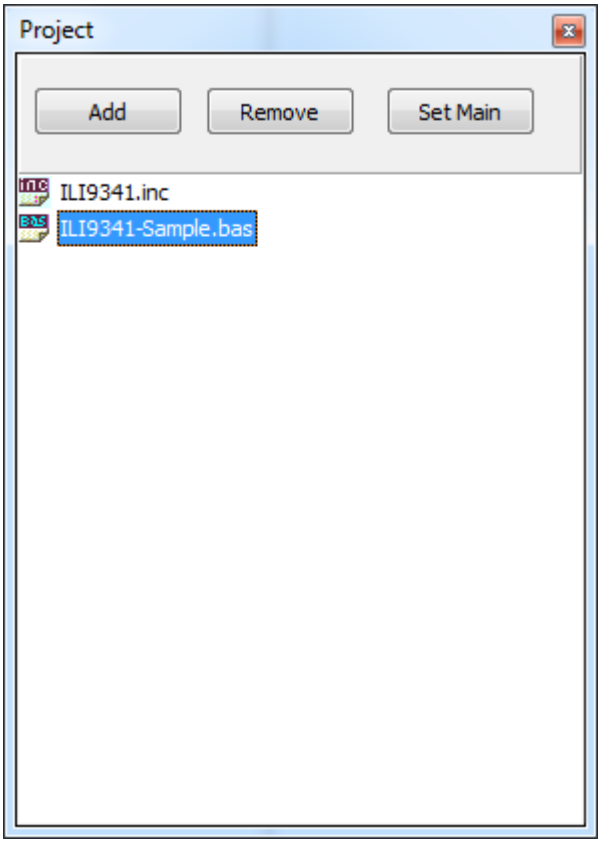
When you close a window, it will exist until BasCom is closed.
For this reason the 'View Show Alert Windows option exists : you can show the old alerts.

A new alert is always added after the last message. So in this example the first message was the NEW com port.
And when the cable was pulled, the second alert was added.
When multiple alert windows exist, you can use << and >> to scroll through them.

3.37 View Project Files

Action

Shows the Project Explore Window.



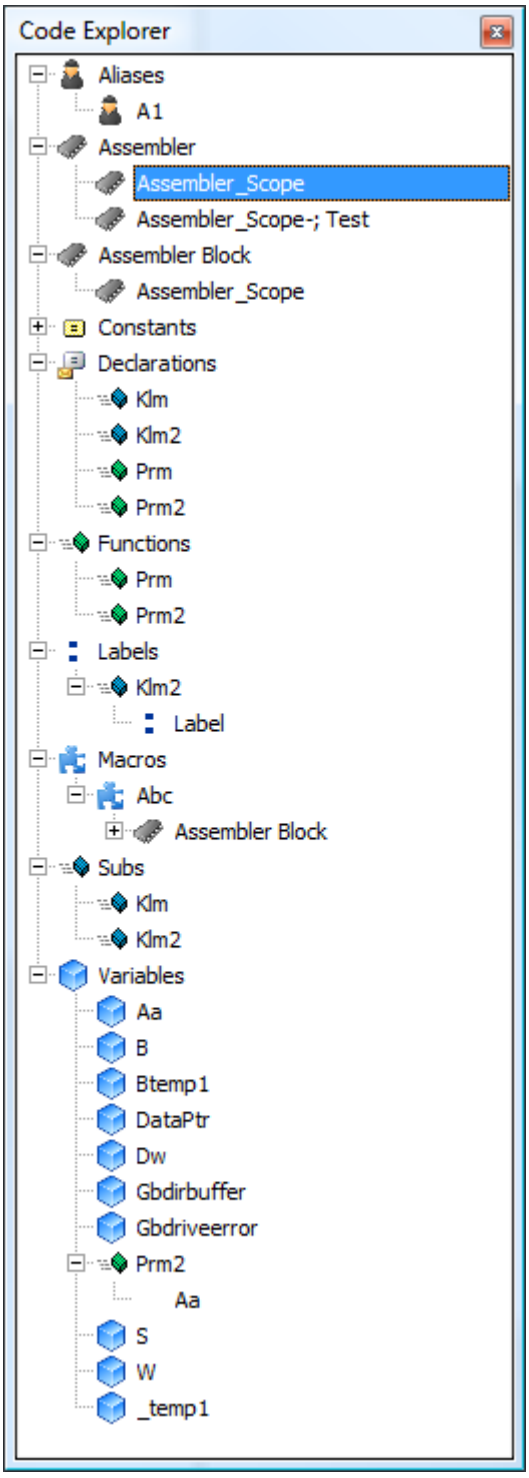
The project explorer window is intended to be used in project mode. Project mode is a mode where all files belong to one project. Here you have a main file and optional include files.

The project explorer is a dock able window. It lists all files assigned to the project. When you double click a file, it will be opened in the editor.
- Use the ADD button to add files to the project
- Use the REMOVE button to remove files from the project. Files you remove are only removed from the project, they are not removed from disk
- Use SET MAIN to set the main project file. The main project file is the file that is compiled.

3.38 View Code Explorer

Action

Shows the Code Explorer Window



The code explorer shows code elements in a tree. By double clicking an element the cursor will be set to the matching code in the editor.
You can also drag an element into the editor window.
By clicking the right mouse a pop up menu will allow you to filter out constants and variables (registers) from the definition file.

The following code elements will be shown in the explorer:
- Aliases. These are the user ALIASes.
- Assembler. This is for single line asm using **!**
- Assembler Block. This is for assembler blocks using $asm .. $end asm. If you add comment after $asm, it will be shown in the tree as well. Example : $asm ; Test

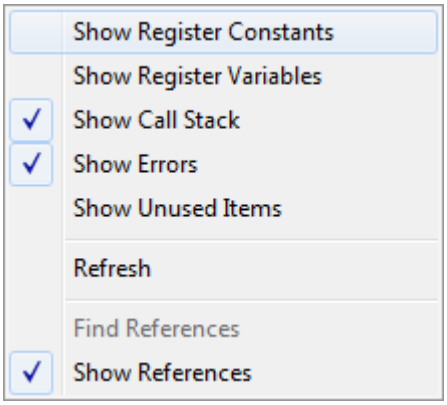
- Constants. Both user defined constants (CONST 1170) and constants from the definition file are shown.
- Declarations. Subs and Functions are both shown. Each with their own color.
- Functions. These are the user function implementations.
- Labels. When labels are used in subs and functions, the sub/functions name is listed first.
- Macros. These are the user macro's created with MACRO 1368.
- Subs. These are the user sub implementations.
- Variables. These are the variables from the user code and definition file. Each shown with their own color. Locals are shown under a branch of the sub/function.
- CallStack. This is optional. Since it takes time to trace the call stack it is turned off by default. Use right mouse click and the pop up menu to activate it.
The call stack shows a tree of the calls you make to user subs and functions. And each sub/function also shows the user functions it calls.
When multiple calls are made, three dots are added for each additional call.
- Types. Declared types with their members are shown. See TYPE.
- Information. Processor, free ERAM and SRAM. Estimated $hwstack, $swstack and $framesize.

The calculated stack settings are based on the program call tree and local variables. This is just a tool to give you an idea about stack usage. Not taken into account is the stack required by the assembler routines. This means that you need to add a certain amount to the calculated values. When your code uses interrupts you need to increase the calculated $HWSTACK by 32. Otherwise increase it by 16.
The $FRAMESIZE should have a minimum value of 24. Add a value of 16 to $SWSTACK.
Applications using AVR-DOS should use a minimum of 128 for all stacks.
A future version will also take the assembler code into account.

When the Code Explorer has the focus, pressing CTRL+F will search in the code explorer and not in the editor.

The code explorer works in a separate thread. It will be updated a few seconds after you have quit typing.
By making the Code Explorer window invisible, the explorer is deactivated.

The popup menu has the following options:



Show Register Constants

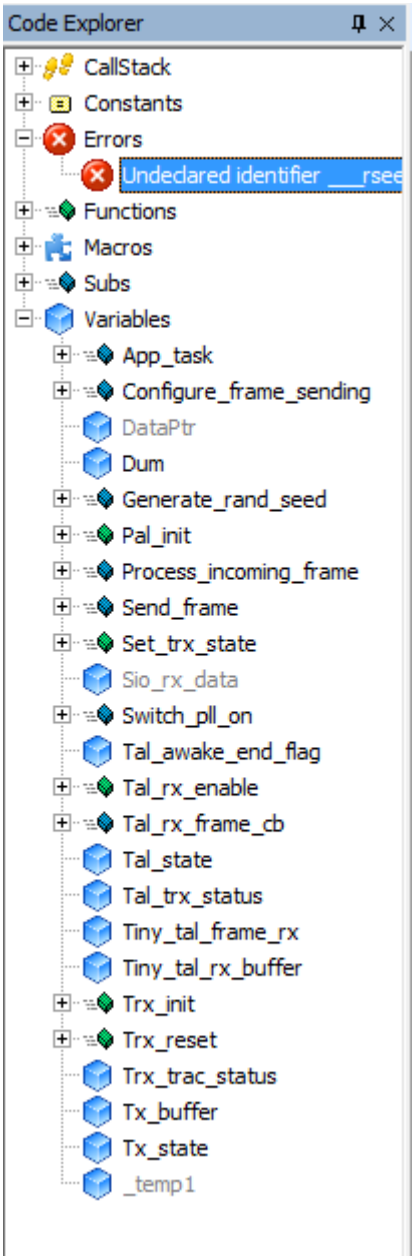
This option can toggle between showing and hiding the register constants. When register constants are shown the tree can become big.
User constants and register constants are shown in a different color.

Show Register Variables
This option can toggle between showing and hiding the register variables. When register variables are shown the tree can become big.
User variables and register variables are shown in a different color.

Show Call Stack
This option can show the Call Stack. This reveals the nesting of your code.

Show Errors
This option deserves a **warning**. The option is turned off by default. It can be useful to find errors but it can also point to errors which are not considered an error for the compiler. The compiler has a separate parser. The parser from the IDE is a different new parser. While in 2080 all DAT files are updated, you still can get errors which are no real errors. You might want to report them to support. Please send a small as possible program that will show the error.

Show Unused Items
When this option is turned on, all unused items will be shown in grey. For example :



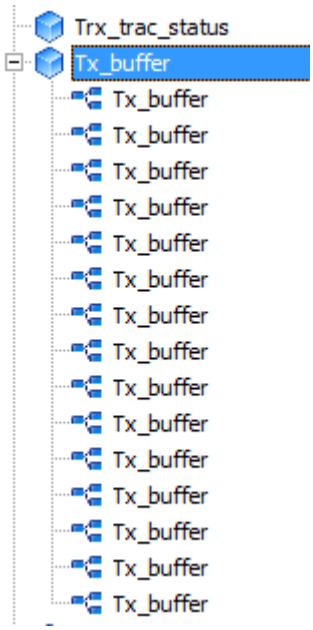
In this sample, _temp1 , so_rx_data and DataPtr are unused or unreferenced. _temp1 is an internal variable and so is DataPtr. They do not occupy any space. But so_rx_data is a user variable which is not referenced. You could remove or remark it.

Refresh

This option will parse the project and update the code explorer tree.

Find References

This option can find all references for an item. For example when you go to Variables, and select a variable the option becomes enabled in the menu. After choosing this option, the references will be added to the tree.



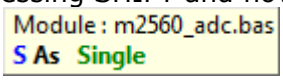
Now by clicking the item you will go to the point in your code where the item is referenced/used.

Show References

This options shows a panel on the bottom of the code explorer tree. When you activate the tooltip keeping SHIFT pressed and hovering an item in the editor, the references panel will be updated with all references of that item. A single click on an item in this list will set the cursor in the IDE to referred item.

Consider this simple piece of code :
```
Dim S As Single
Input "s " , S
Print S
```

When pressing SHIFT and hovering the mouse over the variable S , the tooltip will be shown :



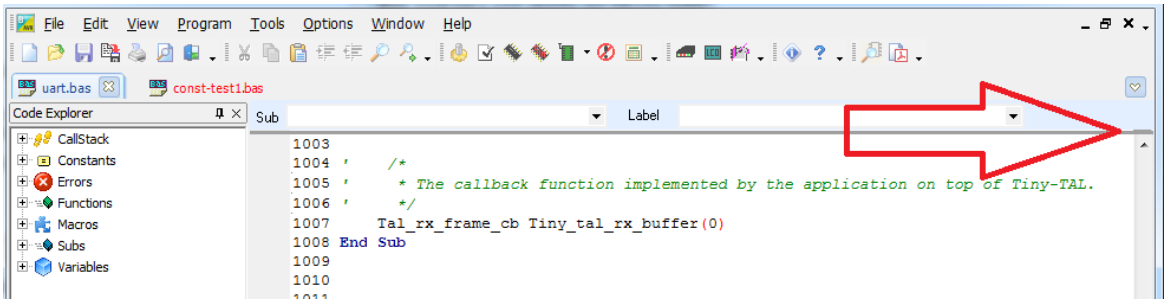
The references list will be updated as well. The item in bold points to the definition, in this case the DIM S.
The following two items in the list point to the INPUT "s ", S and the Print S.
Items shown in **red** are variables that are assigned.
The panel can be shown or hidden using the right click menu from the explorer.

RENAMING

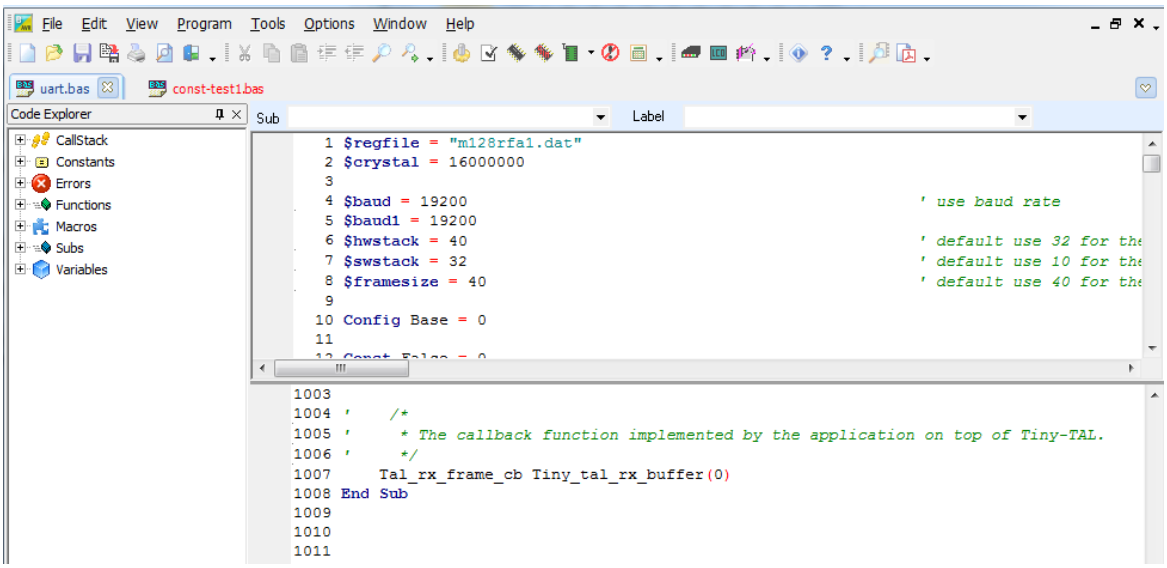
When you right an item in the References List you get a pop up menu : RENAME
When you click the RENAME option you will be asked for a new name.
Enter a new name for the item (variable,constant, etc). All occurrences in your project will be replaced. Not only the ones in loaded files but also in included files on disk.
When the new name you provide is already used in the project you will get an error message and the items will not be replaced.

3.39 View Vertical Splitter

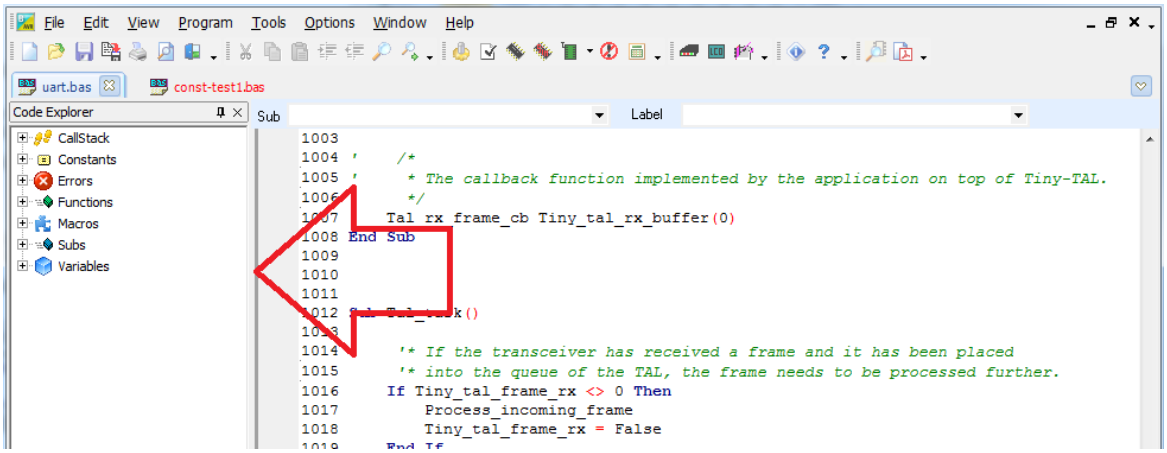
You can split the editor window into two parts. By default you use the horizontal splitter marked with the arrow.



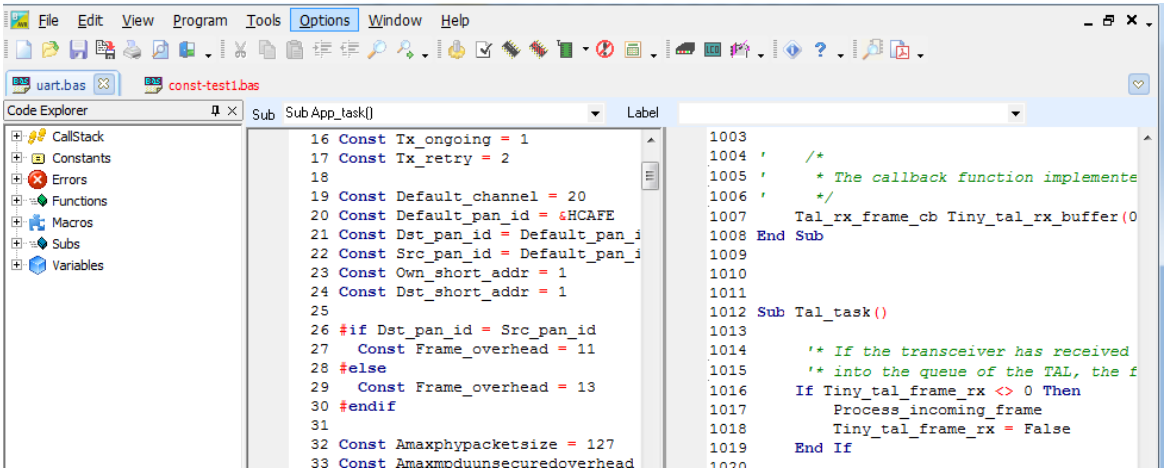
This will create a split screen :



With the option Vertical Splitter, you switch between horizontal en vertical splitter. The splitter is located near the code explorer window.



This will result in a vertical split window.



When you chose the vertical splitter option again the window will be split horizontal again.

Notice that in order to show two different code windows you need to open the two windows and use Tile Vertically 229.

3.40 Program Compile

With this option, you compile your current program.

Your program will be saved automatically before being compiled.

The following files will be created depending on the Option Compiler Settings. 143

File	Description
xxx.BIN	Binary file which can be programmed into the microprocessor.
xxx.DBG	Debug file that is needed by the simulator.
xxx.OBJ	Object file for simulating using AVR Studio. Also needed by the internal simulator.
xxx.HEX	Intel hexadecimal file, which is needed by some programmers.
xxx.ERR	Error file. Only created when errors are found.
xxx.RPT	Report file.
xxx.EEP	EEPROM image file

If a serious error occurs, you will receive an error message in a dialog box and the compilation will end.

All other errors will be displayed at the bottom of the edit window, just above the status bar.

When you click on the line with the error info, you will jump to the line that contains

the error. The margin will also display the sign.

At the next compilation, the error window will disappear or reappear if there are still errors.
See also 'Syntax Check' 109 for further explanation of the Error window.

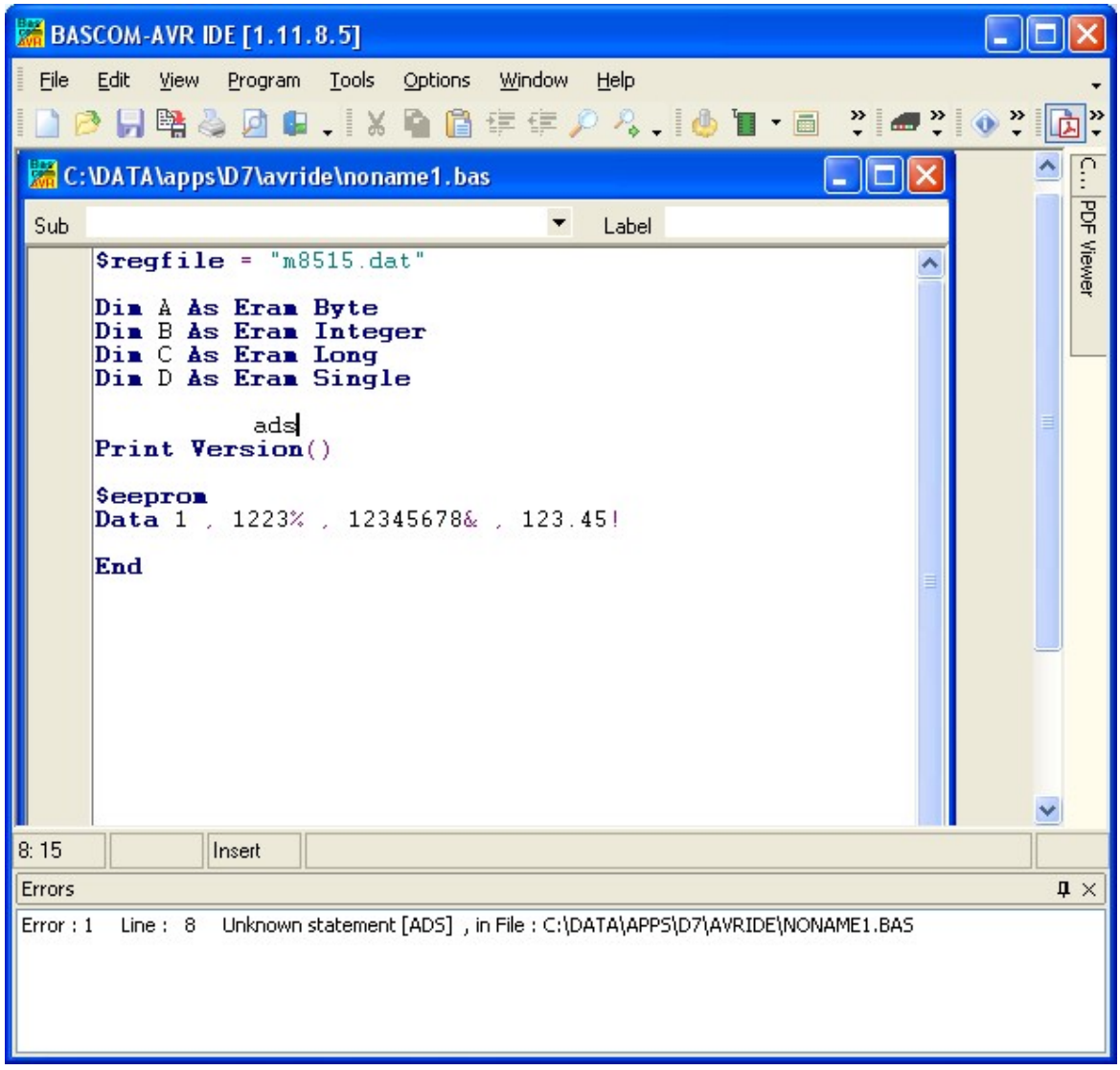
Program compile shortcut: , F7

3.41 Program Syntax Check

With this option, your program is checked for syntax errors. No file will be created except for an error file, if an error is found.

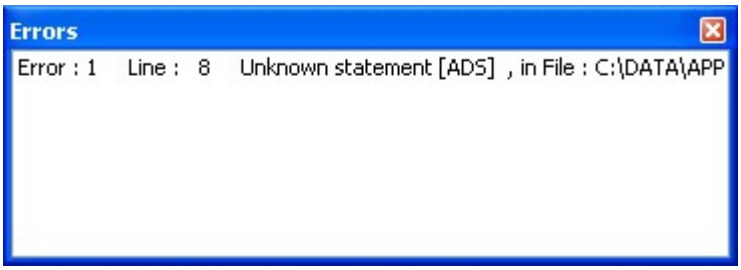
Program syntax check shortcut , CTRL + F7

When there is an error, an error window will be made visible at the bottom of the screen.



You can double click the error line to go to the place where the errors is found. Some errors point to a line zero that does not exist. These errors are caused by references

to the assembler library and are the result of other errors.
The error window is a dockable window that is docked by default to the bottom of the screen. You can drag it outside this position or double click the caption(Errors) to make it undock :



Here the panel is undocked. Like most windows you can close it. But the error must be resolved (corrected and syntax checked/recompiled) for this window can be closed !
By double clicking the caption (top space where the name of the window is show) you can dock it back to it's original position.

When you have closed the window and want to view it again, you can choose the View, Error Panel option from the main menu.

3.42 Program Show Result

Use this option to view information concerning the result of the compilation.

See the Options Compiler Output 145 for specifying which files will be created.

The files that can be viewed are "report" and "error".

File show result shortcut : ,CTRL+W

Information provided in the report:

Info	Description
Report	Name of the program
Date and time	The compilation date and time.
Compiler	The version of the compiler.
Processor	The selected target processor.
SRAM	Size of microprocessor SRAM (internal RAM).
EEPROM	Size of microprocessor EEPROM (internal EEPROM).
ROMSIZE	Size of the microprocessor FLASH ROM.
ROMIMAGE	Size of the compiled program.
BAUD	Selected baud rate.
XTAL	Selected XTAL or frequency.
BAUD error	The error percentage of the baud rate.
XRAM	Size of external RAM if available.
Stack start	The location in memory, where the hardware stack points to. The HW-stack pointer grows downward.
S-Stacksize	The size of the software stack.
S-Stackstart	The location in memory where the software stack pointer points to. The software stack pointer grows downward.

Framesize	The size of the frame. The frame is used for storing local variables.
Framestart	The location in memory where the frame starts.
LCD address	The address that must be placed on the bus to enable the LCD display E-line.
LCD RS	The address that must be placed on the bus to enable the LCD RS-line
LCD mode	The mode the LCD display is used with. 4 bit mode or 8 bit mode.
LCD DB7-DB4	The port pins used for controlling the LCD in pin mode.
LCD E	The port pin used to control the LCD enable line.
LCD RS	The port pin used to control the LCD RS line.
Variable	The variable name and address in memory
Constant	Constants name and value Some internal constants are : _CHIP : number that identifies the selected chip _RAMSIZE : size of SRAM _ERAMSIZE : size of EEPROM _XTAL : value of crystal _BUILD : number that identifies the version of the compiler _COMPILER : number that identifies the platform of the compiler
Warnings	This is a list with variables that are dimensioned but not used. Some of them
EEPROM binary image map	This is a list of all ERAM variables with their value. It is only shown when DATA 1177 lines are used to create the EEP file. (EEPROM binary image).

When the option : Load Report in IDE, is set, the report will be shown as a text file in the IDE.

3.43 Program Simulate

With this option, you can simulate your program. So what exactly is simulating? For BASCOM it means that the generated object code is processed with a virtual AVR processor.
The simulator can simulate the AVR instructions. It can also simulate the hardware for a part. The goal of the simulator is to allow you to debug your code. The goal was not to create 100% virtual AVR hardware.
This means that some hardware is simulated but with different timing.

You can simulate your programs with AVR Studio or any other Simulator available such as ISIS or you can use the built in Simulator.

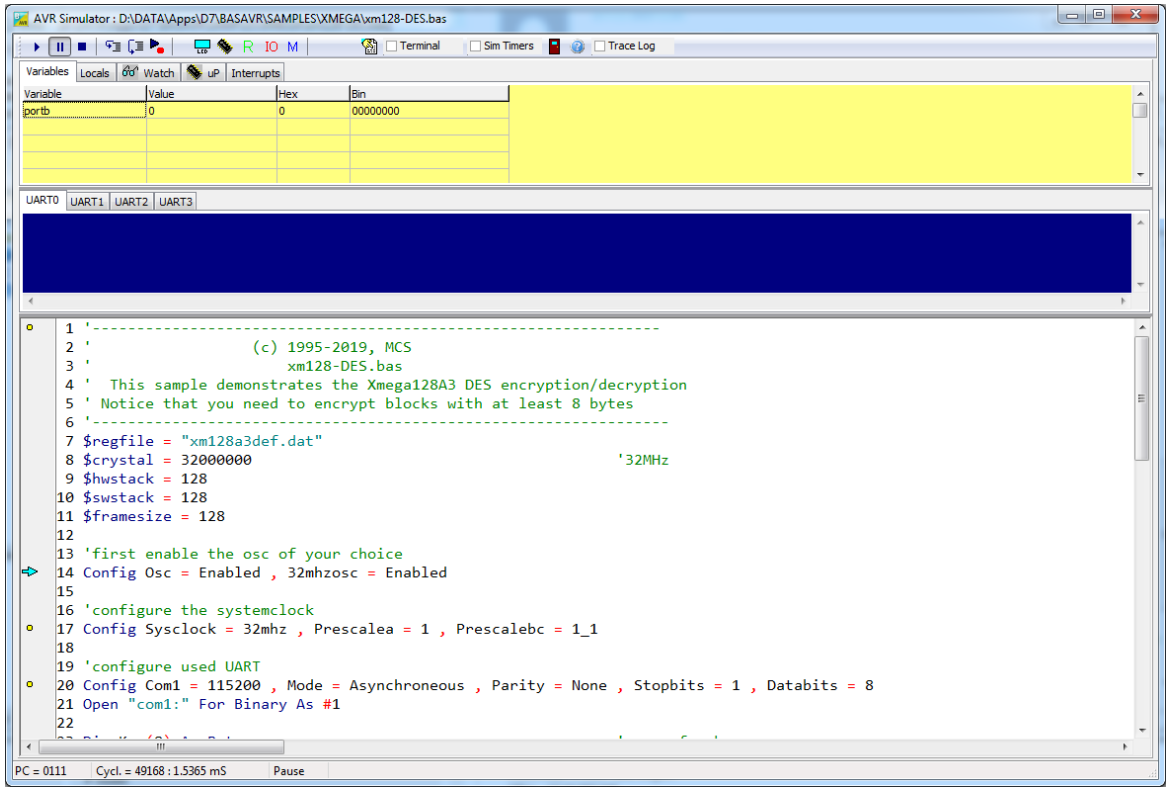
The simulator that will be used when you press F2, depends on the selection you made in the Options Simulator TAB. The default is the built in Simulator.

Program Simulate shortcut :  , **F2**

To use the built in Simulator the files DBG and OBJ must be selected from the Options Compiler Output 145 TAB.

The OBJ file is the same file that is used by the AVR Studio simulator.

The DBG file contains info about variables and many other info required to simulate a program.



The yellow dot means that the line contains executable code. The blue arrow is visible when you start simulating. It will point to the line that will be executed.

The Simulator window is divided into a few sections:

The Toolbar

The toolbar contains the buttons you can press to start an action.

 This is the RUN button, it starts a simulation. You can also press **F5**. The simulation will pause when you press the pause button. It is advised, that you step through your code at the first debug session. When you press **F8**, you step through the code line by line which is a clearer way to see what is happening.

 This is the PAUSE button. Pressing this button will pause the simulation.

 This is the STOP button. Pressing this button will stop the simulation. You can't continue from this point, because all of the variables are reset. You need to press the RUN button when you want to simulate your program again.

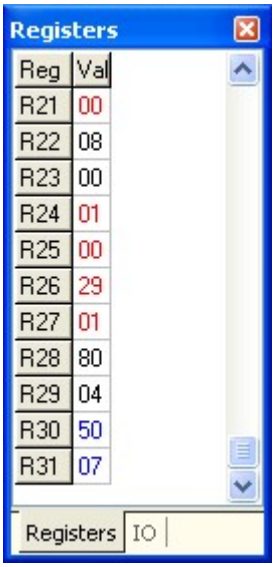
 This is the STEP button. Pressing this button (or **F8**) will simulate one code line of your BASIC program. The simulator will go to the RUN state. After the line is

executed the simulator will be in the PAUSE state. If you press **F8** again, and it takes a long time too simulate the code, press **F8** again, and the simulator will go to the pause state.

 This is the STEP OVER button or **SHIFT+F8**). It has the same effect as the STEP button, but sub programs are executed completely, and the simulator does not step into the SUB program.

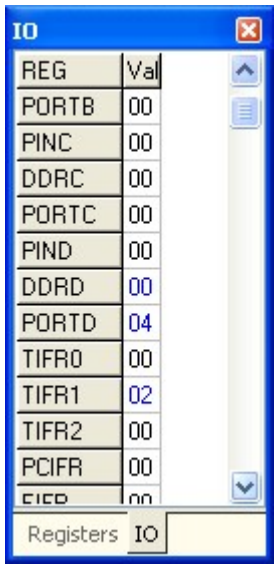
 This is the RUN TO button. The simulator will RUN until it gets to the current line. The line must contain executable code. Move the cursor to the desired line before pressing the button.

 This button will show the processor registers window.



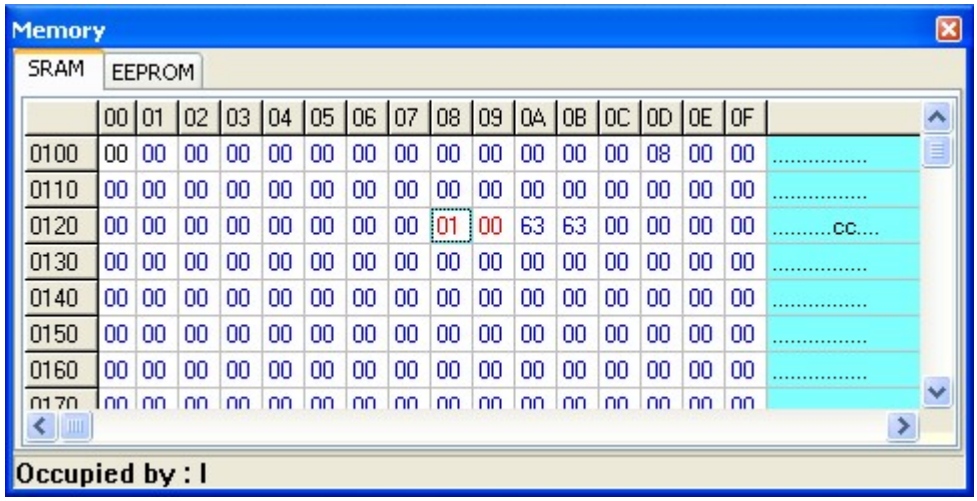
The values are shown in hexadecimal format. To change a value, click the cell in the VAL column, and type the new value. When you right click the mouse, you can choose between the Decimal, Hexadecimal and Binary formats.
The register window will show the values by default in **black**. When a register value has been changed, the color will change into **red**. Each time you step through the code, all changed registers are marked **blue**. This way, the red colored value indicate the registers that were changed since you last pressed F8(step code). A register that has not been changed at all, will remain black.

 This is the IO button and will show processor Input and Output registers.



The IO window works similar as the Register window.
A right click of the mouse will show a popup menu so you can choose the format of the values.
And the colors also work the same as for the registers : black, value has not been changed since last step(F8). Red : the value was changed the last time your pressed F8. Blue : the value was changed since the begin of simulation. When you press the STOP-button, all colors will be reset to black.

Pressing this button shows the Memory window.



The values can be changed the same way as in the Register window.
When you move from cell to cell you can view in the status bar which variable is stored at that address.
The SRAM TAB will show internal memory and XRAM memory.
The EEPROM TAB will show the memory content of the EEPROM.
The colors work exactly the same as for the register and IO windows. Since internal ram is cleared by the compiler at startup, you will see all values will be colored blue.
You can clear the colors by right clicking the mouse and choosing 'Clear Colors'.


The refresh variables button will refresh all variables during a run (**F5**). When you use the hardware simulator, the LEDS will only update their state when you have

enabled this option. Note that using this option will slow down simulation. That is why it is an option. When you use **F8** to step through your code you do not need to turn this option on as the variables are refreshed after each step.

Sim Timers When you want to simulate the processors internal timers you need to turn this option on. Simulating the timers uses a lot of processor time, so you might not want this option on in most cases. When you are debugging timer code it is helpful to simulate the timers.
The simulator supports the basic timer modes. As there are many new chips with new timer modes it is possible that the simulator does not support all modes. When you need to simulate a timer the best option may be to use the latest version of AVR Studio and load the BASCOM Object file.
Even AVR Studio may have some flaws, so the best option remains to test the code in a real chip.

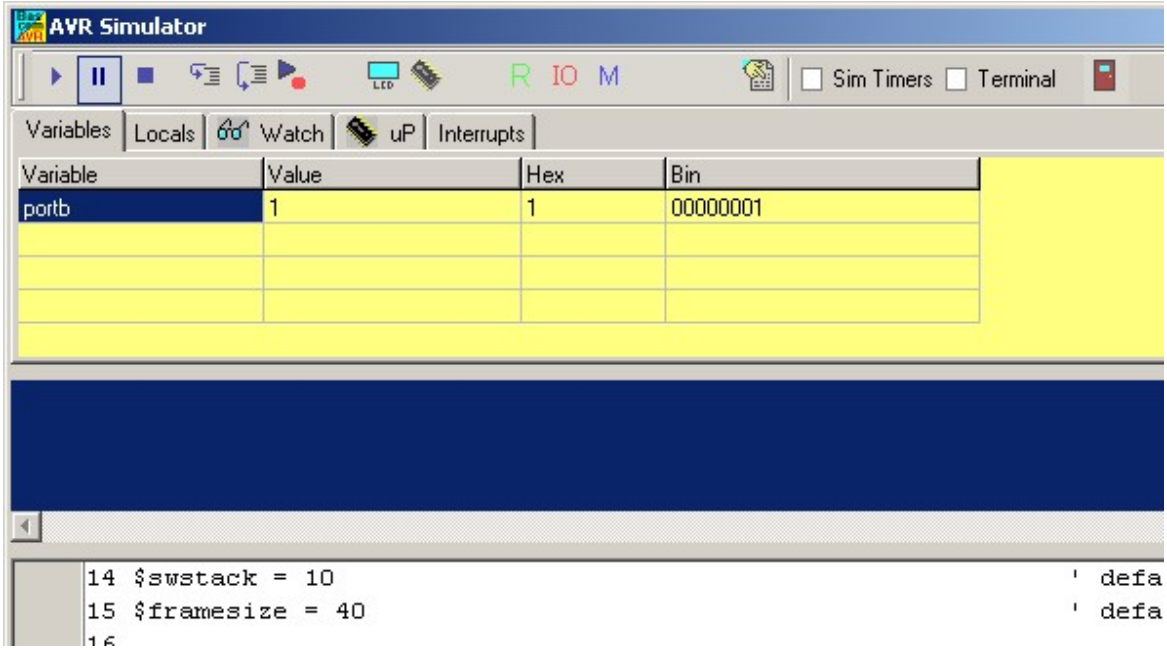
The TIMER simulation only simulates TIMER0 and 16 bit TIMER1. And only counting/time modes are supported. PWM mode is not simulated.

Terminal This option allows you to use a real terminal emulator for the serial communication simulation.
Normally the simulator send serial output to the blue window, and you can also enter data that needs to be sent to the serial port.
When you enable the terminal option, the data is sent to the actual serial port, and when serial data is received by the serial port, it will be shown.

Trace Log This option turns on/off trace information. When enabled, a file with the name of your project will be created with the .TRACELOG extension.
This file will contain the file, line number and source code that is executed. It is intended to check which parts of your code execute.

Under the toolbar section there is a TAB with a number of pages:

VARIABLES



This section allows you to see the value of program variables. You can add variables by double clicking in the Variable-column. A list will pop up from which you can select the variable.
To watch an array variable, type the name of the variable with the index.

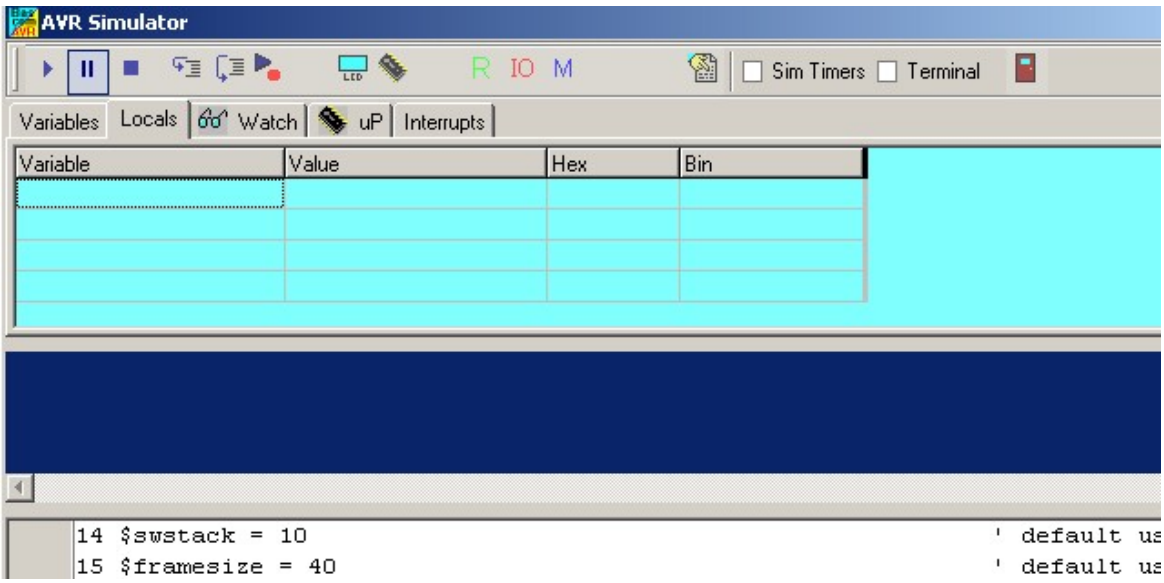
During simulation you can change the values of the variables in the Value-column, Hex-column or Bin-column. You must press ENTER to store the changes.

To delete a variable, you can press CTRL+DEL.

To enter more variables, press the DOWN-key so a new row will become visible.

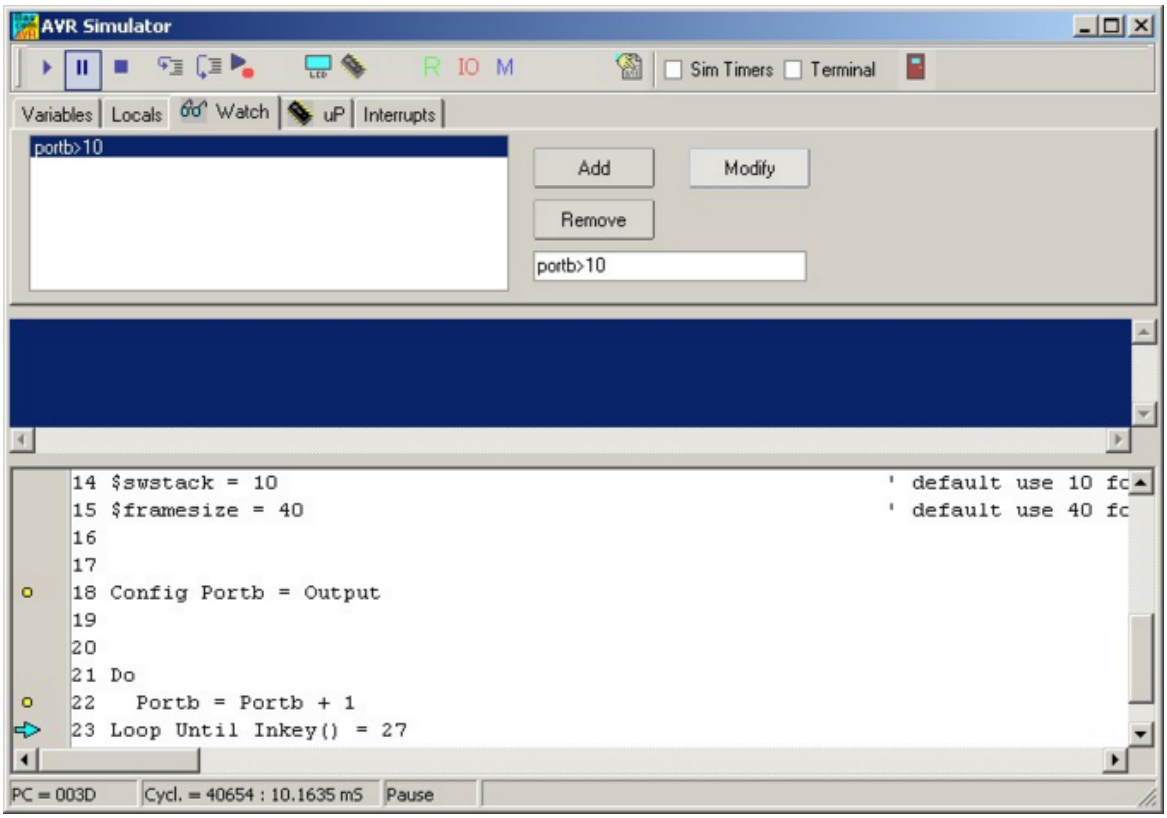
It is also possible to watch a variable by selecting it in the code window, and then pressing enter. It will be added to the variable list automatically.
Notice that it takes time to refresh the variables. So remove variables that do not need to be watched anymore for faster simulation speed.

LOCALS



The LOCALS window shows the variables found in a SUB or FUNCTION. Only local variables are shown. You can not add variables in the LOCALS section.
Changing the value of local variables works the same as in the Variables TAB.

WATCH

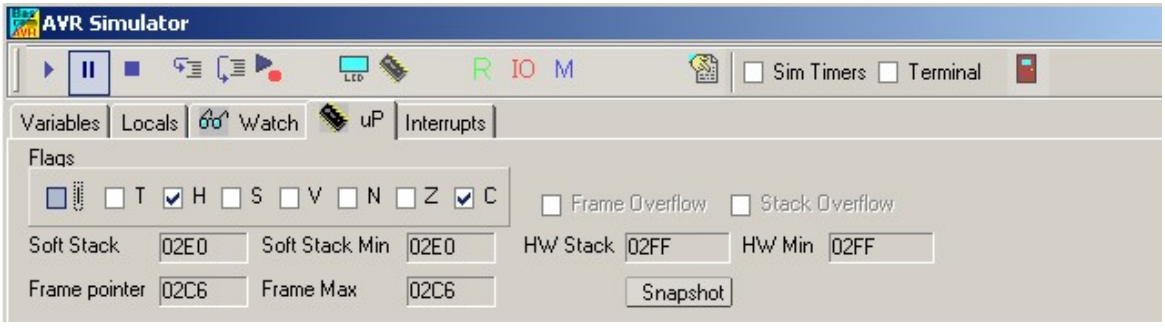


The Watch-TAB can be used to enter an expression that will be evaluated during simulation. When the expression is true the simulation is paused.

To enter a new expression, type the expression in the text-field below the Remove button, and press the Add-button.
When you press the Modify-button, the current selected expression from the list will be replaced with the current typed value in the text field.

To delete an expression, select the desired expression from the list, and press the Remove-button.
During simulation when an expression becomes true, the expression that matches will be selected and the Watch-TAB will be shown.

uP



This TAB shows the value of the microprocessor status register (SREG).

The flags can be changed by clicking on the check boxes.

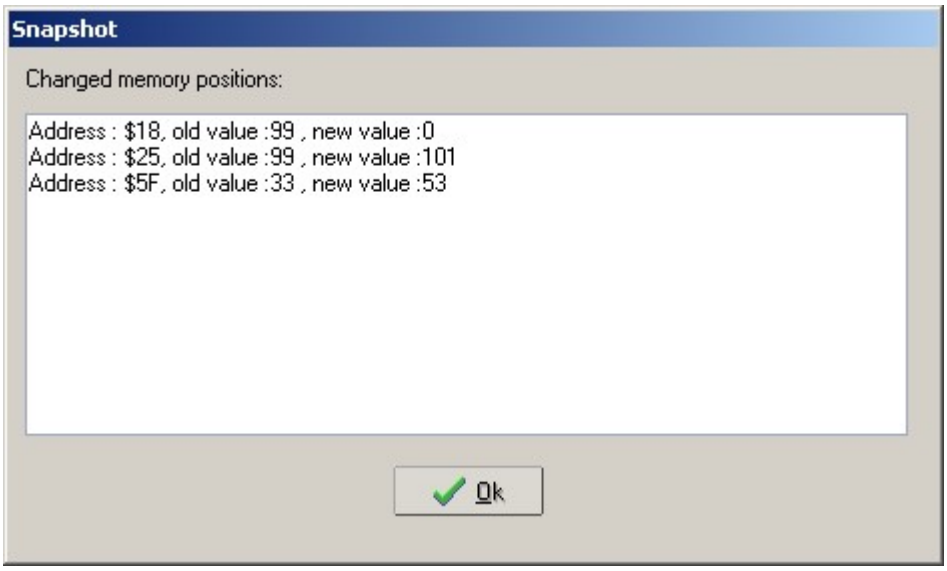
The software stack, hardware stack, and frame pointer values are shown. The minimum or maximum value that occurred during simulation is also shown. When

one of these data areas enter or overlap another one, a stack or frame overflow occurs.
This will be signaled with a pause and a check box.

Pressing the snapshot-button will save a snapshot of the current register values and create a copy of the memory.
You will notice that the Snapshot-button will change to 'Stop'

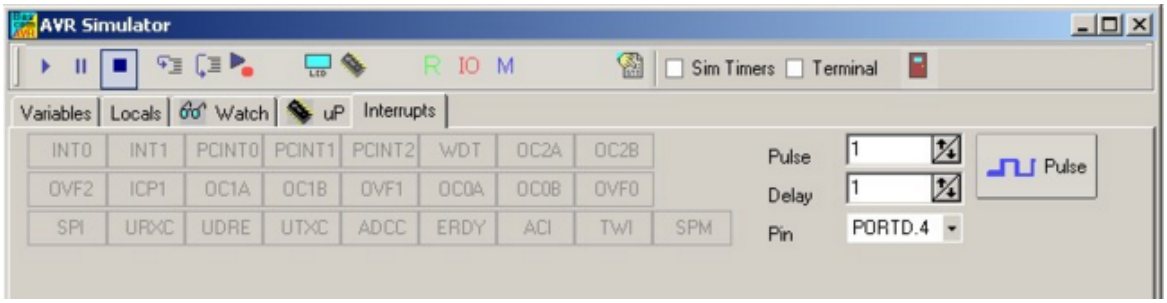
Now execute some code by pressing F8 and press the Snapshot-button again.

A window will pop up that will show all modified address locations.
This can help to determine which registers or memory a statement uses.



When you write an ISR (Interrupt Service Routine) with the NOSAVE option, you can use this to determine which registers are used and then save only the modified registers.

INTERRUPTS



This TAB shows the interrupt sources. When no ISR's are programmed all buttons will be disabled.
When you have written an ISR (using ON INT...), the button for that interrupt will be enabled. Only the interrupts that are used will be enabled.

By clicking an interrupt button the corresponding ISR is executed.
This is how you simulate the interrupts. When you have enabled 'Sim Timers' it can also trigger the event.

The pulse generator can be used to supply pulses to the timer when it is used in

counter mode.
First select the desired pin from the pull down box. Depending on the chip one or more pins are available. Most chips have 2 counters so there will usually be 2 input pins.
Next, select the number of pulses and the desired delay time between the pulses, then press the Pulse-button to generate the pulses.

The delay time is needed since other tasks must be processed as well.

The option 'Sim timers' must be selected when you want to simulate timers/counters.

TERMINAL Section

Under the window with the TABS you will find the terminal emulator window. It is the dark blue area.

In your program when you use PRINT, the output will be shown in this window.

When you use INPUT in your program, you must set the focus to the terminal window and type in the desired value.

You can also make the print output go directly to the COM port.
Check the Terminal option to enable this feature.
The terminal emulator settings will be used for the baud rate and COM port.
Any data received by the COM port will also be shown in the terminal emulator window.

Notice that most microprocessors have only 1 UART. The UART0-TAB is used to communicate with tis UART. The UART1-TAB need to be selected in order to view the UART1 output, or to send data to UART1.
In version 2083, UART0-UART3 are simulated. Unavailable UARTS are not shown.

Software UARTS are not supported by the simulator. They can not be simulated.

UART0

UART0 has some specific options. When you right click the mouse you will get a popup menu.
- Serial Input File.
This option selects a file with the SI extension. It must be named the same as your main file but having the SI extension. The content will be used as serial data input. Each time the processor checks UART0 it will read a byte fom the file as if it were sent.

- Load custom serial Input File
This option allows you to select a specific SI file. An Open Dialog will be shown and you can select the file.

- Copy
This option copies data sent to the simulated terminal.

- Paste
This option sends data to the simulated terminal.

- Log to File
This option creates a file with the LOG extension. It will have the name of your main file with the LOG extension. All data send to the simulated UART terminal will be send to the log file as well.

- Show in HEX

This option shows output in HEX format between brackets like [45] [6E] etc.

SOURCE Section

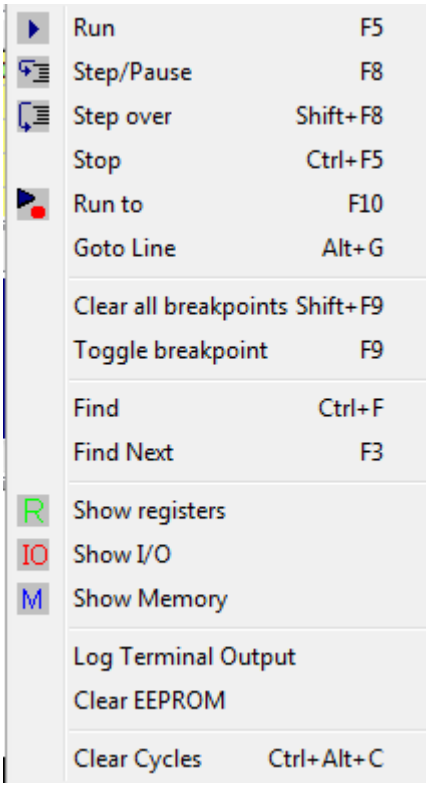
Under the Terminal section you find the Source Window.
It contains the source code of the program you are simulating. All lines that contain executable code have a yellow point in the left margin.
You can set a breakpoint on these lines by selecting the line and pressing **F9**.

By holding the mouse cursor over a variable name, the value of the variable is shown in the status bar.
If you select a variable, and press ENTER, it will be added to the Variable window.

In order to use the function keys (**F8** for stepping for example), the focus must be set to the Source Window.
In version 2083, the simulator source window will have the same fonts as the editor window. The source window is read only. You an not change the source code in the simulator!

A blue arrow will show the line that will be executed next.

When you right click a menu will be shown with the following options:



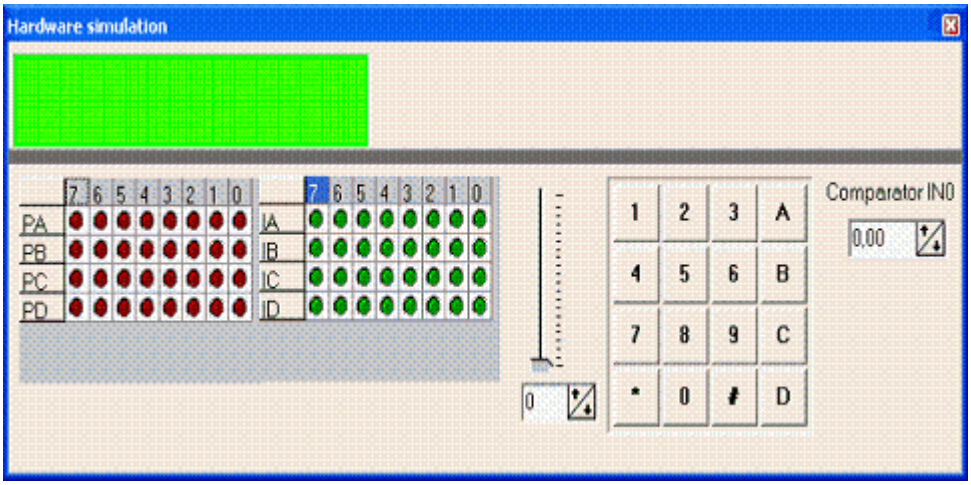
Option	Description
Run (F5)	Run code.
Step /Pause (F8)	Step through code or pause running code
Step Over (SHIFT+F8)	Step code but step over sub routines and functions..
Run To (F10)	Run to the current line. This line should have a yellow dot (contains executable code)

Goto Line (ALT+G).	This option let you chose a line to jump to. Use this with care since it will jump right to the code. This means that some parts of your code are not executed.
Clear All Breakpoints	This option clears all breakpoints set with F9.
Toggle breakpoint (F9)	This option will toggle a break point. It will only work on a line with executable code.
Find (CTRL+F)	Option to find text, similar to the function in the source editor
Find Next (F3)	Option to find next instance similar to the function in the source editor.
Show Registers	Option to show/hide internal registers R0-R31
Show IO	Option to show/hide IO registers
Show Memory	Option to show/hide memory content for SRAM and EEPROM
Log Terminal output	This option let you select a file name for the simulator output log file.
Clear EEPROM	This option will reset the EEPROM content to empty(FF). This is required sometimes since between sessions the EEPROM content is saved in an EEP file when this option is checked in Options, Simulator, Save EEPROM state. And when you restart simulation the EEP content is read. This option will clear the content.

The hardware simulator.

By pressing the hardware simulation button 

the windows shown below will be displayed.



The top section is a virtual LCD display. It works to display code in PIN mode, and bus mode. For bus mode, only the 8-bit bus mode is supported by the simulator.

Below the LCD display area are LED bars which give a visual indication of the ports.

By clicking an LED it will toggle.
PA means PORTA, PB means PORTB, etc.

IA means PINA, IB means PINB etc. (Shows the value of the Input pins)
It depends on the kind of microprocessor you have selected, as to which ports will be shown.

Right beside the PIN led's, there is a track bar. This bar can be used to simulate the input voltage applied the ADC converter. Note that not all chips have an AD converter. You can set a value for each channel by selecting the desired channel below the track bar.

Next to the track bar is a numeric keypad. This keypad can be used to simulate the GETKBD() function.

When you simulate the Keyboard, it is important that you press/click the keyboard button **before** simulating the getkbd() line !!!

To simulate the Comparator, specify the comparator input voltage level using Comparator IN0.

Enable Real Hardware Simulation

By clicking the  button you can simulate the actual processor ports in-circuit!
The processor chip used must have a serial port.

In order simulate real hardware you must compile the basmon.bas file.

To do this, follow this example:
Lets say you have the DT006 simmstick, and you are using a 2313 AVR chip.

Open the basmon.bas file and change the line $REGFILE = "xxx" to $REGFILE = "2313def.dat"
Now compile the program and program the chip.
It is best to set the lock bits so the monitor does not get overwritten if you accidentally press F4.
The real hardware simulation only works when the target micro system has a serial port. Most have and so does the DT006.

Connect a cable between the COM port of your PC and the DT006. You probably already have one connected. Normally it is used to send data to the terminal emulator with the PRINT statement.

The monitor program is compiled for 19200 baud. The Options Communication settings must be set to the same baud rate!
The same settings for the monitor program are used for the Terminal emulator, so select the COM port, and the baud rate of 19200.

Power up or reset the DT006. It probably already is powered since you just previously compiled the basmon.bas program and stored it in the 2313.
When you press the real hardware simulation button now the simulator will send and receive data when a port, pin or DDR register is changed.

This allows you to simulate an attached hardware LCD display for example, or something simpler, like an LED. In the SAMPLES dir, you will find the program DT006. You can compile the program and press F2.

When you step through the program the LED's will change!

All statements can be simulated this way but they have to be able to use static timing. Which means that 1-wire will not work because it depends on timing. I2C has

a static bus and thus will work.

NOTE: It is important that when you finish your simulation sessions that you click the button again to disable the Real hardware simulation.

When the program hangs it probably means that something went wrong with the serial communication. The only way to escape is to press the Real hardware Simulation  button again.

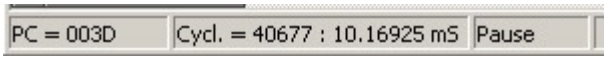
The Real Hardware Simulation is a cost effective way to test attached hardware.

 The refresh variables button will refresh all variables during a run(F5). When you use the hardware simulator, the LEDS will only update their state when you have enabled this option. Note that using this option will slow down the simulation.

Watchdog Simulation

Most AVR chips have an internal Watchdog. This Watchdog timer is clocked from an internal oscillator. The frequency is approximately 1 MHz. Voltage and temperature variations can have an impact on the WD timer. It is not a very precise timer. So some tolerance is needed when you refresh/reset the WD-timer. The Simulator will warn you when a WD overflow will occur. But only when you have enabled the WD timer.

The status bar



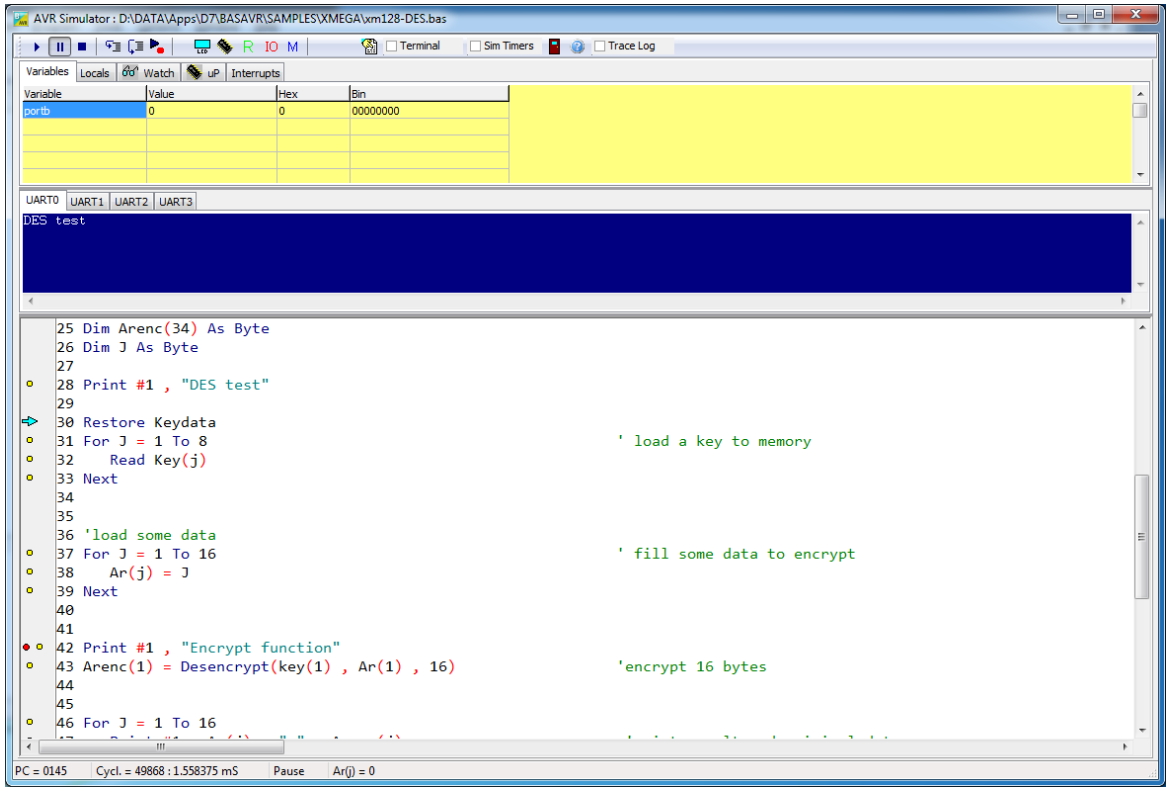
The status bar shows the PC (program counter) and the number of cycles. You can reset the cycles by positioning the mouse cursor on the status bar and then right click. You will then get a pop up menu with the option to reset the cycles. You can also double click the cycles to reset it to 0.

You can use this to determine how much time a program statement takes.
Do not jump to a conclusion too quick, the time shown might also depend on the value of a variable.
For example, with WAITMS var this might be obvious, but with the division of a value the time might vary too.

Start Simulation

To start a simulation the program need to be compiled. So typically you press F7 to compile your code. Make sure that the BIN, DBG and OBJ files are created (Options, Compiler, Output).
When the code is compiled without errors, you can simulate your project. To do so press F2.
By default the simulator is in STOP mode. The status bar will show PC = 0 (program counter) and Cycles = 0. Some instructions use more cycles than other. A NOP for example takes 1 cycle. When the processor has an oscillator running on 8 MHz, and the 8 DIV fuse is set, it means the processor will have a clock of 1 MHz. Meaning that each second, 1 million cycles can be executed. So you could execute a million NOP instructions in 1 second.
The simulator however is not able to do so. The simulator reads the object data, and decodes the data and simulates the instructions and the hardware. Also, the software need to give time to windows otherwise the code will stall windows and your other programs.

When the AVR is initialized, the RAM is cleared. This will takes time. So when you press F8 the first time, you will notice that the blue arrow will be visible on the first line of the main project. It depends on the used processor how long it will take till the initialization is done. When done, the simulator will go into PAUSE mode.
Press F8 again to step through the code. You will notice that the blue arrow will jump only to code with a yellow dot which indicates that the line contains executable code. DIM statements for example are important for the compiler but do not create code. So these statements will be skipped.
Using the BREAK instruction you can pause the simulator. This is a good way when instead of F8, you use F5 to RUN the code. You can also set a break point using F9. This will be visible with a red dot.



When your code uses INC modules, the simulator will show the name of the current module.

3.44 Program Send to Chip

Program send to chip shortcut  , F4

When you access the programmer from the main menu, you will notice the submenu. From the sub menu you can choose 'Program' or 'Manual Program'.
Program will erase and program the processor without any user intervention.
Manual Program will only show the programmer window. You can manually choose the options to program the chip when the programmer supports it.

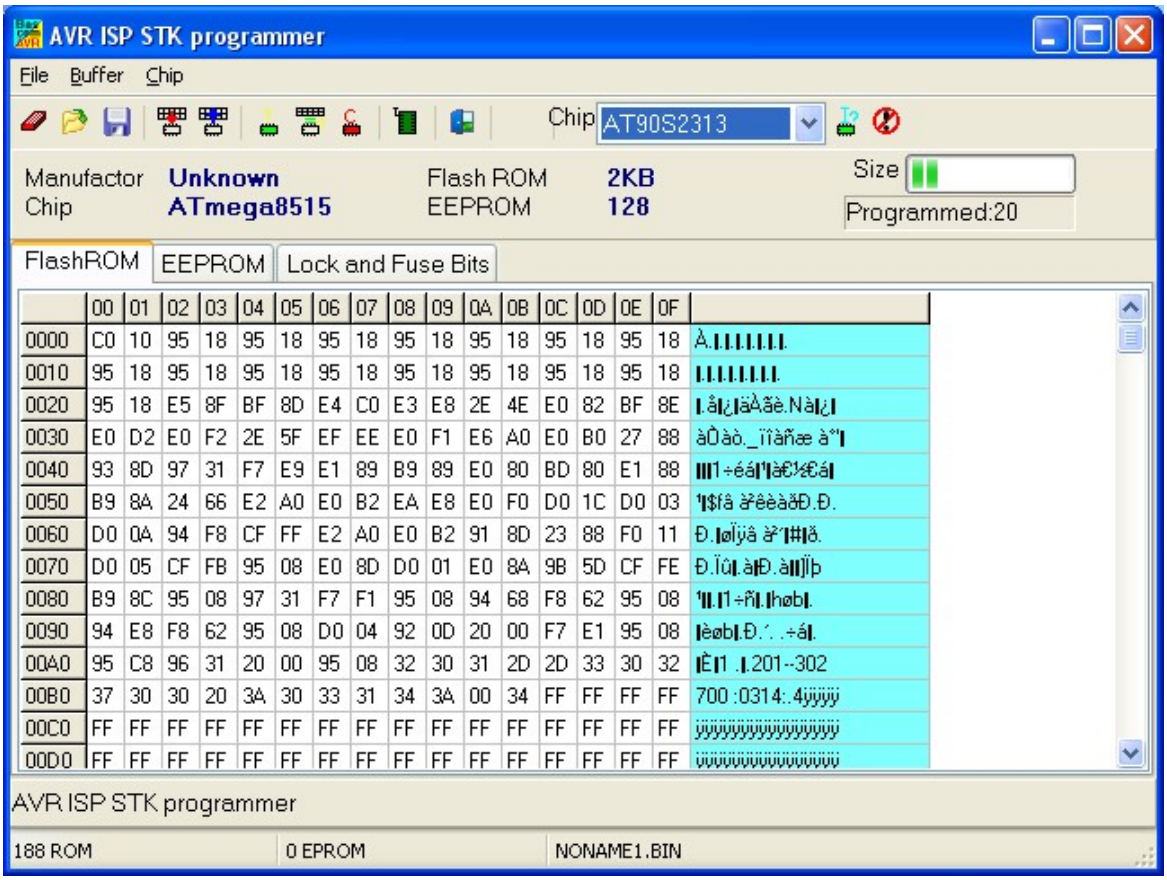
Auto Program also needs the option 'Auto Flash' to be set in the Programmer options 162.

The following section applies to the Programmer window (program chip directly NOT selected) otherwise this is not shown to the user.

"Buffer" below refers to the buffer memory that holds data to be programmed to, or read from the chip.

Menu item	Description
File Exit	Return to editor
File, Test	With this option you can set the logic level to the LPT pins. This is only intended for the Sample Electronics programmer.
Buffer Clear	Clears buffer
Buffer Load from file	Loads a file into the buffer
Buffer Save to file	Saves the buffer content to a file
Chip Identify	Identifies the chip
Write buffer into chip	Programs the buffer into the chip ROM or EEPROM
Read chip code into buffer	Reads the code or data from the chips code memory or data memory
Chip blank check	Checks if the chip is blank or erased
Chip erase	Erase the content of both the program memory and the data memory
Chip verify	Verifies if the buffer is the same as the chip program or data memory
Chip Set lock bits	Writes the selected lock bits LB1 and/or LB2. Only an erase will reset the lock bits
Chip auto program	Erases the chip and programs the chip. After the programming is completed, verification is performed.

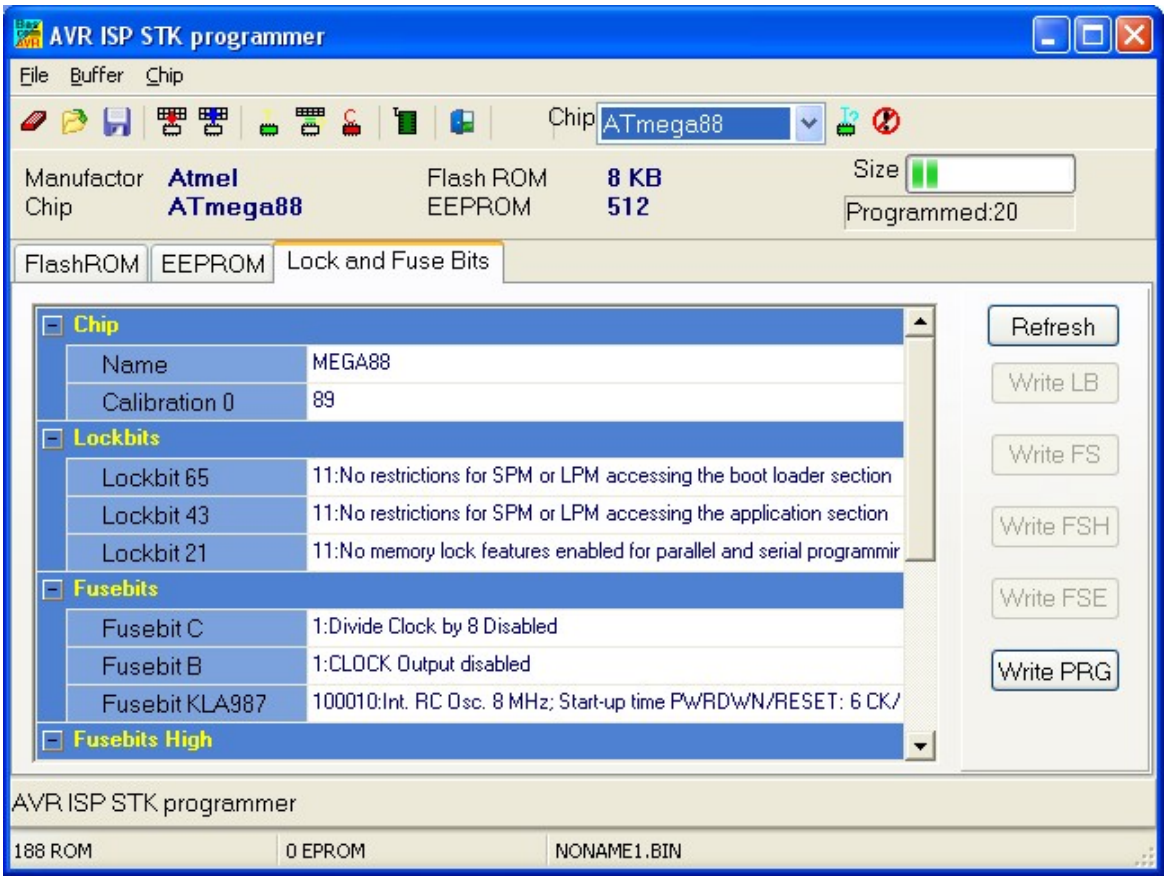
The following window will be shown for most programmers:



Note that a chip must be ERASED before it can be programmed.

By default the Flash ROM TAB is shown and the binary data is displayed.
When you have an EEPROM in your project, the EEPROM TAB will show this data too.

The most important TAB is in many cases the Lock & Fuse Bits TAB.
When you select it , the lock and fuse bits will be read.



These Lock and Fuse bits are different in almost every chip !
You can select new settings and write them to the chip. But be careful ! When you select a wrong oscillator option , you can not program the chip anymore without applying an external clock signal.
This is also the solution to communicate with the chip again : connect a clock pulse to the oscillator input. You could use an output from a working micro, or a clock generator or simple 555 chip circuit.

When you found the right settings, you can use $PROG 689 to write the proper settings to new, un-programmed chips. To get this setting you press the 'Write PRG' button.
After a new chip is programmed with $PROG, you should remark the line for safety and quicker programming.

The 'Write PRG' will write the settings, read from the Microprocessor, it will NOT insert the unsaved settings you have made manual. Thus, you must first use the 'Write XXX' buttons to write the changed fuse bits settings to the chip, then you can use the 'Write PRG'.

Notice that the Write xxx buttons are disabled by default. Only after you have changed a lock or fuse bit value, the corresponding button will be enabled. You must click this button in order to apply the new Lock or Fuse bit settings.

Many new chips have an internal oscillator. The default value is in most cases 8 MHz. But since in most cases the 'Divide by 8' option is also enabled, the oscillator value will be 1 MHz. We suggest to change the 'Divide by 8' fuse bit so you will have a speed of 8 MHz.
In your program you can use $crystal 625 = 8000000 then.

 $crystal will only inform the compiler which oscillator speed you have selected.

This is needed for a number of statements. $crystal will NOT set the speed of the oscillator itself.

Do not change the fuse bit that will change the RESET to a port pin. Some chips have this option so you can use the reset pin as a normal port pin. While this is a great option it also means you can not program the chip anymore using the ISP.

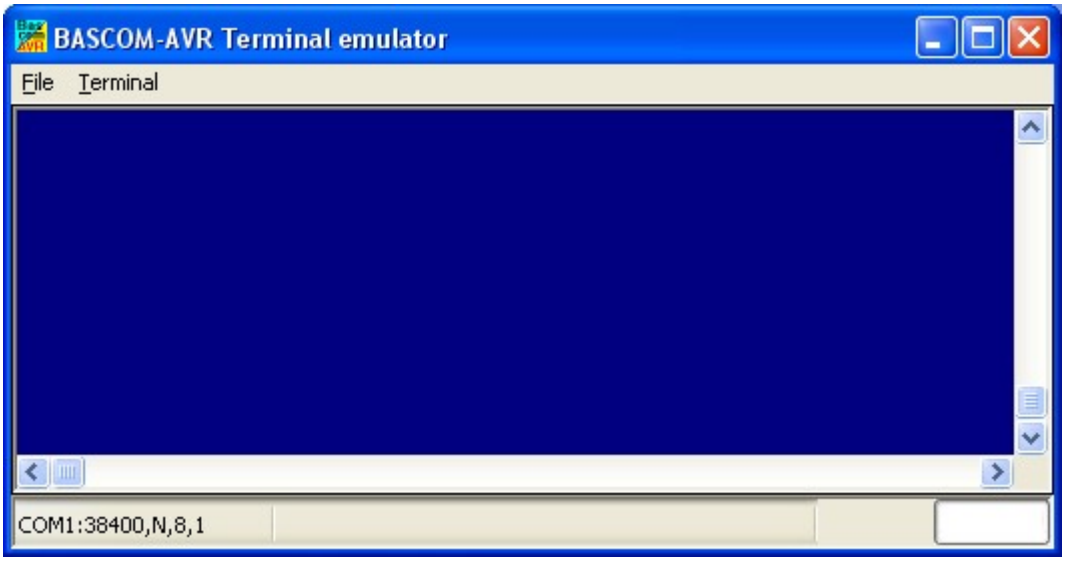
3.45 Program Reset Chip

This menu options will let the programmer reset the processor.
Shortcut : SHIFT+F4

The MCS UPDI programmer will perform a reset using UPDI protocol. It could be done using a DTR or RTS pin from the serial port but using UPDI has the advantage that you can program the RESET pin to be used as a normal IO pin. That is important on processors that do not have a dedicated RESET pin.

3.46 Tools Terminal Emulator

With this option you can communicate via the RS-232 interface to the microcomputer. The following window will appear:



Information you type and information that the computer board sends are displayed in the same window.

Note that you must use the same baud rate on both sides of the transmission. If you compiled your program with the Compiler Settings at 4800 baud, you must also set the Communication Settings to 4800 baud.

The setting for the baud rate is also reported in the report file.

NOTE: The focus MUST be on this window in order to see any data (text, etc)

sent from the processor. You will NOT see any data sent by the processor right after a reset. You must use an external hardware reset AFTER the terminal Emulator window is given focus in order to see the data. Using the Reset  shortcut, you will not be able to see any data because pressing the shortcut causes the Terminal emulator to lose focus. This is different than "Hyper Terminal" which always receives data even when the Hyper terminal window does not have focus. Use Hyper terminal if you need to see the program output immediately after programming or reset. Or use the option 'Keep terminal emulator open' from the Options, Communication.

File Upload

Uploads the current program from the processor chip in HEX format. This option is meant for loading the program into a monitor program for example. It will send the current compiled program HEX file to the serial port.

File Escape

Aborts the upload to the monitor program.

File Exit

Closes terminal emulator.

Terminal Clear

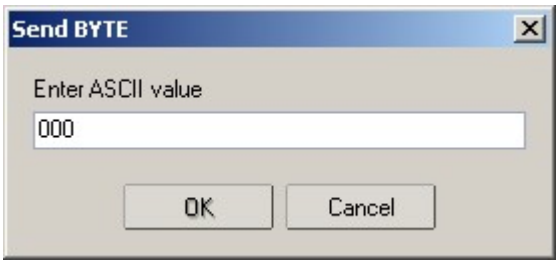
Clears the terminal window.

Terminal Open Log

Opens or closes the LOG file. When there is no LOG file selected you will be asked to enter a filename or to select a filename. All info that is printed to the terminal window is captured into the log file. The menu caption will change into 'Close Log' and when you choose this option the file will be closed.

Terminal Send ASCII

This option allows you to send any ASCII character you need to send. Values from 000 to 255 may be entered.



Terminal Send Magic number

This option will send 4 bytes to the terminal emulator. The intention is to use it together with the boot loader examples. Some of the boot loader samples check for a number of characters when the chip resets. When they receive 4 'magic' characters after each other, they will start the boot load procedure. This menu options send these 4 magic characters.

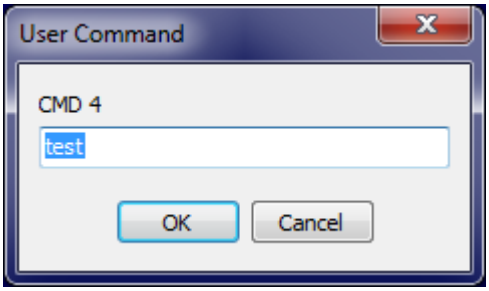
Terminal Setting

This options will show the terminal settings so you can change them quickly.
It is the same as Options, Communication 149.

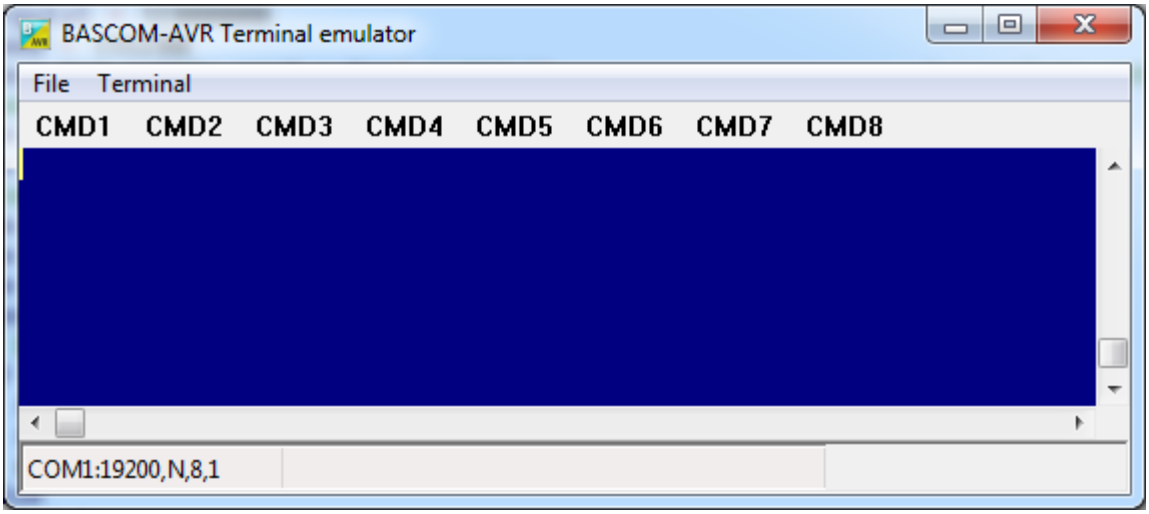
Terminal User Commands

This option will show or hide the toolbar with the user definable command buttons.

There are 16 user definable buttons named CMD1-CMD16. When you hover the mouse cursor above the button, the button data will be shown.
When you right click the mouse above the button, you can enter the data for the button.
Example for CMD4:



In the sample above the data "test" will be sent. No carriage return(CR) or line feed (LF) will be sent. If you want to send them as well you need to include them as special characters.
Special characters are entered with their 3 digit ASCII value between brackets : {xxx}
For example to send CR + LF you wend enter {013}{010}



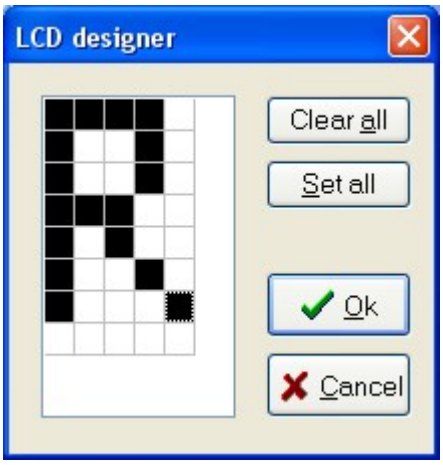
See Also

Options, Communication 149

3.47 Tools LCD Designer

With this option you can design special characters for LCD-text displays.

The following window will appear:



The LCD-matrix has 7x5 points. The bottom row is reserved for the cursor but can be used.
You can select a point by clicking the left mouse button. If a cell was selected it will be unselected.

Clicking the Set All button will set all points.
Clicking the Clear All button will clear all points.

When you are finished you can press the Ok button : a statement will be inserted in your active program-editor window at the current cursor position. The statement looks like this :

Deflcdchar ?,1,2,3,4,5,6,7,8

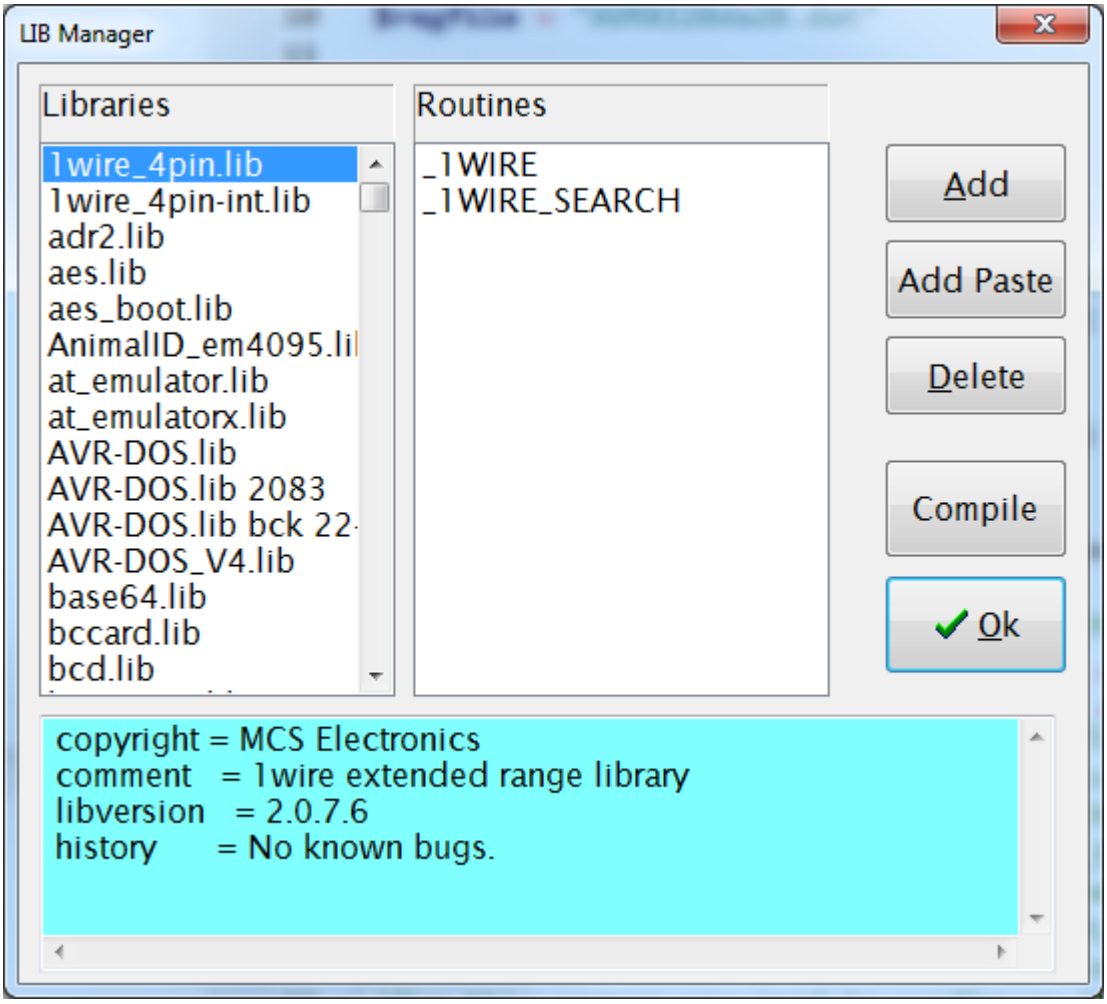
You must replace the ?-sign with a character number ranging from 0-7.
The eight bytes define how the character will appear. So they will be different depending on the character you have drawn.

See Also

Font Editor 238

3.48 Tools LIB Manager

With this option the following window will appear:



The Libraries are shown in the left pane. When you select a library, the routines that are in the library will be shown in the right pane.

After selecting a routine in the left pane, you can DELETE it with the DELETE button..

Clicking the ADD button allows you to add an ASM routine to the library. The ADD Paste button will add a routine from the clipboard instead of a file on disk.

The COMPILE button will compile the lib into an LBX file. When an error occurs you will get an error. By watching the content of the generated lbx file you can determine the error.

A compiled LBX file does not contain comments and a huge amount of mnemonics are compiled into object code. This object code is inserted at compile time of the main BASIC program. This results in faster compilation time.

The DEMO version comes with the compiled MCS.LIB file which is named MCS.LBX. The ASM source (MCS.LIB) is included only with the commercial edition.

With the ability to create LBX files you can create add on packages for BASCOM and sell them. For example, the LBX files could be distributed for free, and the ASM

source could be sold.

Some library examples :

- MODBUS crc routine for the modbus slave program.
- Glcd.lib contains the graphical LCD asm code

Commercial packages available from MCS:

- I2CSLAVE library
- BCCARD for communication with www.basiccard.com chipcards

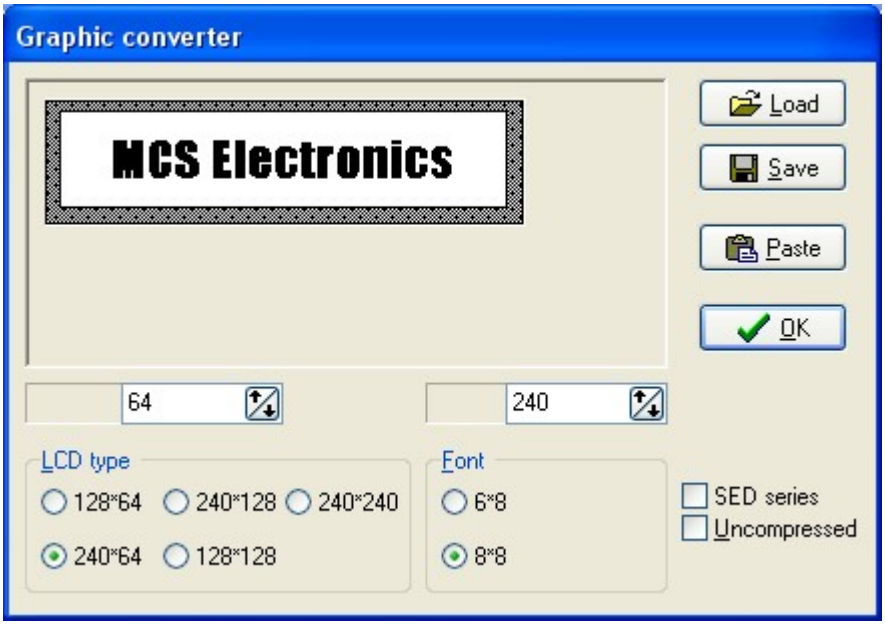
See Also

$LIB for writing your own libraries

3.49 Tools Graphic Converter

The Graphic converter is intended to convert BMP files into BASCOM Graphic Files (. BGF) that can be used with Graphic LCD displays.

The following dialog box will be shown:



To load a picture click the Load button.
The picture can be maximum 128 pixels high and 240 pixels width.

When the picture is larger it will be adjusted.

You can use your favorite graphic tool to create the bitmaps and use the Graphic converter to convert them into black and white images.

When you click the Save-button the picture will be converted into black and white. Any non-white color will be converted into black.

The resulting file will have the BGF extension. (bascom graphics format)

You can also paste a picture from the clipboard by clicking the Paste button.

Press the Ok-button to return to the editor.

The picture can be shown with the ShowPic 1355 statement or the ShowpicE 1355 statement.

 The BGF files are RLE encoded to save space.

 It is important that the font selection 6*8 or 8*8 match the font size in the CONFIG GRAPHLCD. For example :
Config Graphlcd = 240x128 , Dataport = Porta , Controlport = Portc , Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , **Mode = 8**
In this case you would use 8*8.

When you use your own drawing routine you can also save the pictures uncompressed by setting the Uncompressed check box. The resulting BGF files can not be shown with the showpic or showpicE statements anymore in that case!

The BGF format is made up as following:

- first byte is the height of the picture
- second byte is the width of the picture
- for each row, all pixels are scanned from left to right in steps of 6 or 8 depending on the font size. The resulting byte in stored with RLE compression

The RLE method used is : byte value, AA(hex), repeats.
So a sequence of 5, AA, 10 means that a byte with the value of 5 must be repeated 16 times (hex notation used)

Option	Description
Height	The height in pixels of the image.
Width	The width in pixels of the image.
Font	The T6963 supports 6x8 and 8x8 fonts. This is the font select that must match the CONFIG statement. For other displays, use 8*8.
Type	The size of the display. When the size is not listed, use one with the same width.
SED Series	If your display is a SEDxxxx chip, select this option.
Uncompressed	Images are RLE encoded. Select this option when you do not want to compress the image.

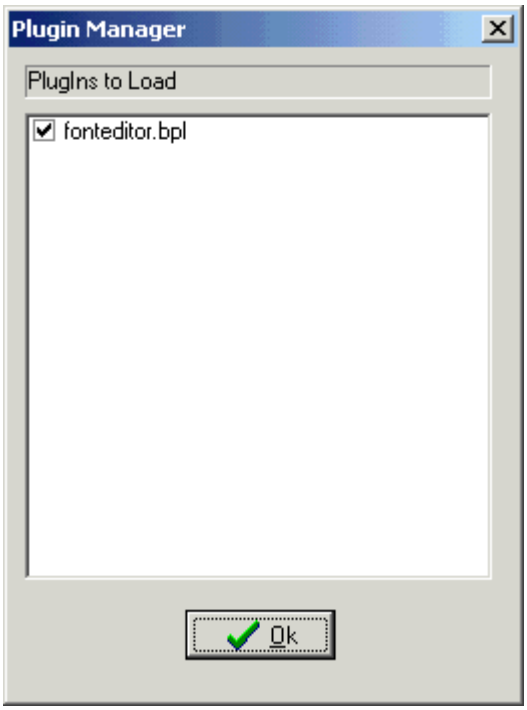
3.50 Tools Stack Analyzer

The Stack analyzer helps to determine the proper stack size.

See $DBG 628 for the proper usage of this option.

3.51 Tools Plugin Manager

The Plug in Manager allows you to specify which Plug-in's needs to be loaded the next time you start BASCOM.



Just select the plug in's you want to load/use by setting the check box.
The plug in's menu's will be loaded under the Tools Menu.

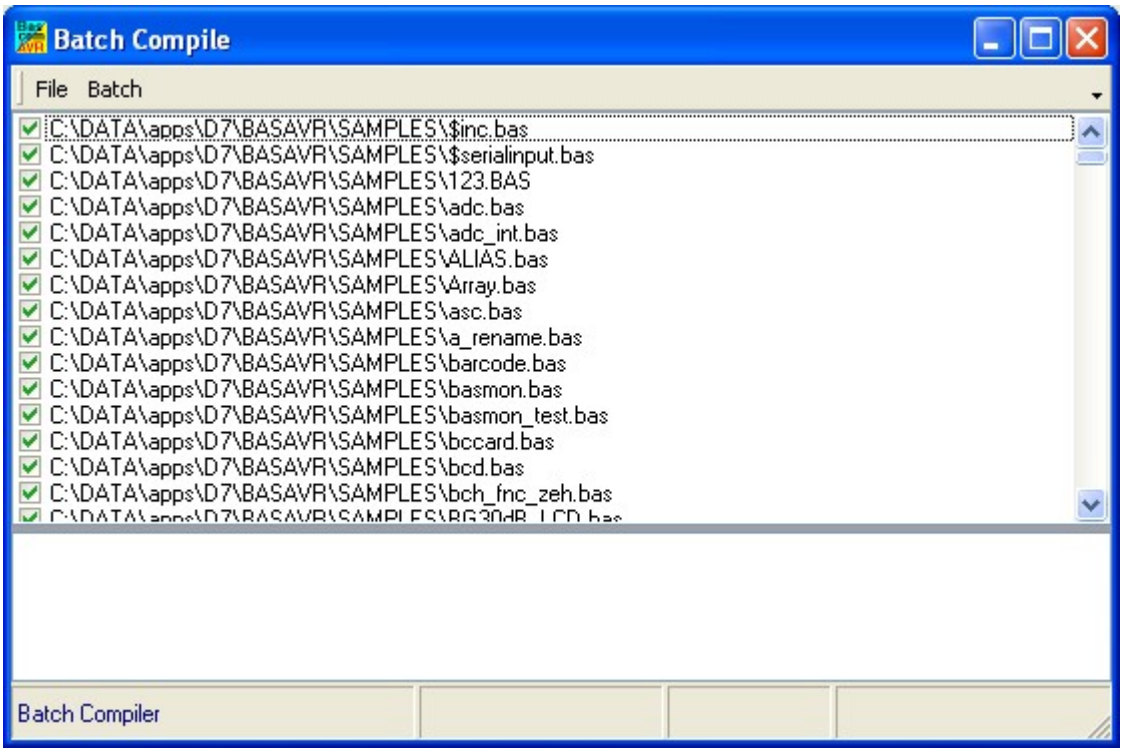
To add a button to the toolbar, right click the mouse on the menu bar, and choose customize.

When you want to write your own plug in's, contact support@mcselec.com

3.52 Tools Batch Compile

The Batch Compiler is intended to compile multiple files.
Shortcut : CTRL+B

The Batch compile option was added for internal test usage. It is used by MCS to test the provided test samples.
The following window is shown :



There are a number of menu options.

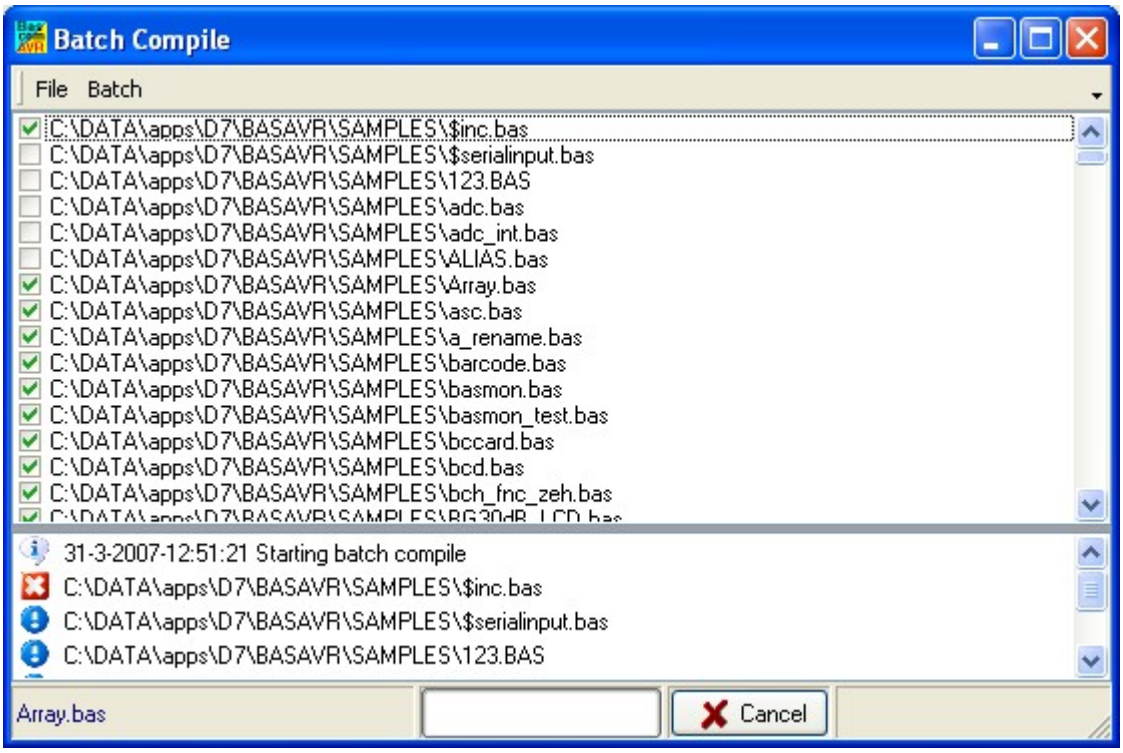
File Load Batch
Load an earlier created and saved batch file list from disk.

File Save Batch
Save a created list of files to disk
When you have composed a list with various files it is a good idea to save it for later re usage.

File Save Result
Save the batch compile log file to disk. A file named batchresult.txt will be saved in the BASCOM application directory.

File Exit
Close window

Batch Compile
Compile the checked files. By default all files you added are checked. During compilation all files that were compiled without errors are unchecked.



This screen print shows that $inc.bas could not be compiled.
And that array.bas was not yet compiled.

Batch Add Files
Add files to the list. You can select multiple *.BAS files that will be added to the list.

Batch Add Dir
Add a directory to the list. All sub directories will be added too. The entire directory and the sub directories are searched for *.BAS files. They are all added to the list.

Batch Clear List
Clear the list of files.

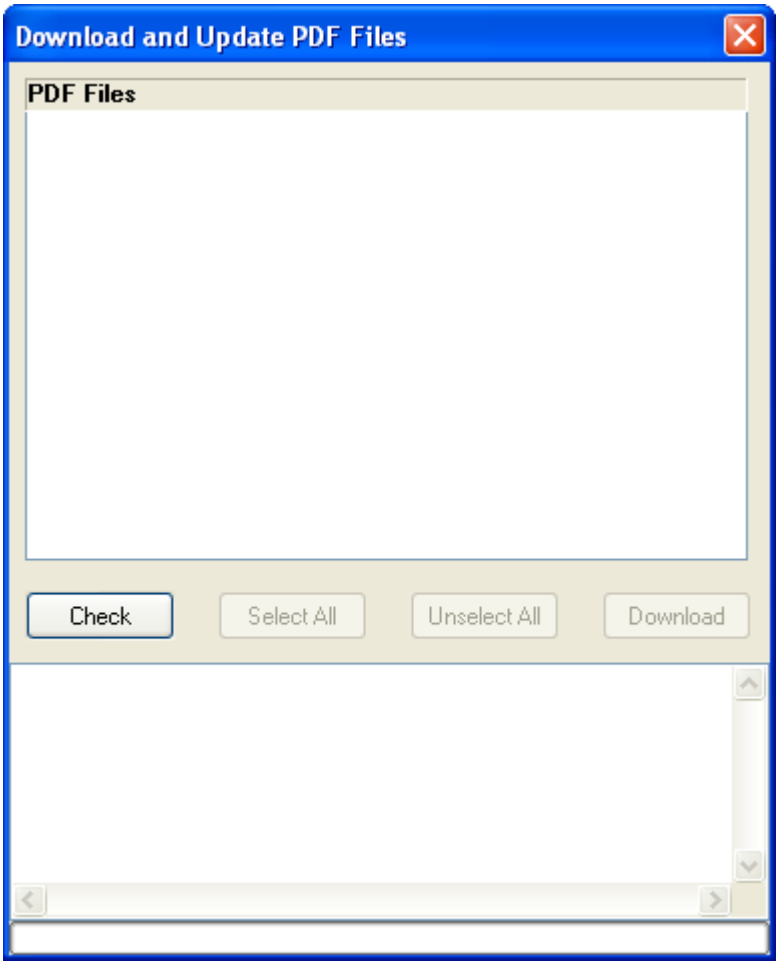
Batch Clear Good
Remove the files that were compiled without error. You will keep a list with files that compiled with an error.

All results are shown in an error list at the bottom of the screen.
When you double click an item, the file will be opened by the editor.

See Also
$NOCOMP 685

3.53 Tools PDF Update

Use this option to update all Atmel PDF files.
The Atmel data sheets are stored in the **\PDF** subdirectory.
The following window will be shown :

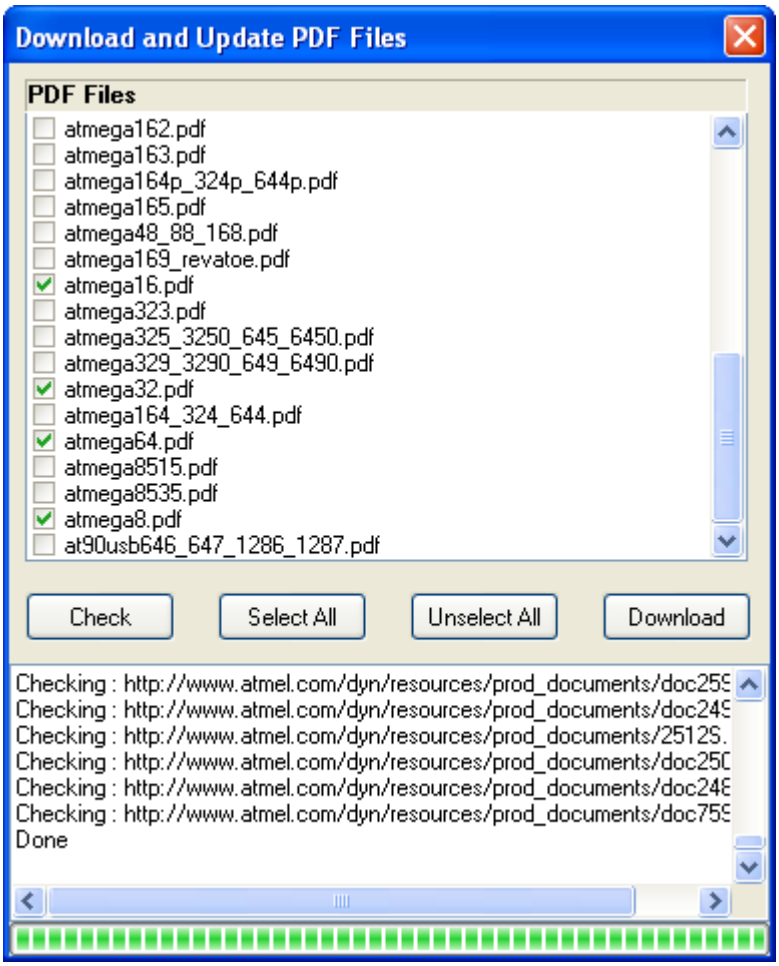


There is only one option available : Check. When you click the Check-button, the MCS server will be checked for newer versions of the PDF documents.
You need to make sure that BASCOM is allowed to contact the internet.
You also need to have port **211** open. This port is used in FTP mode to contact the MCS server.

The MCS server is synchronizing all PDF files each day with the ATMEL server. This means that the copy on the MCS server can be maximum 24 hours old.

The check will read all available DAT files and check if there is a reference to the PDF. When an item is disabled(grayed) then it means there is no link to the PDF in the DAT file.

During the check the window will look like this :



All PDF's that are newer will have a check mark. These need an update.
You can manual unselect or select the PDF's.
In the log window at the bottom of the window you can view which files will be downloaded.

When you want to download the selected files, press the Download-button.
This will close all PDF documents in the PDF viewer. A backup of each PDF file downloaded will be made before it is downloaded. You need to restore it when something goes wrong during the download(server drops the connection for example).
When a document is downloaded, the check mark will be removed.

After all documents are downloaded, they documents are opened again in the PDF viewer.

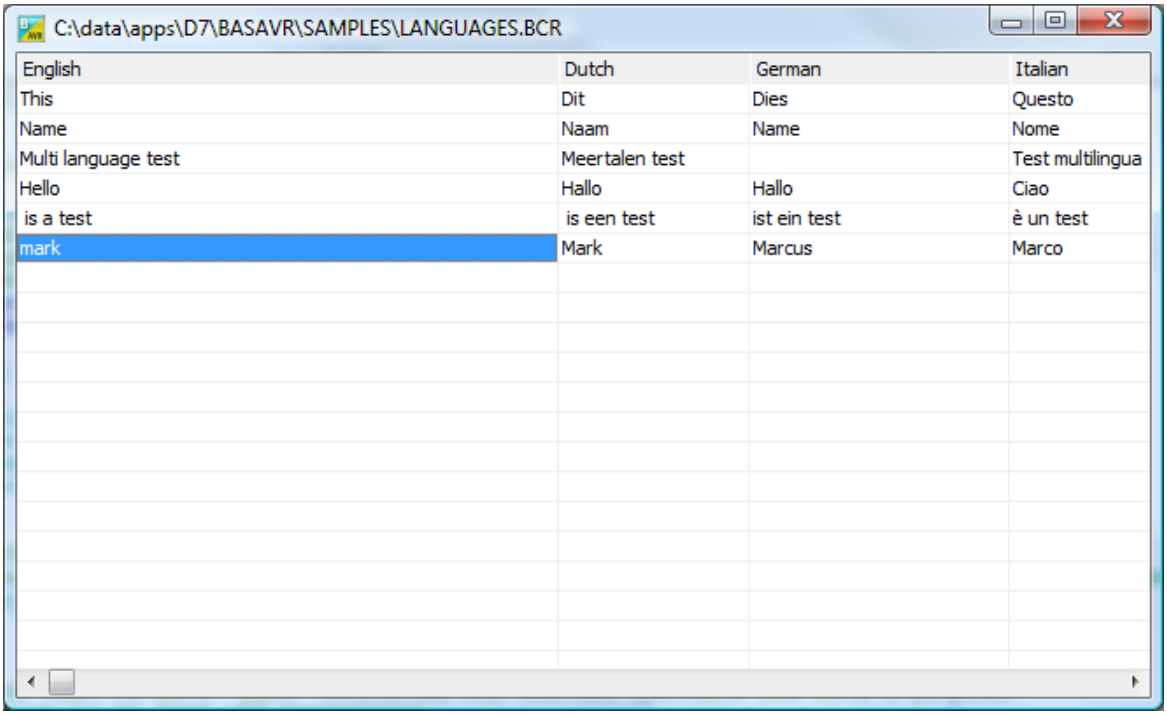
As of version 2077 the PDF documents are downloaded from the MCS Electronics server.
Previously they were downloaded from Atmels webserver. When Atmel change the file name the link is broken and you can not update the file.
To solve this all files are stored on the MCS server and each day all files are synchronized with atmel so all files are maximum 1 day old.

As of version 2079 the PDF files are downloaded using FTP. This results in a better performance. Just make sure port 411 is open in your firewall for outgoing connections.

3.54 Tools Resource Editor

The resource editor can be used to edit the resource strings of your application.
The resource editor will create a <project>.BCR file.
The resource editor is part of the Resource Add On, and is only available when you have this add on installed.

The simplest way to get the resources from your application is to create a BCS file using the DUMP option.
Then import them with the resource editor.



The following options are available when you right click with the mouse in the resource editor.

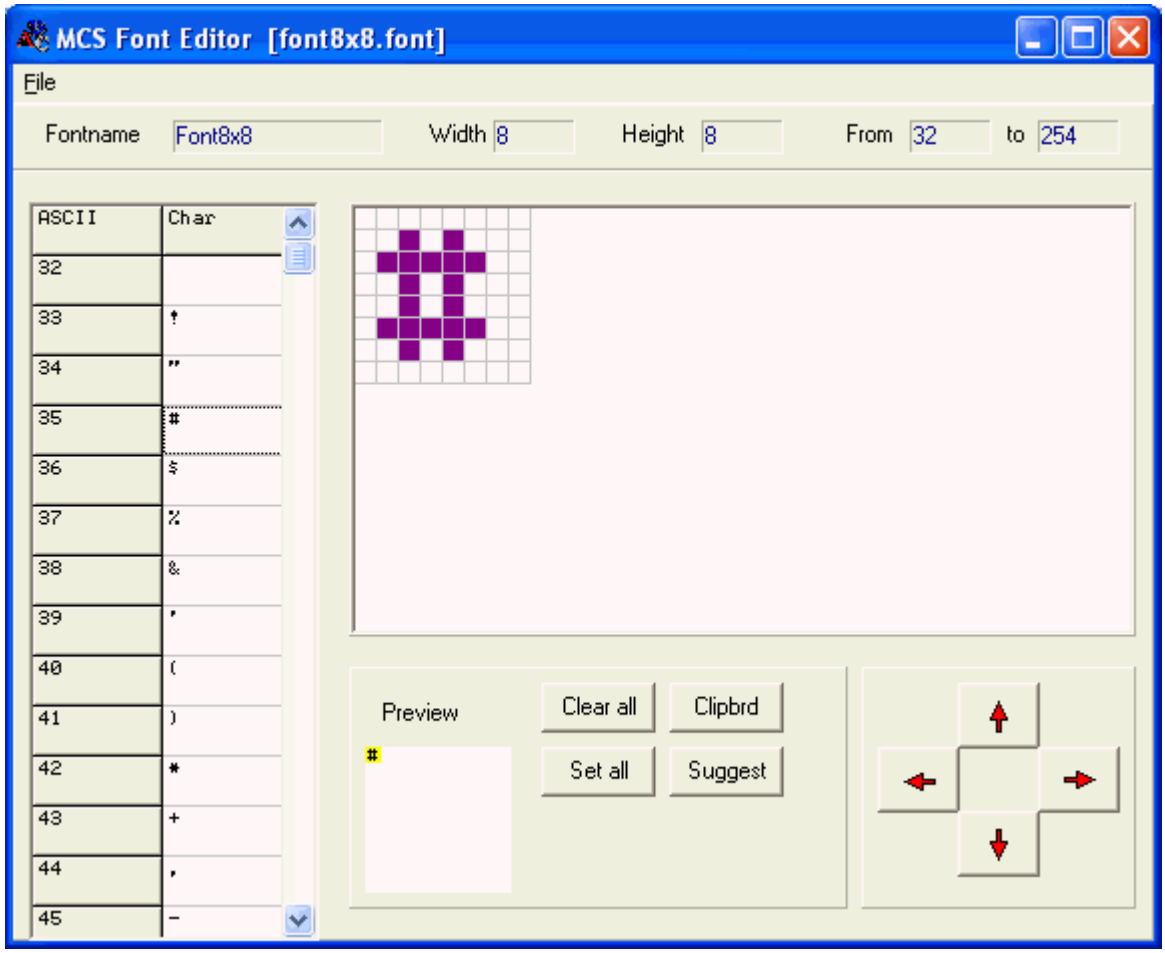
Option	Description
Search	Search for a string.
Find Next	Find next occurrence.
Delete Row	Delete the current row.
Add Row	Add a new row for a new string.
Import	This option will import the BCS file which you can create with the $RESOURCE DUMP option.
Set Language Name	Change the language name of the current language/column.
Add Language	Add a new column for a new language.
Delete Language	Delete the current column (language).

The resource editor is pretty simple. The only task is allow you to edit the various strings. You can also use notepad or Excel to create the BCR file which is explained in the $RESOURCE 694 topic.

3.55 Tools Font Editor

The Font Editor was a Plug in which is now integrated into the Tools menu.
The editor is intended to create Fonts that can be used with Graphical display such as SED1521, KS108, color displays, etc.

When you choose this option the following window will appear:



You can open an existing Font file, or Save a modified file.

The supplied font files are installed in the Samples\lcdgraph folder.
You can copy an image from the clipboard, and you can then move the image up , down, left and right.

When you select a new character, the current character is saved. The suggest button will draw an image of the current selected character.

When you keep the left mouse button pressed, you can set the pixels in the grid.
When you keep the right mouse button pressed, you can clear the pixels in the grid.

When you choose the option to create a new Font, you must provide the name of the font, the height of the font in pixels and the width of the font in pixels.

The Max ASCII is the last ASCII character value you want to use. Each character will occupy space. So it is important that you do not choose a value that is too high and will not be used.

When you display normal text, the maximum number is 127 so it does not make sense to specify a value of 255.

A font file is a plain text file.
Lets have a look at the first few lines of the 8x8 font:

```
Font8x8:
$asm
.db 1,8,8,0
.db 0,0,0,0,0,0,0,0 ;
.db 0,0,6,95,6,0,0,0 ; !
```

The first line contains the name of the font. With the SETFONT 1351 statement you can select the font. Essential, this sets a data pointer to the location of the font data.

The second line ($ASM) is a directive for the internal assembler that asm code will follow.
All other lines are data lines.

The third line contains 4 bytes: 1 (height in bytes of the font) , 8 (width in pixels of the font), 8 (block size of the font) and a 0 which was not used before the 'truetype' support, but used for aligning the data in memory. This because AVR object code is a word long.

This last position is **0** by default. Except for 'TrueType' fonts. In BASCOM a TrueType font is a font where every character can have it's own width. The letter 'i' for example takes less space then the letter 'w'. The EADOG128 library demonstrates the TrueType option.
In order to display TT, the code need to determine the space at the left and right of the character. This space is then skipped and a fixed space is used between the characters. You can replace the 0 by the width you want to use. The value 2 seems a good one for small fonts.

All other lines are bytes that represent the character.
Graphical LCD uses 1 byte to set 8 pixels black/white.
The LCD driver will for load the height in bytes i the example this is 1 byte.
Then the driver will load the width in bytes of the character. in the example this is 8.
This means that data lines are 8 bytes long.
And finally the block size in bytes of one character will be loaded. This is simply the first parameter times the second parameter. In the example this is 8 too. The last parameter is loaded to see if TT spacing must be used.

Depending on the ASCII value of the character to show, the driver will located the proper place in the table by multiplying the block size with the ASCII value. This will be added to the start of the table address.

Now the driver will load and write a byte and thus will set 8 pixels.
The pixels are shown from top to bottom like this :

```
x
x
x
x
x
x
x
x
```

Then the next byte will be loaded and send to the lcd.

xy
xy
xy
xy
xy
xy
xy
xy

and this will repeat for the number of bytes specified.

Then if the height is more than 1 byte, the next block will be loaded.
This also means that each font height must be a multiple of 8 pixels.
This restriction is only because writing a full byte is the fastest way to update an LCD.

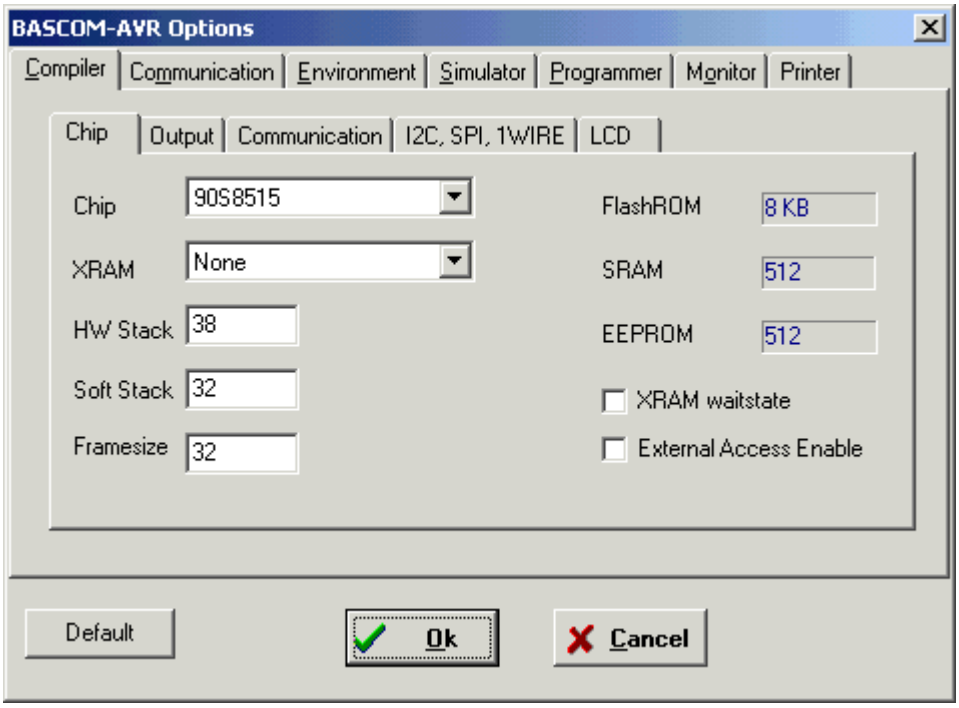
3.56 Options Compiler

With this option, you can modify the compiler options.

The following TAB pages are available:

Options Compiler Chip 143
Options Compiler Output 145
Options Compiler Communication 146
Options Compiler I2C , SPI, 1WIRE 147
Options Compiler LCD 148

3.56.1 Options Compiler Chip

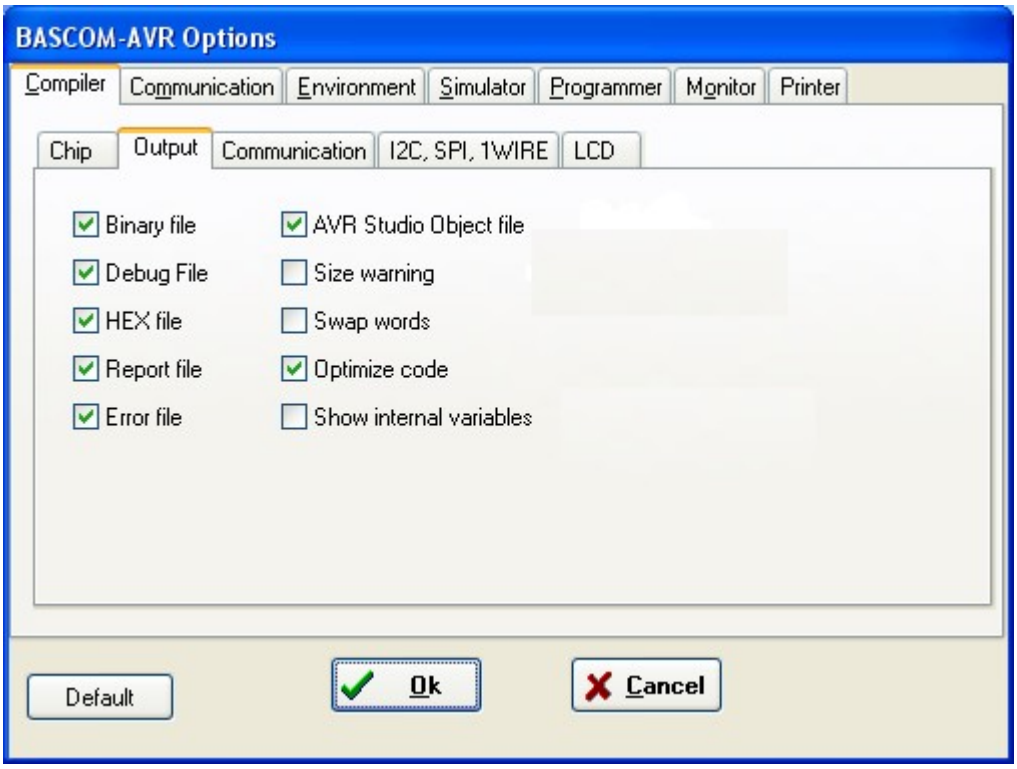


The following options are available:

Options Compiler Chip

Item	Description
Chip	Selects the target chip. Each chip has a corresponding x.DAT file with specifications of the chip. Note that some DAT files are not available yet.
XRAM	Selects the size of the external RAM. KB means Kilo Bytes. For 32 KB you need a 62256 STATIC RAM chip.
HW Stack	The amount of bytes available for the hardware stack. When you use GOSUB or CALL, you are using 2 bytes of HW stack space. When you nest 2 GOSUB's you are using 4 bytes (2*2). Most statements need HW stack too. An interrupt needs 32 bytes.
Soft Stack	Specifies the size of the software stack. Each local variable uses 2 bytes. Each variable that is passed to a sub program uses 2 bytes too. So when you have used 10 locals in a SUB and the SUB passes 3 parameters, you need 13 * 2 = 26 bytes.
Frame size	Specifies the size of the frame. Each local variable is stored in a space that is named the frame space. When you have 2 local integers and a string with a length of 10, you need a frame size of (2*2) + 11 = 15 bytes. The internal conversion routines used when you use INPUT num, or STR(), or VAL(), etc, also use the frame. They need a maximum of 16 bytes. So for this example 15+16 = 31 would be a good value.
XRAM wait state	Select to insert a wait state for the external RAM.
External Access enable	Select this option to allow external access of the micro. The 8515 for example can use port A and C to control a RAM chip. This is almost always selected if XRAM is used
Default	Press or click this button to use the current Compiler Chip settings as default for all new projects.

3.56.2 Options Compiler Output

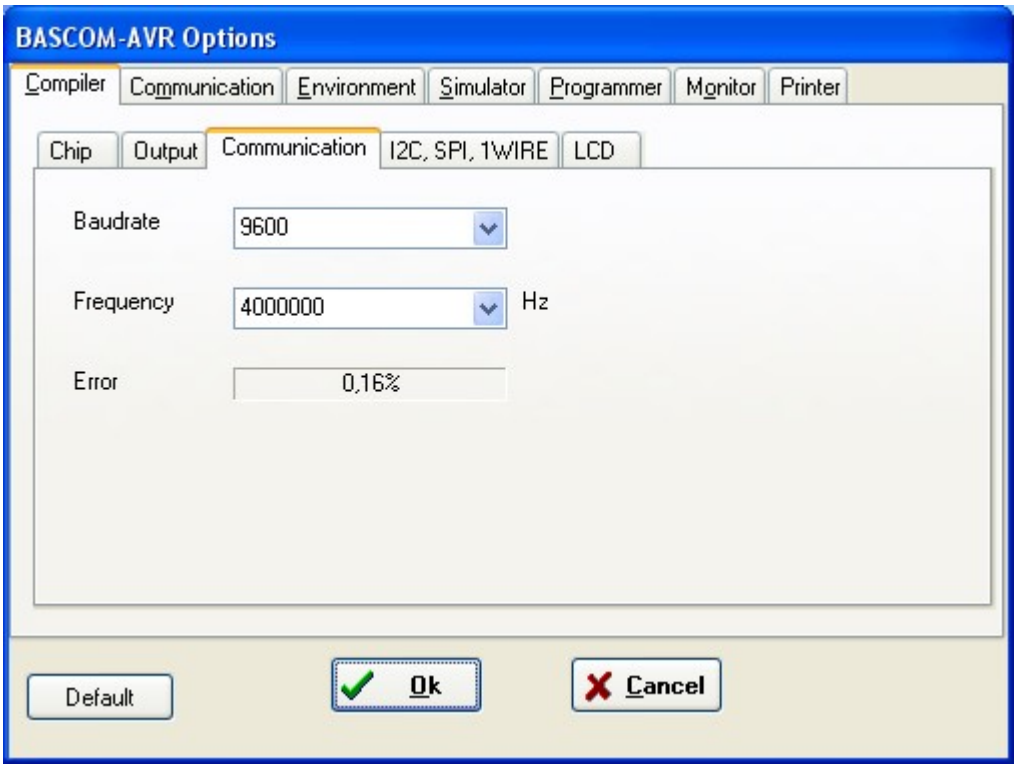


Options Compiler Output

Item	Description
Binary file	Select to generate a binary file. (xxx.bin)
Debug file	Select to generate a debug file (xxx.dbg)
Hex file	Select to generate an Intel HEX file (xxx.hex)
Report file	Select to generate a report file (xxx.rpt)
Error file	Select to generate an error file (xxx.err)
AVR Studio object file	Select to generate an AVR Studio object file (xxx.obj) Using the OBJ file you can debug with AVR Studio. This also allows to use tools like ICE. In Studio 6.0 (fixed in 6.1) you need to make these changes in Studio : Locate the file atmelstudio.pkgundef under the installation folder for Atmel. Studio. Remove (or remark) the below lines from the file and save the file. [$RootKey$\Languages\Language Services\Basic] [$RootKey$\AutomationProperties\TextEditor\Basic]
Size warning	Select to generate a warning when the code size exceeds the Flash ROM size.
Swap words	This option will swap the bytes of the object code words. Useful for some programmers. Should be disabled for most programmers.

	Don't use it with the internal supported programmers.
Optimize code	This options does additional optimization of the generated code. Since it takes more compile time it is an option.
Show internal variables	Internal variables are used. Most of them refer to a register. Like _TEMP1 = R24. This option shows these variables in the report.

3.56.3 Options Compiler Communication



Options Compiler Communication

Item	Description
Baud rate	Selects the baud rate for the serial communication statements. You can also type in a new baud rate. It is advised to use $BAUD 607 in the source code which overrides this setting.
Frequency	Select the frequency of the used crystal. You can also type in a new frequency. It is advised to use $CRYSTAL 625 in the source code which overrides this setting. Settings in source code are preferred since it is more clear.

The settings for the internal hardware UART are:

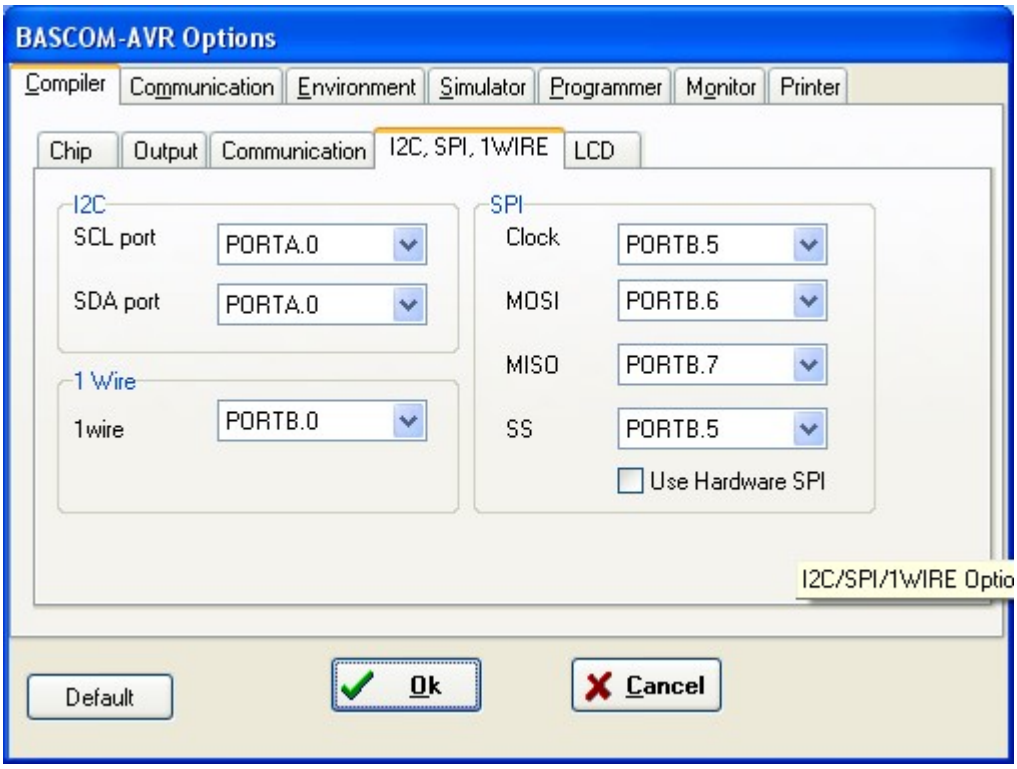
No parity , 8 data bits , 1 stop bit

Some AVR chips have the option to specify different data bits and different stop bits and parity.

Note that these settings must match the settings of the terminal emulator. In the

simulator the output is always shown correct since the baud rate is not taken in consideration during simulation. With real hardware when you print data at 9600 baud, the terminal emulator will show weird characters when not set to the same baud rate, in this example, to 9600 baud.

3.56.4 Options Compiler I2C, SPI, 1WIRE

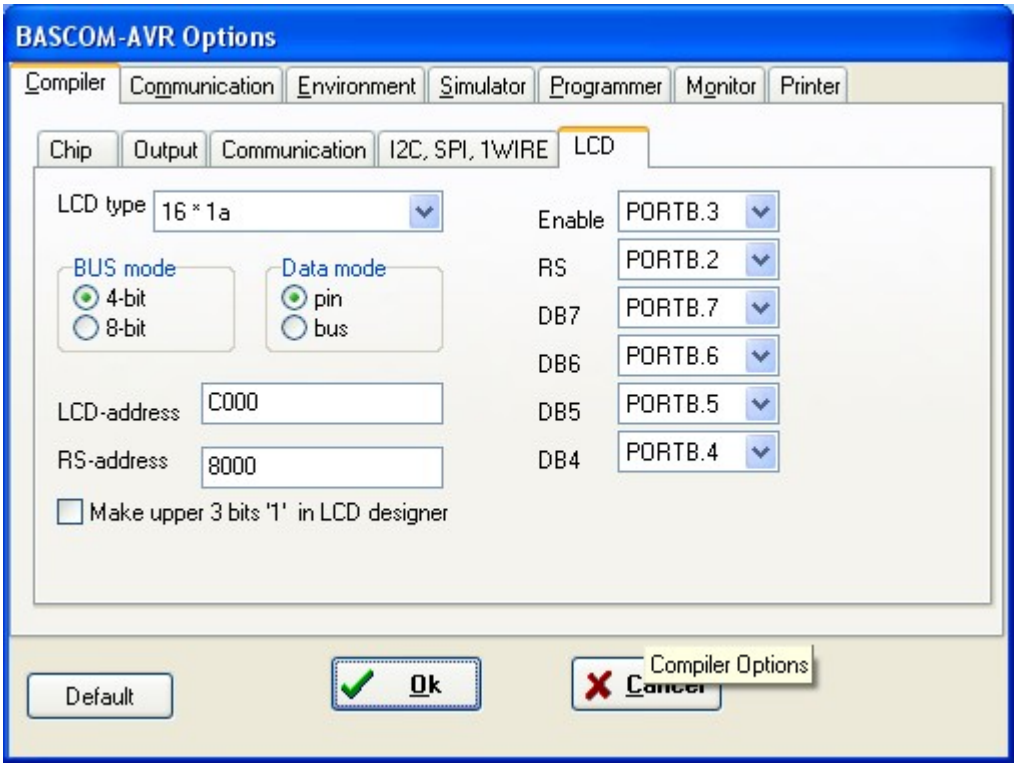


Options Compiler I2C, SPI, 1WIRE

Item	Description
SCL port	Select the port pin that serves as the SCL-line for the I2C related statements.
SDA port	Select the port pin that serves as the SDA-line for the I2C related statements.
1WIRE	Select the port pin that serves as the 1WIRE-line for the 1Wire related statements.
Clock	Select the port pin that serves as the clock-line for the SPI related statements.
MOSI	Select the port pin that serves as the MOSI-line for the SPI related statements.
MISO	Select the port pin that serves as the MISO-line for the SPI related statements.
SS	Select the port pin that serves as the SS-line for the SPI related statements.
Use hardware SPI	Select to use built-in hardware for SPI, otherwise software emulation of SPI will be used. The 2313 does not have internal HW SPI so it can only be used with software SPI mode. When you do use hardware SPI, the above settings are not used anymore since the SPI pins are dedicated pins and can not be chosen by the user.

It is advised to use the various CONFIG commands in your source code. It make more clear in the source code which pins are used.

3.56.5 Options Compiler LCD



Options Compiler LCD

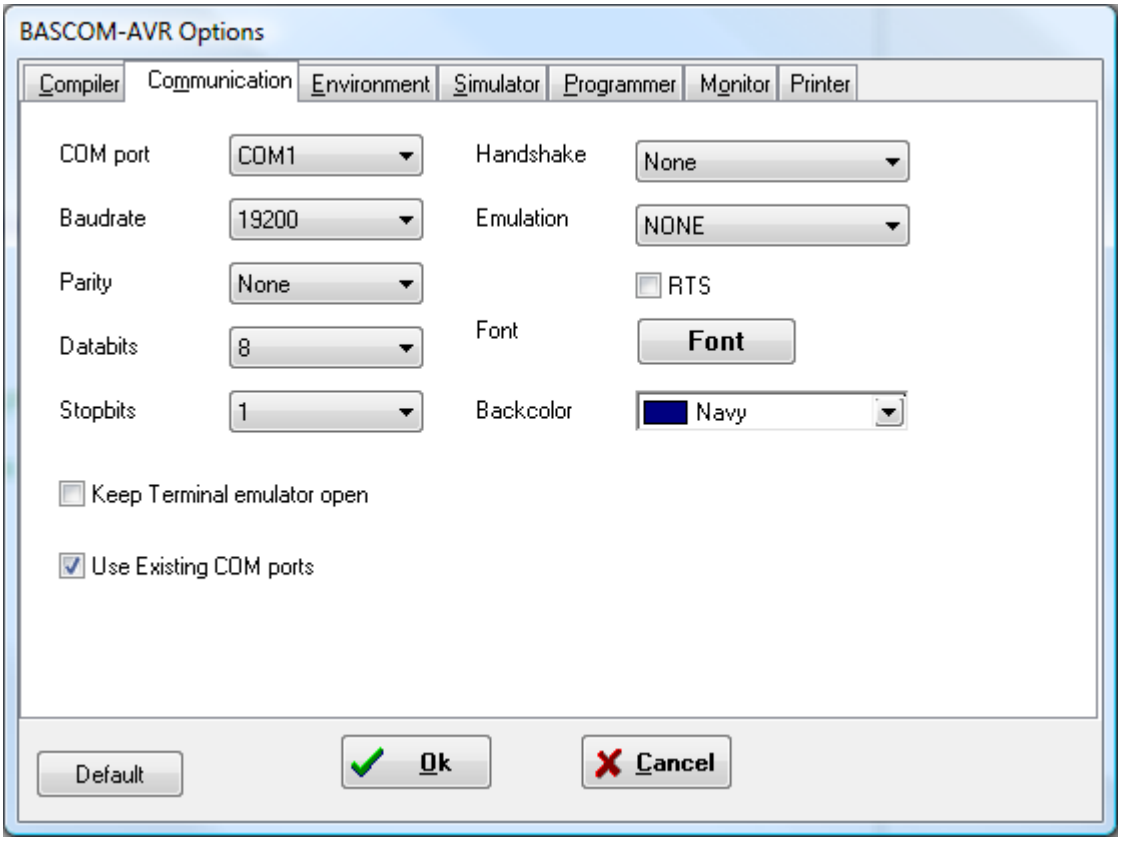
Item	Description
LCD type	The LCD display used.
Bus mode	The LCD can be operated in BUS mode or in PIN mode. In PIN mode, the data lines of the LCD are connected to the processor port pins. In BUS mode the data lines of the LCD are connected to the data lines of the BUS. Select 4 when you have only connect DB4-DB7. When the data mode is 'pin' , you should select 4.
Data mode	Select the mode in which the LCD is operating. In PIN mode, individual processor pins can be used to drive the LCD. In BUS mode, the external data bus is used to drive the LCD.
LCD address	In BUS mode you must specify which address will select the enable line of the LCD display. For the STK200, this is C000 = A14 + A15.
RS address	In BUS mode you must specify which address will select the RS line of the LCD display. For the STK200, this is 8000 = A15
Enable	For PIN mode, you must select the processor pin that is connected to the enable line of the LCD display.
RS	For PIN mode, you must select the processor pin that is

	connected to the RS line of the LCD display.
DB7-DB4	For PIN mode, you must select the processor pins that are connected to the upper four data lines of the LCD display.
Make upper 3 bits high in LCD designer	Some displays require that for setting custom characters, the upper 3 bits must be 1. Should not be used by default.

It is advised to use the CONFIG LCD command. This way the settings are stored in your source code and not in the separate CFG file.

3.57 Options Communication

With this option, you can modify the communication settings for the terminal emulator.



Item	Description
Comport	The communication port of your PC that you use for the terminal emulator.
Baud rate	The baud rate to use.
Parity	Parity, default None.
Data bits	Number of data bits, default 8.
Stop bits	Number of stop bits, default 1.
Handshake	The handshake used, default is none.
Emulation	Emulation used, default TTY and VT100.
Font	Font type and color used by the emulator.
Back color	Background color of the terminal emulator.

Keep TE open	This option will keep the terminal emulator COM port open when you close the window or move the focus away. Some serial programmers which close the COM port when they need to program, will not work in this mode when they use the same COM port.
Use Existing COM ports	When you select this option, you will get a list with the available COM ports only at places you can select a COM port. When you insert an USB virtual COM port, it will be added to list automatically. Removing virtual COM ports will also update the available COM port list. When you do not select this option you get a list with COM1-COM255.

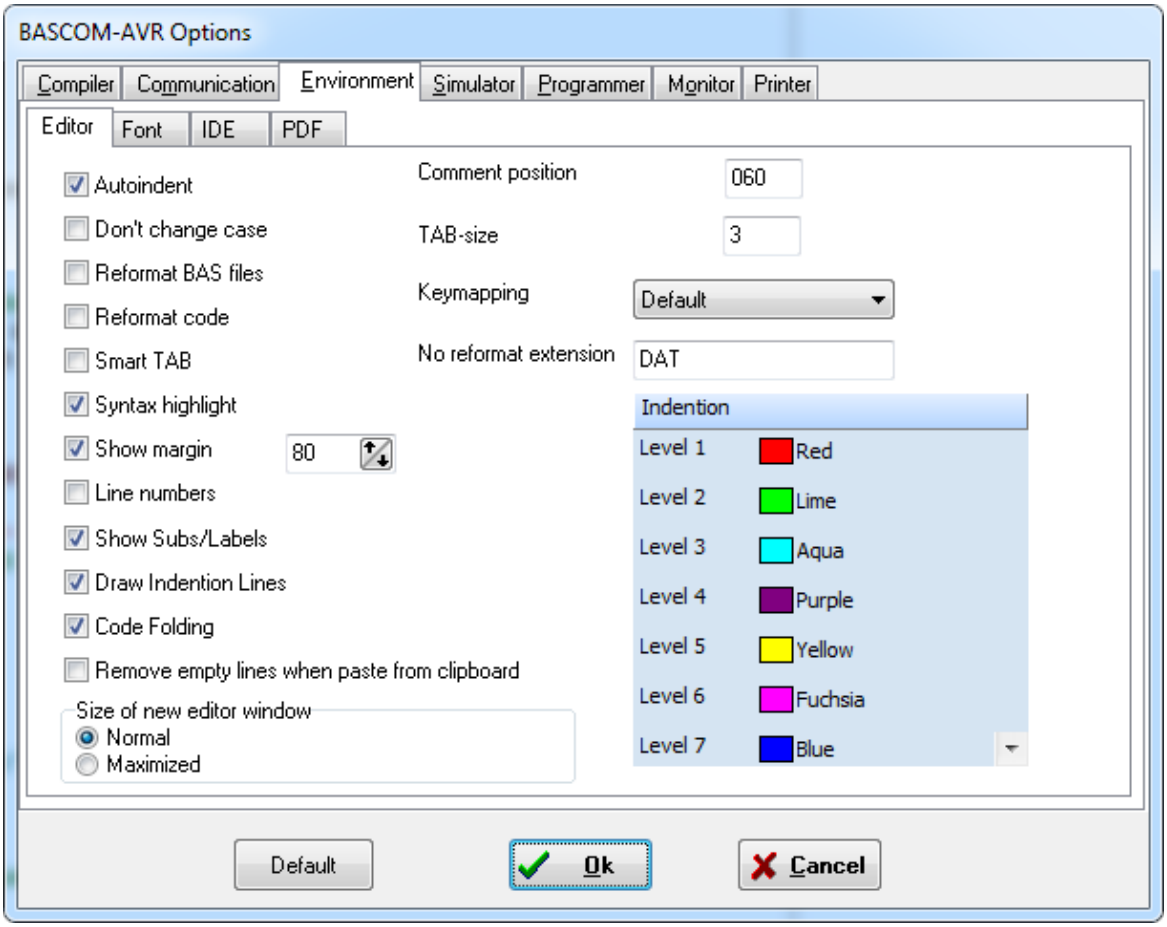
Note that the baud rate of the terminal emulator and the baud rate setting of the compiler options 146, must be the same in order to work correctly.

The reason why you can specify them both to be different is that you can use the terminal emulator for other purposes too.

3.58 Options Environment

The Environment TAB has a few TABS of it's own.

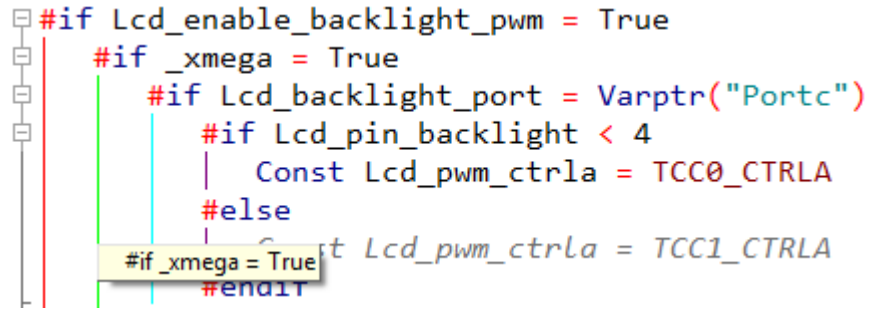
Options Environment Editor



OPTION	DESCRIPTION
Auto Indent	When you press return, the cursor is set to the next line at the current column position.
Don't change case	When set, the reformat won't change the case of the line after you have edited it. Default is that the text is reformatted so every word begins with upper case.
Reformat BAS files	Reformat files when loading them into the editor. All lines are reformatted so that multiple spaces are removed. This is only necessary when you are loading files that where created with another editor. Normally you won't need to set this option.
Reformat code	Reformat code when entered in the editor. The reformat option will change the modified line. For example a = a + 1 will be changed into : a = a + 1 . When you forget a string end marker **"**, one will be added, and endif will be changed into End If. And finally, ? is changed into Print.
Smart TAB	When set, a TAB will place the cursor to the column where text starts on the previous line.
Syntax highlighting	This options highlights BASCOM statements in the editor.
Show margin	Shows a margin on the right side of the editor. You can specify the position. By default this is 80.
Comment	The position of the comment. Comment is positioned to the right of your source code. Except when comment is first character of a line.
TAB-size	Number of spaces that are generated for a TAB.
Key mapping	Choose default, Classic, Brief or Epsilon.
No reformat extension	File extensions separated by a space that will not be reformatted when loaded. For example when DAT is entered, opening a DAT file can be done without that it is reformatted.
Size of new editor window	When a new editor window is created you can select how the windows will be created. Normal or Maximized (full window)
Line Numbers	Show line numbers in the margin.
Show Subs/Labels	This option will show sub modules/functions and labels at the top of the editor window in a drop down box. To get more screen space you can disable this option.
Remove Empty Lines	This option will remove empty lines when you paste data from the clipboard into the editor. When you copy & paste text from the help file (or any other source) you will find that windows inserts empty lines. This option will change two CR+LF into one.

Indention

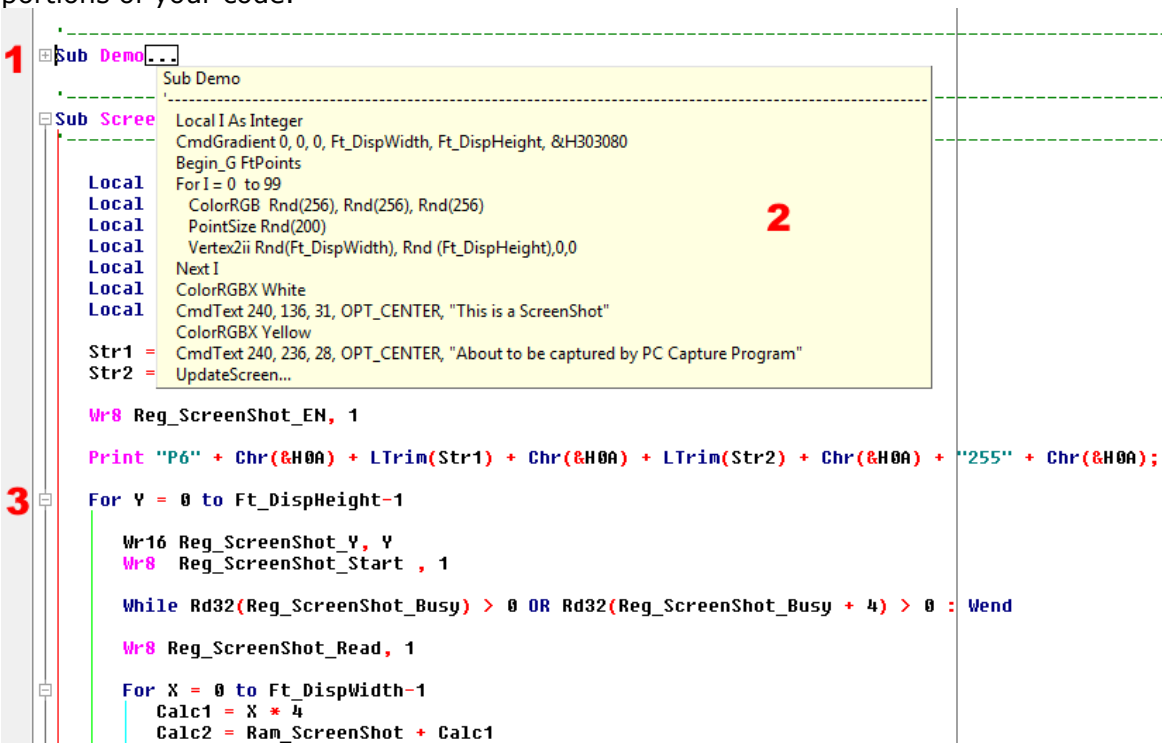
When indention lines are drawn, you can select the color of each level. The default is gray.
When you move the mouse over an indention line, the tooltip will show the start of the structure.



The sample above shows the info for the green indention line.
Obvious when the code fits into the screen, it is simple to see that the green line belongs to #IF _XMEGA. But when there is a lot of code in the editor, and you can not see all of the code, it can be a big help.

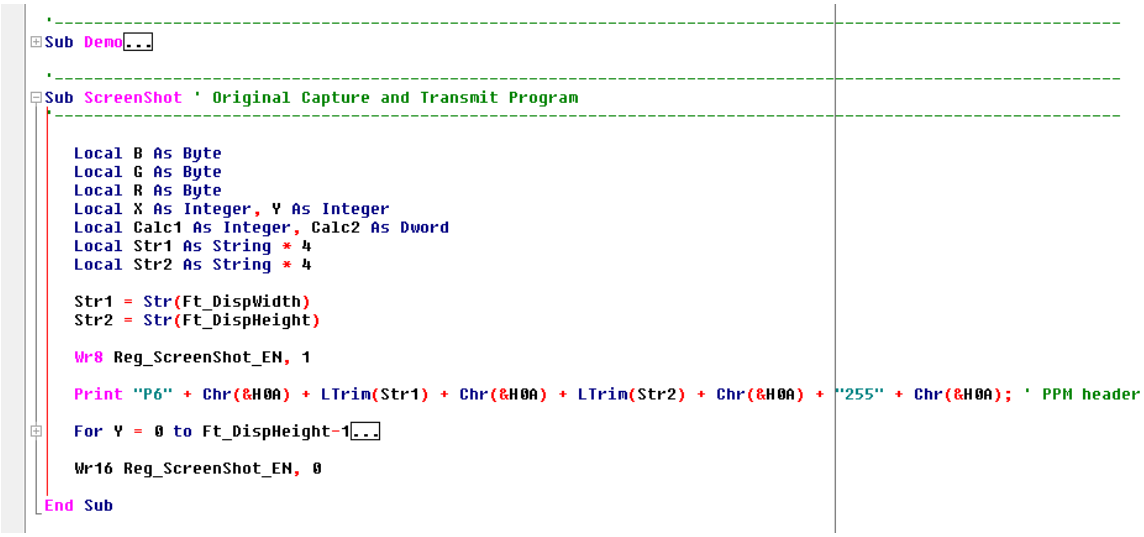
Code Folding

This option activates so called Code Folding. Code Folding allows you to hide/fold portions of your code.



The screen shot above shows :
1 - The Sub DEMO is folded. So you only see Sub Demo in your code. To indicate that the sub is folded there is a marker at the end of the line (3 dots)
Another indicator is the + sign. This means that the node is folded.
2 - When you put the cursor above the marker, you get a hint with the folded text/ code.
3 - The minus means that you can fold that node. When you click the - it will turn into a + and the code is folded.

This is how it looks when the node at (3) is clicked:



```
'-----------------------------------------------------------------------------------------------
Sub Demo...

'-----------------------------------------------------------------------------------------------
Sub ScreenShot ' Original Capture and Transmit Program
'-----------------------------------------------------------------------------------------------

  Local B As Byte
  Local G As Byte
  Local R As Byte
  Local X As Integer, Y As Integer
  Local Calc1 As Integer, Calc2 As Dword
  Local Str1 As String * 4
  Local Str2 As String * 4

  Str1 = Str(Ft_DispWidth)
  Str2 = Str(Ft_DispHeight)

  Wr8 Reg_ScreenShot_EN, 1

  Print "P6" + Chr(&H0A) + LTrim(Str1) + Chr(&H0A) + LTrim(Str2) + Chr(&H0A) + "255" + Chr(&H0A); ' PPM header

  For Y = 0 to Ft_DispHeight-1...

  Wr16 Reg_ScreenShot_EN, 0

End Sub
```

When folding code, all child code (all levels under the node) will be folded/unfolded as well.
A node is a point in your code that is part of a structure like sub/end sub , function/ end function, for/next, do/loop, while/wend. Remarks , Dim, Const and Config can also be folded.

When you press F11, the current SUB or FUNCTION will be folded/unfolded. The Editor menu also has options to fold/unfold all code.

When you want to fold code that normally would not fold you can use a trick. When you define a constant you can use this for code folding:
Const fold=1
#IF Fold
  print "fold this"
  print "end this too"
#ENDIF

Conditional compilation is used to fold the code.

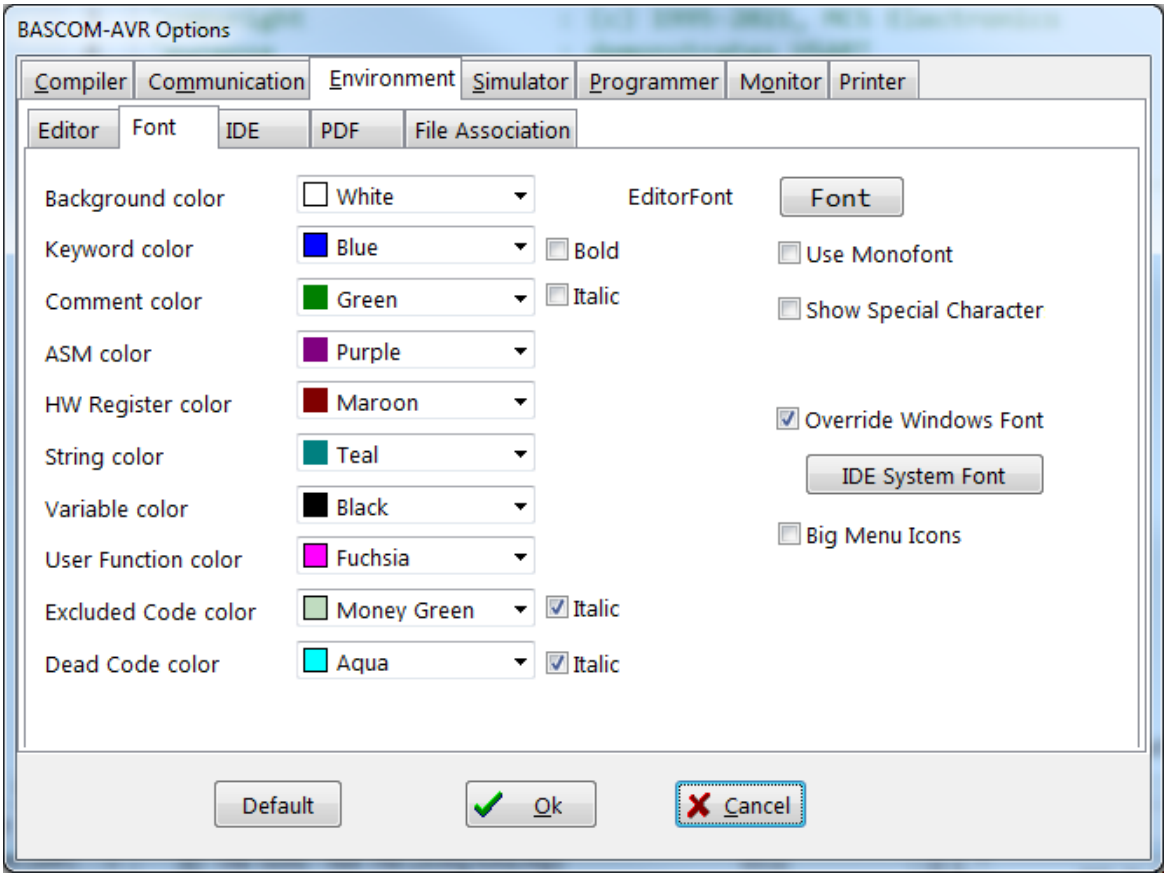
Draw Indention Lines

This option will draw vertical indent lines for structures.

```
51 do
52    Print #1 , "test xmega"
53
54    For J = 0 To 120 Step 1                                  'notice that we scan
55       I2cstart #4                                           'send start
56       I2cwbyte J , #4                                       'send value of J
57       If Err = 0 Then                                         ' no errors
58          Print #1 , "FOUND : " ; Hex(j)
59          if j.0 = 0 then                                    'ONLY if R/W bit is n
60             I2cwbyte 100 , #4                               'just write to values
61             I2cwbyte 101 , #4
62          else                                               'read
63             I2crbyte b , Ack , #4 : print #1 , "GOT : " ; b 'read 2 bytes
64             I2crbyte b , nAck , #4 : print #1 , "GOT : " ; b
65          end if
66       End If
67       I2cstop #4                                            'done
68    Next
69    waitms 2000                                              'wait some time
70 loop
```

Drawing indention lines may result in slower screen painting. Errors in your code might result in wrong painting of the lines.

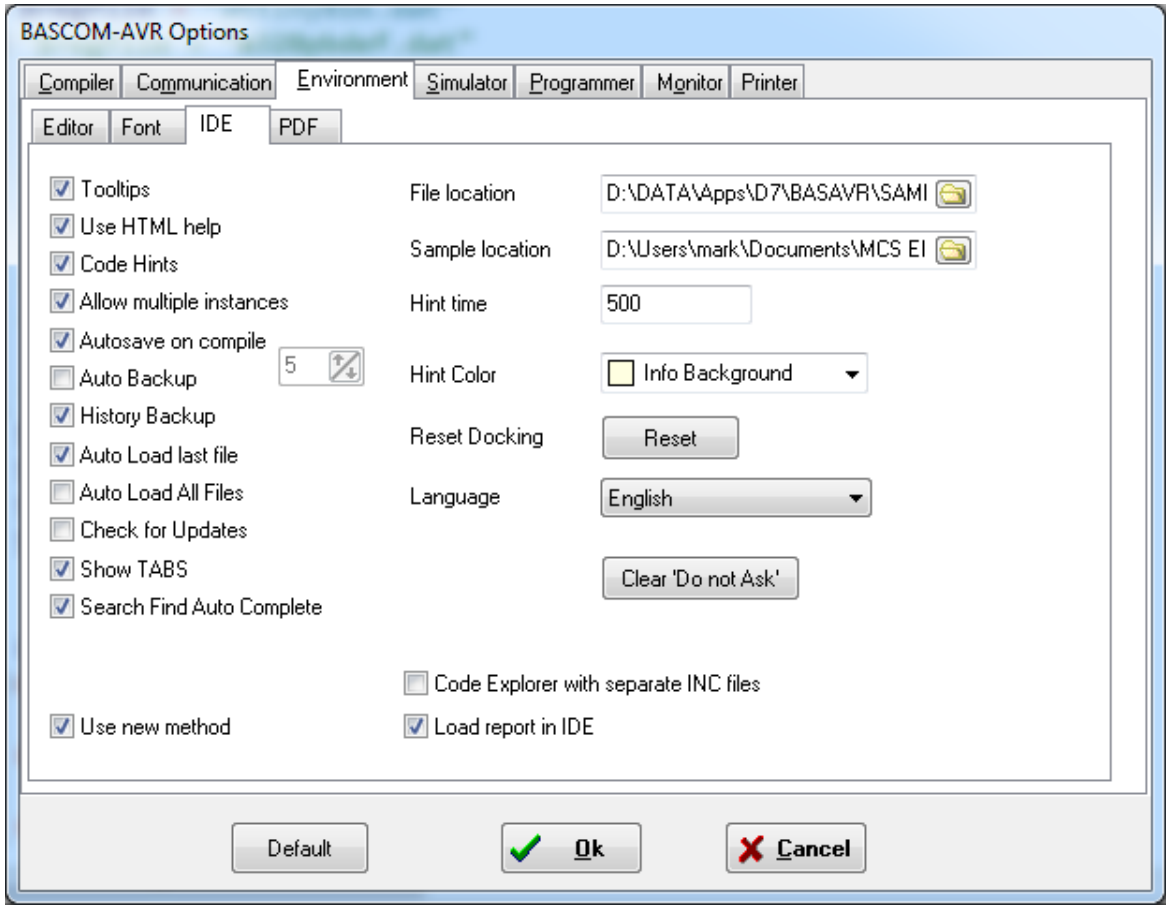
Options Environment Font



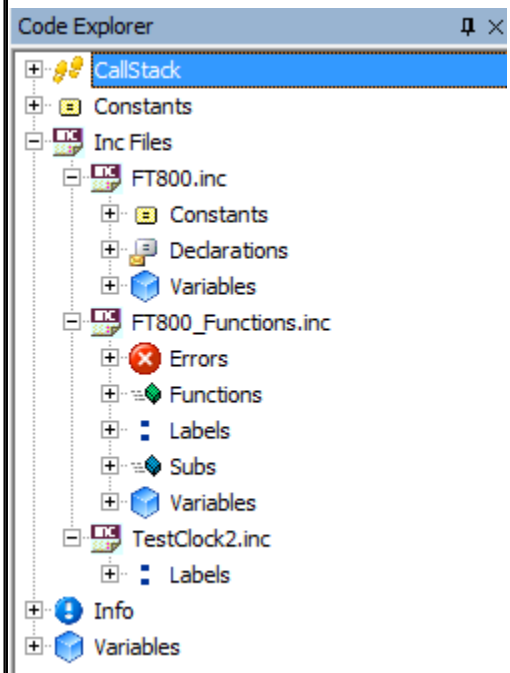
OPTION	DESCRIPTION
Background color	The background color of the editor window. Choose a color that is the same as your background. In a white room, using white would be best for your eyes.
Keyword color	The color of the reserved words. Default Navy. The keywords can be displayed in **bold** too.
Comment color	The color of comment. Default green. Comment can be shown in *Italic* too.
ASM color	Color to use for ASM statements. Default purple.
HW registers color	The color to use for the hardware registers/ports. Default maroon.
String color	The color to use for string constants : "test"
Variable color	The color to use for variables. Default is black.
User Function Color	The color to use for user SUBS and FUNCTIONS. The default is fuchsia .
Excluded Code Color	The color to use for Excluded code (code not compiled because of conditional compilation).

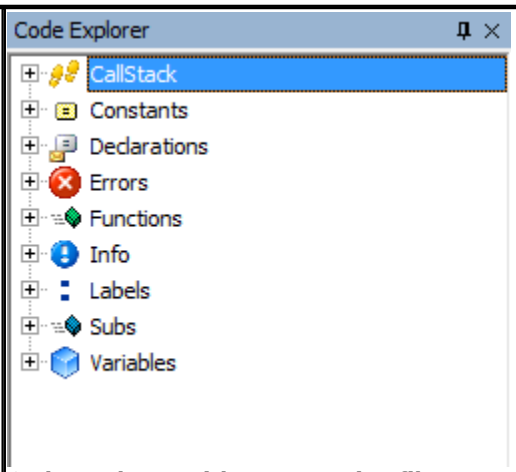
OPTION	DESCRIPTION
Dead Code Color	The color to use for Dead Code. (code that is not used)
Editor font	Click on this button to select another font for the editor window. A good choice is Fixedsys.
Use Monofont	When checked and the selected font is a monofont the font will be drawn with monofont properties. Otherwise it will be shown as non monofont. Use this for compatibility with old bascom versions and fonts.
Show Hidden Characters	This option will show special characters in the editor. Special characters are characters such as CR and LF. And all characters with an ASCII value above 127. You can use this option to find odd characters in your code which could result in compilation errors.
Override Windows Font	This setting will override the Windows default font. You can select a font by clicking the IDE system font button. It is recommended to select a font like SEGOU UI, normal, 10 points. This font is used by all forms of the IDE. It is independent of the editor font.
Big Menu Icons	This option will use bigger icons for the IDE. The normal default size is 16x16. The bigger size is 32x32. This will give better images when you have a high resolution monitor setting.

Options Environment IDE



OPTION	DESCRIPTION
Tool tips	Show tool tips when hovering over form elements such as buttons.
File location	Click to select a directory where your program files are stored. By default Windows will use the **My Documents** path.
Sample Location	Click to select the folder where the SAMPLE files are located. They are either stored in a sub folder of the application, or in a folder under the Documents\MCS Electronics\BASCOM-AVR\samples folder
Use HTML Help	Chose between old help and CHM Help. CHM is the preferred help file. Since HLP is not supported under Vista, it is advised to switch to CHM/ HTML Help. The HLP file is not distributed but using the UpdateWiz you can still download the HLP file.
Code hints	Select this option to enable code hints. You can get code hints after you have typed a statement that is recognized as a valid statement or function.
Hint Time	The delay time in mS before a code hint will be shown.
Hint Color	The background color of the hints.
Allow multiple Instances	Select this option when you want to run multiple instances of BASCOM. When not enabled, running a second copy will terminate the first instance.
Auto save on compile	The code is always saved when you compile. When you select this option, the code is saved under the same name. When this option is not selected, you will be prompted for a new filename.
Auto backup	Check this option to make periodic backups. When checked you can specify the backup time in minutes. The file will also be saved when you press the compiler button.
History Backup	This option creates a history backup of the source file each time you save it. When you Compile code, the active source will be saved too before compilation and hence it will create a history file as well. The history file is a version of the code saved in the HISTORY folder. This folder is located in the same folder as the main project. The file will be named <FILE>~yymmdd hhnnss.hst Where <FILE> is the original file name, and yymmdd is the date and hhNNss is the time.
Auto load last file	When enabled, this option will load the last file that was open into the editor, when you start BASCOM.
Auto load all files	When enabled, this option will load all files that were open when you closed BASCOM.
Check for updates	Select this option to check for updates when the IDE is started.
Show TABS	This option will enable/disable the TAB for multiple windows. While the TAB is convenient to switch between windows, it will also consume screen space. You can disable this option to get more screen space.
Reset docking	This will reset the dockable windows to the default position.
Search Find Auto Complete	This option can enable/disable the auto completion in the Find dialog. When it is active and you type some text, based on historical input, the text will be completed. This is not always desired and can be disabled.
Language	This will set the language in the main menu to the selected language. Not all listed languages are supported/translated yet.
Clear Do not Ask	Some messages have a 'do not ask again' option. To reset this and thus show the messages, you can click this button.
Use New Parser	When compiling a project, the main file is searched for some settings like $regfile, $hwstack, $swstack and $framesize. This information is passed

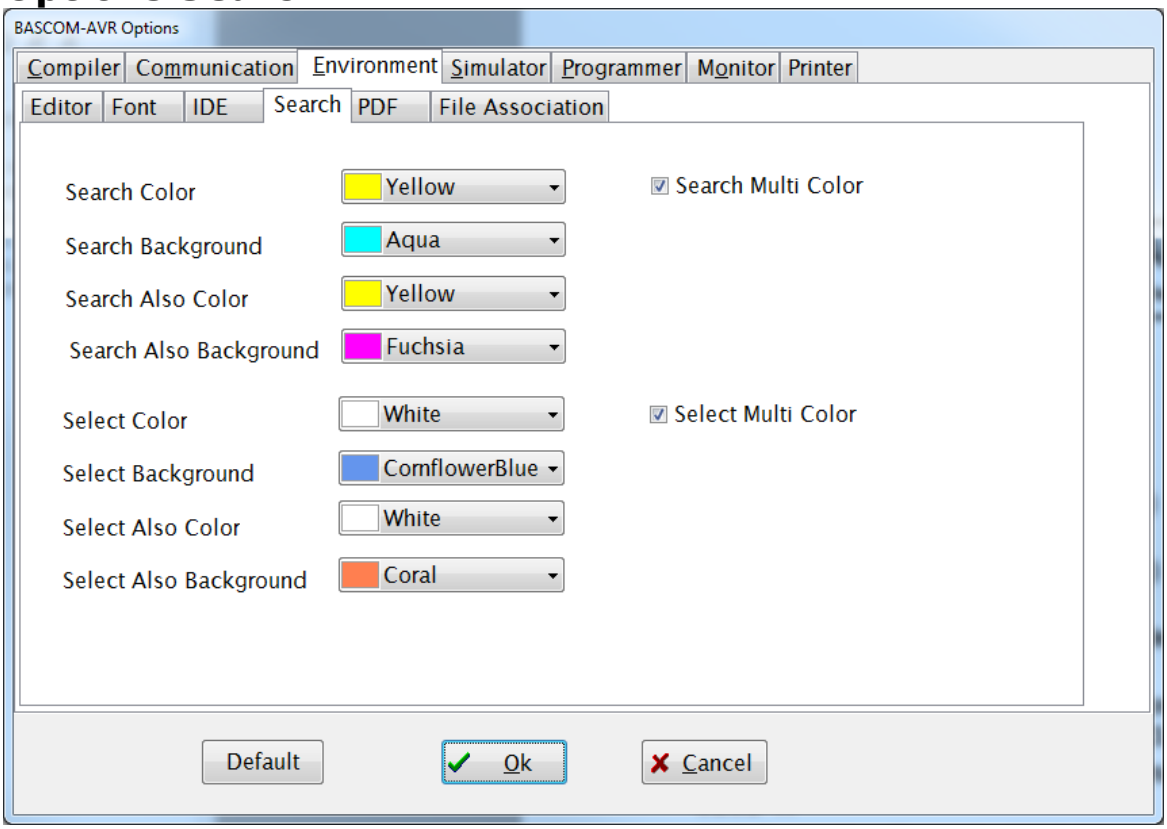
	to the compiler DLL. This search is fast but simple : it will not work correct when using directives such as : #IF someConditon $regfile = "m88def.dat" #ELSE $regfile = "m2650def.dat" #ENDIF The parser used for the code explorer is capable to get the information but requires more time because it will parse the entire project. So you have the option to choose the old method or the new method. In version 2087 the new parser is made default. It is good practice to start your project with the required info : $regfile = "yourmicro.dat" $hwstack=32 ' values shown as sample $swstack=32 $framesize=32 We recommend that you always use 'New Parser' since the old method will be disregarded in a future update. This does mean that your main project code always need to contain the most important settings : $REGFILE, $HWSTACK, $SWSTACK and $FRAMESIZE. These settings override the optional project configuration settings. A future version will not use the configuration file settings since it is best that these settings are stored in the code. This setting is also required for the $PROGRAMMER 690 directive.
Code Explorer with separate INC files	The Code explorer will put all elements in one tree without file names. Setting this option however will create a tree of elements with all file names under a branch named 'Inc Files'. 



Code explorer with separate inc files

Code explorer without separate inc files

Options Search



Using the Search and select options you can customize the search and select colors.
You can also enable the option : Search Multi Color and Select Multi Color.
When search multi color is enabled all other matches will be highlighted too.

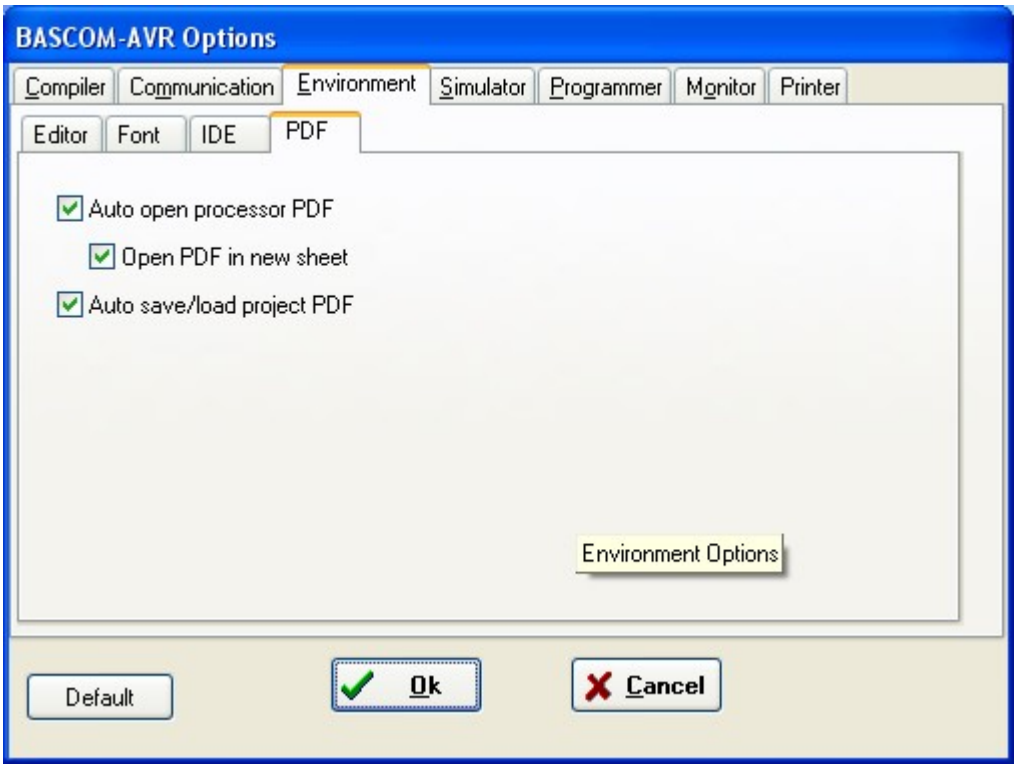
```
Const Serial_number_byte0 = &H0E
Const Serial_number_byte1 = &H0F
Const Serial_number_byte2 = &H10
Const Serial_number_byte3 = &H11
Const Serial_number_byte4 = &H12
Const Serial_number_byte5 = &H13
Const Serial_number_byte6 = &H14
Const Serial_number_byte7 = &H15
Const Serial_number_byte8 = &H16
Const Serial_number_byte9 = &H17
```

This sample shows what happens when you search for 'serial'. Note that that text is gray because of the 'Dead Code' option.

```
'some constants for the signature row
Const Device_signature_byte1 = 0
Const Device_signature_byte2 = 2
Const Device_signature_byte3 = 4
```

When you enable Select Multi Color, all text you select using the keyboard, mouse, or double click, will be highlighted the usual way. All other occurrences will be highlighted too.
This can be confusing.
The multi select coloring will do only what the name suggests : it will color the selecting text and all other occurrences. But when you perform an operation like replace, it will only be performed on the active selected text.

Options Environment PDF



OPTION	DESCRIPTION
Auto open processor PDF	This option will automatic load the PDF of the selected micro processor in the PDF viewer. The $REGFILE value determines which data sheet is loaded. The PDF must exist otherwise it can not be loaded.
Open PDF in new sheet	Every time you change the value of the $REGFILE the processor PDF can be shown in the same sheet, or a new sheet can be shown with the PDF. A good option in case your project uses multiple processors.
Auto save/load project PDF	Load all PDF's when the project is opened that were loaded when the project was closed.

Custom ShortCuts

When you want to define your own short cuts you can create an ini file named **shortcuts.ini**.
This ini file is just a text file you can create with notepad. Store this file in the BASCOM application folder.
To enable the user shortcuts you need to make an option named : ENABLED with the value -1.

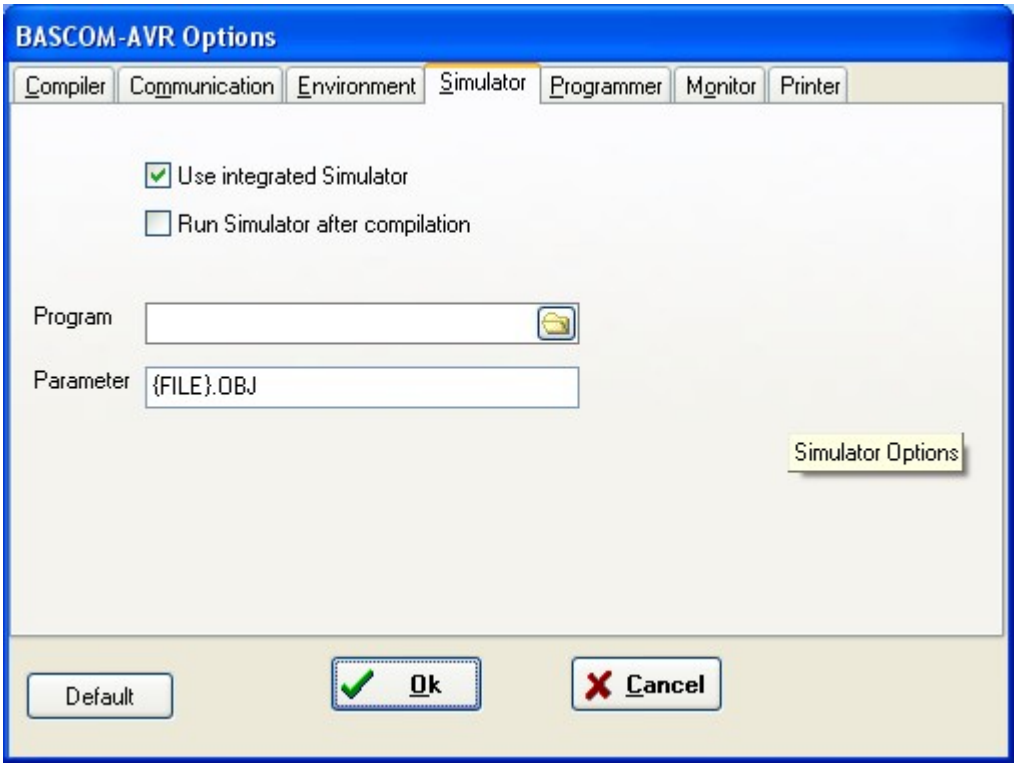
This is an example of the default values
[MENU]
enabled=-1
FileNew=CTRL+N
FileSave=CTRL+S
FilePrint=CTRL+P
EditUndo=CTRL+Z
EditRedo=SHIFT+CTRL+Z
EditCut=CTRL+X
EditCopy=CTRL+C

EditPaste=CTRL+V
EditFind=CTRL+F
EditFindNext=F3
EditReplace=CTRL+R
EditGoto=CTRL+G
EditIndent=SHIFT+CTRL+I
EditUnIndent=SHIFT+CTRL+U
EditUnremarkBlock=CTRL+M
EditProperIndent=CTRL+ALT+P
Compile=F7
SyntaxCheck=CTRL+F7
ProgramShowResult=CTRL+W
ProgramSimulate=F2
ProgramSendToChip=F4
ProgramResetChip=SHIFT+F4
TerminalEmulator=CTRL+T
LCD_Designer=CTRL+L
LibManager=CTRL+I
BatchCompile=CTRL+B
ShowDevMng=CTRL+D

To turn the custom shortcuts off set the ENABLED to 0.

3.59 Options Simulator

With this option you can modify the simulator settings.

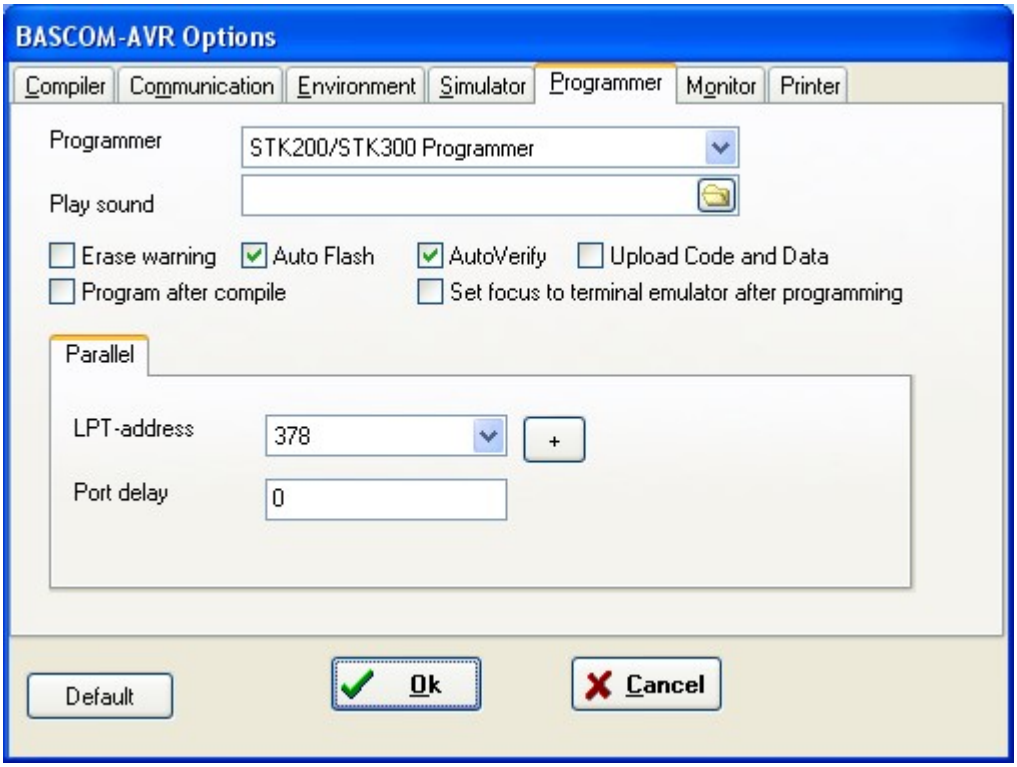


OPTION	DESCRIPTION

Use integrated simulator	Set this option to use BASCOM's simulator. You can also use AVR Studio by clearing this option.
Run simulator after compilation	Run the selected simulator after a successful compilation.
Program	The path with the program name of the external simulator.
Parameter	The parameter to pass to the program. {FILE}.OBJ will supply the name of the current program with the extension . OBJ to the simulator.

3.60 Options Programmer

With this option you can modify the programmer settings.



OPTION	DESCRIPTION
Programmer	Select one from the list.
Play sound	Name of a WAV file to be played when programming is finished. Press the directory button to select a file.
Erase Warning	Set this option when you want a confirmation when the chip is erased.
Auto flash	Some programmers support auto flash. Pressing F4 will program the chip without showing the programmer window.
Auto verify	Some programmers support verifying. The chip content will be verified after programming.
Upload code and data	Set this option to program both the FLASH memory and the EEPROM memory
Program after compile	When compilation is successful, the chip will be programmed

Set focus to terminal emulator	When the chip is programmed, the terminal emulator will be shown
	Parallel printer port programmers
LPT address	Port address of the LPT that is connected to the programmer.
Port delay	An optional delay in uS. It should be 0. But on some systems a delay might be needed.
	Serial port programmer
COM port	The com port the programmer is connected to.
STK500 EXE	The path of stk500.exe. This is the full file location to the files stk500.exe that comes with the STK500.
USB	For mkII and other Atmel USB programmers you can enter the serial number here. Or you can look it up from the list.
Baud	Some serial programmers have an optional baud rate you can chose. The usual values are supported. When you want special custom baud rate you can replace the baud rate by creating a file named : **progbaud.lst** In this text file you can place all the baud rates. For example : 123 300 600 1200 2400 4800 9600 9610 19200 38400 57600 128000 256000 115200 500000 Then save the file. The file must reside in the bascom-avr application folder. The file is loaded when you run bascom. It will depend on your PC hardware/driver if the baud you use will actually work.
	Other
Use HEX	Select when a HEX file must be sent instead of the bin file.
Program	The program to execute. This is your programmer software.
Parameter	The optional parameter that the program might need. Use {FILE} to insert the binary filename(file.bin) and {EEPROM} to insert the filename of the generated EEP file. When 'Use Hex' is checked the filename (file.hex) will be inserted for {FILE}. In all cases a binary file will be inserted for {EEPROM} with the extension .EEP Use {CHIP} to insert the official device name of the chip. The

	device name is required by some programmers.

See Also

Supported programmers 164

3.60.1 Supported Programmers

BASCOM supports the following programmers

AVR ICP910 based on the AVR910.ASM application note 175

STK200 ISP programmer from Atmel 175

The PG302 programmer from Iguana Labs 166

The simple cable programmer from Sample Electronics. 166

KITSRUS KIT122 Programmer 167

MCS Universal Interface Programmer 168

The MCS Universal Interface supports a number of programmers as well. In fact it is possible to support most parallel printer port programmers.

STK500 programmer and Extended STK500 programmer. 170

Lawicel BootLoader 174

USB-ISP Programmer 175

MCS Bootloader 184

PROGGY 186

MyAVR/MK2/AVR910 programmer 174

FLIP 187

USBprog Programmer / AVR ISP mkII 190 (AVRISP)

KamProg for AVR 190

USBASP 193

STK600 194

ARDUINO 197

BIPOM MINI-MAX/C 200

mySmartUSB light 200

UPDI Programmer 202

MCS EDBG Programmer 207

3.60.1.1 ISP programmer

BASCOM supports the STK200 and STK200+ and STK300 ISP programmer from Atmel.

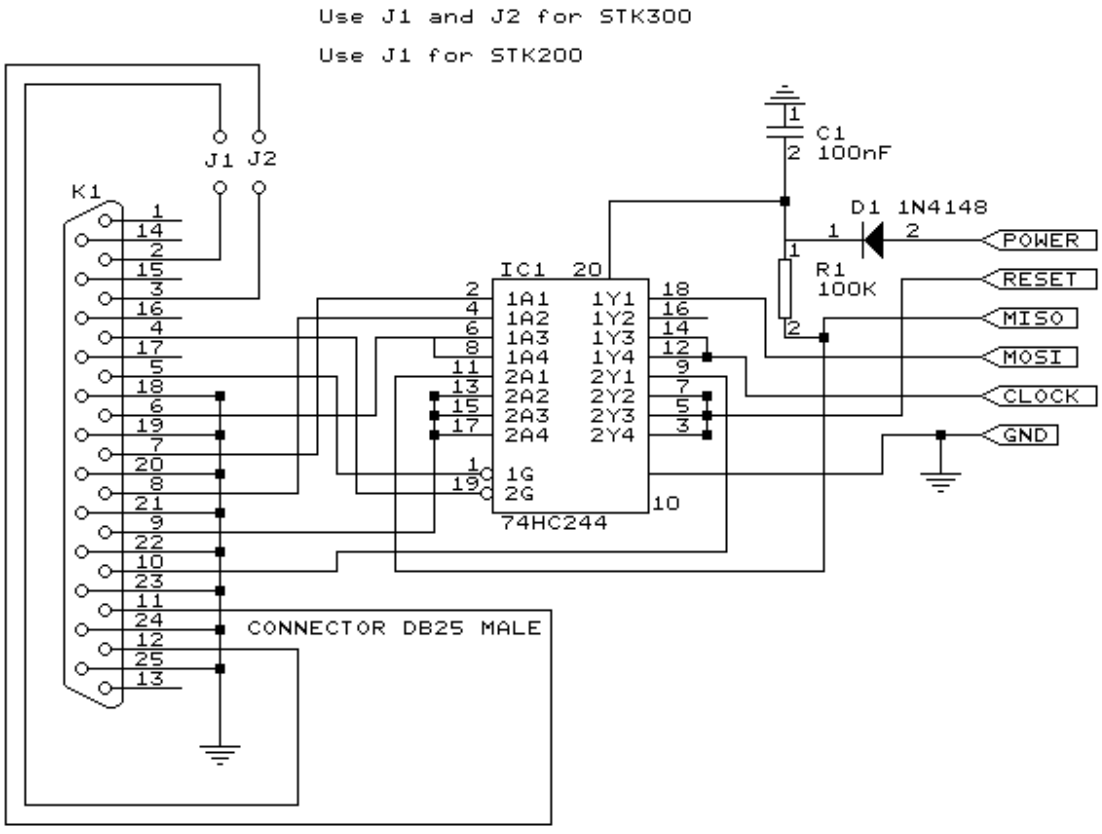
This is a very reliable parallel printer port programmer.
The STK200 ISP programmer is included in the STK200 starter kit.
Most programs were tested with the STK200.

For those who don't have this kit and the programmer the following schematic shows how to make your own programmer:

The dongle has a chip with no identification but since the schematic is all over the web, it is included. MCS also sells a STK200 compatible programmer.

Here is a tip received from a user :

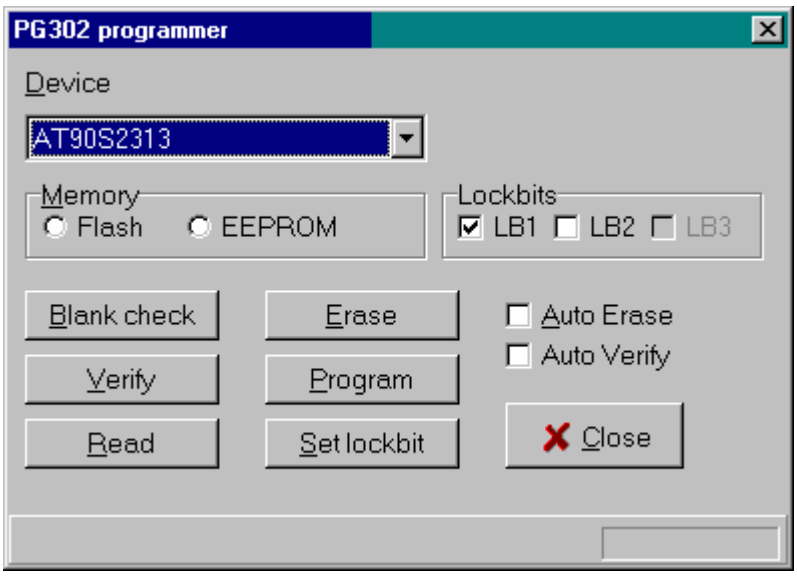
If the parallel port is disconnected from the interface and left floating, the '244 latch outputs will waver, causing your micro controller to randomly reset during operation. The simple addition of a 100K pull-up resistor between pin 1 and 20 of the latch, and another between pin 19 and 20, will eliminate this problem. You'll then have HIGH-Z on the latch outputs when the cable is disconnected (as well as when it's connected and you aren't programming), so you can use the MOSI etc. pins for I/O.



Since parallel printer ports do not exist in new equipment you better use an USB programmer.

3.60.1.2 PG302 programmer

The PG302 is a serial programmer. It works and looks exactly as the original PG302 software.



Select the programmer from The Option Programmer menu or right click on the  button to show the Option Programmer 162 menu

THIS PROGRAMMED IS MARKED FOR REMOVAL. Send a note to support if you use it.

3.60.1.3 Sample Electronics cable programmer

Sample Electronics submitted the simple cable programmer.

They produce professional programmers too. This simple programmer you can make yourself within 10 minutes.
What you need is a DB25 centronics male connector, a flat cable and a connector that can be connected to the target MCU board.

The connections to make are as following:

DB25 pin	Target MCU pin (AT90S8535)	Target MCU M103/M128	Target MCU pin 8515	DT104
2, D0	MOSI, pin 6	PE.0, 2	MOSI, 6	J5, pin 4
4, D2	RESET, pin 9	RESET, 20	RESET, 9	J5, pin 8
5, D3	CLOCK, pin 8	PB.1,11	CLOCK, 8	J5, pin 6
11, BUSY	MISO, pin 7	PE.1, 3	MISO, 7	J5, pin 5
18-25,GND	GROUND	GROUND	GND,20	J5, pin 1

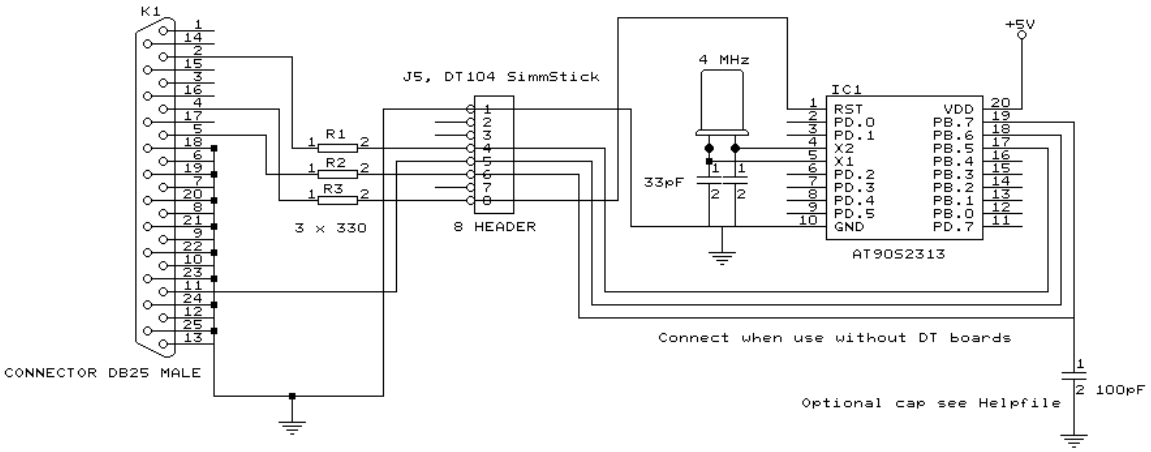
The MCU pin numbers are shown for an 8535! And 8515
Note that 18-25 means pins 18,19,20,21,22,23,24 and 25

You can use a small resistor of 100-220 ohm in series with the D0, D2 and D3 line in order not to short circuit your LPT port in the event the MCU pins are high.

It was tested without these resistors and no problems occurred.

Tip : when testing programmers etc. on the LPT it is best to buy an I/O card for your PC that has a LPT port. This way you don't destroy your LPT port that is on the motherboard in the event you make a mistake!

The following picture shows the connections to make. Both a setup for the DT104 and stand-alone PCB are shown.



I received the following useful information:

I have been having spurious success with the simple cable programmer from Sample Electronics for the AVR series.
After resorting to hooking up the CRO I have figured it out (I think). When trying to identify the chip, no response on the MISO pin indicates that the Programming Enable command has not been correctly received by the target.

The SCK line Mark/Space times were okay but it looked a bit sad with a slow rise time but a rapid fall time. So I initially tried to improve the rise
time with a pull-up. No change ie still could not identify chip. I was about to add some buffers when I came across an Atmel app note for their serial programmer "During this first phase of the programming cycle, keeping the SCK line free from pulses is critical, as pulses will cause the target AVR to loose synchronization with the programmer. When synchronization is lost, the only means of regaining synchronization is to release the RESET line for more than 100ms."

I have added a 100pF cap from SCK to GND and works first time every time now. The SCK rise time is still sad but there must have been enough noise to corrupt the initial command despite using a 600mm shielded cable.

3.60.1.4 KITSRUS Programmer

The K122 is a KIT from KITSRUS. (www.kitsrus.com)

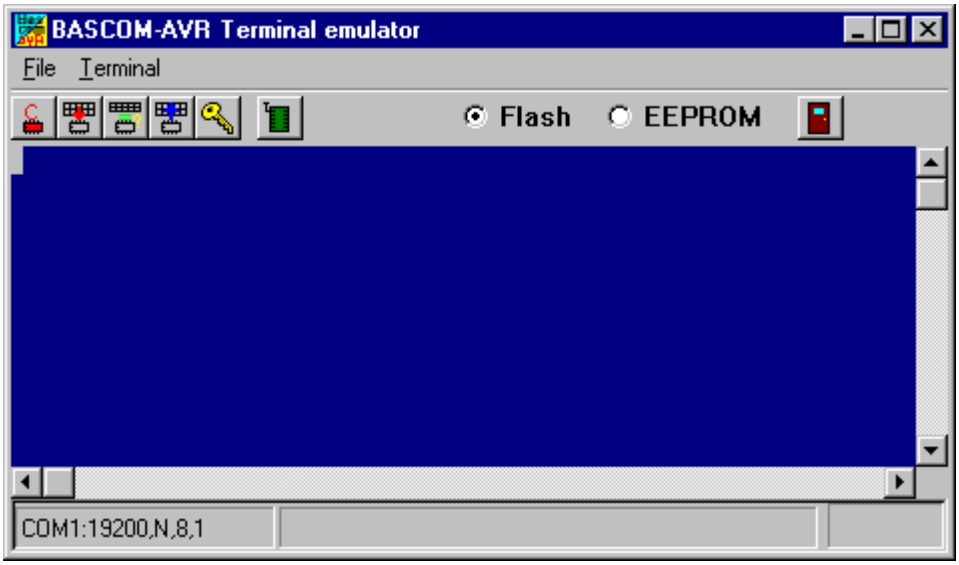
The programmer supports the most popular 20 and 40 pins AVR chips.

On the Programmer Options tab you must select this programmer and the COM port it

is connected to.

On the Monitor Options tab you must specify the upload speed of 9600, Monitor delay of 1 and Prefix delay 1.

When you press the Program button the Terminal Emulator screen will pop up:



A special toolbar is now visible.
You must press the Program enable button to enable the programmer.
When you enable the programmer the right baud rate will be set.
When you are finished you must press the Enable button again to disable it.
This way you can have a micro connected to your COM port that works with a different BAUD rate.
There is an option to select between FLASH and EEPROM.
The prompt will show the current mode which is set to FLASH by default.

The buttons on the toolbar allow you to :
ERASE, PROGRAM, VERIFY, DUMP and set the LOCK BITS.
When DUMP is selected you will be asked for a file name.
When the DUMP is ready you must CLOSE the LOGFILE where the data is stored. This can be done to select the CLOSE LOGFILE option form the menu.

THIS PROGRAMMED IS MARKED FOR REMOVAL. Send a note to support if you use it.

3.60.1.5 MCS Universal Interface Programmer

The MCS Universal Interface programmer allows you to customize the pins that are used for the ISP interface. The file prog.settings stores the various interfaces.

The content :

```
;how to use this file to add support for other programmers

;first create a section like [newprog]

; then enter the entries:

; BASE= $hexaddress
```

```
; MOSI= address in form of BASE[+offset] , bit [,inverted]

; CLOCK= same as MOSI

; RESET=same as MOSI

; MISO=same as MOSI

; The bit is a numer that must be written to set the bit

; for example 128 to set bit 7

; Optional is ,INVERTED to specify that inverse logic is used

; When 128 is specified for the bit, NOT 128 will be written(127)


[FUTURELEC]

;tested and ok
BASE=$378
MOSI=BASE+2,1,inverted
CLOCK=BASE,1
RESET=BASE,2
MISO=BASE+1,64

[sample]
;tested and ok
BASE=$378
MOSI=BASE,1
CLOCK=BASE,8
RESET=BASE,4
MISO=BASE+1,128,INVERTED


[stk200]
;tested and ok
BASE=$378
MOSI=BASE,32
CLOCK=BASE,16
RESET=BASE,128
MISO=BASE+1,64
```

Four programmers are supported : Futurelec, Sample and STK200/STK300 and WinAVR/ SP12.
To add your own programmer open the file with notepad and add a new section name. For the example I will use stk200 that is already in the file.

[stk200]
The LPT base address must be specified. For LPT1 this is in most cases $378. $ means hexadecimal.

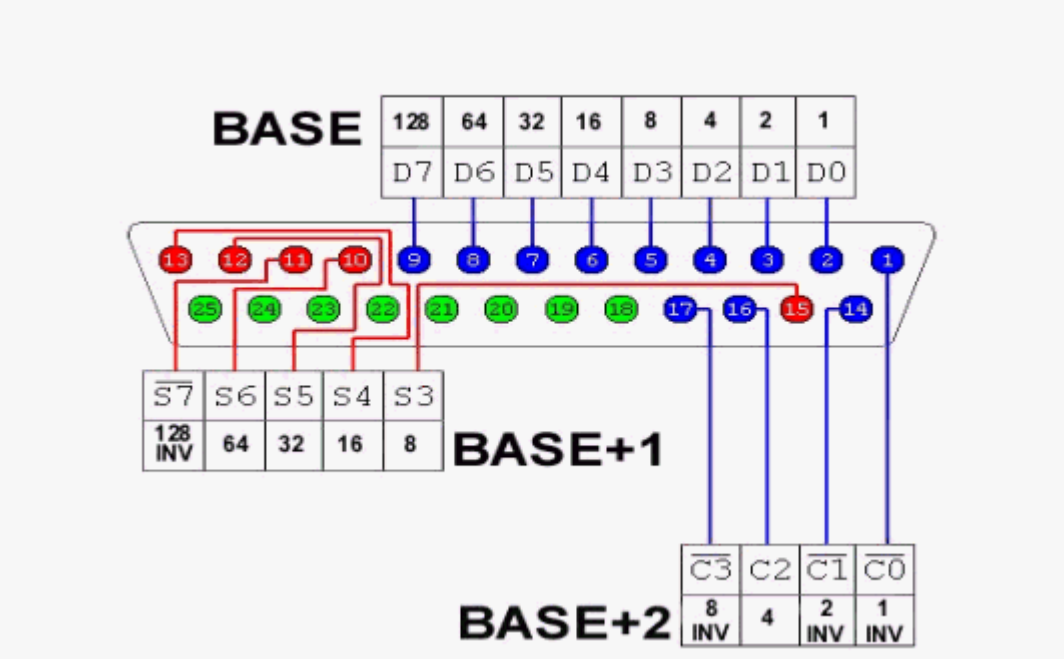
The pins that are needed are MOSI, CLOCK, RESET and MISO.
Add the pin name MOSI =

After the pin name add the address of the register. For the STK200 the data lines are

used so BASE must be specified. After the address of the register, specify the bit number value to set the pin high. Pin 0 will be 1, pin 1 would be 2, pin 2 would be 4 etc. D5 is used for the stk so we specify 32.

When the value is set by writing a logic 0, also specify, INVERTED.
After you have specified all pins, save the file and restart BASCOM.
Select the Universal Programmer Interface and select the entry you created.
After you have selected an entry save your settings and exit BASCOM. At the next startup of BASCOM, the settings will be used.

The following picture shows the LPT connector and the relation of the pins to the LPT registers.



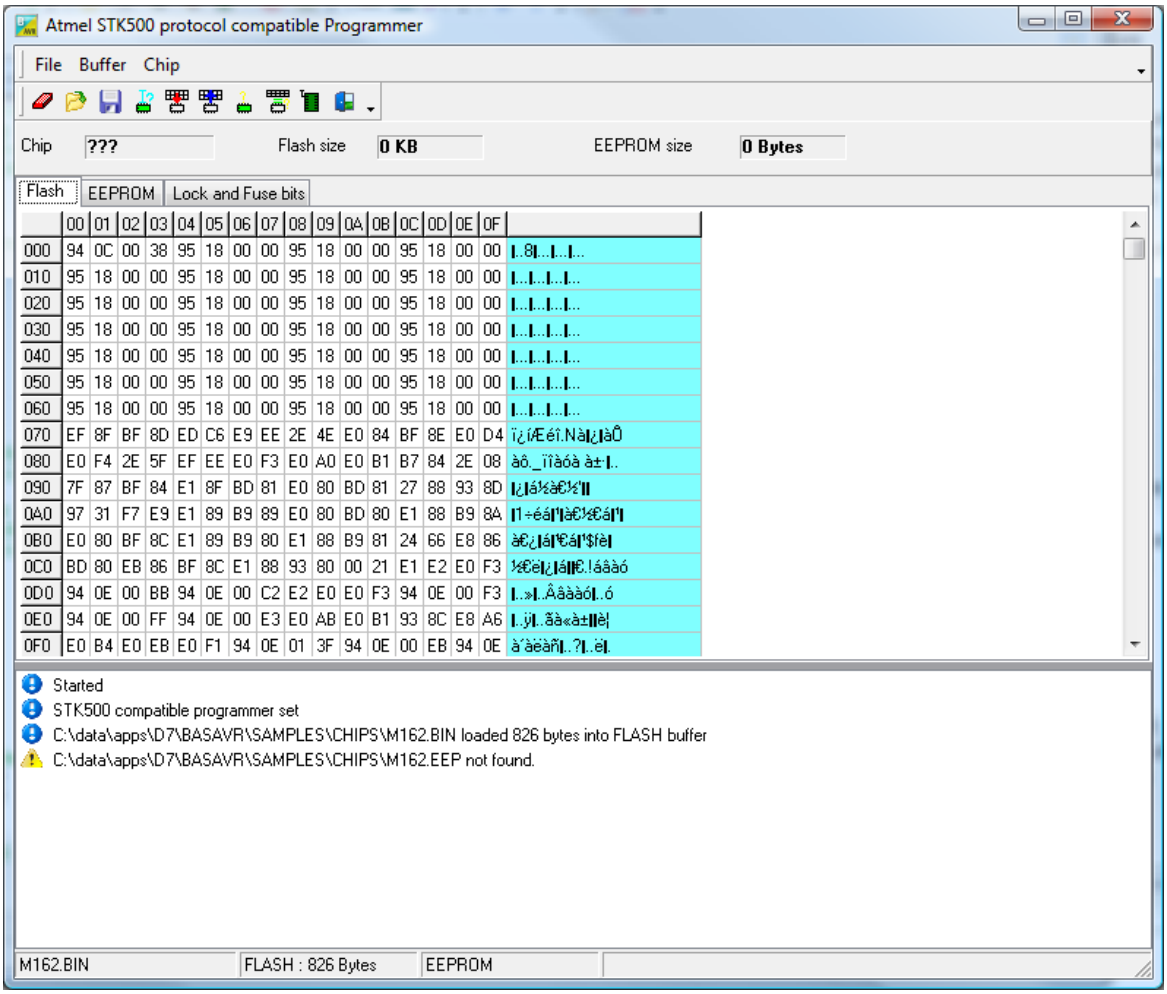
Always add your entry to the bottom of the file and email the settings to support@mcselec.com so it can be added to BASCOM.

3.60.1.6 STK500 Programmer

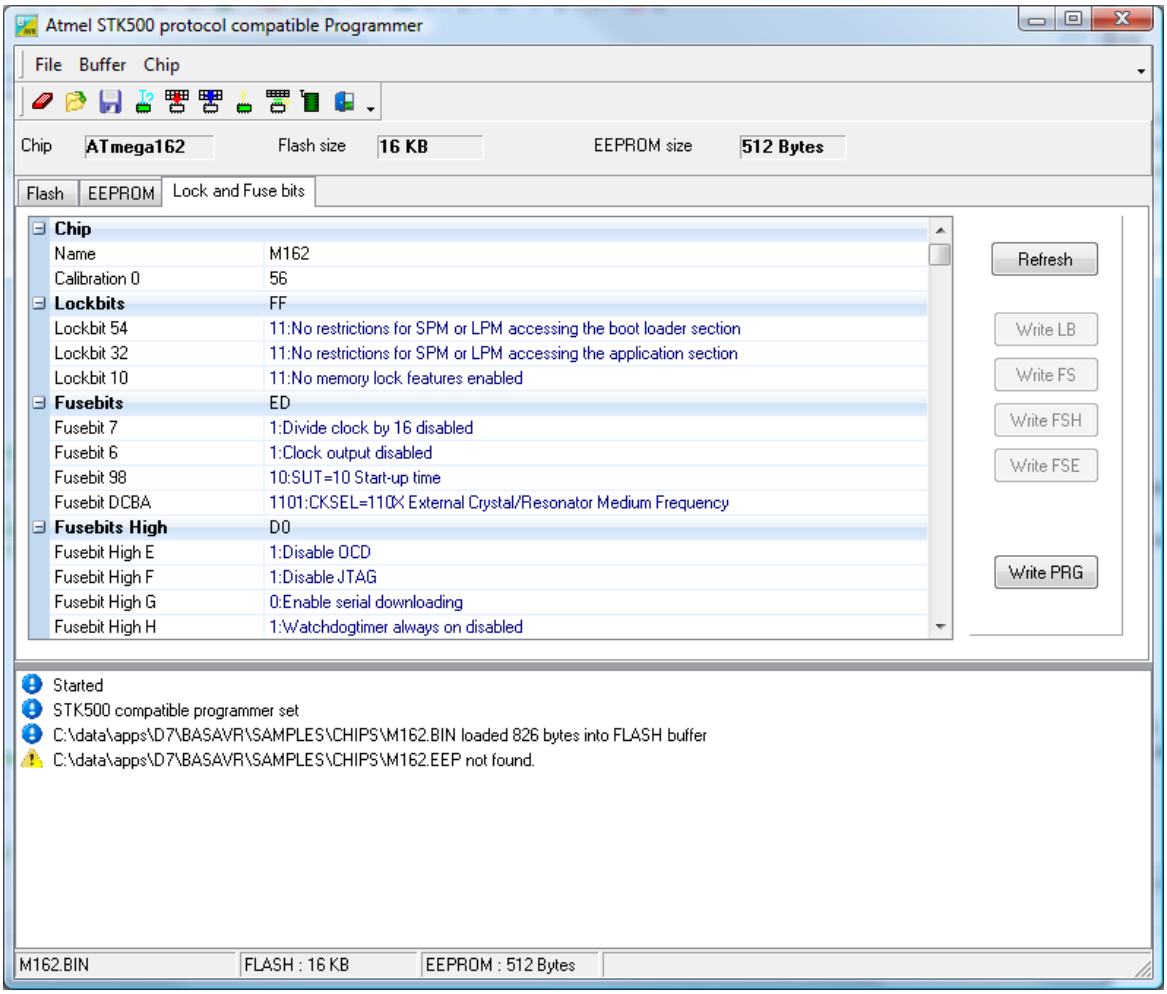
When you select the STK500 programmer, BASCOM will run the file named stk500.exe that is installed with AVR Studio.

That is why you have to specify the file location of the stk500.exe
The normal STK500 support will erase, and program the flash.
The STK500.EXE supports a number of Atmel programmers which all use the STK500 V1 or V2 protocol.
For the AVR ISP mkII, you need to supply the serial number of the USB programmer. The USB port will be used then instead of the serial port.

You can also use the **native** driver which does not use/need the stk500.exe
If you select this programmer, you will see the following window when you launch the programmer with F4(manual program)



When the source code is compiled and the BIN file exists, it is loaded automatic into the buffer.
When an EEPROM image file exists (EEP), it is loaded too into the EEPROM buffer.
When it does not exist you will see a warning which you can ignore.
When the target device is not read yet, the CHIP will be unidentified which is marked as ???.
In the status bar you can see the loaded file, and the size of the file. Notice that 16000 will be shown as 16 KB.
You can select the EEPROM-TAB to view the EEPROM image. Memory locations can be altered. Select a cell, and type a new value. Then press ENTER to confirm. You can immediately see the new value.
When you select the Lock and Fusebits-TAB the lock and fuse bits will be read.



As you can see that as soon as the target chip is determined, the chip name is shown under the tool bar.
The FLASH size and EEPROM size are shown also.
As soon as you alter a lock or fuse bit, the corresponding Write-button will be enabled. You need to click it to write the new value. The lock and fuse bits are read again so you can see if it worked out. The lock and fuse bits shown will depend on the used chip. Every chip has different fuse bits. Some fuse bits can not be altered via the serial programming method. The native stk500 driver uses the serial programming method. Some fuse bits require the parallel or high voltage programming method. For example the fuse bit 'enable serial downloading' can not be changed with the serial programming method.
Fuse bits of interest are : the clock divider and the oscillator fuse bits. When you select a wrong oscillator fuse bit (for example you select an external oscillator) the chip will not work anymore till you connect such an external oscillator! Of course a simple 555 chip can generate a clock signal you can use to 'wake' a locked chip.

Once you have all settings right, you can press the 'Write PRG' button which will insert some code into your program at the current cursor position. This is the $PROG directive.
For example : $prog &HFF , &HED , &HD0 , &HFF
When you compile your program with the $PROG 689 directive it will generate a PRG file with the lock and fuse bit settings.
If you then auto program(see later) a chip, it will use these settings.
$PROG is great to load the right lock and fuse bits into a new chip. But be careful : do not enable $PROG till you are done with development. Otherwise programming will be slow because of the extra reading and writing steps.

The following menu options are available:

Option	Description
File	
Exit	Close programmer.
Buffer	
Clear	Clear buffer. Will put a value of 255 (FF hex) into each memory location. When the FLASH-TAB has the focus, the FLASH buffer will be cleared. When the EEPROM-TAB has the focus, the EEPROM buffer will be cleared. 255 is the value of an empty memory location.
Load from File	This will shown an open file dialog so you can select a binary file (BIN)
	The file is loaded into the buffer.
Save to File	Will save the current buffer to a file.
Reload	Reloads the buffer from the file image.
Chip	
Identify	Will attempt to read the signature of the chip. When the signature is unknown(no DAT file available) or there is no chip or other error, you will get an error. Otherwise the chip name will be shown.
Write buffer to chip	This will write the active buffer(FLASH or EEPROM) into the chip.
Read chipcode	When the chip lock bit is not set you can read the FLASH or EEPROM into the buffer.
Blank check	Check if the chip FLASH or EEPROM is empty.
Erase	Erases the chip FLASH. It depends on the fusebits if the EEPROM is erased too. Normally the EEPROM is erased too but some chip have a fuse bit to preserve EEPROM when erasing the chip. A chip MUST be erased before it can be programmed.
Verify	Checks if the buffer matches the chip FLASH or EEPROM.
Auto program	This will eraser, and program the FLASH and EEPROM and if $PROG is used, it will set the lock and fusebits too.

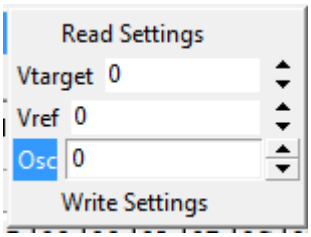
Under Options, you can find a setting to change the clock frequency.

 The clock frequency should not be higher then a quarter of the oscillator frequency.
This means that a chip with an internal 8 MHz oscillator which has the 8-divider fuse enabled, will have a clock frequency of 1 Mhz.
The programming clock may not exceed 250 KHz in this case.

STK500 board

When using the STK500 board, you can change the target voltage and the reference voltage. In 2081 you can also change the board oscillator frequency.
The BOARD menu has a sub menu named STK500. This sub menu has a few options :
- Read Settings : you should do this first
- Vtarget : this is the target voltage. Make sure the chip can handle the voltage you enter

- Vref : this is the reference voltage. It may not exceed Vtarget.
- Osc : this is the oscillator frequency.
- Write settings : this will write the new settings to the STK500 board. After doing so, read back the settings to see if the values are correct. You will notice that not all values you enter are possible. This is exactly the same when you use AVR Studio.

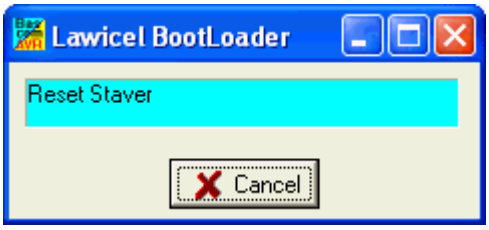


3.60.1.7 Lawicel BootLoader

The Lawicel Boot loader must be used with the StAVeR. The StAVeR contains a boot loader so you only need a serial interface, no parallel programmer or other programmers.

You can also use Hyper terminal.

When you have selected the Lawicel Boot loader from the Options, Programmer, the following window will appear when you press F4.



As the window suggests, press the reset button on the activity board or StAVeR, and the chip will be programmed. This is visible by a second wind that will be shown during programming.

When the programming succeeds, both windows will be closed.

When an error occurs, you will get an error message and you can clock the Cancel button in order to return to the Editor.

THIS PROGRAMMED IS MARKED FOR REMOVAL. Send a note to support if you use it.

3.60.1.8 MyAVR/MK2/AVR910 programmer

This programmer is an ISP programmer from MYAVR which supports the AVR910/911 protocol.
It has an USB interface with serial port interface. You need to select the proper port and a baud rate of **19200** baud.

3.60.1.9 AVR ISP Programmer

The AVRISP programmer is AVR ICP910 based on the AVR910.ASM application note.

The old ICP910 does not support Mega chips. Only a modified version of the AVR910. ASM supports Universal commands so all chips can be programmed.

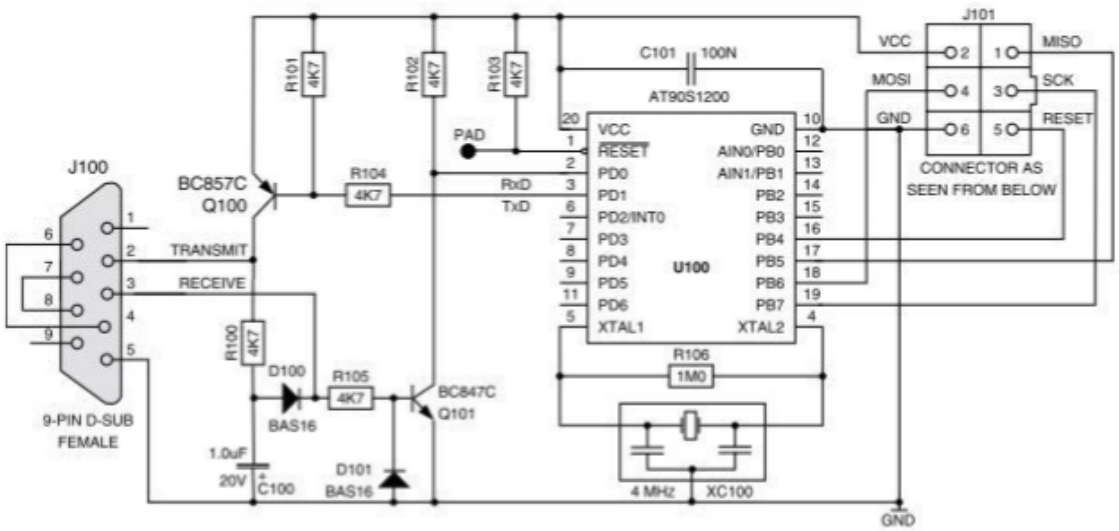
The new AVRISP from Atmel that can be used with AVR Studio, is not compatible! You need to select STK500 programmer 170 because the new AVRISP programmer from Atmel, uses the STK500 protocol.

When you do not want to use the default baud rate that AVR910 is using, you can edit the file bascavr.ini from the Windows directory.
Add the section [AVRISP]
Then add: COM=19200,n,8,1

This is the default. When you made your own dongle, you can increase the baud rate

You need to save the file and restart BASCOM before the settings will be in effect.

Figure 4. A Low-cost In-System Programmer



This programmer is not available from Atmel/Microchip and is not recommended.

3.60.1.10 USB-ISP Programmer

The USB-ISP Programmer is a special USB programmer that is fully compatible with BASCOM's advanced programmer options.
Since many new PC's and especial Laptop's do not have a parallel programmer anymore, MCS selected the USB-ISP programmer from EMBUD.

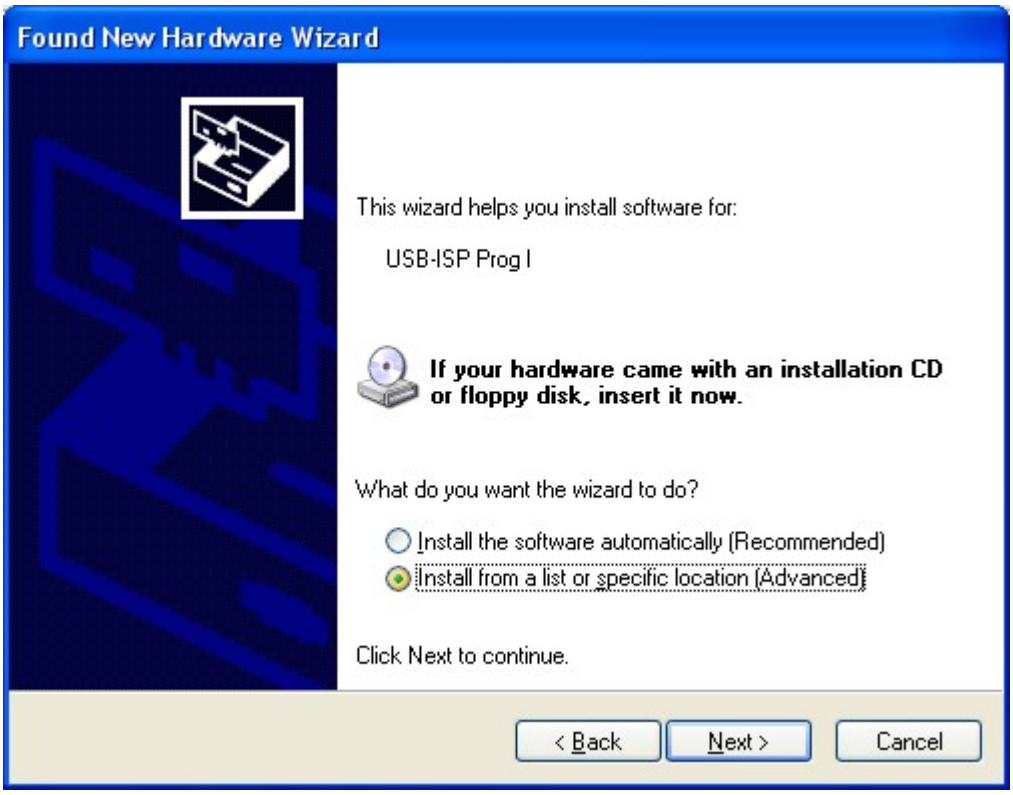
The drivers can be downloaded from the MCS Electronics website.
Please download from
http://www.mcselec.com/index.php?option=com_docman&task=doc_download&gid=204&Itemid=54
After downloading, unzip the files in the BASCOM-AVR application directory in a sub directory named USB.

When you connect the programmer, Windows (98, ME, 2000, XP) will recognize the new device automatically.

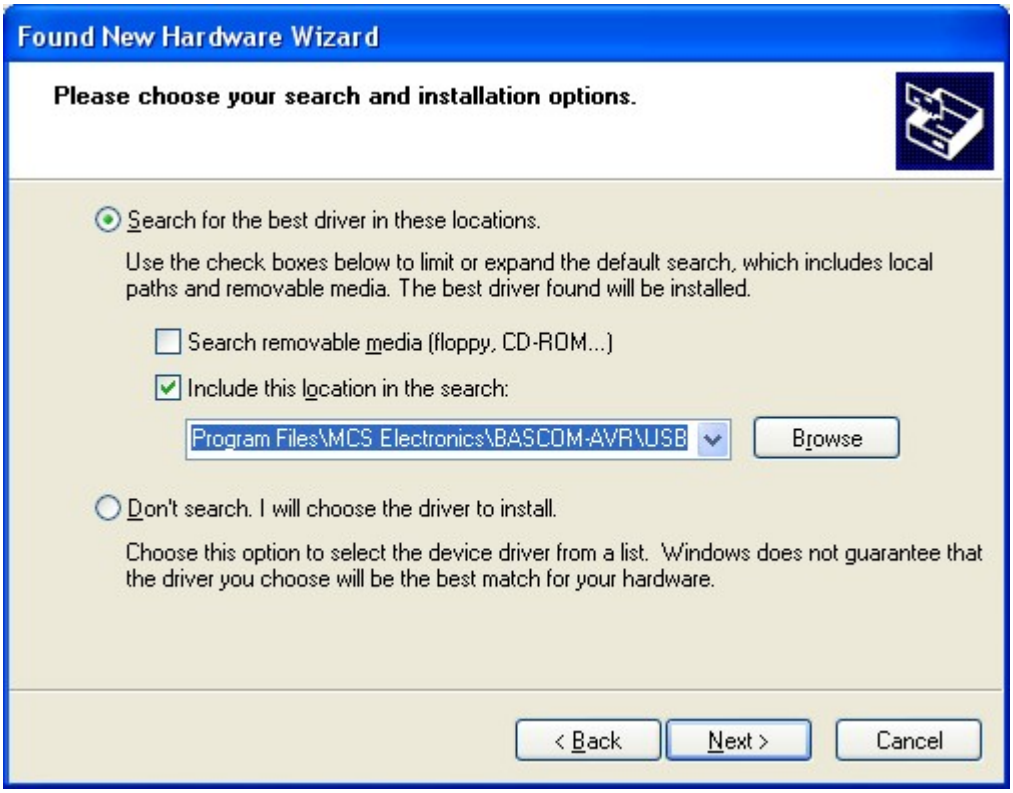
Then the Hardware wizard will be started :

Select 'No, not this time' and click Next, as there is no driver at Microsoft's web.

The Wiz will show :



You need to select 'Install from a list or specific location' and click Next.



You can specify the path of the USB driver. This is by default :

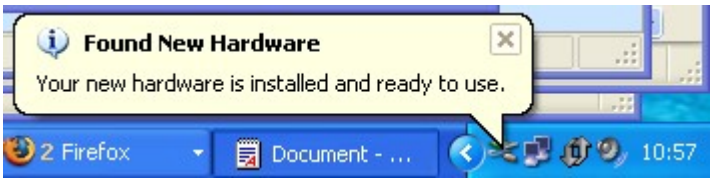
C:\Program Files\MCS Electronics\BASCOM-AVR\USB

Use the Browse-button to select it, or a different location, depending on your installation.

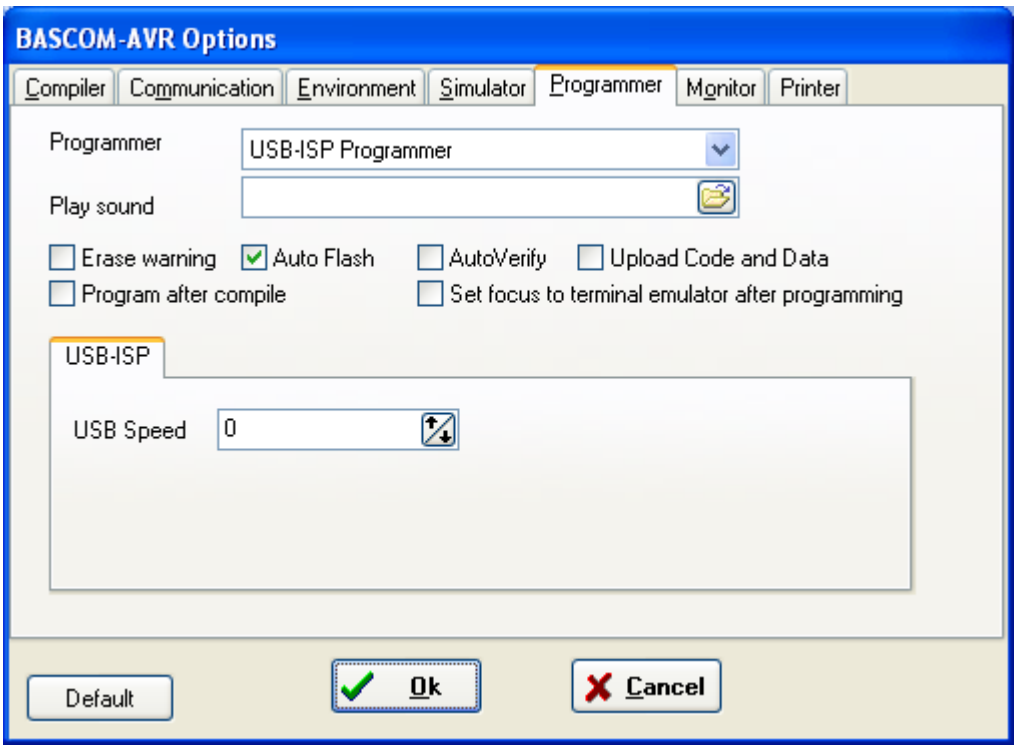
As the driver is not certified by Micros ft, you will see the following window:

You need to select 'Continue Anyway'. A restore point will be made if your OS supports this and the driver will be installed.
After installation you must see the following window :

After you press Finish you will see Windows can use the programmer :



In BASCOM , Options, Programmer you can select the new programmer now.

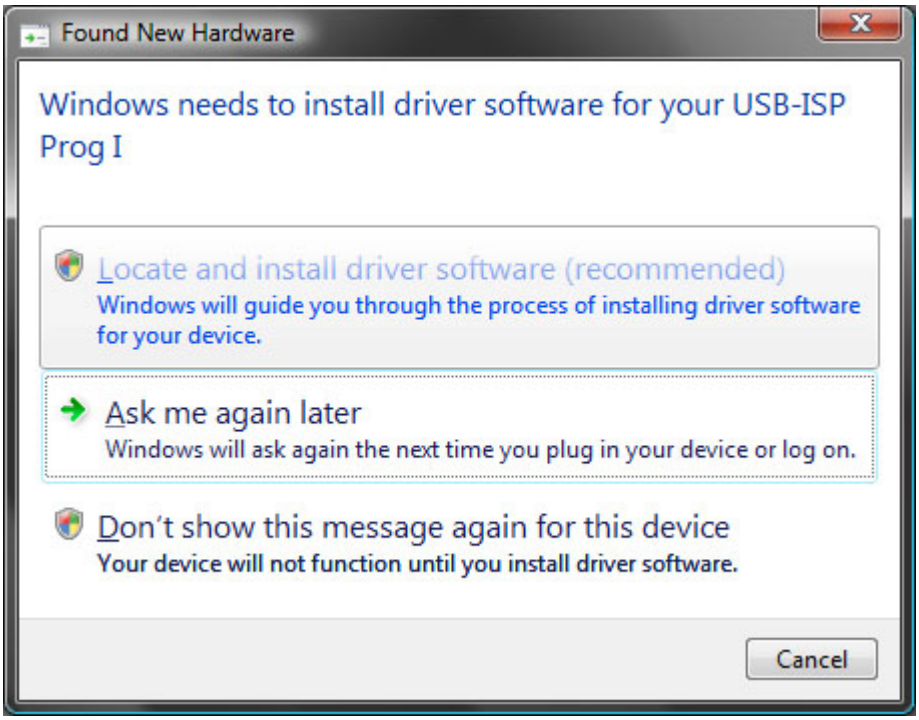


New models of the USB programmer allow to set the speed.
The USB-ISP programmer is very quick and supports all options that the Sample Electronics and STK200 programmers support. It is good replacement for the STK200.

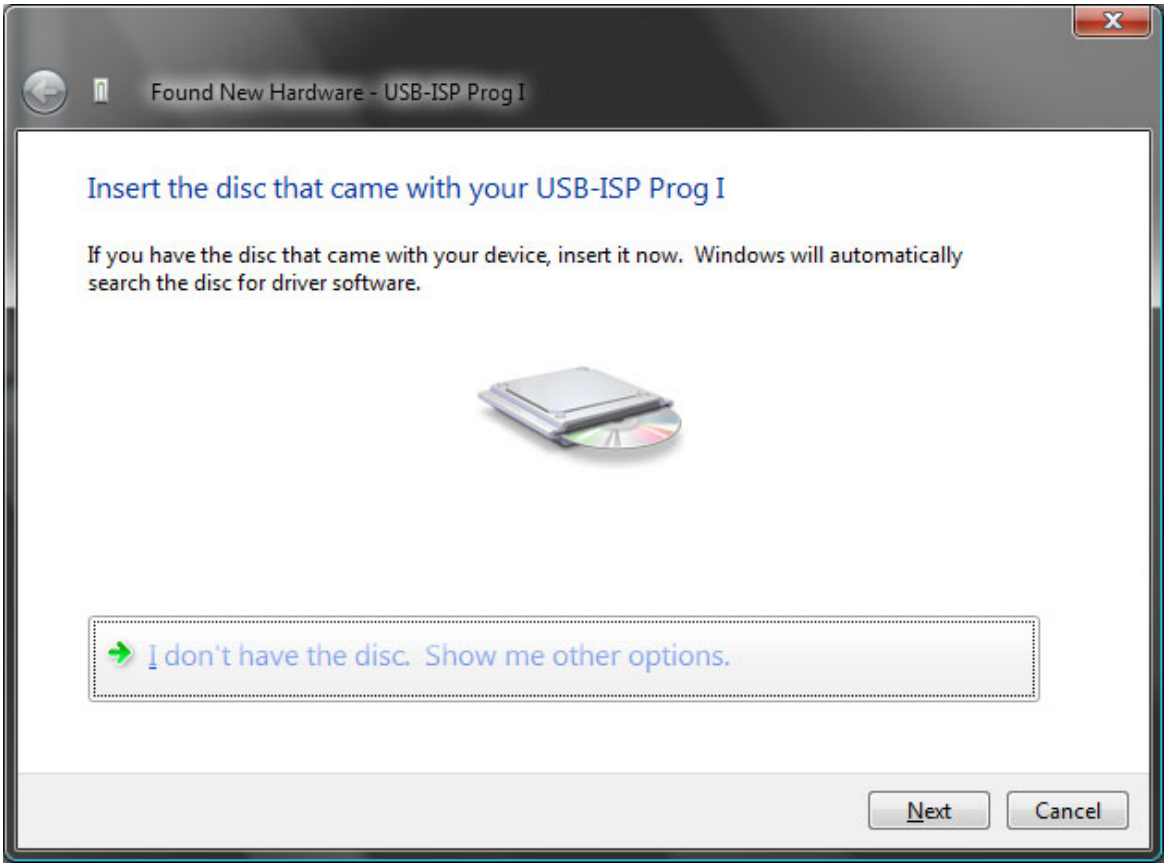
When you use other USB devices that use the FTDI drivers, there might occur a problem. Manual install the drivers of these other devices, then install the USB-ISP driver.

USB-ISP on VISTA

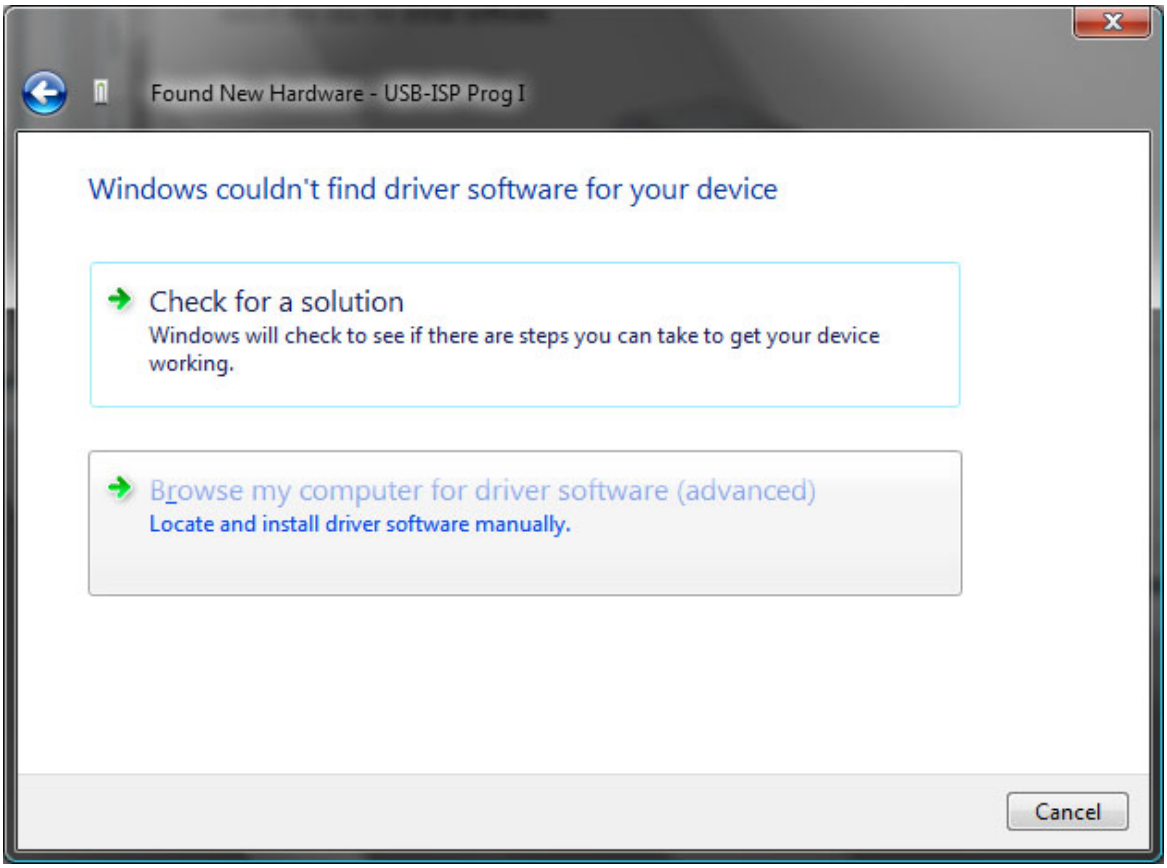
For Vista and Vista 64, please follow the this installation description.



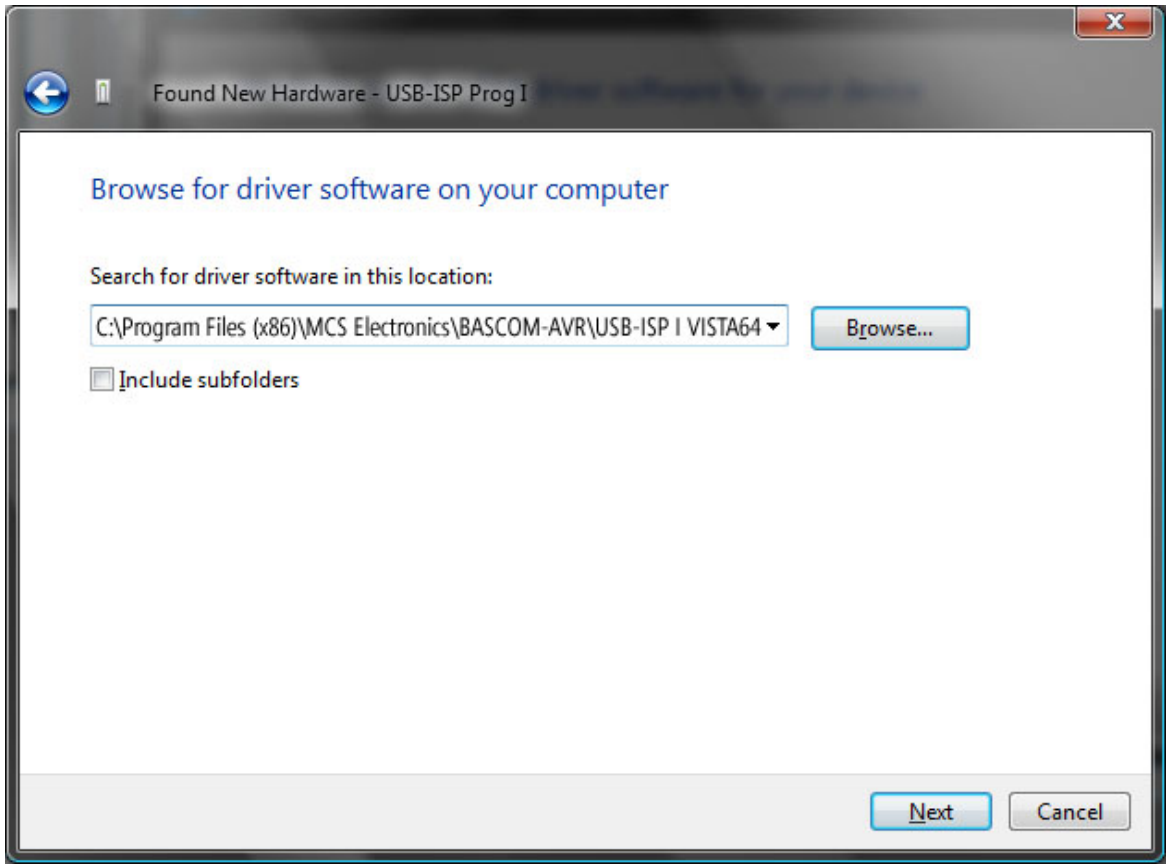
When connection the ISP-PROG I to your PC the following window will show up. Here I have to select the top selection: ***Locate and Install driver software (recommended)*** Vista starts it search for the driver and will come finally with the question to Insert the driver disk.



As we have no driver CD, you have to select: ***I don't have the disc. Show me other options***

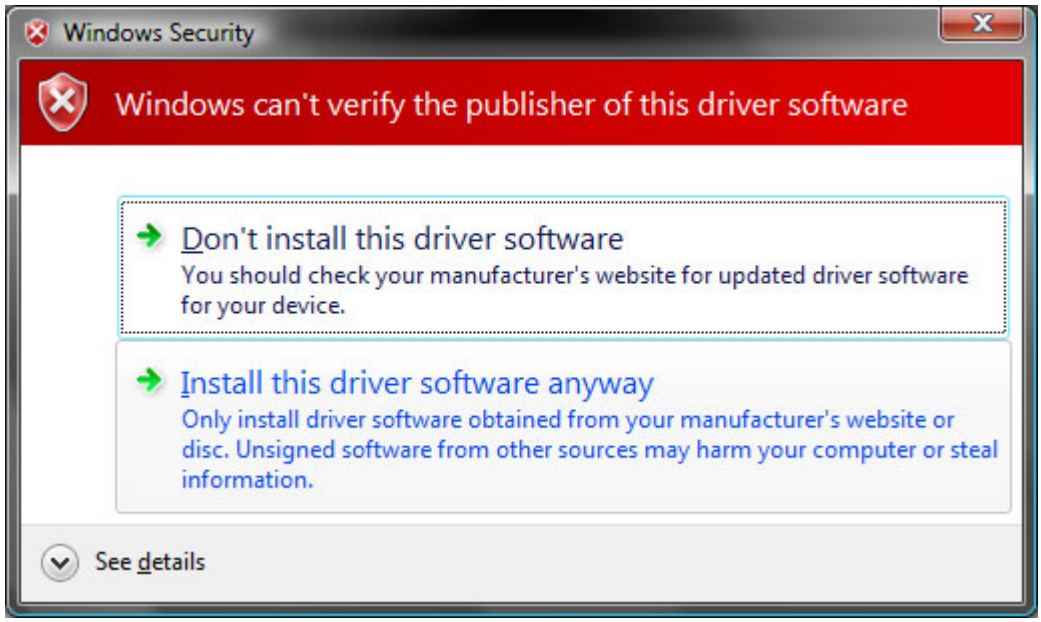


Now we select the Browse selection and locate the driver folder.

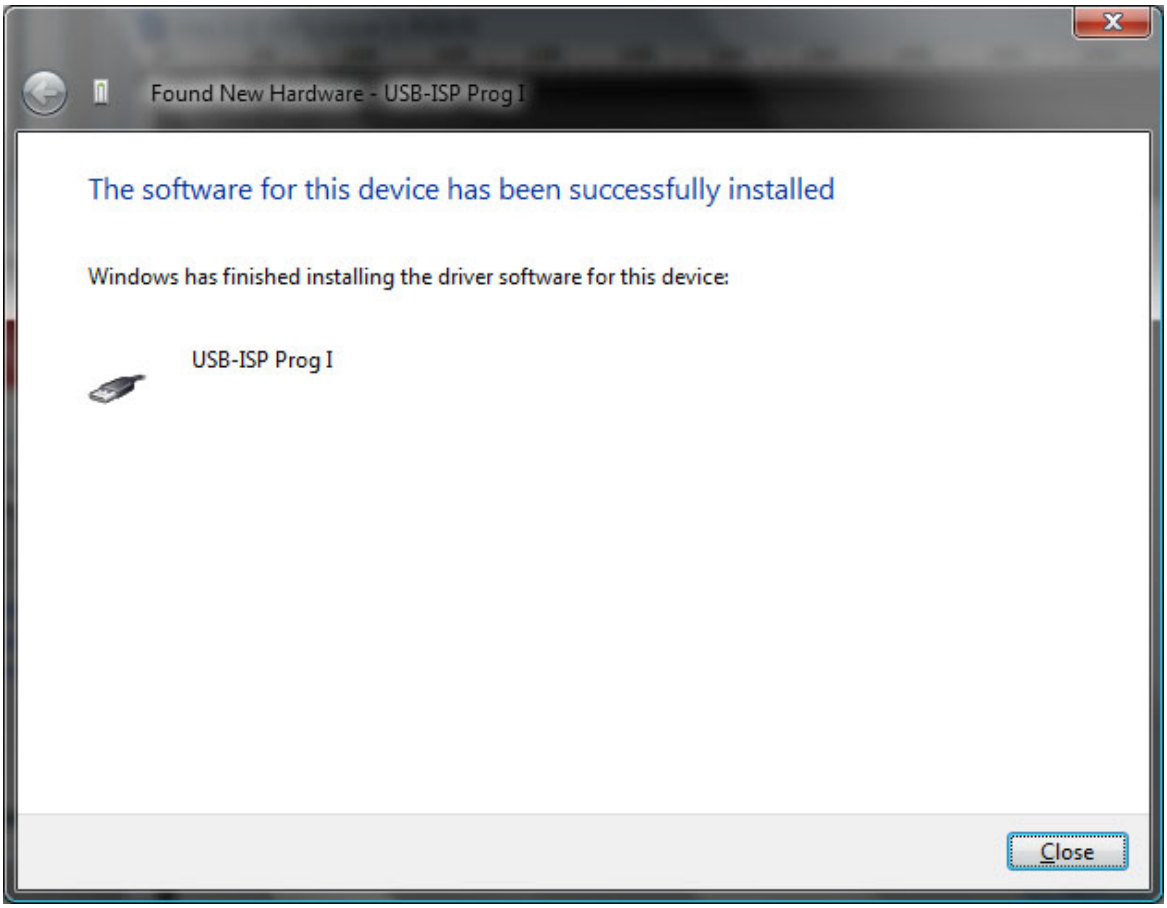


And select ***Next*** button.

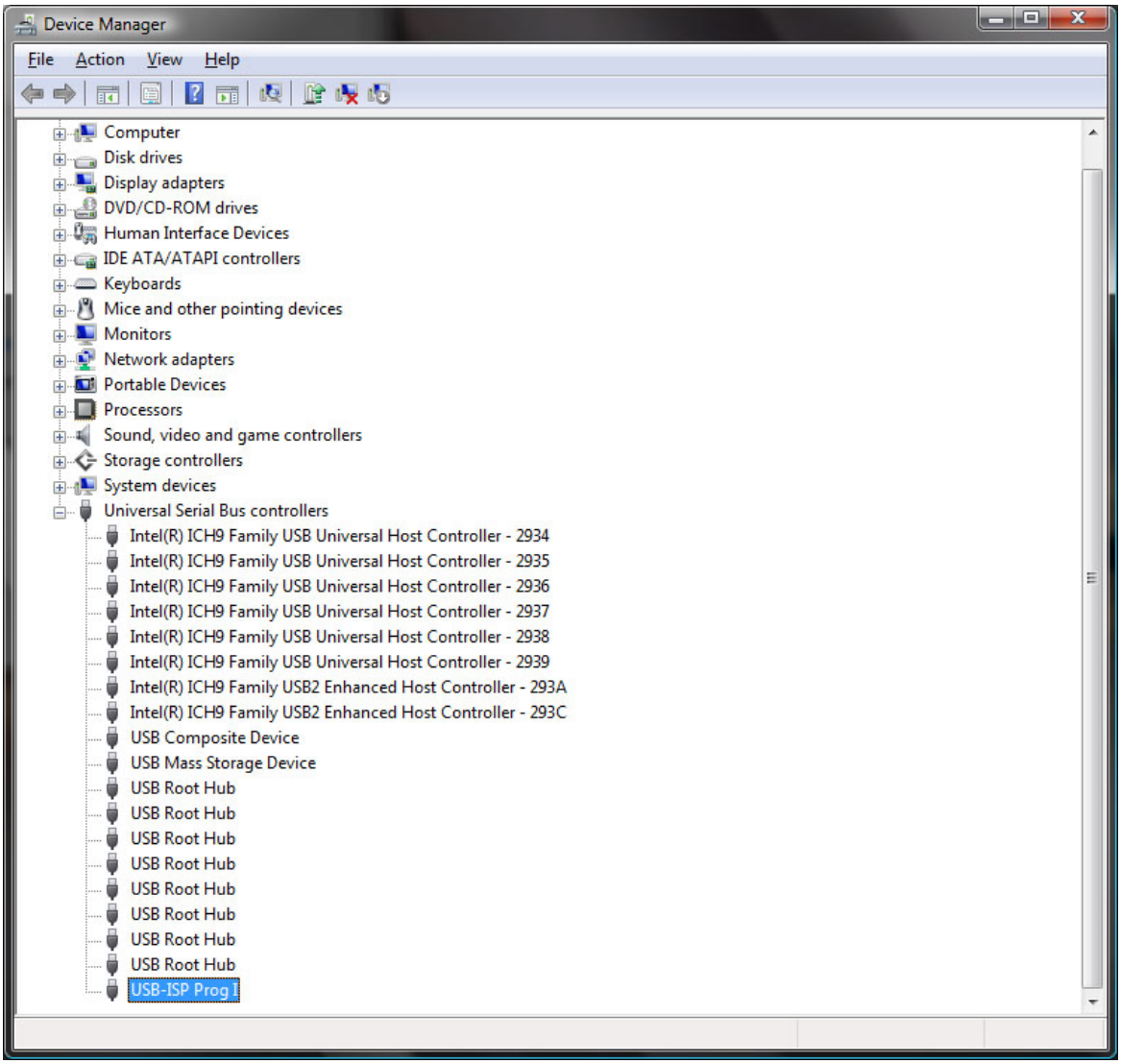
As Vista 64 only allows certified drivers the following message will pop-up.



Just select ***Install this driver software anyway*** and Vista 64 will now start with installing the driver. Be patient as it depends on your system configuration how long it will take.



Finally Vista 64 will tell you that the driver is installed. To check your configuration you can go to your device manager to see if it is there.

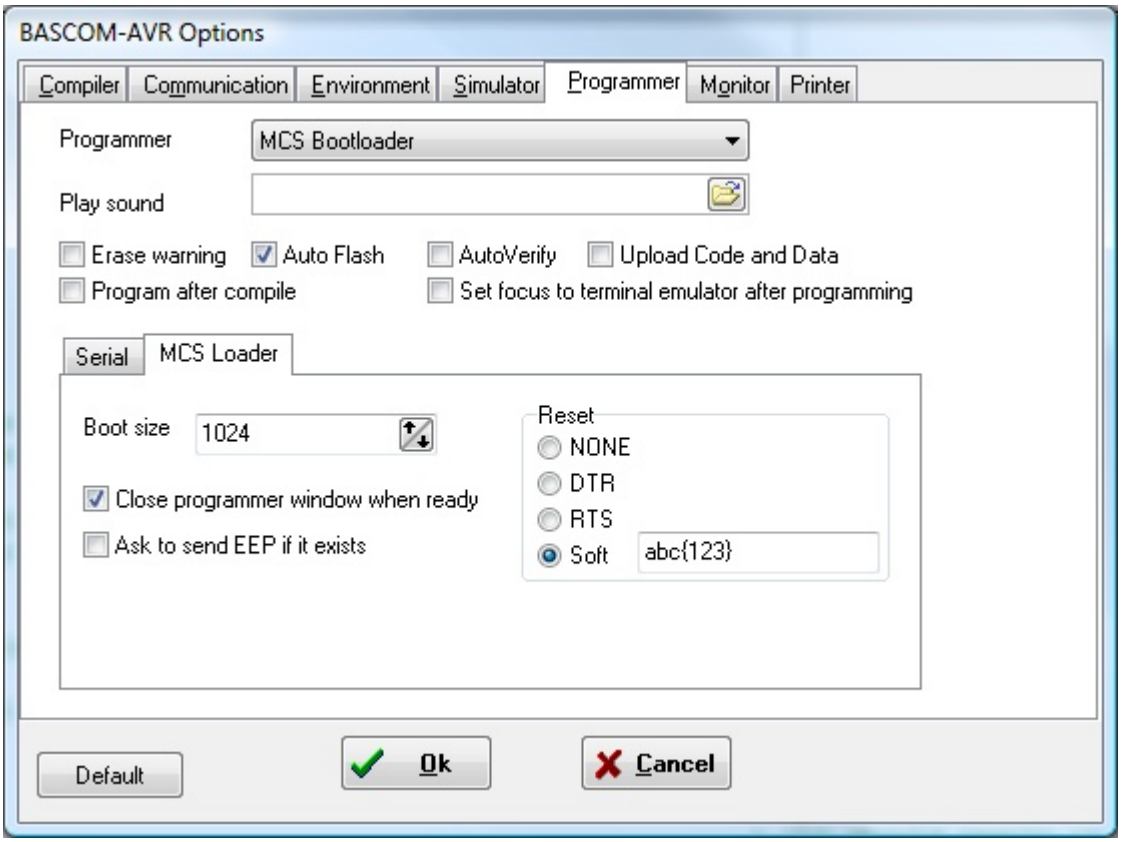


3.60.1.11 MCS Bootloader

The MCS Boot loader is intended to be used with the $LOADER 667 sample for normal AVR and XMega. For Xtiny, MegaX and AVRX check the $ROMSTART 697 topic.
It uses the X-modem Checksum protocol to upload the binary file. It works very quick.
The Boot loader sample can upload both normal flash programs and EEPROM images.
The Boot loader sends a byte with value of 123 to the AVR Boot loader. This boot loader program then enter the boot loader or will jump to the reset vector (0000) to execute the normal flash program.

When it receives 124 instead of 123, it will upload the EEPROM.
When you select a BIN file the flash will be uploaded. When you select an EEP file, the EEPROM will be uploaded.

The Boot loader has some specific options.



BOOTSIZE

You can choose the boot size which is 1024 for the BASCOM $LOADER example. Since this space is used from the normal flash memory, it means your application has 1024 less words for the main application. (A word is 2 byte, so 2KB less)
The XMEGA has a separate boot space so for Xmega you can set the value to 0.

RESET

The boot loader is started when the chip is reset. Thus you need to reset the chip after you have pressed F4(program). But when you have connected the DTR line to the chip reset (with a MAX232 buffer) you can reset the chip automatically. You do need to set the 'Reset via DTR' option then. You can also chose to use the RTS line. When your program does not use the boot vector or needs a special sequence to activate the loader, you can chose the soft reset. To send ASCII characters you can embed them between brackets {}. For example {065} will be sent as the character A or byte with value 65.

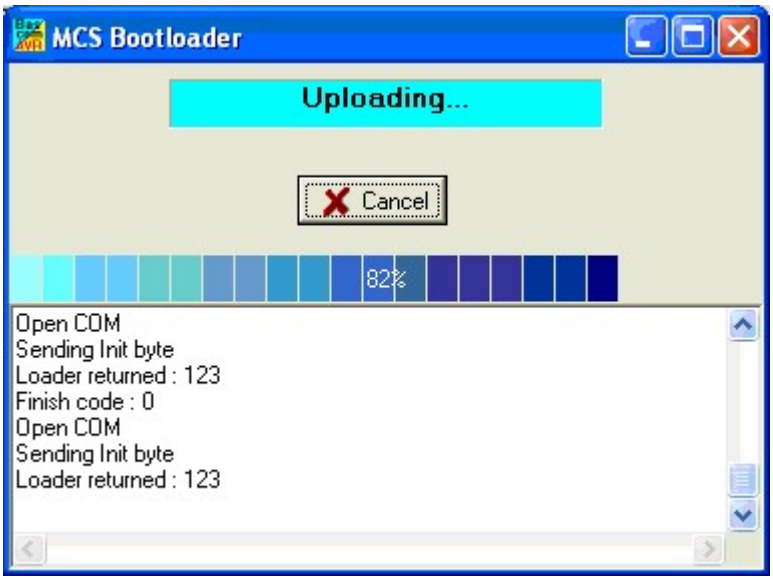
CLOSE

By choosing 'Close programmer window when ready' the window will be closed when the loader returns 0.
In all other cases it will remain opened so you can look at a possible cause.

EEP

If an EEP (EEPROM image file) exists, the loader can send this file instead of the flash binary file. If you enable this option, you will be asked if you want to send the EEP instead of the BIN file.

After you have pressed F4 to following window will appear :



As you can see the loader sends a byte with value of 123.
You need to reset the chip, and then you will see that the loader returned 123 which means it received the value.
It will start the upload and you see a progress bar. After the loader is ready, you see a finish code of 0.
A finish code of 0 means that all wend well.
Other finish codes will not close the window even if this option is enabled.
You need to manual close the window then.

ERROR CODES

-6001 - Bad format in file name
-6002 - file not found
-6003 - file not found in folder
-6004 - folder not found
-6005 - canceled
-6006 - time out
-6007 - protocol error
-6008 - too many errors
-6009 - block sequence error
-6016 - session aborted

The most likely error is -6006 when the bootloader is not present or does not respond timely after the initial handshake. Increase the $timeout in the boot loader in that case.

3.60.1.12 PROGGY

PROGGY is a popular USB programmer written by Red_Mamba.

You need to install it and make sure that the registry key :
HKEY_CURRENT_USER\Software\Red_Mamba\Atmel programator exists with the parameter : **InstallPath**

InstallPath should point to the executable which name is atme.exe
When you install PROGGY, it will be handled for you. When you have an older version, you need to update.

BASCOM will call the programmer with the following options : -p -s -e

The -e will cause the programmer to exit after the programming.

THIS PROGRAMMED IS MARKED FOR REMOVAL. Send a note to support if you use it.

3.60.1.13 FLIP

FLIP is a free USB bootloader from Atmel. With FLIP you can program an AVR without additional (ISP) programmer hardware.
Because it is a USB bootloader it only work with AVR with built in USB functionality.

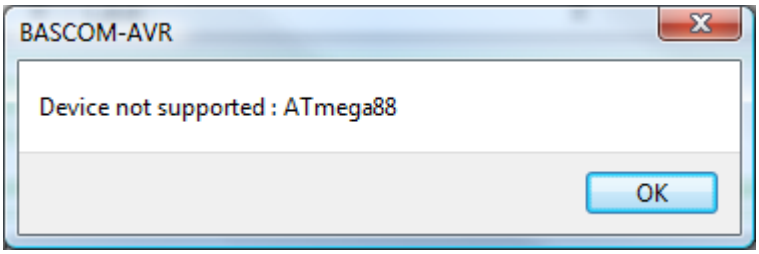
FLIP is supported by the BASCOM-IDE so you can use it direct by pressing the Program Chip (F4) button and download a HEX file.

FLIP can be downloaded from the Atmel site.
Search for "FLIP bootloader" on the Atmel Website for the latest version:
https://www.microchip.com/developmenttools/ProductDetails/flip

1. Download FLIP from Atmel Website
2. Install FLIP
3. In BASCOM-IDE Select FLIP from **Options** >>> **Programmer** , in order to program quickly without the FLIP executable
4. Now you can press Program Chip (F4) to program the HEX file into the chip

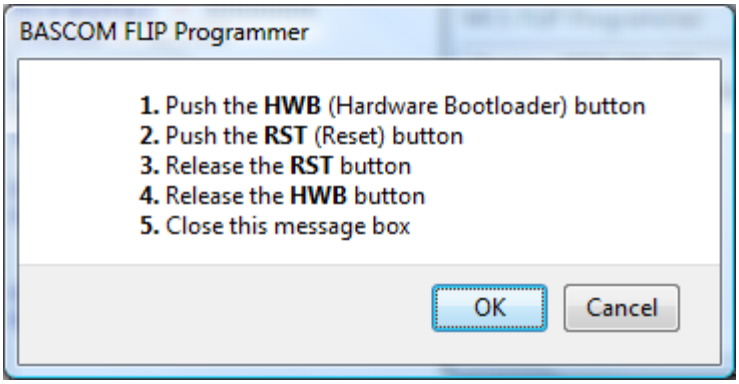
As with other programmers, you press F4 to program the HEX file into the chip. A small window will become visible.

A number of dialogs are possible:



In this case, you try to program a chip which is not supported by FLIP. The Mega88 is not an USB chip so the error makes sense.

If you are using an USB AVR you could get following dialog box:
This dialog informs you about a missing DFU device and/or the device is not in boot loader mode:



In this case, the boot loader is not found. You can run the boot loader by following the sequence from the dialog box.

In order to make this work, the HWB (Hardware Bootloader Button) and RST (Reset Button) input both need a small switch to ground.
When HWB is pressed(low) during a reset, the boot loader will be executed.

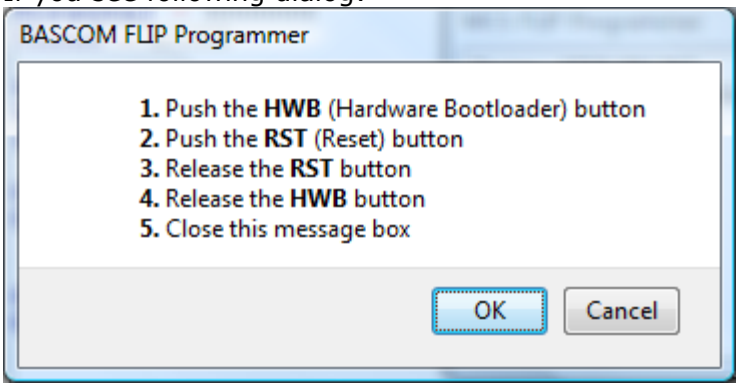
Abbreviations:

- ISP: In-system programming
- RST: Rest
- USB: Universal serial bus
- DFU: Device firmware upgrade
- FLIP: Flexible in-system programmer

FAQ - Using FLIP with XMEGA-A3BU Xplained Board from Atmel (under Windows 7 32-Bit)

1. Read Atmel App Note: AVR1916: USB DFU Boot Loader for XMEGA
2. Download FLIP
3. Install FLIP 3.4.5 or higher for Windows (Java Runtime Environment included)
4. Connect the USB Cable during pressing **Switch0 SW0** (Hardware Bootloader button) on the XMEGA-A3BU Xplained board
5. The USB Driver can be found in the FLIP Software directory (e.g.: C:\Program Files\Atmel\Flip 3.4.5\usb)
6. You can also search for DFU ATXMEGA256A3BU in the Windows 7 device manager and reinstall the driver by pointing it to this directory (e.g.: C:\Program Files\Atmel\Flip 3.4.5\usb)
7. Then you will find this here in the device manager Atmel USB Devices >>>> ATxmega256A3BU
8. In BASCOM-IDE Select FLIP from Options >>> Programmer , in order to program quickly without the FLIP executable
9. Now you can press Program Chip (F4) to program the HEX file into the chip

If you see following dialog:



Just connect the USB Cable during pressing Switch0 SW0 on the XMEGA-A3BU Xplained board
Hit OK button then the XMEGA will be programmed.

First example for XMEGA-A3BU board:

```
$regfile = "XM256A3BUDEF.DAT"
$crystal = 32000000                                    '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80

Config Osc = Enabled , 32mhzosc = Enabled              '32MHz
```

```
'configure      the      systemclock
Config    Sysclock = 32mhz ,     Prescalea = 1 ,     Prescalebc = 1_1


Config Porte.4 = Output
Backlight Alias Porte.4                                    'LCD      Backlight

Config Portr.0 = Output
Led0 Alias Portr.0                                         'LED  0


Config Portr.1 = Output
Led1 Alias Portr.1                                         'LED  1


Do

 Waitms  500
 Reset  Led0
 Set  Led1



  Waitms  500
 Set  Led0
 Reset  Led1


Loop

End                                                        'end     program
```

FAQ - FLIP with BASCOM-IDE

On former versions like FLIP 3.3.1 there was on VISTA a problem with loading some of the FLIP DLL's.
In case you get an error, copy the FLIP DLL's to the BASCOM application directory.
You need to copy the following files :

- atjniisp.dll
- AtLibUsbDfu.dll
- msvcp60.dll
- msvcrt.dll

You can also create a command file for that task like: flipDLLcopy.cmd to copy these files.
The content of the command file :

copy "c:\program files\atmel\flip 3.3.1\bin\atjniisp.dll" .
copy "c:\program files\atmel\flip 3.3.1\bin\AtLibUsbDfu.dll" .
copy "c:\program files\atmel\flip 3.3.1\bin\msvcp60.dll" .
copy "c:\program files\atmel\flip 3.3.1\bin\msvcrt.dll" .
pause

The last line pauses so you can view the result. Notice the . (dot) that will copy the file to the current directory, which is the reason that you need to run this file from the BASCOM application directory.
You also need to adapt the version of FLIP in the command file.

In order to use BASCOM's FLIP support, you must have running FLIP successfully first !
Here is a good tip from a user :

IMO he Flip 3.3.1 Installer is a little bit stupid.
The dll´s are located in the Path ...\Atmel\Flip 3.3.1\bin .
The Installer has set a correct Path-Variable in Windows for this path.
But, the libusb0.dll isn´t in that location. It is in ...\Atmel\Flip 3.3.1\USB !
So I moved the libusb0.dll into the \bin dir and Flip runs without the errors. (GRRRR)

In the ...\Atmel\Flip 3.3.1\USB dir I have also detected the missing .inf File. After installing this, Windows detects the AT90USB162 and Flip can connect the device.

3.60.1.14 USBprog Programmer / AVR ISP mkII

The USBprog programmer is a neat small USB programmer which is fully compatible with the AVR ISP mkII programmer.
When you select this programmer, you will get the same interface as for the STK500 native 170 programmer.
F4 will launch the programmer. For more details read the help section for the STK500 programmer.

When programming XMEGA chips the interface for the fuse bits will be different. See STK600 194 programmer for a description.

The default clock is 125 KHz. This because most/all chips ship with a clock frequency of 1 MHz. And since the clock frequency maximum is a quarter of the oscillator frequency, the default is 125 KHz, low enough to be able to program all chips. Once your chip runs at say 8 MHz, you can select 2 MHz as the maximum.

You must have the LIBSUSB 216 drivers installed on your PC. Without it, it will not work.

Options

In the Configuration options you can adjust the clock speed and the timeout of the USB.
When you are using USB 1.1 and a lot of devices that generate a lot of USB traffic, you might need to increase the default timeout of 100 (msec).

XMEGA

When used in PDI mode, take care about the following for some of the processors:

JTAG is activated by default which preventing from using the PDI because both interfaced share the same pins. In this case :

1 - Disable the JTAG before using the PDI. > You need a JTAG programmer

2 - Use a 47Kohm resistor to Pull down the clock pin to ground which allow you to have both JTAG and PDI working simultaneously.

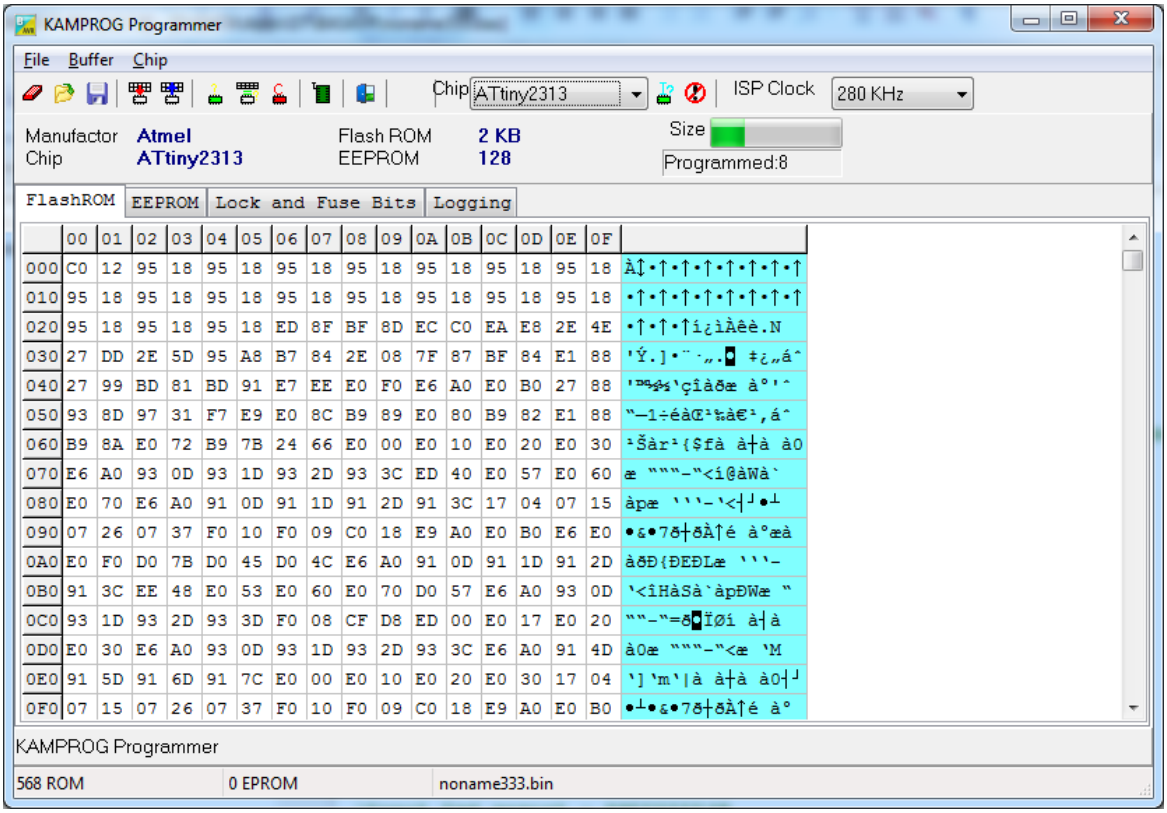
3.60.1.15 KamProg for AVR

KamProg for AVR is an USB programmer from Kamami.
You need to install the software that comes with the KamProg.
You can download the software from the web site of the manufacturer or from MCS Electronics web shop Kamprog product page.
KamProg can be used with BASCOM but also with AVR Studio.

BASCOM will use the KamProg software to either automatic or manual program the chip.

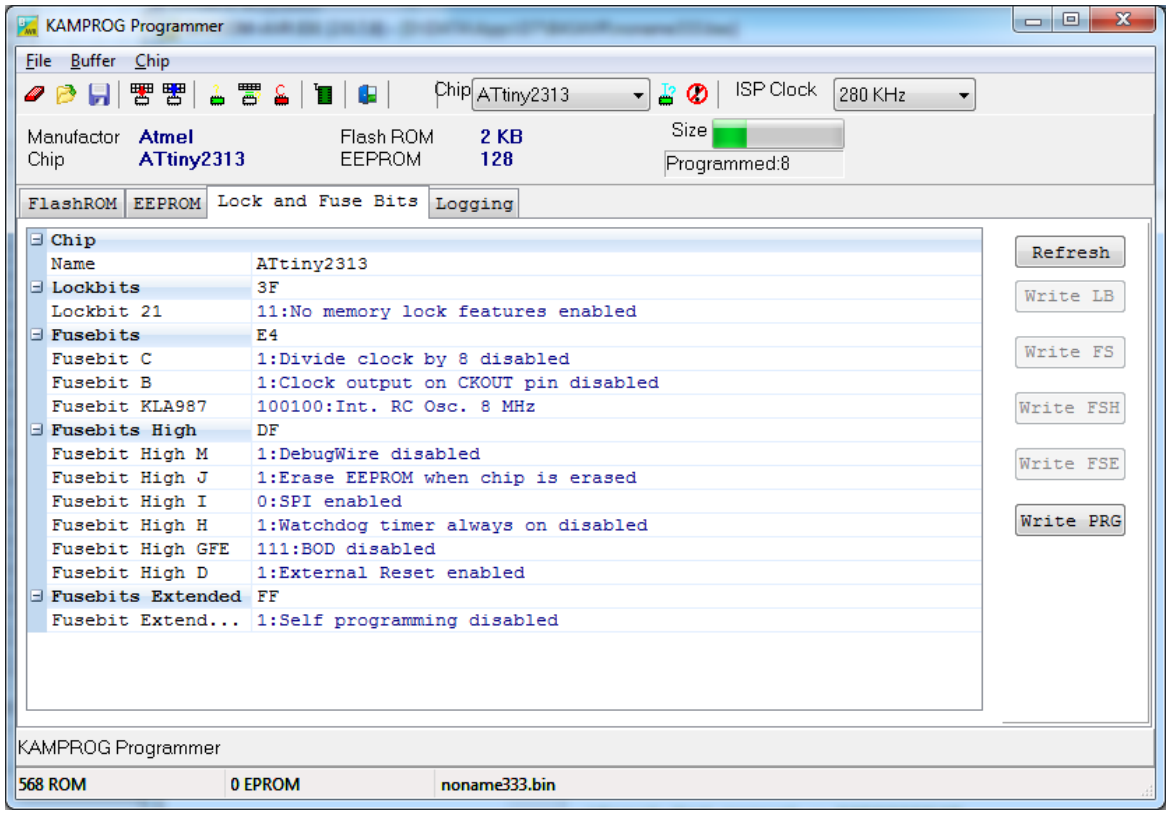
The Kamprog programmer works on Vista32 and Vista64 and requires no special drivers. It has also been tested with Win7 and Win8 and Win8.1
The KamProg programmer is available from MCS Electronics webshop.

When you use Auto program, you will see a small progress window while the processor is programmed.
When you chose manual program, you will see a familiar window, known from USB-ISP.



When the source code is compiled and the BIN file exists, it is loaded automatic into the buffer.
When an EEPROM image file exists (EEP), it is loaded too into the EEPROM buffer.
When it does not exist you will see a warning which you can ignore.
When the target device is not read yet, the CHIP will be unidentified which is marked as ???.
In the status bar you can see the loaded file, and the size of the file. Notice that 16000 will be shown as 16 KB.

You can select the EEPROM-TAB to view the EEPROM image. Memory locations can be altered. Select a cell, and type a new value. Then press ENTER to confirm. You can immediately see the new value.
When you select the Lock and Fusebits-TAB the lock and fuse bits will be read.



As soon the target chip is determined, the chip name is shown under the tool bar. The FLASH size and EEPROM size are shown too.
When you alter a lock or fuse bit, the corresponding Write-button will be enabled. You need to click it to write the new value. The lock and fuse bits are read again so you can see if it worked out.
The lock and fuse bits shown will depend on the used chip. Every chip has different fuse bits. Some fuse bits can not be altered via the serial programming method. For example the fuse bit 'enable serial downloading' can not be changed using the serial programming method.

Fuse bits of interest are : the clock divider and the oscillator fuse bits. When you select a wrong oscillator fuse bit (for example you select an external oscillator) the chip can not programmed anymore until you connect such an external oscillator! Of course a simple 555 chip can generate a clock signal you can use to 'wake' a locked chip.

Once you have all settings right, you can press the 'Write PRG' button which will insert some code into your program at the current cursor position. This is the $PROG directive.
For example : $prog &HFF , &HED , &HD0 , &HFF
When you compile your program with the $PROG 689 directive it will generate a PRG file with the lock and fuse bit settings.
If you then auto program(see later) a chip, it will use these settings.
$PROG is great to load the right lock and fuse bits into a new chip. But be careful : do not enable $PROG till you are done with development. Otherwise programming will be slow because of the extra reading and writing steps.

The following menu options are available:

Option	Description
File	
Exit	Close programmer.
Buffer	
Clear	Clear buffer. Will put a value of 255 (FF hex) into each memory location. When the FLASH-TAB has the focus, the FLASH buffer will be cleared. When the EEPROM-TAB has the focus, the EEPROM buffer will be cleared. 255 is the value of an empty memory location.
Load from File	This will shown an open file dialog so you can select a binary file (BIN)
	The file is loaded into the buffer.
Save to File	Will save the current buffer to a file.
Reload	Reloads the buffer from the file image.
Chip	
Identify	Will attempt to read the signature of the chip. When the signature is unknown(no DAT file available) or there is no chip or other error, you will get an error. Otherwise the chip name will be shown.
Write buffer to chip	This will write the active buffer(FLASH or EEPROM) into the chip.
Read chipcode	When the chip lock bit is not set you can read the FLASH or EEPROM into the buffer.
Blank check	Check if the chip FLASH or EEPROM is empty.
Erase	Erases the chip FLASH. It depends on the fusebits if the EEPROM is erased too. Normally the EEPROM is erased too but some chip have a fuse bit to preserve EEPROM when erasing the chip. A chip MUST be erased before it can be programmed.
Verify	Checks if the buffer matches the chip FLASH or EEPROM.
Auto program	This will eraser, and program the FLASH and EEPROM and if $PROG is used, it will set the lock and fusebits too.

In the toolbar you can also alter the ISP clock frequency.

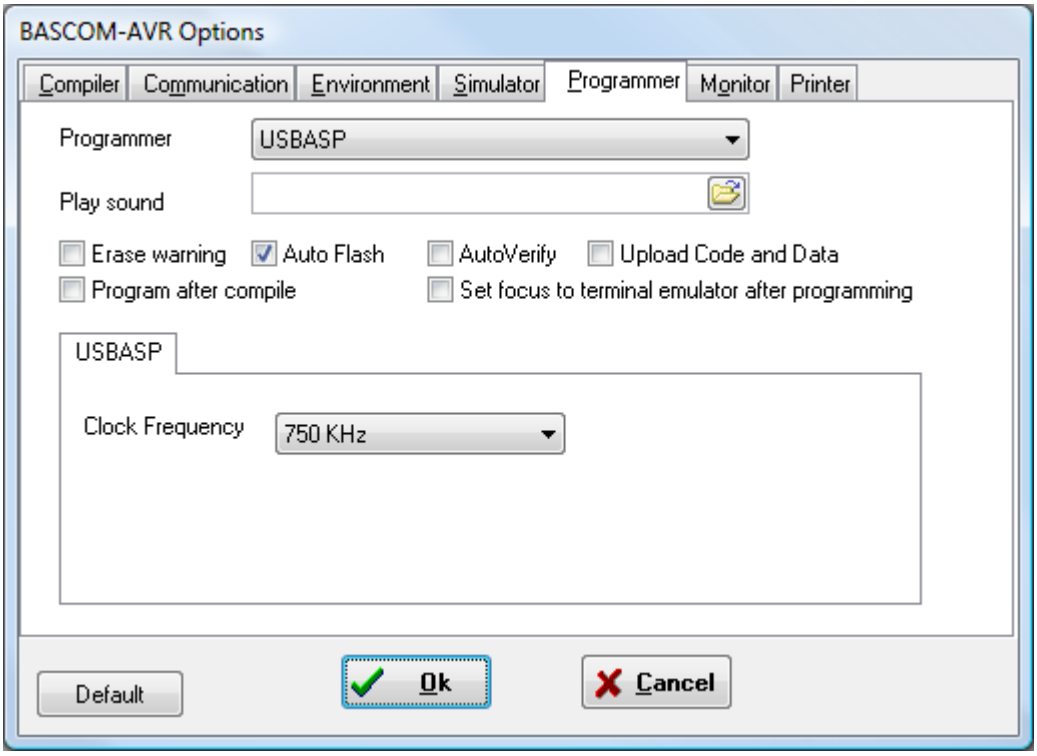
 The clock frequency should not be higher than a quarter of the oscillator frequency.
This means that a chip with an internal 8 MHz oscillator which has the 8-divider fuse enabled, will have a clock frequency of 1 Mhz.
The programming clock may not exceed 250 KHz in that case.

3.60.1.16 USBASP

The USBASP is a popular USB programmer created by Thomas Fischl
The programmer uses a Mega8 or other AVR chip as an USB device.
You can find the programmer at Thomas website : http://www.fischl.de/usbasp

Make sure when programming the fuse and lock bits that the selected clock frequency is not too high. The clock frequency of the ISP programmer should be less then one quarter of the oscillator frequency. When your micro is running at 8 MHz, you can

select up to 2 MHz. On the safe size, 125 KHz is always ok.
By default most AVR processors run at 8 MHz with an 8-divider resulting in 1 MHz clock frequency. So 250 KHz is a safe value for most processors.



You can select various clock frequencies.

See also LIBUSB 216 for installation of LIBUSB

3.60.1.17 STK600

The STK600 is a development board from Atmel. It uses a similar protocol as the STK500 and has an integrated USB programmer on board.
The programmer can be connected with a cable to the STK600 board itself, but also to an external board.

The STK600 replaces the STK500 and is advised for XMEGA development. For regular AVR chips we would recommend the STK500.

The STK600 has actual 3 different programmers on board : ISP, PDI and JTAG. the ISP/PDI protocols are combined and placed on one connector.

When programming XMEGA chips, the BASCOM programmers will automatic switch to the PDI protocol. The ISP protocol can not be used with XMEGA chips.
For other chips, (non-xmega), the ISP protocol will be used.
There are affordable PDI programmers available.

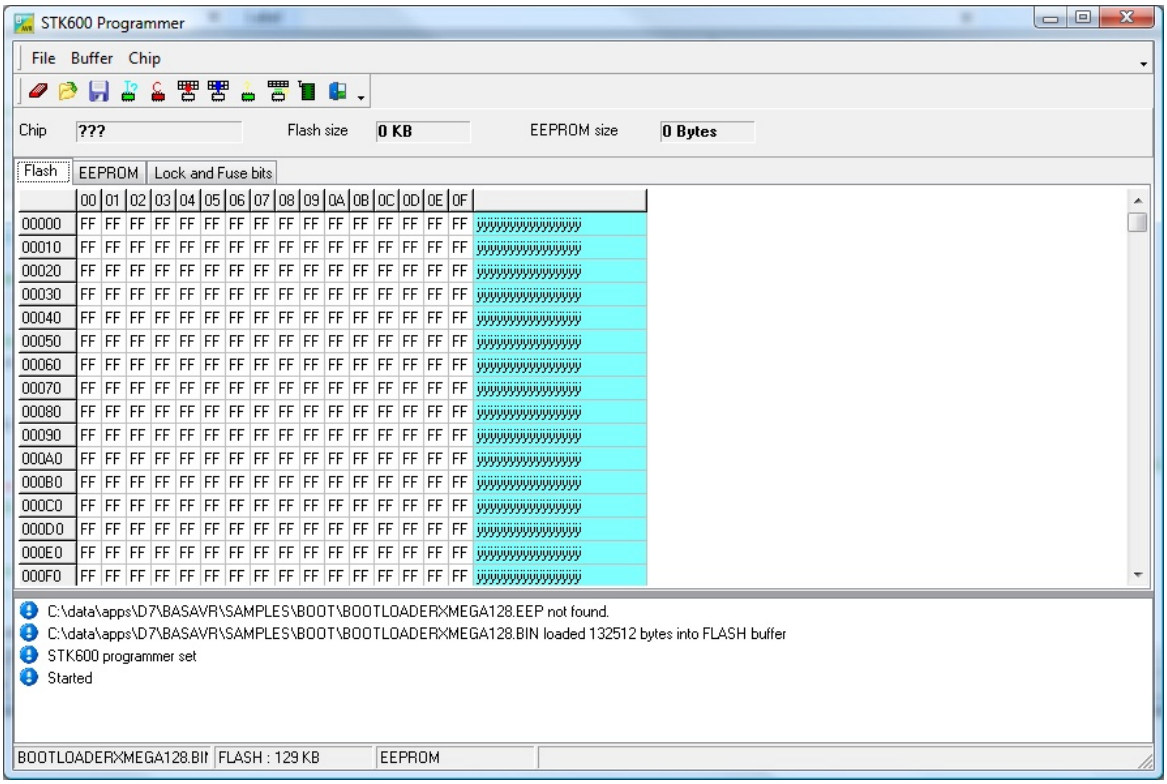
The following description is also true for the AVRISP/mkII programmer which also supports the PDI protocol.

In order to use the STK600 protocol you need to have LIBSUSB 216 installed.

Identification

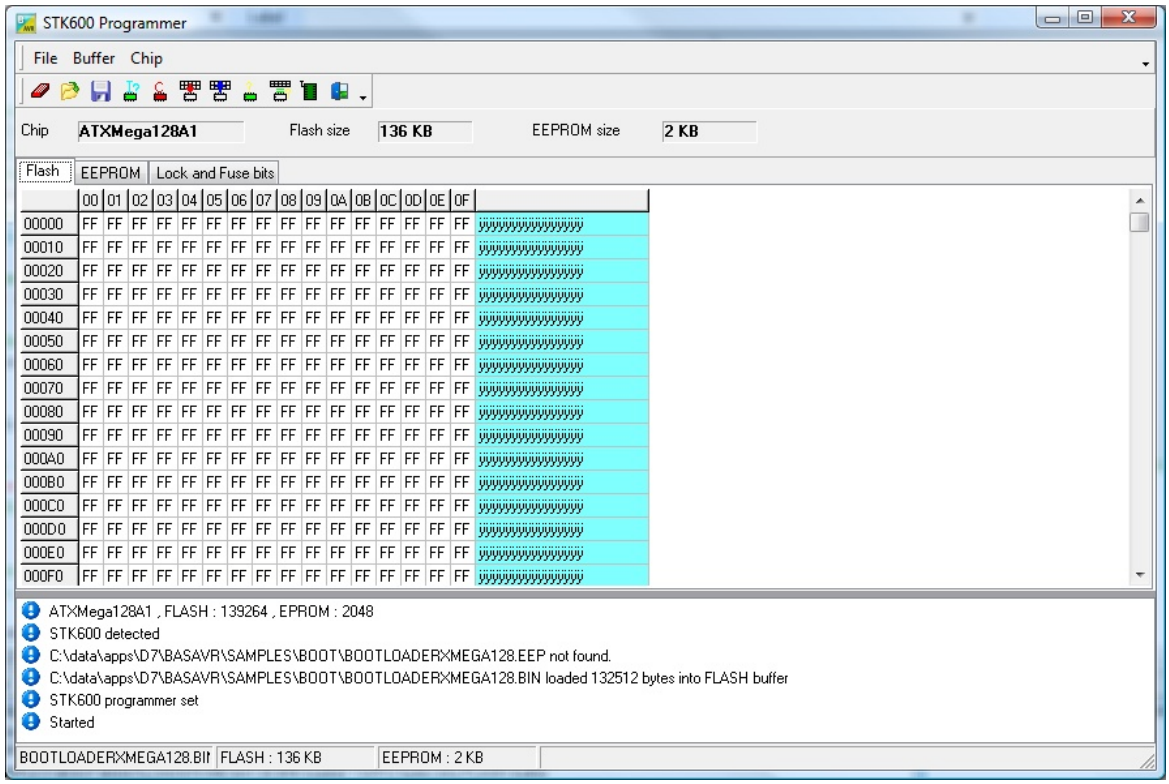
The BASCOM programmers always try to identify the chip before an action is performed. This is needed to check the size and to check if your program is intended for the selected chip.
It would not be a good idea for example to program an attiny13 with xmega128a1 code.

When you chose manual programming, you will get the following window:



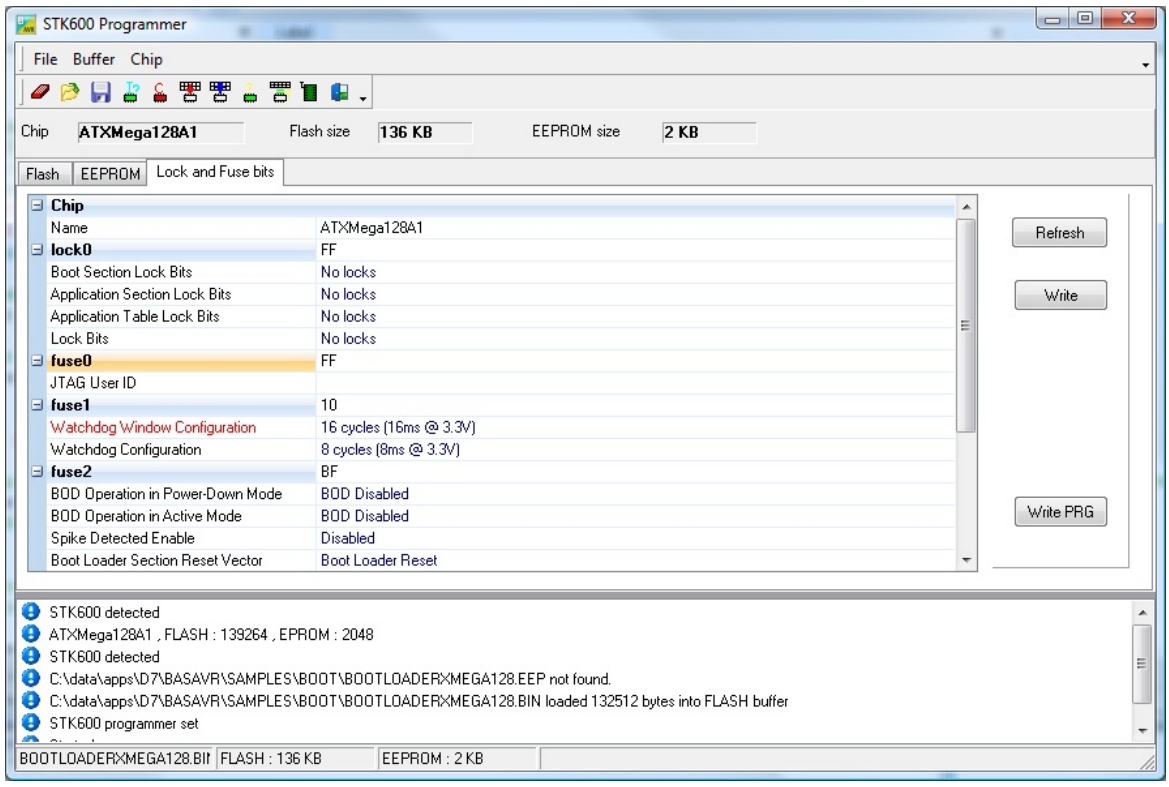
As you can see, the binary image is loaded and if an EEPROM EEP binary image was available it would have been loaded too.

When you click the Identify button, the programmer will read the device id. The same will happen for any other action you chose.



The Device ID is now read and you can see the ATXMEGA128A1 is detected.

The programmer has the same options as the STK500 programmer. Only the lock and fuse byte differ for the Xmega.
When you select the Lock and Fuse bits, you will get a similar screen:



The XMEGA has one lock byte and 6 fuse byes named FUSE0-FUSE5.
Not all fuse bytes are used. The options depend on the XMEGA chip you use.

In the screen shot from above you can see that under the FUSE1 section, the 'Watchdog Window Configuration' is colored red.
When you change an option and move focus or enter, a change will result in the option to be shown in red.

When you have selected all values you can select the WRITE button to write the lock and fuse bytes.
After this the values will be read again and updated.

The WRITE PRG button will insert a $PROG directive into your code with all lock and fuse bytes.

A description of the fuse bytes you can find in the PDF of the processor.

3.60.1.18 ARDUINO

The ARDUINO is a hardware platform based on AVR processors. ARDUINO boards/ chips are programmed with a bootloader. This bootloader is the old STK500 protocol, not longer supported by Atmel in Studio. There are various programmers for ARDUINO, AVRDUDE is probably the most versatile.

BASCOM also supports the ARDUINO/STK500 v1 protocol. the DTR/RTS lines are used to reset the board.
You can program/read flash/EEPROM but you can not read/write fuse/lock bytes. The STK500 bootloader for ARDUINO does not support this.

Under options you only need to select the programmer, and the COM port. Since an FTDI chip is used on most ARDUINO boards, this is a virtual COM port. Only present when the USB cable is connected to your PC.
Select **57600** baud for the baud rate. Older ARDUINO boards work with **19200** baud.

ARDUINO V2

The developers of the ARDUINO finally implemented the STK500V2 protocol. This protocol is supported by Atmel and of course by BASCOM.
Select the ARDUINO STK500V2 programmer in BASCOM programmer options to use this protocol.
A board like the MEGA2560 R3 uses this protocol and probably all newer AVR based ARDUINO boards will support this protocol. The baud rate should be 115200 but could be different for your board.

ARDUINO Leonardo

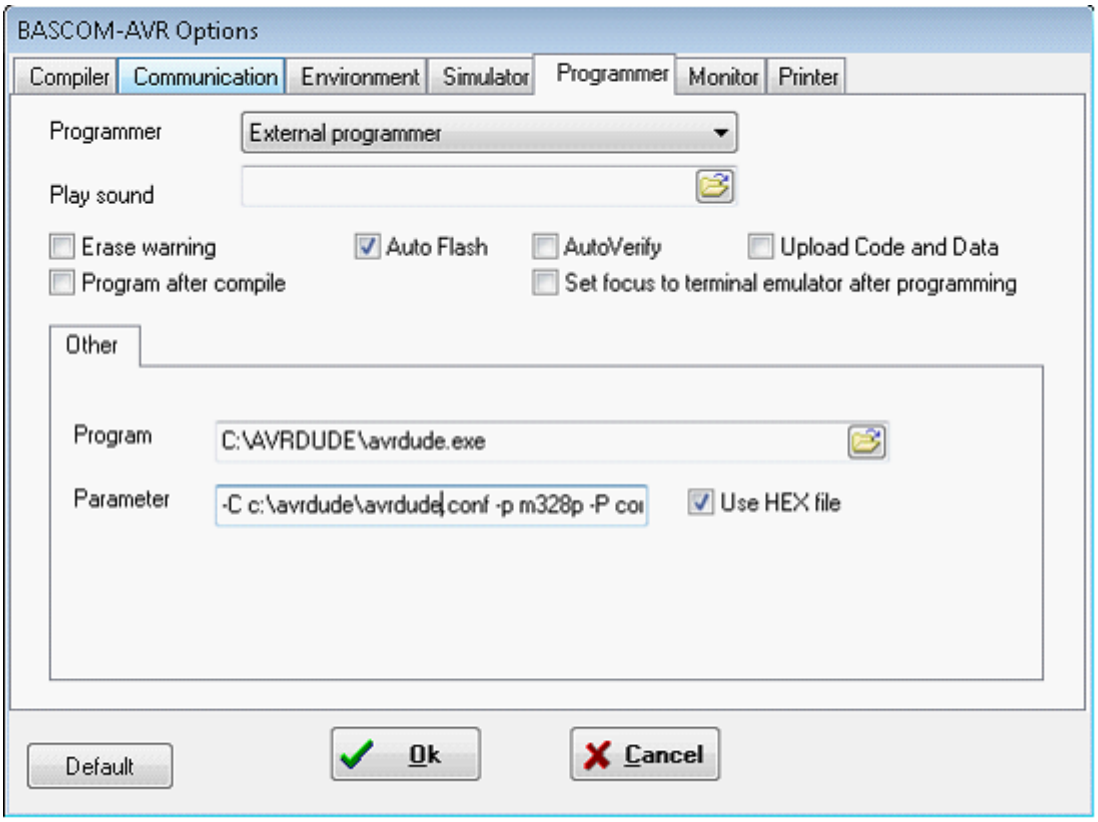
For some reason each arduino board seems to use a different bootloader method. The leonardo implements a virtual COM port. When opened at 1200 baud, the board resets into another virtual COM device with a different COM port number.
In BASCOM you need to chose the **myAVR MK2 / AVR910** programmer since Leonardo uses the AVR910 loader from Atmel.
You need to select the COM port that you get at Boot time. The baud is 115200.
To program, press the reset button, wait till the USB is enumerated and the Virtual COM port is ready, then press F4 to program the processor.

Using Bascom-AVR with Arduino Optiboot Bootloader (under Windows 7)

For more information on Optiboot visit following website: http://code.google.com/p/optiboot/

1. Download AVRDUDE from http://www.nongnu.org/avrdude/
2. Latest Windows Version (April 2012): avrdude-5.11-Patch7610-win32.zip
   Complete link:
http://download.savannah.gnu.org/releases/avrdude/avrdude-5.11-Patch7610-win32.zip
3. Create a folder like c:\AVRDUDE
4. Copy the content of avrdude-5.11-Patch7610-win32.zip in this new folder
5. Open Bascom-AVR
6. Click on **Options** >>> **Programmer**
7. Choose External programmer
8. Checkmark Use HEX file
9. Include the path to avrdude.exe
10. User Parameter:

-C c:\avrdude\avrdude.conf -p m328p -P com19 -c arduino -b 115200 -U flash:w:{FILE}:i



Explanation of Parameter:
-C
c:\avrdude\avrdude.conf The config file tells avrdude about all the different ways it can talk to the programmer.

-p

m328p This is just to tell it what microcontroller its programming. For example, if you are programming an Atmega328p, use m328p as the partnumber

-P

com19 This is the communication port to use to talk to the programmer (COM19) in this case. Change it to your COM port.

-c

arduino

Here is where we specify the programmer type, if you're using an STK500 use stk500, use arduino for Optiboot

-b

115200

Set serial baudrate for programmer. Use 115200 baud for Optiboot.

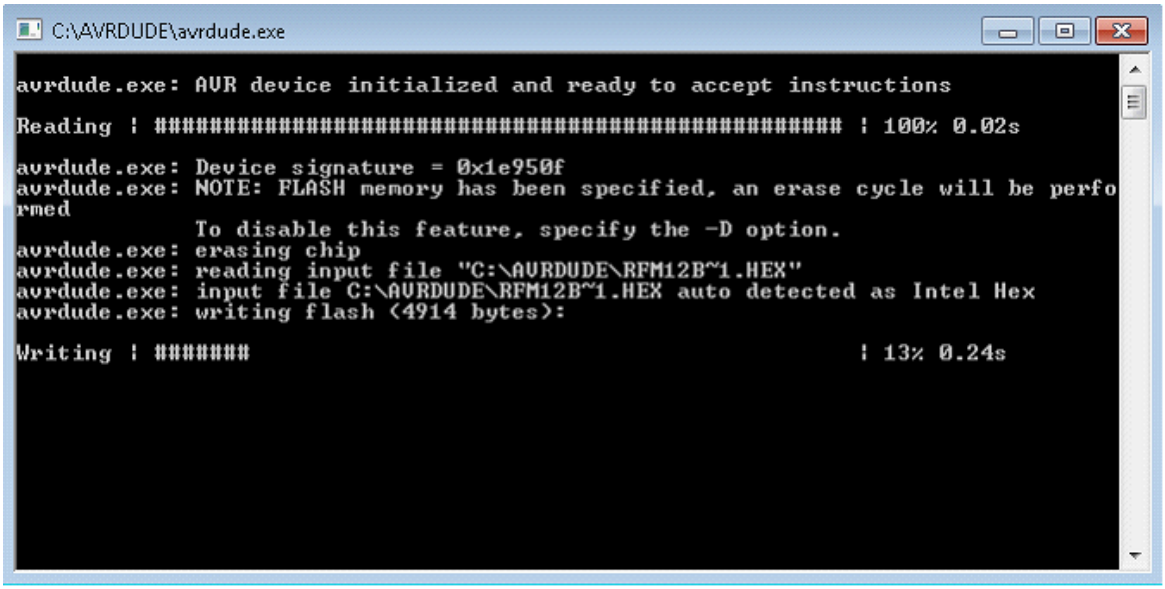
-U

flash:w:{FILE}:i

You define here:

- the memory type: flash or eeprom (this could be also hfuse, lfuse or effuse if you want to verfiy this)
- r (read), w (write) or v (verify)
- Use {FILE} to insert the filename {EEPROM} to insert the filename of the generated EEP file.
- i = Intel Hex File

After clicking on the F4 (Program Chip) Button in Bascom-AVR you see the CMD window of Windows 7 until AVRDUDE is ready flashing the Arduino.



Complete documentation of AVRDUDE parameters:
http://www.nongnu.org/avrdude/user-manual/avrdude_4.html#Option-Descriptions

3.60.1.19 BIPOM MINI-MAX/C

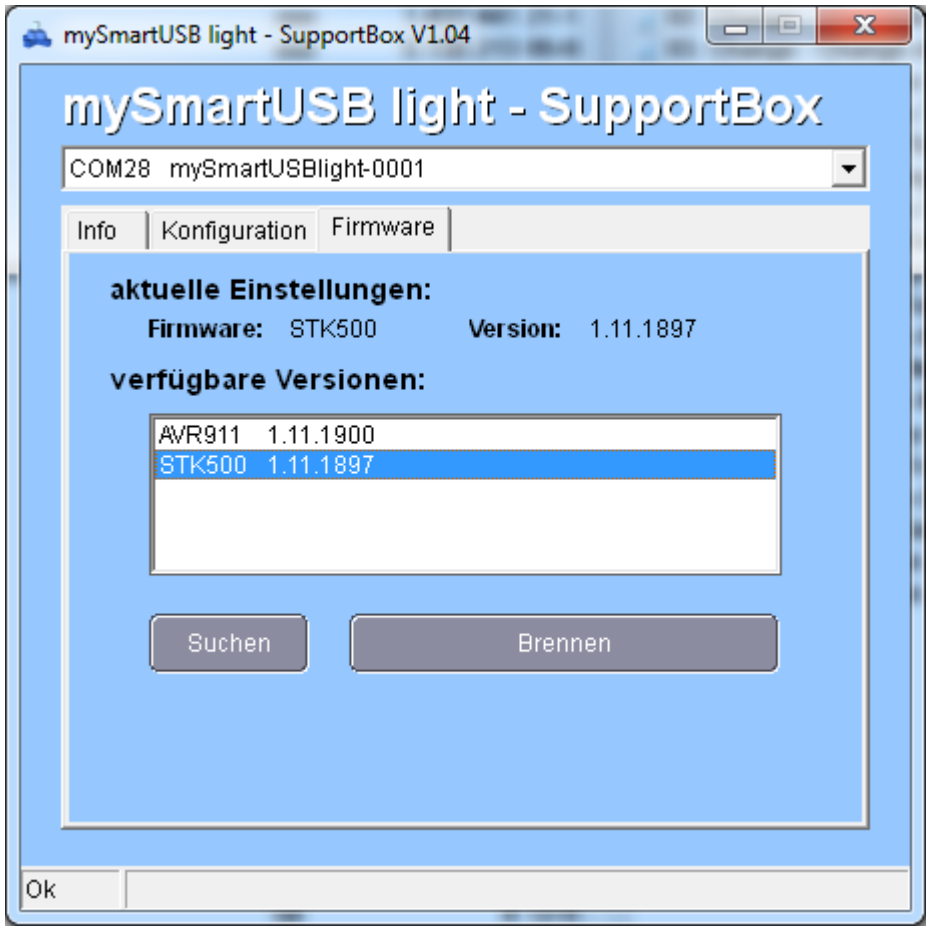
The BiPOM MINI-MAX/AVR-C board from www.bipom.com can be set into PROGRAM and RUN modes.
In programming mode, the board uses the STK500V2 protocol for program downloads.
Selecting the BiPOM MINI-MAX/AVR-C programmer and the COM port is sufficient.
Baud rate is fixed at 115200 baud.
The IDE automatically handles switching between PROGRAM and RUN modes.
If you press F4, the board will be put in PROGRAM mode, the firmware will be uploaded, and the board will be set back to RUN mode.

3.60.1.20 mySmartUSB Light

The mySmartUSB Light programmer is an affordable and versatile ISP programmer. It supports the AVR911 and STK500V2 protocols.
The mySmartUSB Light programmer is available form the MCS Webshop. It is an USB programmer that requires a virtual COM port driver. When your PC is connected to the internet, the driver will be installed automatically by Windows.

The programmer is either shipped with the AVR911 protocol or the STK500V2 protocol.
The support in BASCOM is for the STK500V2 mode.
MyAVR has a simple utility that you can use to check and/or change the firmware.
Download it here

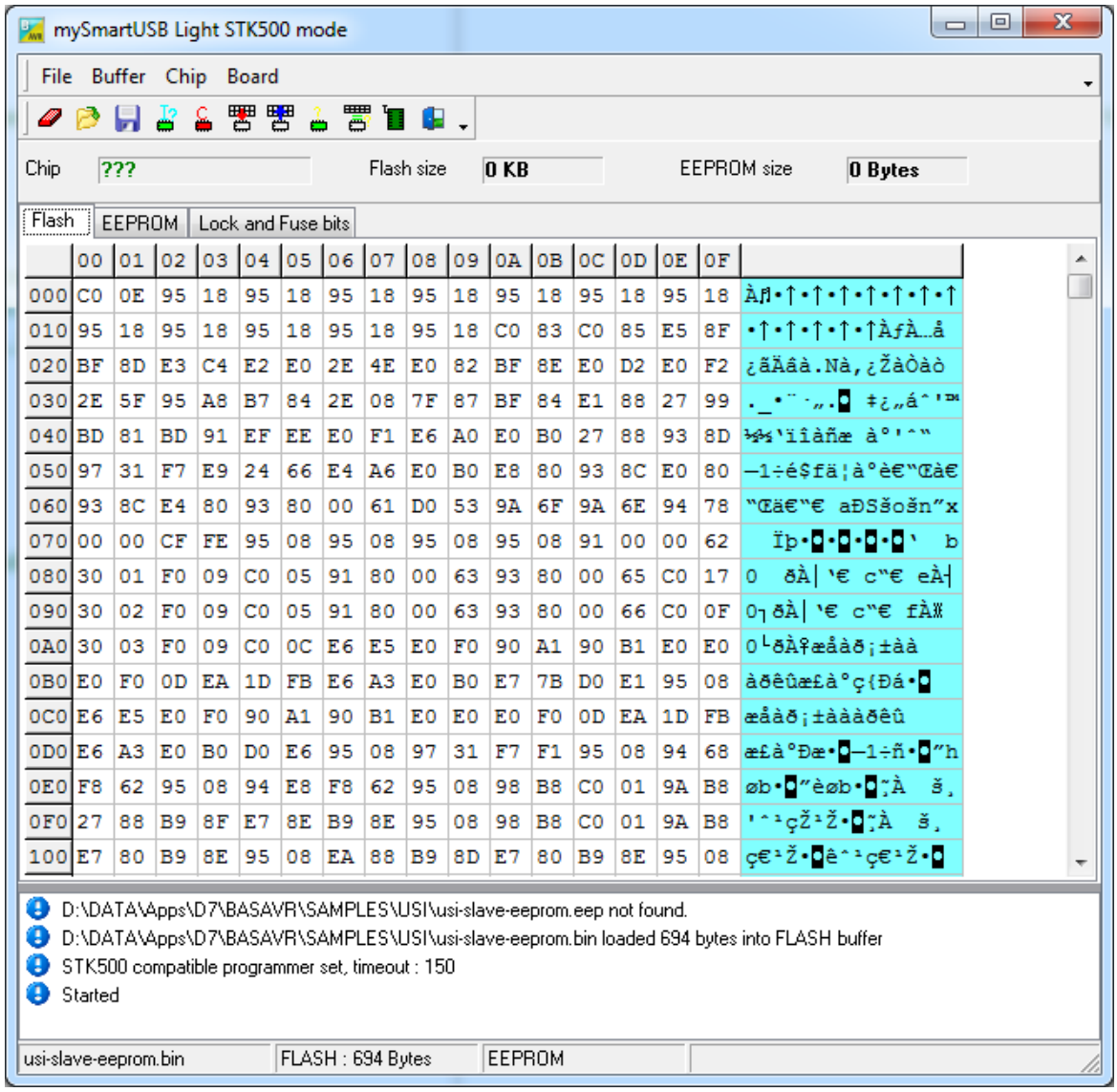
When you run the tool you get a window similar to this one:



The window above shows that the current firmware is STK500 which is OK.
When the version is AVR911, you can change it by selecting the STK500 1.11.xxxx in the list and click 'BRENNEN' (burning)

The tool also allows to set the voltage of the programmer to 3V or 5V.
And you can turn on the power while burning (this will use internal USB power)

The above options are available from BASCOM as well.
When you press manual program, the following window will be shown:



The usual options are available. Please read STK500 Programmer 170 for more info.

The MyAVR programmer has a special menu accessible from the Board menu.
Board, MyAVR, Voltage, 3V or 5V. This selects the output voltage of the programmer
Board, MyAVR, Power On Program. This option can be set and cleared. When set, the programmer will route power to the target circuit during programming.
Board, MyAVR, Board Power, turn on/off. These options can be used to power the target board while not programming.

When using the options to power the circuit, you should notice that this power is taken from the USB bus. You should take care that your circuit does not draw too much current.

For the manual see : http://www.myavr.info/download/produkte/mysmartusb_light/techb_mySmartUSB-light_de_en.pdf

3.60.1.21 UPDI Programmer

The MCS UPDI programmer is a serial based programmer.
You need to select 115200 BAUD and the COM port which is connected to the UPDI interface.
In version 2084 you can select up to 225000 baud. This is the maximum

recommended baud from microchip with the default clock.
In version 2086 the maximum baud of 1600000 is supported. Notice that only DA/DB processors support this speed.

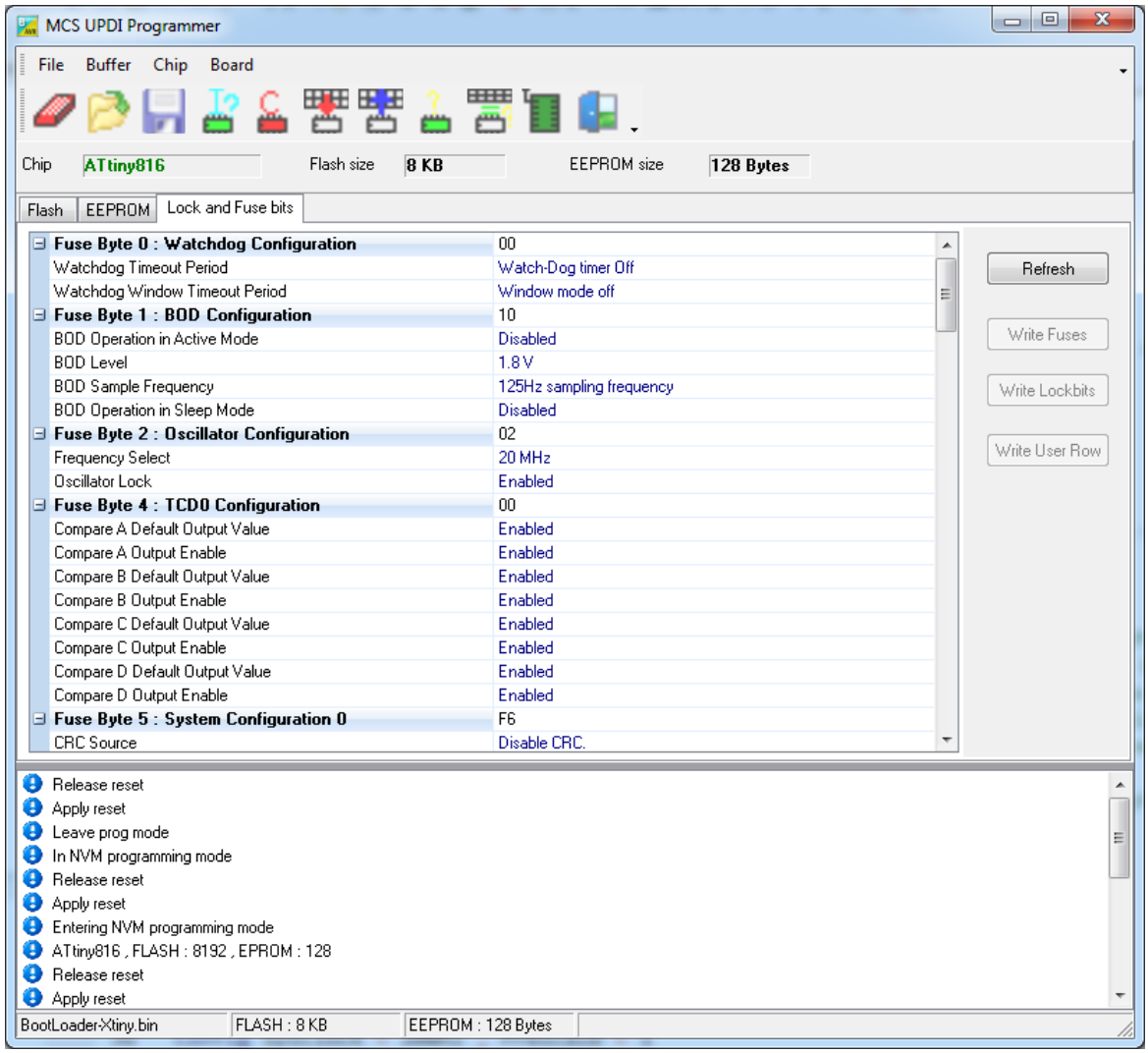
The UPDI interface is very simple : all you need is a TX, RX and a resistor.
Connect TX from the PC UART to a 4K7 resistor. The other side of the resistor is connected to the PC RX and to the UPDI pin of the processor.
We use DTR to switch the TX and RX from the PC to the processor. This allows to use the PC COM port to be used for serial communication and as a UPDI programmer.

Note : some modules will not give proper signals. A 1K resistor will bring better results.

Please notice that you need a MAX232 or other level converter between the PC communication pins to create the proper voltage level! Like the circuit shown below.

The programmer works similar as the other supported programmers : you can program the FLASH, EEPROM and the fuse/lock bytes

In version 2083 you can also write the fuse bytes.



When you change the values of a fuse the WRITE-FUSES button will be enabled.
When you change the value of the LOCK fuse, the WRITE-LOCK bits button will be

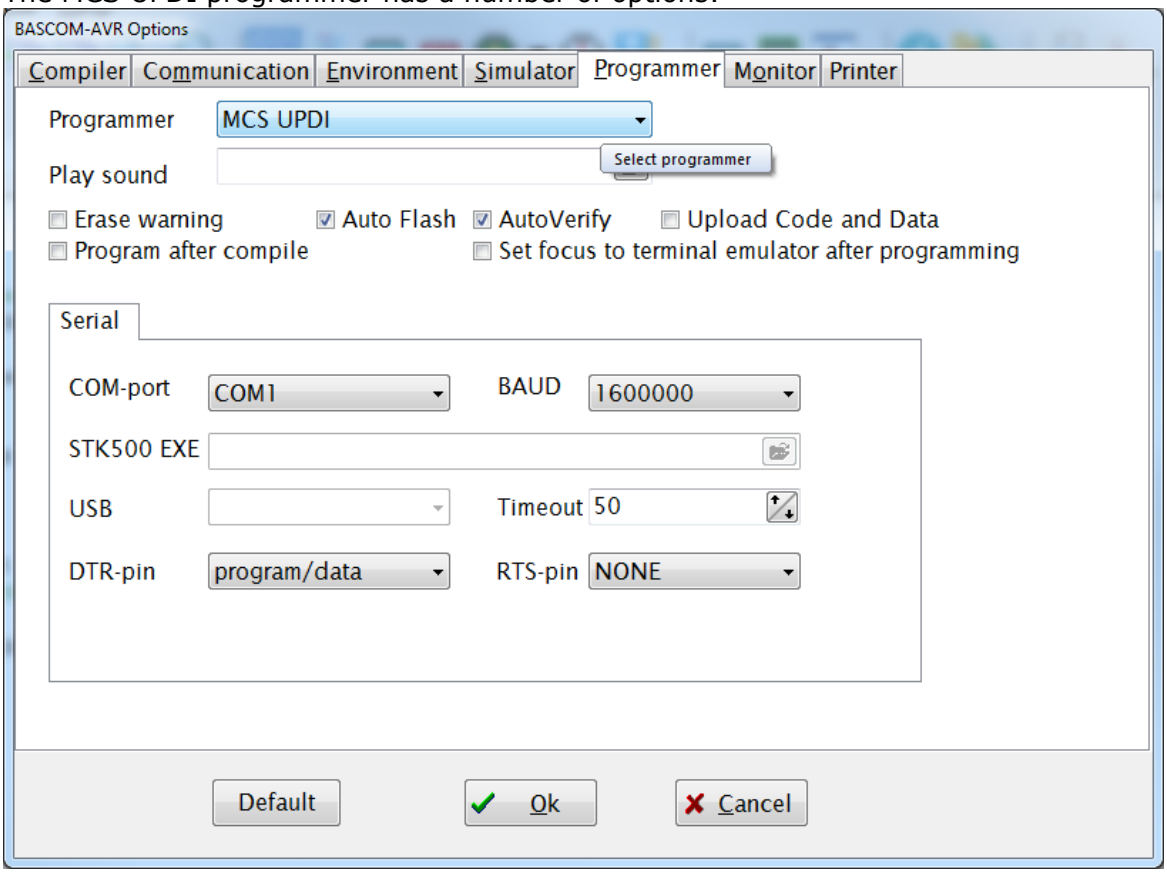
enabled.
When you change the value of the user fuses, the WRITE USER ROW button will be enabled.

When you write the fuses, the fuse values will be re-read (refreshed). And the same for the other fuses.

See CONFIG FUSES for information on how to automatic program fuse bytes.

Programmer Options

The MCS UPDI programmer has a number of options:



Since this is a serial programmer you need to select the COM port.
The BAUD also need to be selected.
115200 should always work
22500 also should work for all processors
The maximum speed for the XTINY platform is 900.000 (900 KB)
The maximum speed for the MEGAX platform is also 900 KB
The maximum speed for the AVRX (DA/DB) platform is 1.600.000 (1.6 MB)

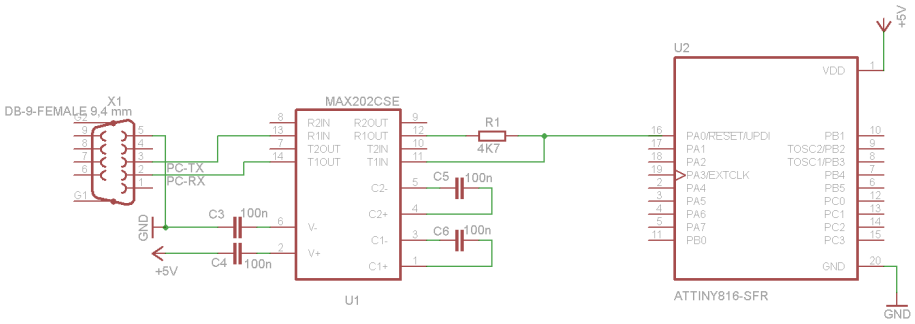
The timeout can best be set to 50.
Then there is an option to control what the DTR and RTS pins should do.
You can chose :
- NONE : DTR/RTS is not used
- program/data : DTR/RTS is used to switch between programming and normal mode. This way you can use a MUX and use the same TX/RX pins of your serial port for programming and data
- HV program : DTR/RTS is used to generate a pulse on the UPDI pin. You should include some circuit that inserts 12V using DTR line.

Since you have 2 pins you can chose DTR for switch between program/data and RTS for the 12V pulse.

Unlock

The UPDI prgrammer also has a Chip Unlock option. When the chip is locked, you need this option to fully erase the chip.

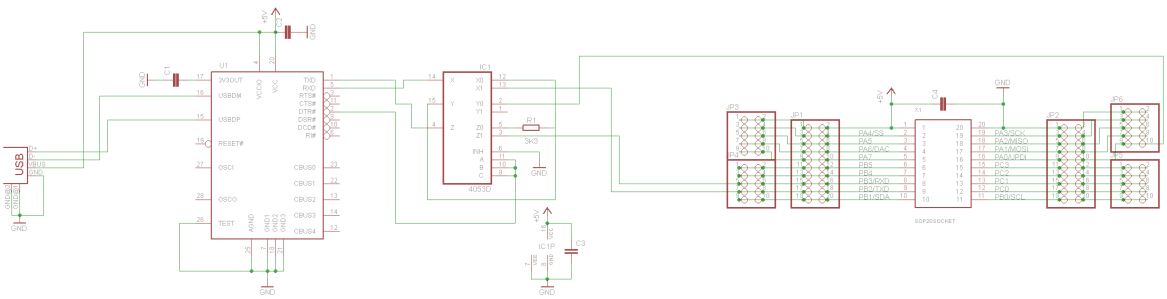
A typical connection for the UPDI programmer :



A MAX232 level converter will convert the RS232 levels to 5V.
The TX from the PC/max232 is connected with a 4K7(or 1K) resistor to the UPDI pin.
The RX from the PC/max232 is connected directly to the UPDI pin.

you can also use an USB virtual com port chip such as the FT232 or CP2102.

Using a serial port just for programming is a bit of a waste. Often you also like to have serial communications.
So a more practical programmer will switch the TX/RX lines between the UPDI pin and the TX-RX USART pins of the processor.



Notice that the USB circuit shown is not complete, you should check it with the chip of your choice like FT232RL, CP2102, etc. The main purpose of the USB part is to show the TX/RX and DTR pins.

The TX pin and RX pins are connected to a 4053 switch. This is an analog switch. The DTR line selects the XYZ-0 or XYZ-1 side of the switch.
The UPDI pin is also connected to a MUX switch. This simple circuit now switches between the UPDI mode and the TX and RX pins of the processor.
The BASCOM-UPDI programmer will automatically switch the DTR line.

Some processors have a dedicated UPDI pin. Other processors share this pin with a normal IO pin.
The RESET pin can also be dedicated or shared. When shared you must program the RESET function since this is not enabled by default.

A virtual USB COM port is used most likely since a DB-9 serial connector is not found on most modern PC/laptop.

FT232/CP2102

We have tested using the **CP2102** but FT232 should also work. As user EDC found out the FT232 requires some driver changes.
- Buffers need to be changed from 4096 to 64
- Latency need to be set from 16 to 2
For more information : mcs forum

User Feedback about USB chips

MCP2221A:
1.1 Bascom AVR does not work with the chipset MCP2221A.... no matter of every change in the circuit or in the UART-settings.

UM**FT230XB**-01:
Works but very slow
See also the info from EDC listed above.

CP2102 (MCS chip of choice)
- works perfect!

FT232RL
It works but it programs slow.
See also EDC settings.

CH340G
works perfect & fast

Comment from other user:
CH340 doesn't work.
FT232 works perfectly.

This is of course contradicting. Could be the driver, other settings?
MCS UPDI programmer is considered a nice free alternative to program the processor.
For better results you best get the Microchip SNAP programmer.

See Also

Using a BOOTLOADER 285 , MCS SNAP Programmer

3.60.1.22 MCS EDBG Programmer

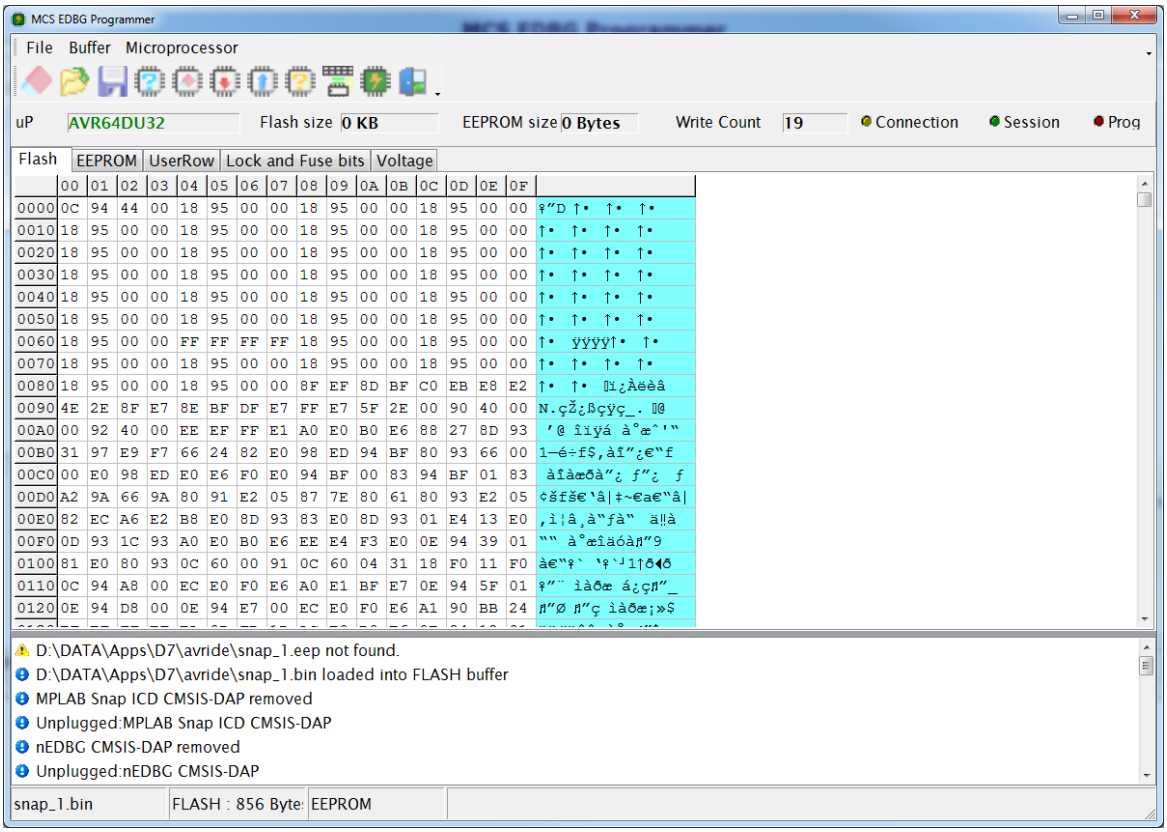
The MCS EDBG programmer supports the Microchip EDBG programmers. We named it MCS EDBG Programmer since we implemented the EDBG protocol in BASCOM-AVR.
So what is an EDBG programmer? EDBG is a protocol for programming and debugging.
The ones that BASCOM supports must work in USB HID device mode.
Originally the SNAP programmer was used for testing but Microchip made an update to the firmware using a different protocol which is not supported.
You can still use the SNAP programmer but you need to replace the firmware. This means doing the recovery mode procedure, then using AS7 to load the firmware for AVR.
But it does not run out of the box and for this reason we do not recommend it for beginners. The SNAP programmer is however cheap and supports all AVR programmer modes.

Programmers can be obtained from Microchip. It does not require an additional windows driver since it is a so called HID device which is always supported.
You need the AVR UPDI license in order to use this programmer in BASCOM. Without UPDI license you can not code for the UPDI processors anyway.
The programmer should supports all Microchip EDBG UPDI programmers that identify as an Atmel device.
We tested with SNAP programmer V1, JTAGICE3, AVR128DB48-CNANO KIT and ATTINY817 Xplained. The explained series have an integrated programmer.
The new curiosity : AVR64DU32 Curiosity Nano is a cheap test board from Microchip that also has an integrated programmer. You can separate some tracks and use it to program other processors as well.
Details can be found in the documentation from Microchip.

The goal of the EDBG programmer support was to support UPDI mode. Some programmers support ISP and PDI mode. But BASCOM does not support these modes yet.
The ISP mode is in beta. It works for the ISP curiosity series. But no extensive tests were done.
The PDI mode is in beta as well. It is tested on the xmega128A3. The DAT files are not adapted and thus it will not work correct on processors with a different memory configuration.

The curiosity programmer only supports UPDI mode. SNAP supports all modes but does not work out of the box.
The great thing about EDBG programmers/hardware is that the protocol supports hardware debugging. BASCOM will support this too in a future version.

The programmer interface looks familiar:



The FLASH TAB will show the content of the FLASH buffer. The content is loaded automatically when a BIN file exists of your project.
The EEPROM TAB will show the content of the EEPROM buffer. This is loaded automatically when an EEP file exists of your project.
The USERROW TAB will show the content of the USERROW buffer. This is loaded automatically when a USR file exists of your project.
The LOCK and FUSE Bits TAB is intended to program the various fuses.
The VOLTAGE TAB can be used to change the target voltage. But only when your programmer supports this.
The INFO window on the bottom will show if a programmer is found. In this case it is the SNAP ICD.

The yellow led will lit when there is a connection with a programmer.
The Session led will lt when there is a session active.
The Prog led wil lit when programming is active.
When using automatic programming the session is created and ended automatically.
When using manual programming you have the option (in Options, Programmer) to keep the session active. This to improve speed. On the other hand the programmer uses USB and the speed is great.

Each time you program the flash memory the write count is increased. The serial number is used and stored in the file named updiserials.prg. The MCS UPDI programmer uses the same file to keep track of writing.

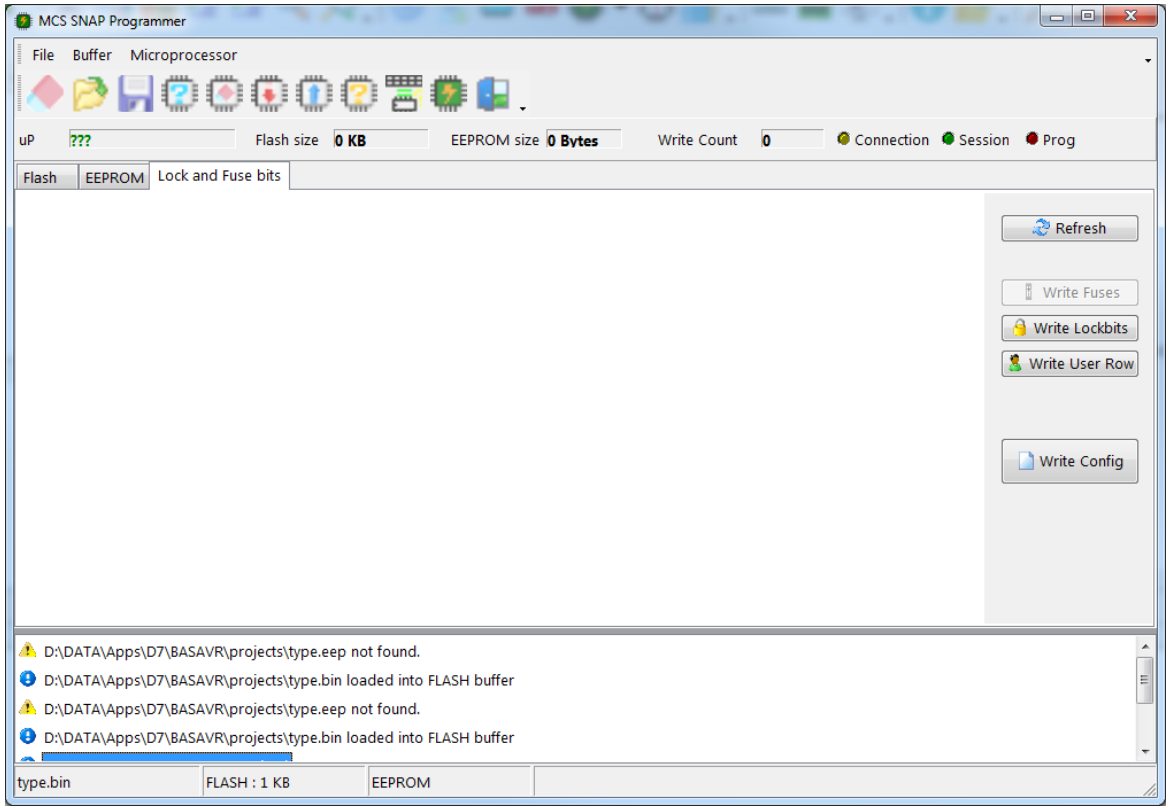
Menu Options

Option	Description
File	
Exit	Close programmer.

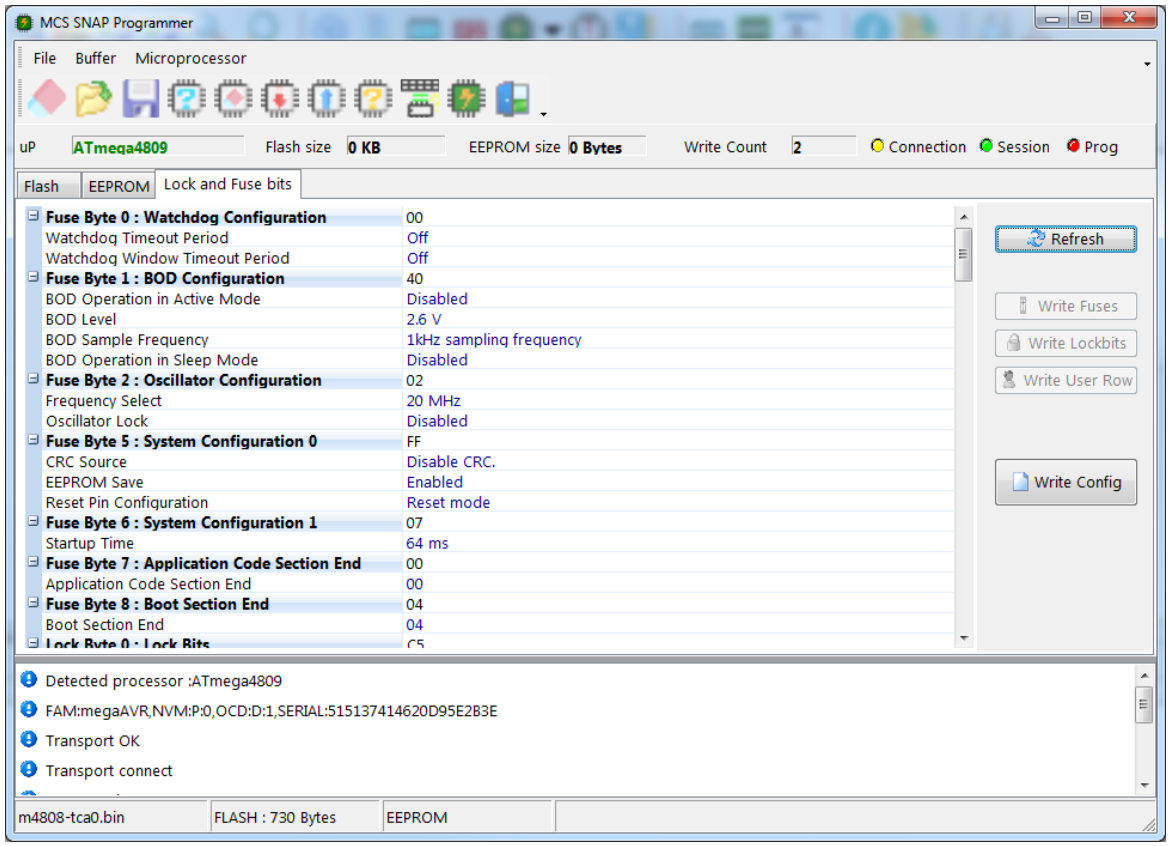
Buffer	
Clear	Clear buffer. Will put a value of 255 (FF hex) into each memory location. When the FLASH-TAB has the focus, the FLASH buffer will be cleared. When the EEPROM-TAB has the focus, the EEPROM buffer will be cleared. When the UserRow TAB has the focus the UserRow buffer will be cleared. 255 is the value of an empty memory location.
Load from File	This will show an open file dialog so you can select a binary file (BIN) or a HEX file in Intel HEX format. The file is loaded into the buffer. You can also manual edit the memory cells.
Save to File	Will save the current buffer to a BIN (binary) or HEX (hexadecimal Intel HEX) file.
Reload	Reloads the buffer from the file image. All programmers will always use the binary image to load files by default.
Microprocessor	
Identify	Will attempt to read the signature of the processor. When the signature is unknown(no DAT file available) or there is no chip or other error, you will get an error. Otherwise the processor name will be shown. It is important that you include $REGFILE with the used processor in your code. For example : $regfile = "AVRX64da64.dat"
Write buffer into processor memory	This will write the active buffer(FLASH, EEPROM or UserRow) into the processor.
Read processor memory into buffer	When the chip lock bit is not set you can read the FLASH , EEPROM or UserRow memory into the buffer.
Blank check	Checks if the chip FLASH , EEPROM or UserRow is empty.
Erase microprocessor	Erases the processor FLASH. It depends on the fusebits if the EEPROM is erased too. Normally the EEPROM is erased too but some processors have a fuse bit to preserve EEPROM when erasing the chip. A processor MUST be erased before it can be programmed. Otherwise depending on the value of the memory cell, the cell can not be programmed.
Erase EEPROM	Erases the EEPROM of the processor. The FLASH memory is not erased. When programming the EEPROM, all pages are automatically erased before programming.
Unlock Microprocessor	Unlocks the processor. This is a special ERASE option. When the processor is locked with its fuse bytes, you can not program the processor. You can only unlock/erase it. This will also erase the fuse bytes!
Verify microprocessor memory with buffer	Checks if the buffer matches the processor FLASH , EEPROM or UserRow.
Auto program	This will erase, and program the FLASH **and** EEPROM. **UPDI MODE** When config FUSES is used in the code with the options set to ON, the lock and fuse bytes will be programmed too. When there is no EEPROM image, the EEPROM will not be programmed.

	For automatic fuse programming consider this : Config Fuses = **On** , Lock = Off , Fuse0 = &H00 , Fuse1 = &H64 , Fuse2 = &H00 , Fuse5 = &HD9 , Fuse6 = &H07 , Fuse7 = &H00 , Fuse8 = &H08 In this case automatic fuse programming is ON and all mentioned fuses will be programmed. Since LOCK is OFF , there will be no locks set. While developing you best set the fuse options to OFF : Config Fuses = **Off** , Lock = **Off** , Fuse0 = &H00 , Fuse1 = &H64 , Fuse2 = &H00 , Fuse5 = &HD9 , Fuse6 = &H07 , Fuse7 = &H00 , Fuse8 = &H08 Remember, after locking a processor you need the UNLOCK option to be able to reprogram the processor. **ISP** and **PDI MODE** $PROG directive is used for ISP and PDI programming.

The Lock and Fuse bits TAB



By default the lock and fuse bytes are **not** loaded. You need to click the **REFRESH** button to load the values.



When you alter the value of a fuse, depending on the kind of fuse you alter, the WRITE FUSES button, WRITE LOCK BITS or WRITE USER ROW button becomes enabled.
Clicking the button will update the fuse value(s). And the Lock and Fuse bits are read again. Reading all values can take some time.

The WRITE CONFIG button will write the current Lock and Fuse bit value to the editor. That is : the value as read from the processor. This is NOT the value as you alter it. The idea is that you first test the settings. When all works, you use the WRITE CONFIG button which will write a line like this to the current editor position :

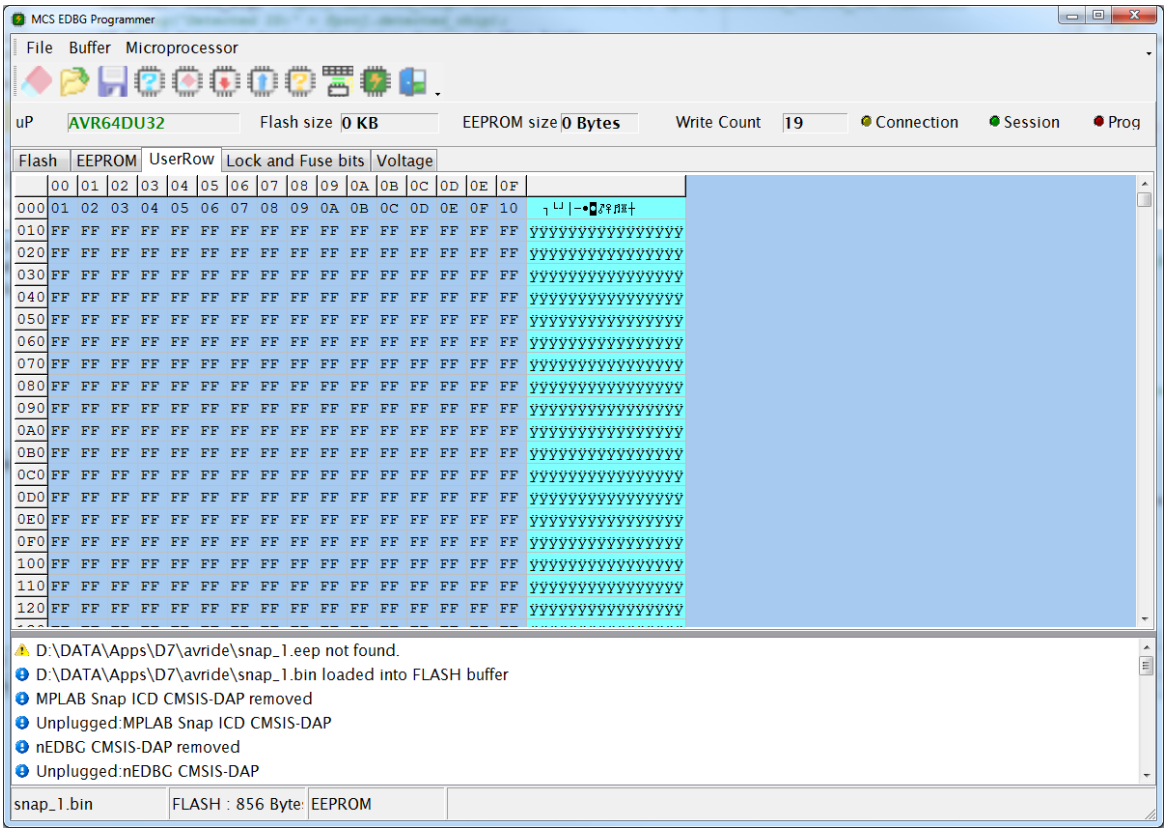
Config Fuses=**Off**,Lock=**OFF**,Fuse0=&H00,Fuse1=&H40,Fuse2=&H02,Fuse4=&H00,
Fuse6=&H07,Fuse7=&H00,Fuse8=&H04,UROW0=&H00,UROW1=&H00,
UROW2=&H00,UROW3=&H00,UROW4=&H00,UROW5=&H00,UROW6=&H00,
UROW7=&H00,UROW8=&H00,UROW9=&H00,UROW10=&H00,UROW11=&H00,
UROW12=&H00,UROW13=&H44,UROW14=&H00,UROW15=&H00,UROW16=&H00,
UROW17=&H00,UROW18=&H00,UROW19=&H00,UROW20=&H00,UROW21=&H00,
UROW22=&H00,UROW23=&H00,UROW24=&H00,UROW25=&H00,UROW26=&H00,
UROW27=&H00,UROW28=&H00,UROW29=&H00,UROW30=&H00,UROW31=&H00,
UROW32=&H00,UROW33=&H00,UROW34=&H00,UROW35=&H00,UROW36=&H00,
UROW37=&H00,UROW38=&H00,UROW39=&H00,UROW40=&H00,UROW41=&H00,
UROW42=&H00,UROW43=&H00,UROW44=&H00,UROW45=&H00,UROW46=&H00,
UROW47=&H00,UROW48=&H00,UROW49=&H00,UROW50=&H00,UROW51=&H00,
UROW52=&H00,UROW53=&H00,UROW54=&H00,UROW55=&H00,UROW56=&H00,
UROW57=&H00,UROW58=&H00,UROW59=&H00,UROW60=&H00,UROW61=&H00,
UROW62=&H00,UROW63=&H00

The CONFIG FUSES line only contains values that differ from &HFF meaning that one of the bits is set.
By default FUSES and LOCK are always set to OFF. This means that the programmer will not process the fuses and locks.

When your project is done and you want to program more processors you can change the FUSES=OFF into FUSES=ON. Then recompile so this info is written into the project PRG file. And now when you auto program the processor, the fuses are also programmed.
The same applies for the locks : when you want to lock the processor, you change LOCK=OFF into LOCK=ON, then recompile. When you program the processor, the lock byte(s) is also programmed.
When the lock byte is set, only an UNLOCK allows to reprogram the processor. UNLOCK will erase the processor. So all content (EEPROM too) will be erased.

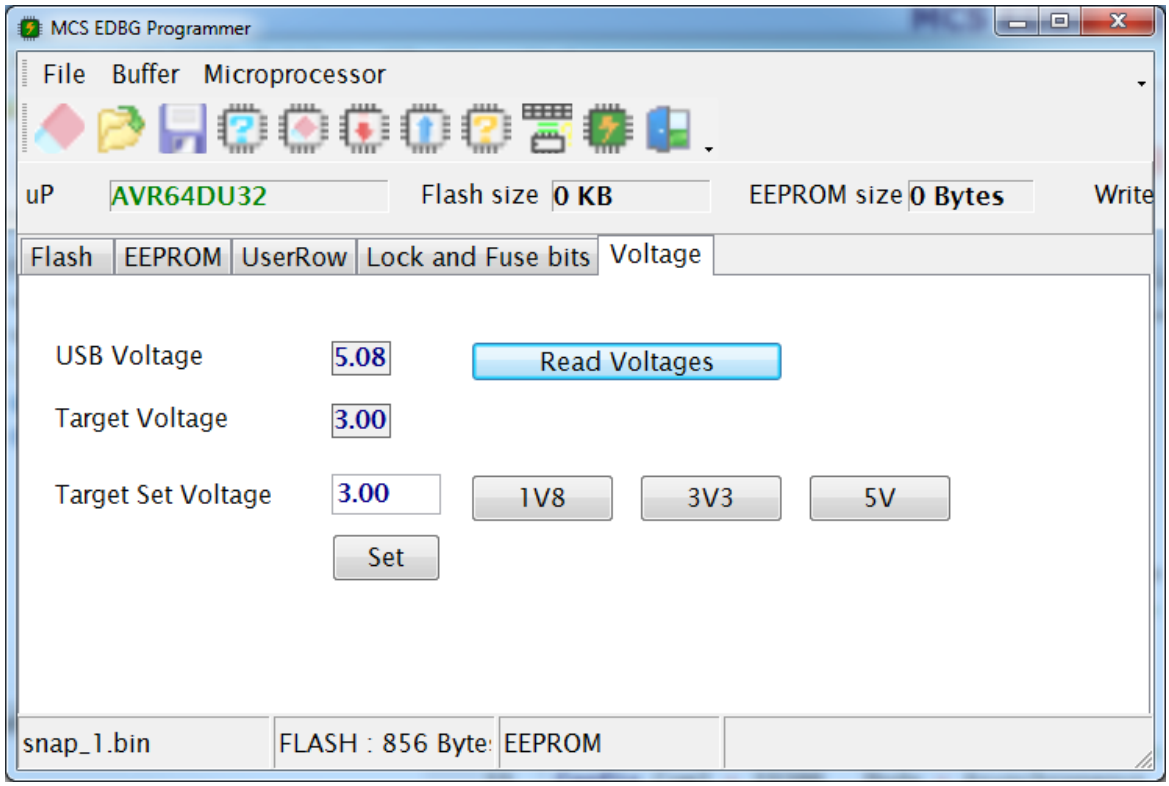
Some processors have a large UserRow memory. For this reason you can also program the UserRow using the UserRow TAB.

USERROW



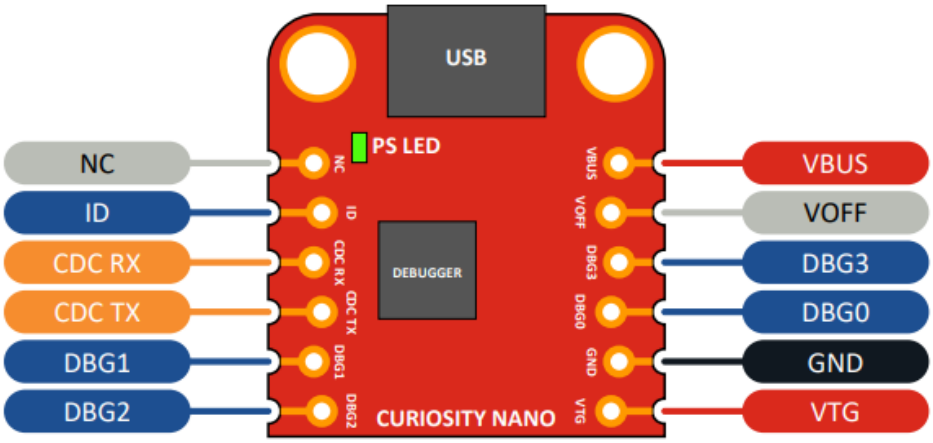
It works just the same as the EEPROM TAB.
When you create a binary .USR file the content is loaded automatically into the buffer.
Use the $USER 711 directive in combination with DATA lines to create the binary .USR file.

Voltages



The voltages TAB can read the USB voltage and the target voltage. When the programmer supports it you can also set the target voltages.
Use the 1V8, 3V3 and 5V preset buttons to set the voltage. Click the SET button to actual set the voltage.

Curiosity Nano Debugger Pinout



PIN	DESCRIPTION
NC	Not connected
ID	ID line for extension
CDC RX	USB CDC RX line
CDC TX	USB CDC TX line
DBG1	Debug clock line
DBG2	Debug GPIO0/SW0
VBUS	VBUS voltage for external use

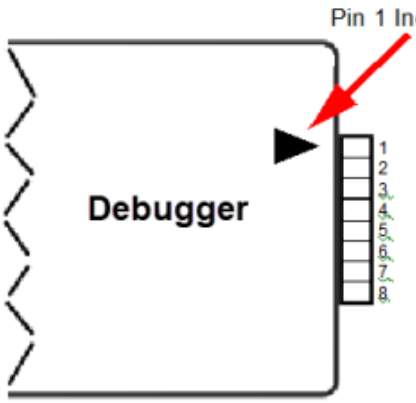
VOFF	Voltage Off input. Disables the target regulator and target voltage when pulled low. Make it low to activate.
DBG3	Reset Line
DBG0	Debug data line. Connect to UPDI of the target processor
GND	Common ground.
VTG	Target Voltage

For programming you connect GND to GND of the target. Connect VTG to the target processor when your target circuit does not need much power. Do not connect when your target has it's own power.
And connect DBG0 to the UPDI pin of the target processor. Optional you can connect DBG3 to the reset pin of the processor. Most processors have a fuse that must be programmer first in order to select reset for the pin. Otherwise it is just a normal pin.

The image shown is the programmer part of the curiosity board. You need to separate the PCB before you can use it to program other processors. Instruction on how to cut tracks you can find in the documentation of the curiosity board. By default the programmer/debugger is connected to the on board target processor. When you cut tracks or board you need to make wired connections in order to program in again.

The ISP mode was tested with both Xplained boards : Xplained 168PB and Xplained 328PB.
The ISP mode has a fixed clock frequency for most boards we tried. In AS7 you will also not find a way to change the clock.

SNAP programmer info



Debugger Pin Descriptions

1 = $\overline{\text{MCLR}}$
2 = VDD
3 = Ground
4 = PGD
5 = PGC
6 = Do not connect
7 = Unused for ICSP
8 = Unused for ICSP

Pinouts for Interfaces

The programming connector pin functions are different for various devices and interfaces. Refer to the following pinout tables for debug and data stream interfaces.

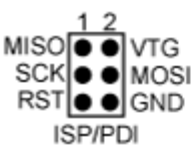
Note: Refer to the data sheet for the device you are using as well as the application notes for the specific interface for additional information and diagrams.

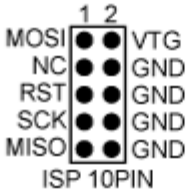
Table 10-4. Pinouts for Debug Interfaces

MPLAB Snap			DEBUG								
Connector	Pin #	Pin Name	ICSP (MCHP)	MIPS EJTAG	CORTEX® SWD	AVR® JTAG	AVR ISP(&DW)	UPDI	PDI	debugWIRE	TPI
	1	TVPP	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$						
	2	TVDD	VDD	VIO_REF	VTG	VTG	VTG	VTG	VTG	VTG	VTG
	3	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND
	4	PGD	DAT	TDO	SWO	TDO	MISO	DAT	DAT		DAT
	5	PGC	CLK	TCK	SWCLK	TCK	SCK				CLK
	6	TAUX	AUX			$\overline{\text{RESET}}$	$\overline{\text{RESET}}$		CLK	dW	RST
	7	TTDI		TDI		TDI	MOSI				
	8	TTMS		TMS	SWDIO	TMS					

For all supported mode you need to connect GND(ground) to pin 3 and VTG(vcc) to pin 2.
For UPDI you also need to connect the DAT to pin 4.
For ISP you need to connect MISO to pin 4, CLOCK to pin 5, reset to pin 6 and MOSI to pin 7.
For PDI you need to connect DAT to pin 4 and RESET to pin 6. While the table shows CLK for pin 6, it is actually connected to the RESET pin of the ISP connector.

The pinout of the 6- and 10-pin ISP headers are shown below:





 There is no option to select ISP, UPDI or ISP mode. The mode depends on the DAT file value from the $REGFILE 694 directive in your code.

See Also

$PROGRAMMER 690

3.60.2 LIBUSB

Using USB programmers in BASCOM-AVR

Please read this document completely before starting to install software.

Like every other USB device, an USB programmer requires a windows driver. Some programmers use drivers that are provided (built into) by windows. For example the KamProg uses the HID class and does not require an additional third party driver.

A programmer like the AVRISP mkII does need an additional driver. This device driver is installed when you install AVR Studio.
Studio is using device drivers from JUNGO.
When you plug in the programmer and Windows informs you that it requires a driver you know that you need to install a third party driver.
When Windows does not complain it will use a driver already available on your PC.

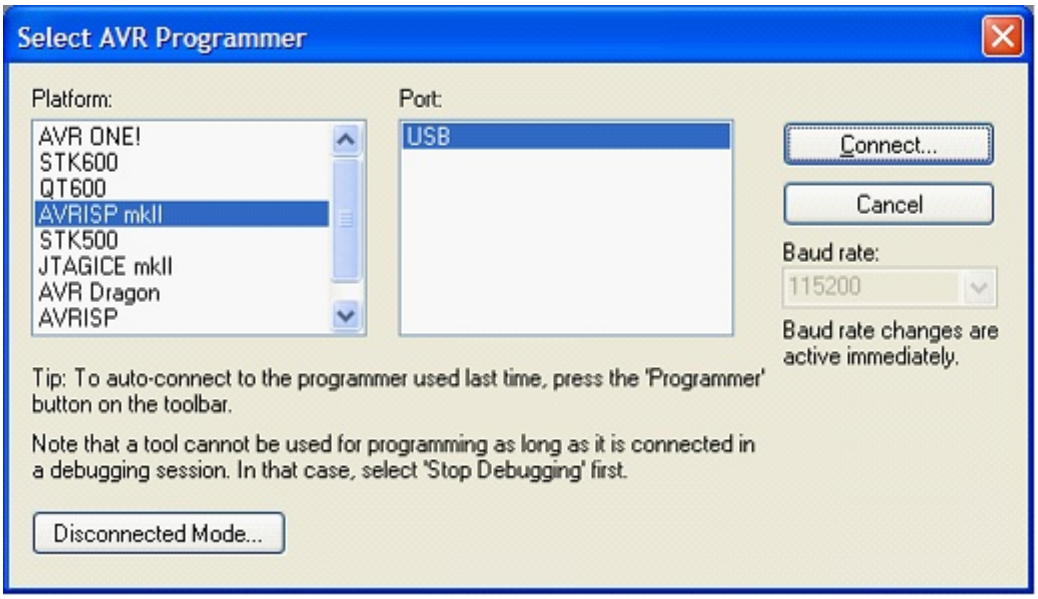
Most USB devices need software installed before you plug them in for the first time. In many cases there is a warning sticker that you should first install the software.

BASCOM uses LIBUSB to access USB devices. LIBUSB is available as a device driver or as a filter driver.
When your device is using a device driver you must access the device with a filter driver.
Some devices do not have a vendor supplied driver (USBASP programmer) and those require a device driver.

Scenario one : you have a 32 bit or 64 bit OS and have a product that uses a device driver.

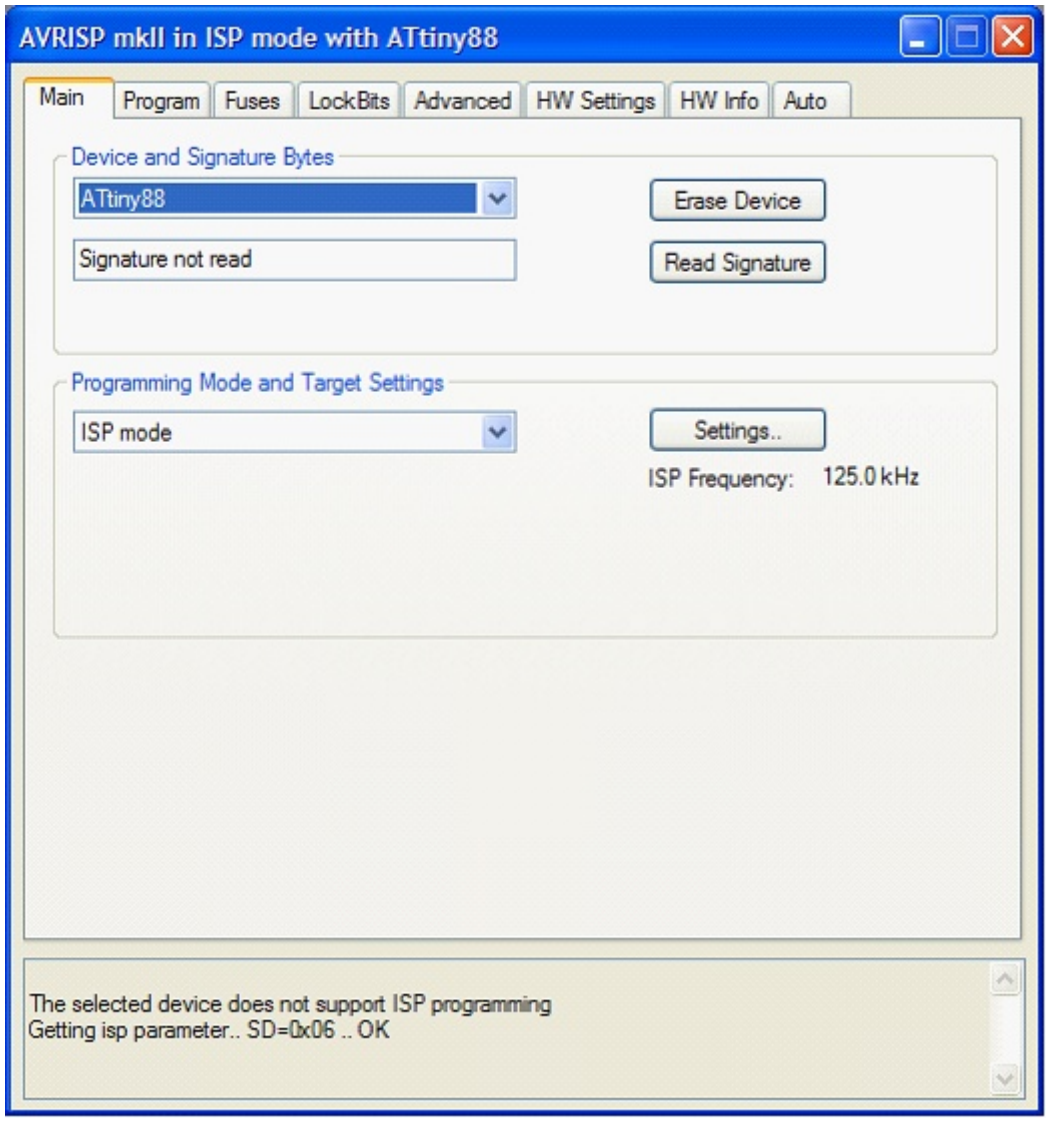
In this example we use the AVRISP mkII that is supported by AVR Studio. When you do not have AVR Studio installed you can download it from Atmels website for free. The original programmer comes with a CD-ROM too. But many imitation/self build devices exist that do not come with a CD. For those you need to download and install AVR Studio.

The next step is to plug your programmer, and see if it works with AVR Studio. Windows will recognize it, and install the device driver.
When windows is ready, press the connect button in Studio.



If you open Studio, and press the CON(nection) button, the window shown above will open.
Now select your programmer, in this sample AVRISP mkII and press Connect

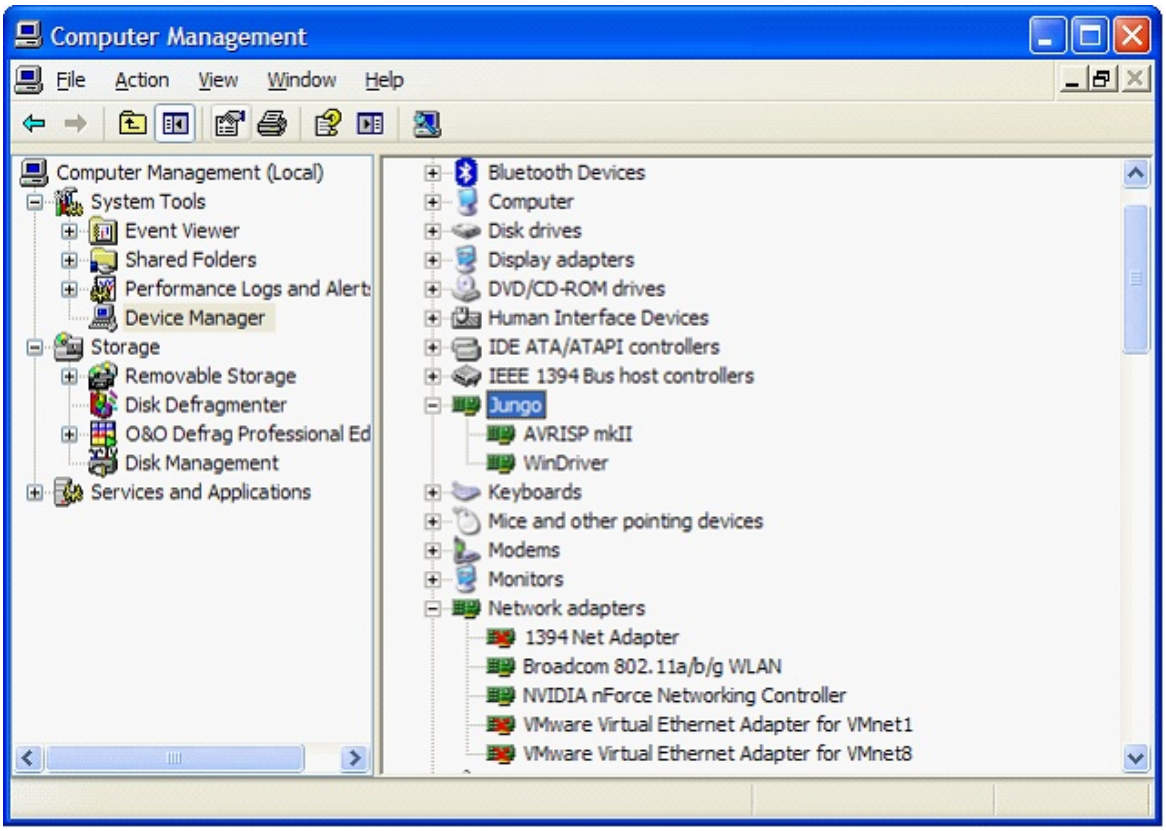
When it functions, a new window will open



You can select the device, the programming mode and ISP frequency. This frequency should be 125 KHz (or better said, should not exceed a quarter of the chip oscillator frequency).

When you do not get this window but you return to the connection window, it means your programmer is not working.
You have to solve this first before you can continue.
The programmer will only work in BASCOM when it functions with the original software!

In the windows device manager, you can find this info: (right click Computer, select manage, and chose device manager)



The screen above shows the JUNGO usb driver which Atmel AVR Studio uses and the AVRISP mkII driver for the AVRISP mkII.
If you install AVR Studio with the USB drivers, it will install JUNGO and the WinDriver. The AVRISP mkII entry you only get when you plug the programmer.

To make it work with BASCOM, you need to install LIBUSB. LIBUSB is used by many different programs. Atmels FLIP is using it too. So there is a big change that it is available on your system already.
You can install **LIBUSB** as a **FILTER** driver or a **DEVICE** driver.
We install the FILTER driver, so we can use the programmer with Studio AND bascom.

Before you install LIBUSB it is a good idea to make a restore point.

When installing the USB driver, disconnect ALL USB devices. Obvious, you can not install from an USB flash drive since this is an USB device as well.

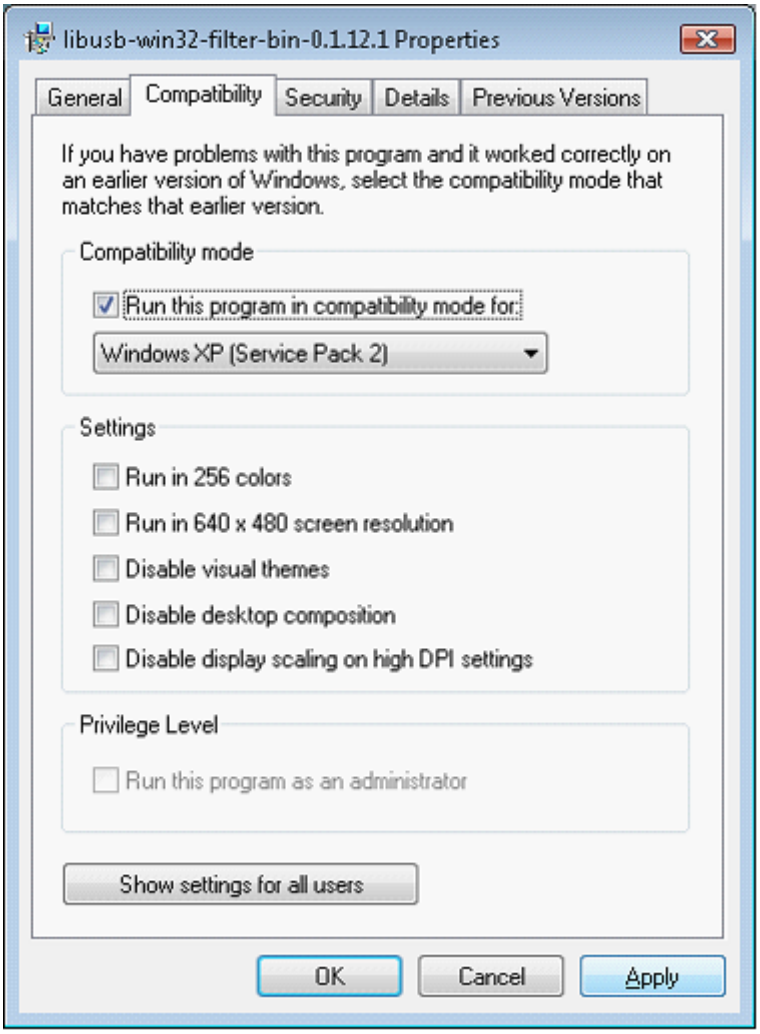
You can read about LIBUSB and download it from :
http://sourceforge.net/apps/trac/libusb-win32/wiki
The last version is :
http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases/1.2.4.0/libusb-win32-devel-filter-1.2.4.0.exe/download

Notice that this an executable you can install. You MUST have ADMIN rights when you install this executable.

After LIBUSB has been installed you can test if it is functional.

- Look in the Program Files\LibUSB-Win32 folder (also named on Windows-7 64 bit !!!)
  You will find a sub folder named **bin** which contains a number of executables.

- Run the **testlibusb-win.exe** application. When LIBUSB is functional you will see a screen with all USB devices.
When it does not work, try to install again with compatibility mode set to XP SP2.
Do this by selecting the the setup exe file properties, and select 'Compatibility'.



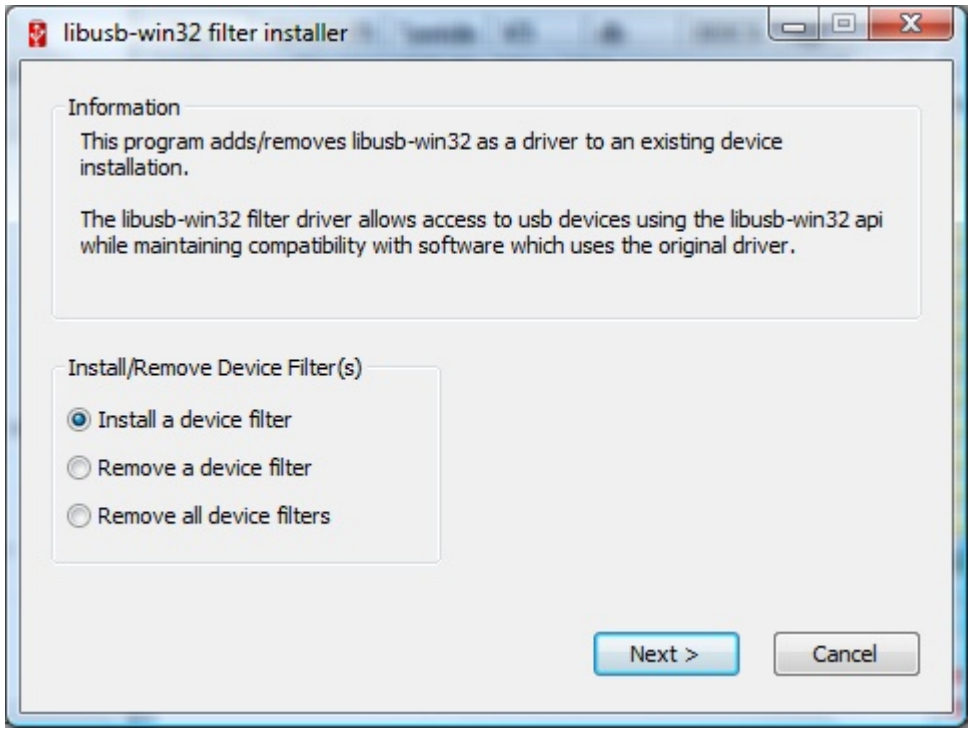
Click Apply and/or OK. And run setup again.

On Windows 7 - 64 bit, this was NOT required.

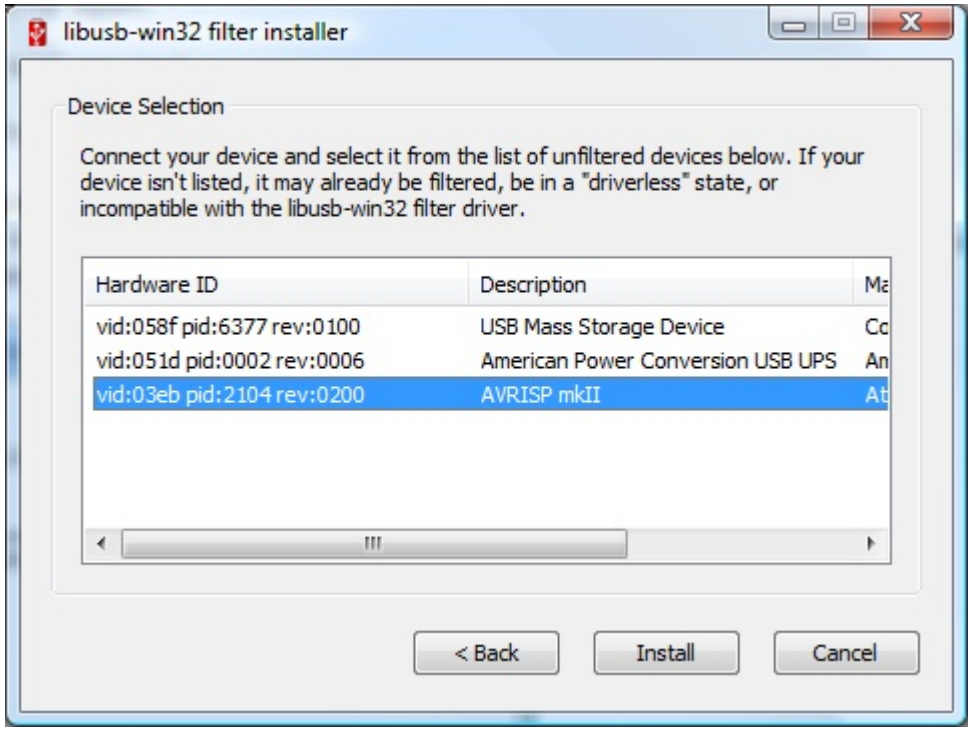
Once the testlibusb-win.exe works, you can continue to the next step.

Install the filter driver for the device

You need to install a filter driver for your programmer. Each different programmer requires it's own filter driver. So you must repeat these steps if you have different programmers.
- Plug in your programmer if it was not plugged in yet
- Run the **install-filter-win.exe** application from the BIN folder.
- You will see this window:

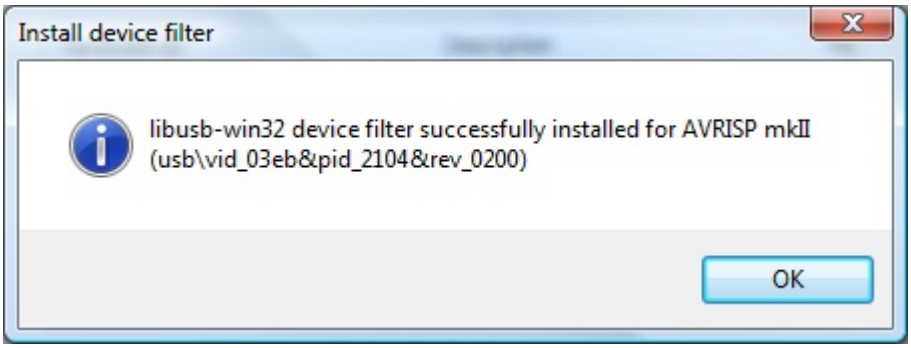


Select 'Install a device filter' and press Next.



Select the programmer and press Install.

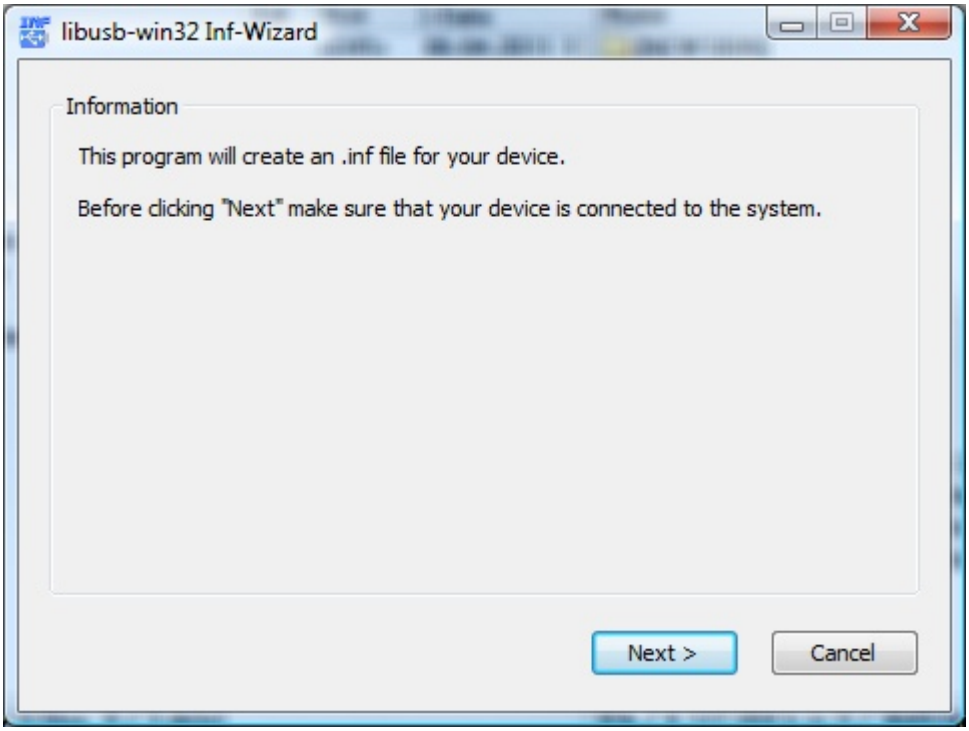
After some moments, you will get a confirmation:



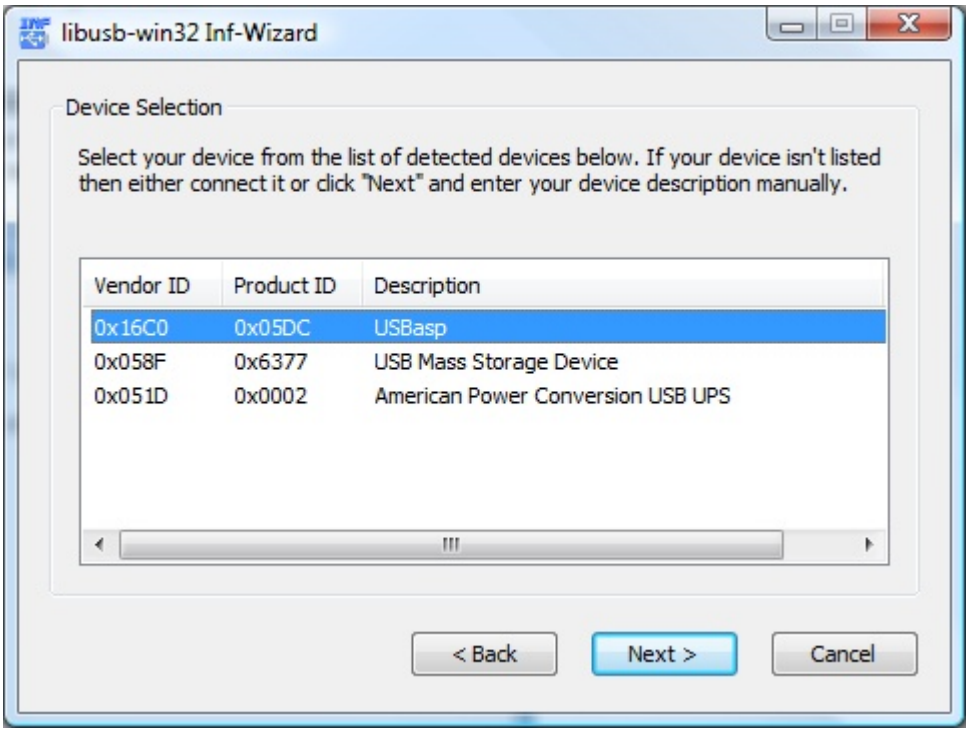
Now the programmer will work in BASCOM. Just select the proper programmer, and timeout of 100 ms. You can try lower time outs too to make it quicker. When you get errors, increase the time out. 100 ms should do for all programmers.

Scenario two: you do not have a device driver.

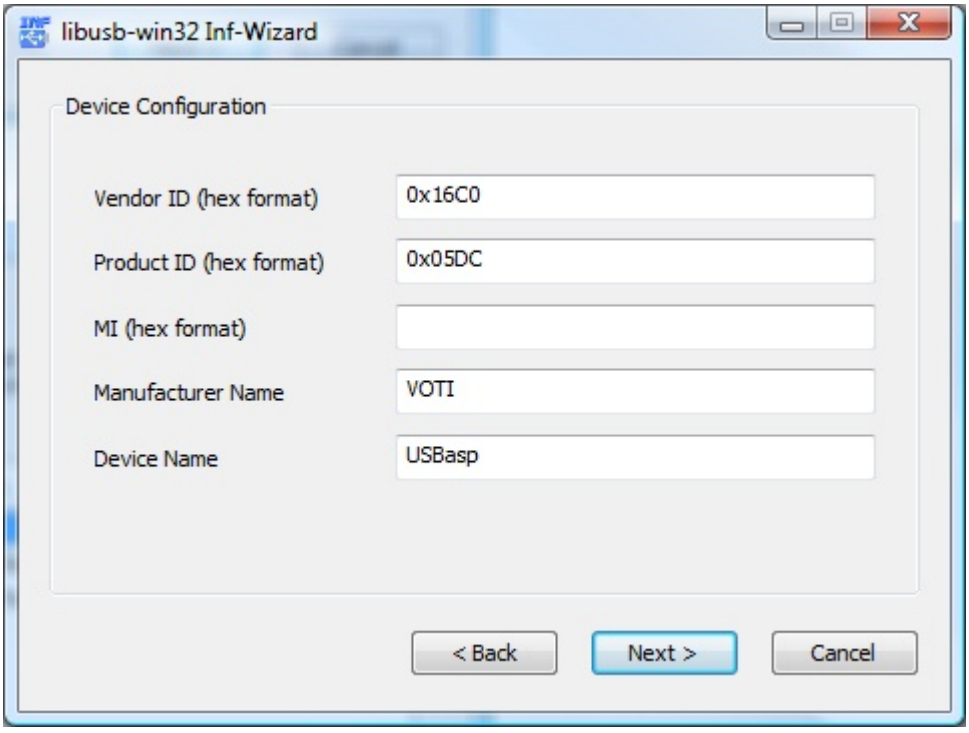
In this case you can follow scenario one till the filter driver installation.
Instead of running **install-filter-win.exe** , you will run **inf-wizard.exe**.



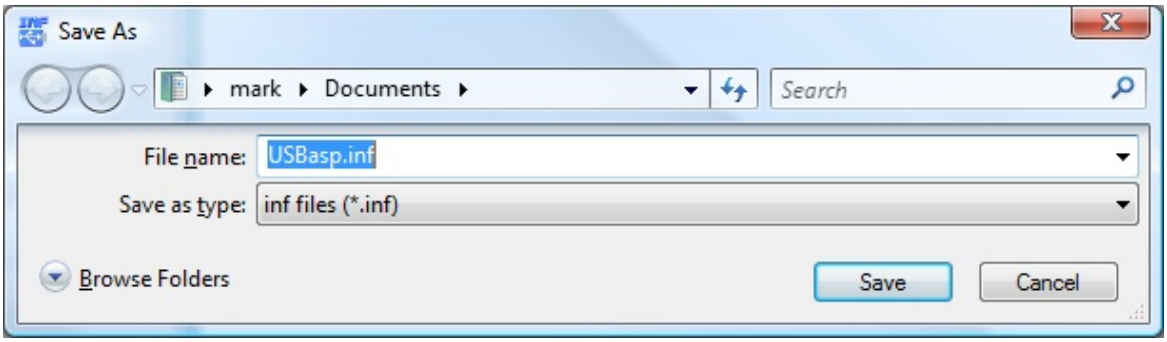
Press Next. And the following window will be shown.



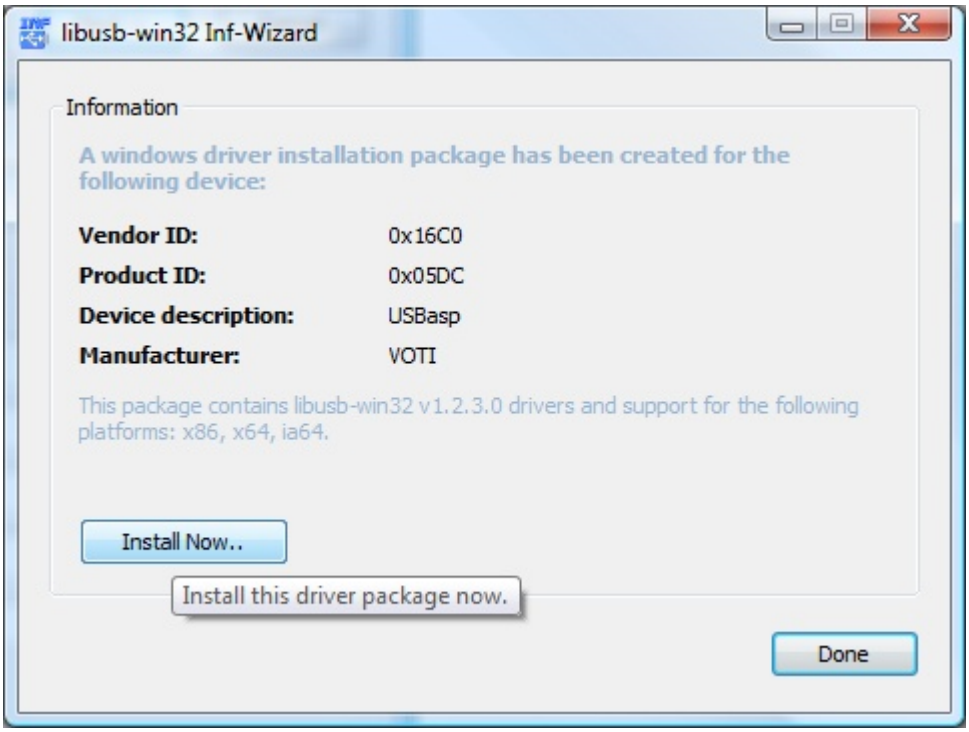
As you can see, the USBASP was inserted in this sample. Select it (or your programmer) and press Next.



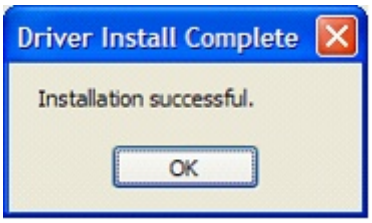
Press Next again and select a folder to store the device driver files. These files are required to install the device.



After you have saved the files, you have the option to install the driver. Press Install Now.. button to do so.



When ready :



Final note

The USB-ISP programmer form EMBUD, uses drivers from FTDI. It does not require LIBUSB.
The Kamprog programmer from KAMAMI uses a HID class and does not require LIBUSB.

Some devices gave a problem in 1.2.3.0. This problem is solved in 1.2.4.0.
http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases/1.2.4.0/libusb-win32-devel-filter-1.2.4.0.exe/download

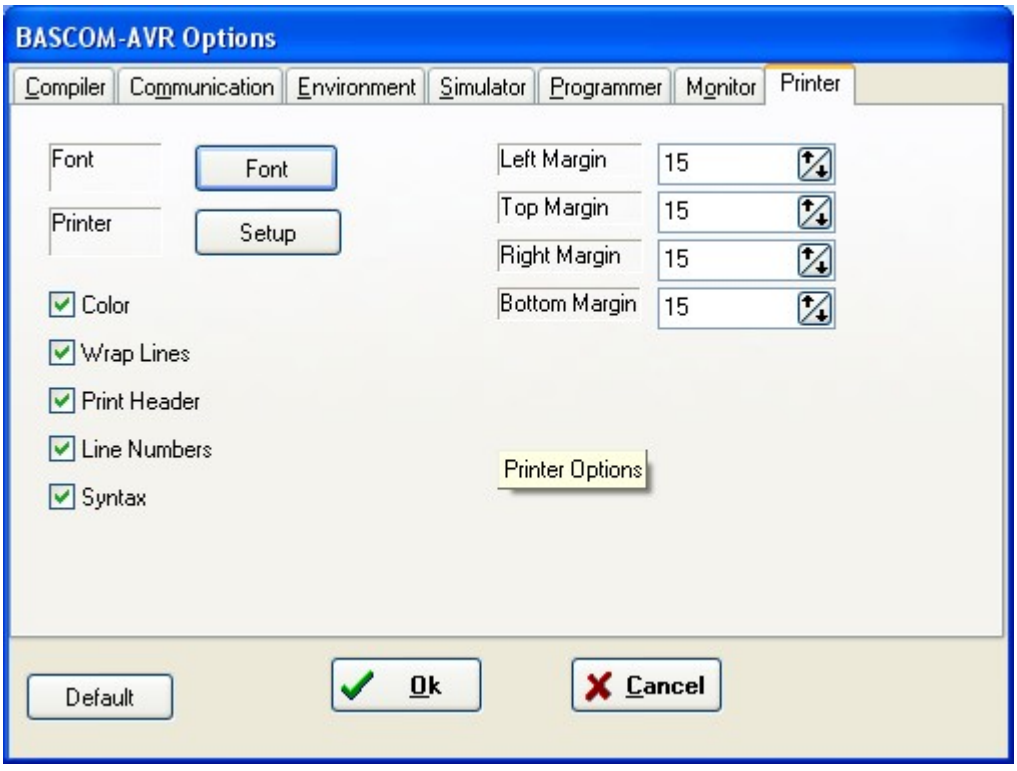
3.61 Options Monitor

With this option you can modify the monitor settings.

OPTION	DESCRIPTION
Upload speed	Selects the baud rate used for uploading
Monitor prefix	String that will be send to the monitor before the upload starts
Monitor suffix	String that us sent to the monitor after the download is completed.
Monitor delay	Time in milliseconds to wait after a line has been sent to the monitor.
Prefix delay	Time in milliseconds to wait after a prefix has been sent to the monitor.

3.62 Options Printer

With this option you can modify the printer settings.



OPTION	DESCRIPTION
Font	Printer font to use when printing
Setup	Click to change the printer setup
Color	Will print in color. Use this only for color printers.
Wrap lines	Wrap long lines. When not enabled, long lines will be partial shown.
Print	Print a header with the filename.

header	
Line numbers	Will be the line number before each line.
Syntax	Enable this to use the same syntax highlighting as the editor
Left margin	The left margin of the paper.
Right margin	The right margin of the paper.
Top margin	The top margin of the paper.
Bottom margin	The bottom margin of the paper.

3.63 Options Select Settings File

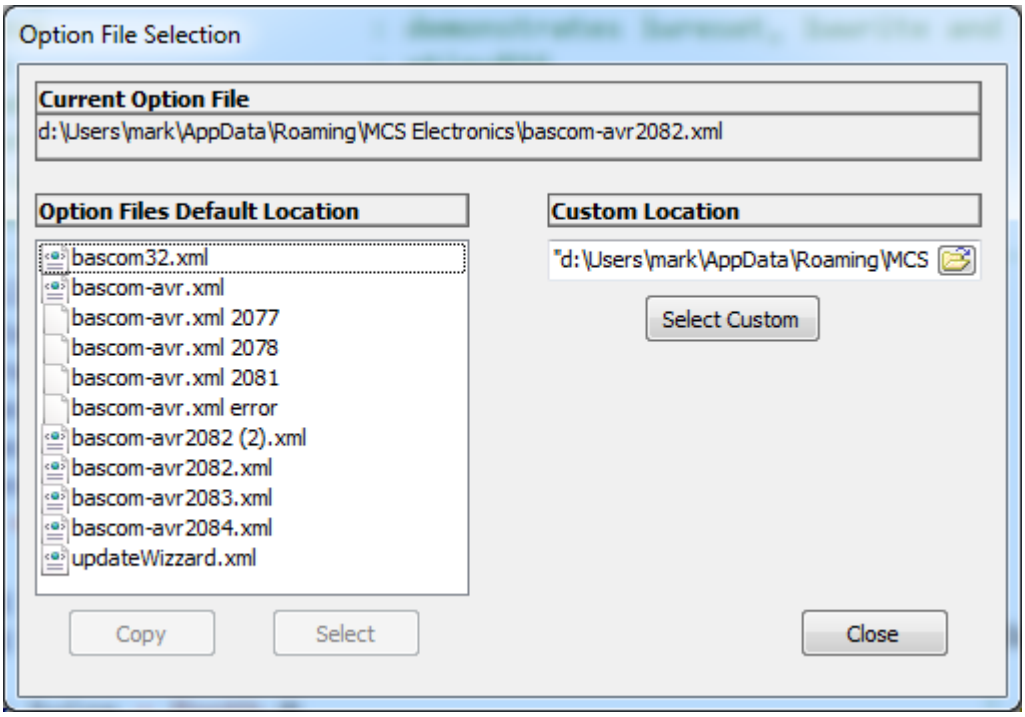
The options are stored in a file named bascom-avrxxxx.xml
The xxxx represent the version. For example 2083.
When you click Help, About, you can click the XML data link that will show you where the file is stored.

In order to make it possible to use different settings file, you can rename these files.
You can select the actual settings file using this menu option.
When you select this option, the status bar will show the current selected file.
This file will also be loaded when you run bascom.

A File Dialog will open and show all XML files. You need to select a bascom-avr xml file.

You can use this option after you have performed an update. Each update will have its own settings file. That way you can use multiple versions that each have their own settings file.

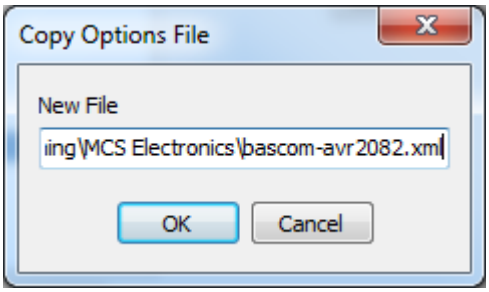
The following dialog window is shown :



The current option file is show : d:\users\mark\AppData\Roaming\MCS

Electronics\bascom-avr2082.xml
This location is also shown when you click the XML data folder link in the Help, About window

At the left you can view all the xml files.
When you select a file, the COPY and SELECT buttons will be enabled.
By using the COPY button you can make a backup of the selected option file.



You must give the file a different name. When the file is created, it will be shown in the list.

When you use the SELECT button, you select the new option file. This is reflected at the CURRENT OPTION FILE value in the top of the window.

The CUSTOM location can be used to store your option file at a custom location. You can enter a file and location or use the button to browse to the file.
You need to click the SELECT CUSTOM button to select this file as the new custom option file.

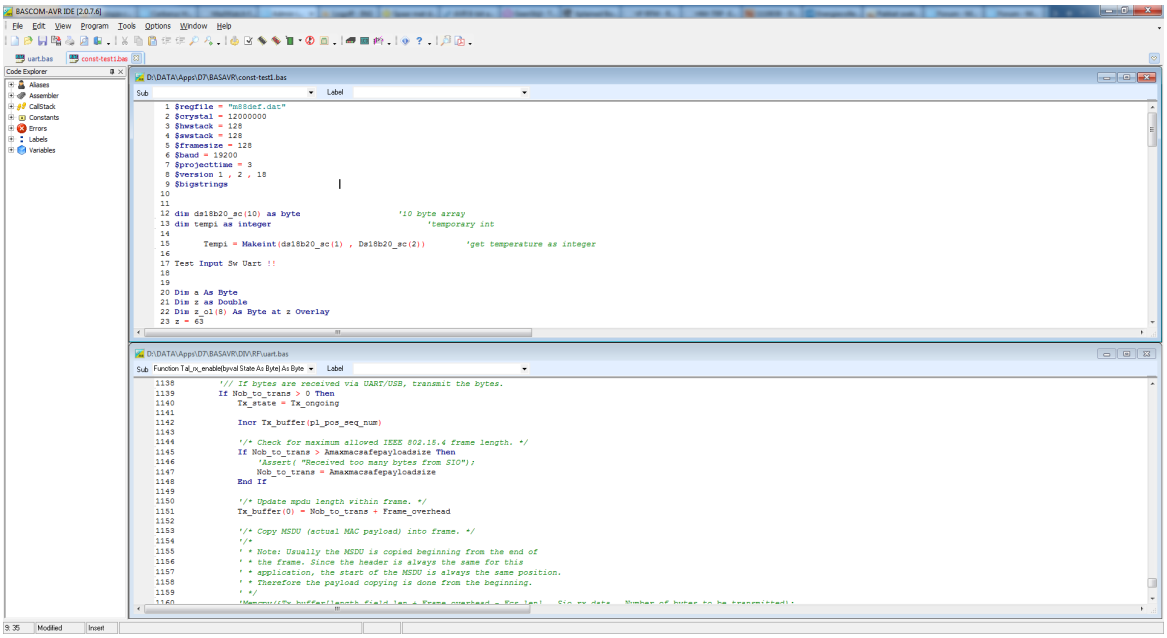
Click the CLOSE button to close the window.

3.64 Window Cascade

Cascade all open editor windows.

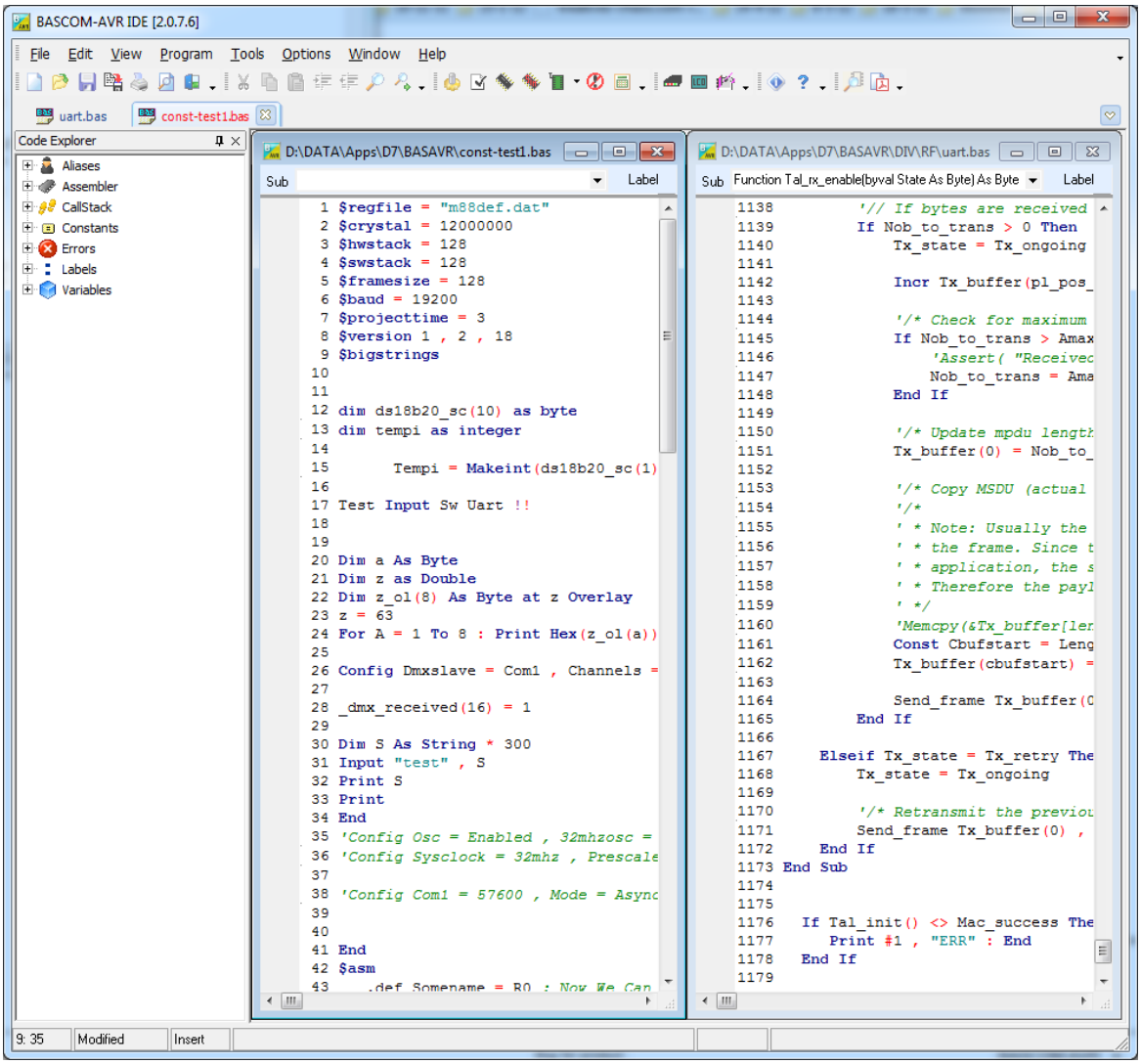
3.65 Window Tile

Tile all open editor windows horizontally.



3.66 Window Tile Vertically

Tile all open editor windows vertically.



3.67 Window Arrange Icons

Arrange the icons of the minimized editor windows.

3.68 Windows Maximize All

Maximize all open editor windows.

3.69 Window Minimize All

Minimize all open editor windows.

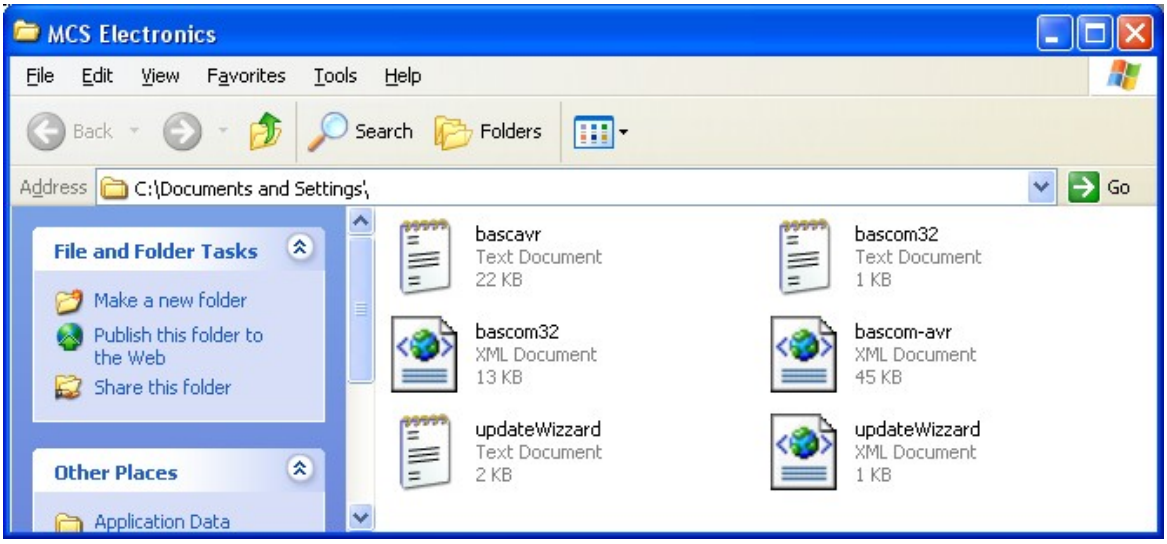
3.70 Help About

This option shows an about box as shown below.

Your serial number is shown on the third line of the about box.
You will need this when you have questions about the product.

The compiler and IDE version numbers are also shown.

When you click the App data dir link, the folder which contains the BASCOM settings will be opened:



It contains the bascom-avr.xml file with all settings and the bascavr.log file. When you need support, you might be asked to email these files.

When you need support, also click the Copy-button. It will copy the following info to the clipboard, which you can paste in your email :

Dont forget that Serial numbers should not be sent to the user list.
Make sure you sent your email to support and not a public list !

Compiler version :1.11.8.3
IDE version :1.11.8.5
Serial number :XX-XXXX-XXXXX
Windows OS :Microsoft Windows XP
Windows SP :Service Pack 2
Explorer :7.0.5730.11
Company :MCS
Owner :Mark Alberts
Windows dir :C:\WINNT
App data dir :C:\Documents and Settings
System dir :C:\WINNT\system32

When you click the support link, your email client will be started and an email to support@mcselec.com will be created.

Click on Ok to return to the editor.

3.71 Help Index

Shows the BASCOM help file.

When you are in the editor window, the current word selected or by the cursor will be used as a keyword.
Notice that when the help window is small, you might need to make the help window bigger to show the whole content.

 The help contains complete sample code and partial sample code.
In all cases the samples are shown to give you an idea of the operation. When trying a program you should always use the samples from the SAMPLES directory. These are updated and tested when new versions are published. The (partial) samples are not all updates, only when they contain errors. So the samples from the help might need some small adjustments while the samples form the SAMPLES dir will work at least on the used chip.

3.72 Help MCS Forum

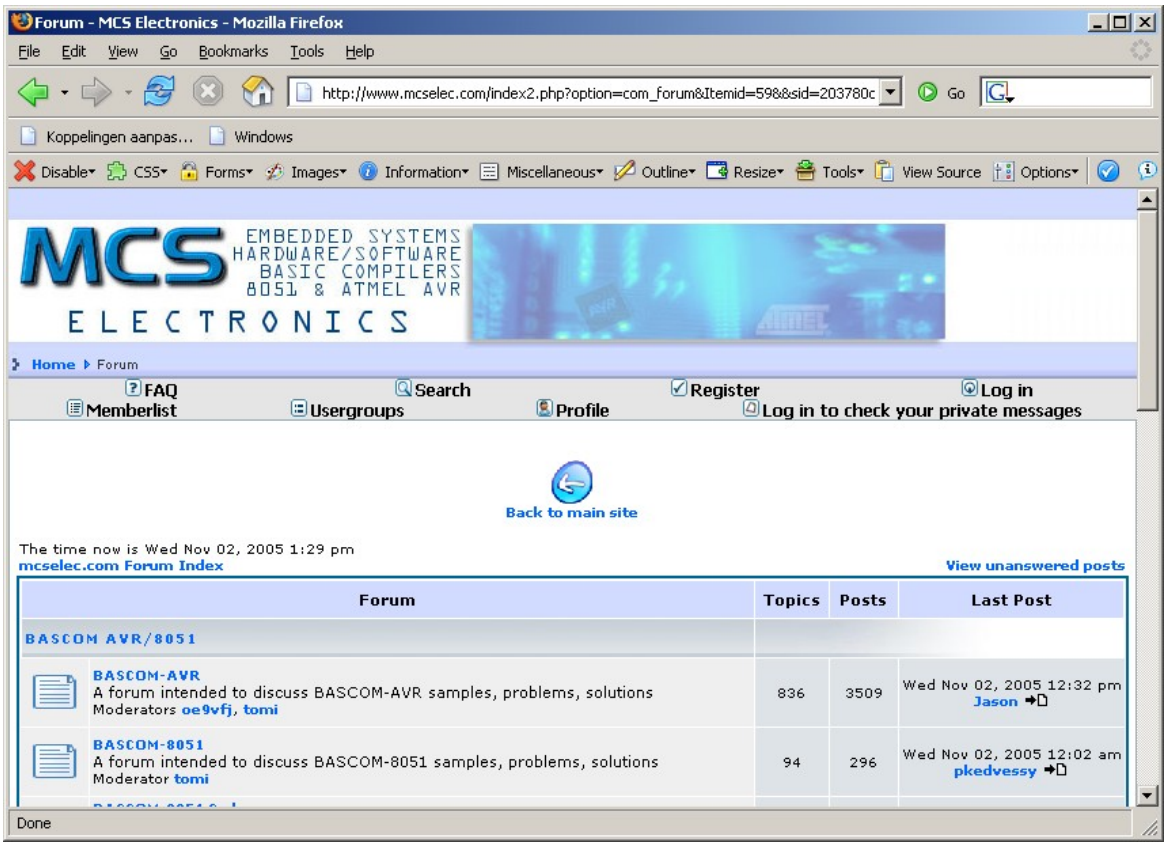
This option will start your default Web browser and direct it to http://www.mcselec.com/index2.php?option=com_forum&Itemid=59

This forum is hosted by MCS Electronics. There are various forums available. You can post your questions there. Do not cross post your questions on multiple forums and to support.

The forum is available for all users : demo or commercial users.
Note that everything you write might be on line for ever. So mind your language.

Users of the commercial version can email MCS support.

The forum allows uploads for code examples, circuits etc.
If you try to abuse the forum or any other part of the MCS web, you will be banned from the site.

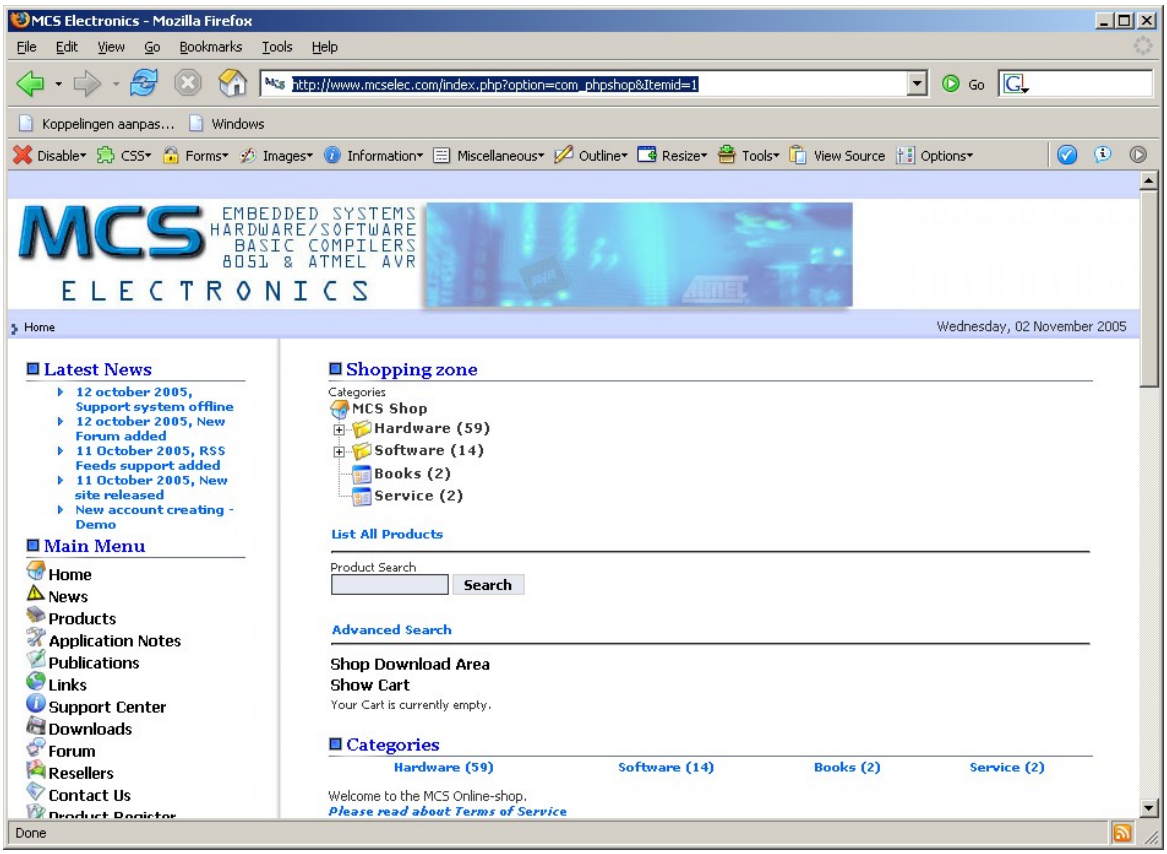


3.73 Help MCS Shop

This option will start your default web browser and direct it to :http://www.mcselec.com/index.php?option=com_phpshop&Itemid=1

You can order items and pay with PayPal. PayPal will accept most credit cards.

Before you order, it is best to check the resellers 1869 page to find a reseller near you. Resellers can help you in your own language, have all MCS items on stock, and are in the same time zone.



Before you can order items, you need to create an account.
Read the following about the new website : http://www.mcselec.com/index.php?option=com_content&task=view&id=133&Itemid=1

3.74 Help Support

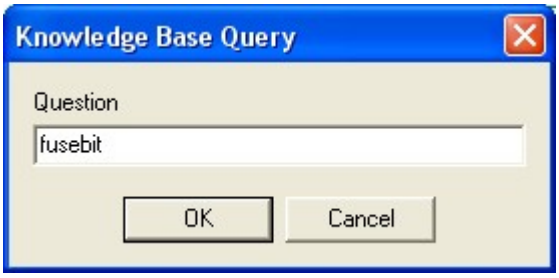
This option will start your default browser with the following URL :

http://www.mcselec.com/support-center/

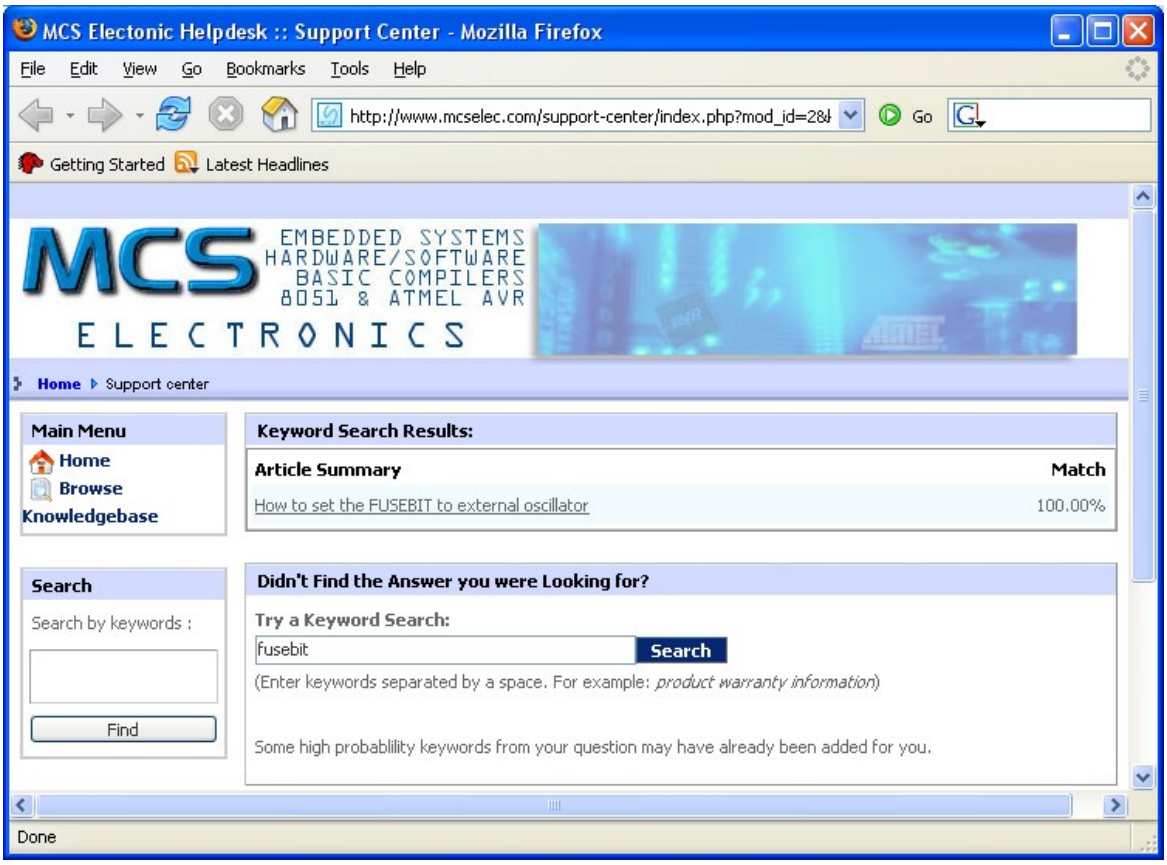
It depends from your browser settings if a new window or TAB will be created.
At the support site you can browse articles. You can also search on keywords.

3.75 Help Knowledge Base

This option will ask you to enter a search string.



This search string will be passed to the MCS support site.
The above example that searches for "FUSEBIT" will result in the following :



You can click one of the found articles to read it.

3.76 Help Credits

BASCOM was "invented" in 1995. Many users gave feedback and helped with tips, code, suggestions, support, a user list, and of course with buying the software. The software improved a lot during the last 25 years and will so during the next decade.

While it is impossible to thank everybody there are some people that deserve credits :

- Peter Maroudas. He wrote and tested the FT80x FTDI display support. FT800 support would not exist without him.
-
- Josef Franz Vögel. He wrote a significant part of the libraries in BASCOM-AVR. He is also author of AVR-DOS.
- Dr.-Ing. Claus Kuehnel for his book 'AVR RISC' , that helped me a lot when I began to study the AVR chips. Check his website at http://www.ckuehnel.ch
- Atmel, who gave permission to use the AVR picture in the start up screen. And for the great tech support. Check their website at http://www.atmel.com
- Brian Dickens, who did most of the Beta testing. He also checked the documentation on grammar and spelling errors. (he is not responsible for the spelling errors i added later :-))
- Jack Tidwell. I used his FP unit for singles. It is the best one available.

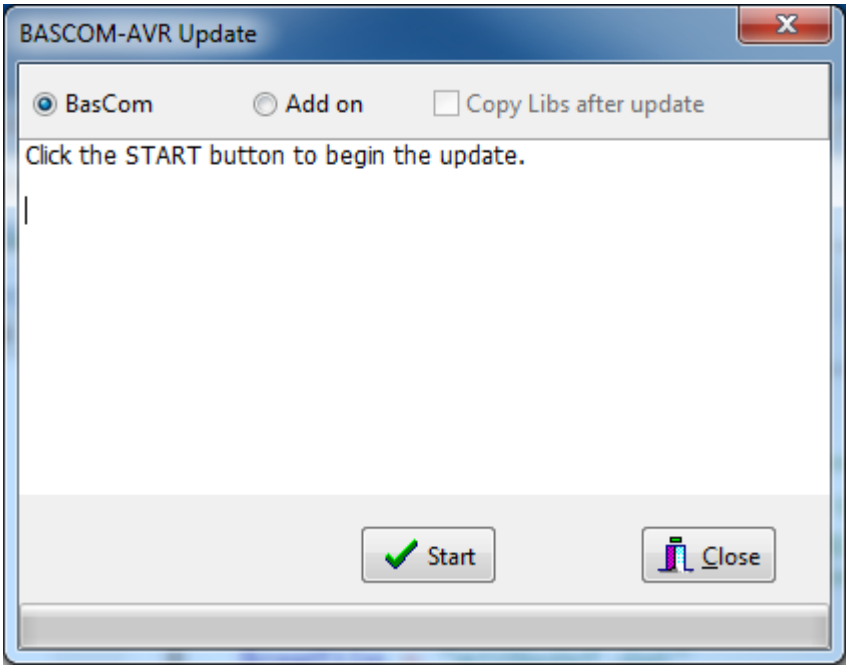
3.77 Help Update

The manual update process is explained here 61.
The Help Update is an automated version.

The DEMO version can not be updated. You can however install the full version into the DEMO folder.

In order to do a successful update you need the following :
- license validated in the register (https://register.mcselec.com)
- working internet connection.
- firewall and anti virus software must allow BASCOM to connect to the internet

When you click Help, Update, the following window will be shown:

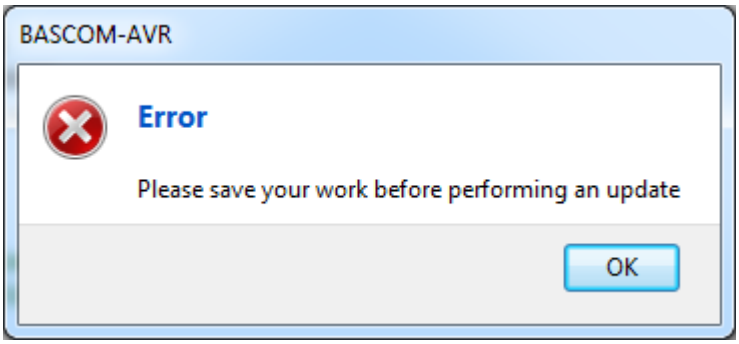


You can select if you want to update BasCom or the Add-On's.
By default BasCom is selected.

BasCom Update

You need to click the **START** button to start the actual update process.

When there are unsaved files, you will get an error message :



Your work/project must be saved since as soon the update download is finished, the setup will be executed and BASCOM is closed.

When there are no unsaved files, the current version will be checked.

Checking for update...
Current version : 2.0.8.0
Latest version : 2.0.7.8
No newer version found

In this case, there is no newer file and nothing happens. You need to click the **CLOSE** button to close the Update window. The IDE will not be closed in this case.

If however a newer version exists, it will be downloaded and unzipped in your windows TEMP folder.
After that **setup.exe** will be executed with admin rights. So you might get a windows security message that setup requires admin rights.
BASCOM will close automatically so the new version can be installed in the same folder. We recommend however to install each version into a different folder.
There is no need to uninstall an older version first.

This setup is the same as you used when you installed the software. But of course the latest version.
You can install into the same folder, but you may also install into a new folder.
When installing into a new folder you must manual install/copy the license file **bscavrl.dll** into the new folder yourself.
The bscavrl.dll file you get when you purchase bascom. It is either on CD-ROM or in the bascom-avr application folder.
Please notice that this license file is offered during purchase. We do not offer it again in the event you lost it.

Add On Update

When you chose to update Add On, you can also specify that libraries will be copied after the update. You do this by checking the 'Copy Libs After Update' check box.
The Add on update will check if you have an add on installed. It will then check if there is an update available. Notice that not all add ons you purchased are supported yet. So those you have to manual download/install.
The supported Add Ons :
- AVRDOS.
- I2CSLAVE
- XTINY

The add on is always ZIPPED and this ZIP file is downloaded to a new SUB folder named ADDONS.
This ADDONS subfolder is created in the BasCom-AVR application folder.
Each Add On is installed into its own sub folder. So for AVRDOS it is installed into ADDONS\AVRDOS
Older versions that might exist are overwritten.

When an Add-On contains a LIB or LBX file, the option 'Copy Libs after update' will be copied to the BasCom-AVR Library folder. The original file will be renamed so you always have a backup.
You do need write access in the BasCom-AVR application folder and sub folder.

3.78 Help Wiki

This option will open the browser at wiki.mcselec.com
The wiki contains the help file, projects and is partial translated into German as well.

When you want to contribute you need to create an account and send an email to tomi@mcselec.com to get the proper access rights.

3.79 BASCOM Editor Keys

Key	Action
LEFT ARROW	One character to the left
RIGHT ARROW	One character to the right
UP ARROW	One line up
DOWN ARROW	One line down
HOME	To the beginning of a line
END	To the end of a line
PAGE UP	Up one window
PAGE DOWN	Down one window
CTRL+LEFT	One word to the left
CTRL+RIGHT	One word to the right
CTRL+HOME	To the start of the text
CTRL+END	To the end of the text
CTRL+ Y	Delete current line
INS	Toggles insert/over strike mode
F1	Help (context sensitive)
F2	Run simulator
F3	Find next text
F4	Send to chip (run flash programmer)
F5	Run
F7	Compile File
F8	Step
F9	Set breakpoint
F10	Run to
F11	Collapse Code Toggle Sub/Function
SHIFT+F11 or CTRL+ENTER	Collapse Code Toggle current block
CTRL+F7	Syntax Check
CTRL+F	Find text
CTRL+G	Go to line
CTRL+K+x	Toggle bookmark. X can be 1-8
CTRL+L	LCD Designer
CTRL+M	File Simulation
CTRL+N	New File
CTRL+O	Load File
CTRL+P	Print File
CTRL+Q+x	Go to Bookmark. X can be 1-8
CTRL+R	Replace text

CTRL+S	Save File
CTRL+T	Terminal emulator
CTRL+P	Compiler Options
CTRL+W	Show result of compilation
CTRL+X	Cut selected text to clipboard
CTRL+Z	Undo last modification
SHIFT+CTRL+Z	Redo last undo
CTRL+INS	Copy selected text to clipboard
SHIFT+INS	Copy text from clipboard to editor
CTRL+SHIFT+J	Indent Block
CTRL+SHIFT+U	Unindent Block
Select text	Hold the SHIFT key down and use the cursor keys to select text. or keep the left mouse key pressed and drag the cursor over the text to select.
CTRL+SPACE	Code help.
SHIFT + MOUSE	Hover on indention lines to see to which group they belong. Hover on an element in your code to get info about that element.
CTRL+BACKSPACE	Jump back
CTRL+CLICK	Hold the CTRL key down and hover with the mouse till an element is underlined like an URL. Click the left mouse to jump to the implementation.

3.80 Program Development Order

- Start BASCOM
- Open a file or create a new one
- ! Important ! Check the chip settings, baud rate and frequency settings for the target system
- Save the file
- Compile the file (this will also save the file !!!)
- If an error occurs fix it and recompile (F7)
- Run the simulator(F2)
- Program the chip(F4)

3.81 PlugIns

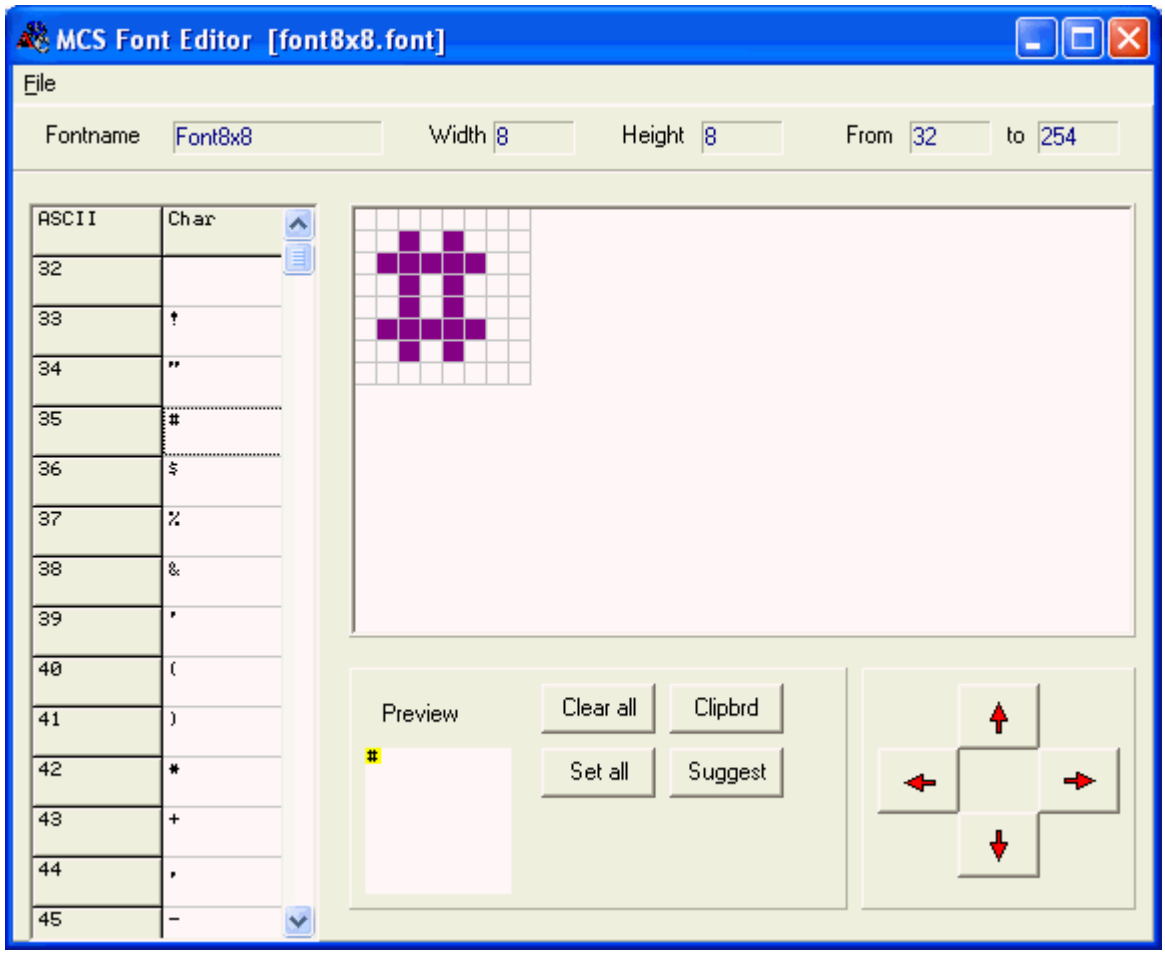
3.81.1 Font Editor

In **version 2079** the Font Editor plugin is replaced by the integrated Font Editor from the Tools menu 141. It has the same options.

The Font Editor is a Plug in that is intended to create Fonts that can be used with Graphical display such as SED1521, KS108, color displays, etc.

When you have installed the Font Editor , a menu option becomes available under the Tools menu : Font Editor.

When you choose this option the following window will appear:



You can open an existing Font file, or Save a modified file.

The supplied font files are installed in the Samples directory.
You can copy an image from the clipboard, and you can then move the image up , down, left and right.

When you select a new character, the current character is saved. The suggest button will draw an image of the current selected character.

When you keep the left mouse button pressed, you can set the pixels in the grid.
When you keep the right mouse button pressed, you can clear the pixels in the grid.

When you choose the option to create a new Font, you must provide the name of the font, the height of the font in pixels and the width of the font in pixels.

The Max ASCII is the last ASCII character value you want to use. Each character will occupy space. So it is important that you do not choose a value that is too high and will not be used.

When you display normal text, the maximum number is 127 so it does not make sense to specify a value of 255.

A font file is a plain text file.
Lets have a look at the first few lines of the 8x8 font:

```
Font8x8:
$asm
```

```
.db 1,8,8,0
.db 0,0,0,0,0,0,0,0 ;
.db 0,0,6,95,6,0,0,0 ; !
```

The first line contains the name of the font. With the SETFONT 1351 statement you can select the font. Essential, this sets a data pointer to the location of the font data.

The second line ($ASM) is a directive for the internal assembler that asm code will follow.
All other lines are data lines.

The third line contains 4 bytes: 1 (height in bytes of the font) , 8 (width in pixels of the font), 8 (block size of the font) and a 0 which was not used before the 'truetype' support, but used for aligning the data in memory. This because AVR object code is a word long.

This last position is **0** by default. Except for 'TrueType' fonts. In BASCOM a TrueType font is a font where every character can have it's own width. The letter 'i' for example takes less space then the letter 'w'. The EADOG128 library demonstrates the TrueType option.
In order to display TT, the code need to determine the space at the left and right of the character. This space is then skipped and a fixed space is used between the characters. You can replace the 0 by the width you want to use. The value 2 seems a good one for small fonts.

All other lines are bytes that represent the character.

Part

4 BASCOM HARDWARE

4.1 Additional Hardware

Of course just running a program on the chip is not enough. You will probably connect many types of electronic devices to the processor ports.
BASCOM supports a lot of hardware and so it has lots of hardware related statements. Before explaining about programming the additional hardware, it might be better to talk about the chip.

The AVR internal hardware 242

Attaching an LCD display 266

Using the I2C protocol 297

Using the 1WIRE protocol 310

Using the SPI protocol 314

You can connect additional hardware to the ports of the microprocessor.
The following are hardware related:

I2CSEND 1305 and I2CRECEIVE 1303 and other I2C related statements.

CLS, 1322 LCD, 1335 DISPLAY 1329 and other related LCD-statements.

1WRESET 718 , 1WWRITE 729 and 1WREAD 720

There are many more hardware specific statements and functions.

Adding XRAM 251

Adding XRAM to XMEGA using EBI 255

Adding SRAM 4-port Non Multiplexed 257

Using a BOOTLOADER 285

4.2 AVR Internal Hardware

The AVR chips all have internal hardware that can be used.

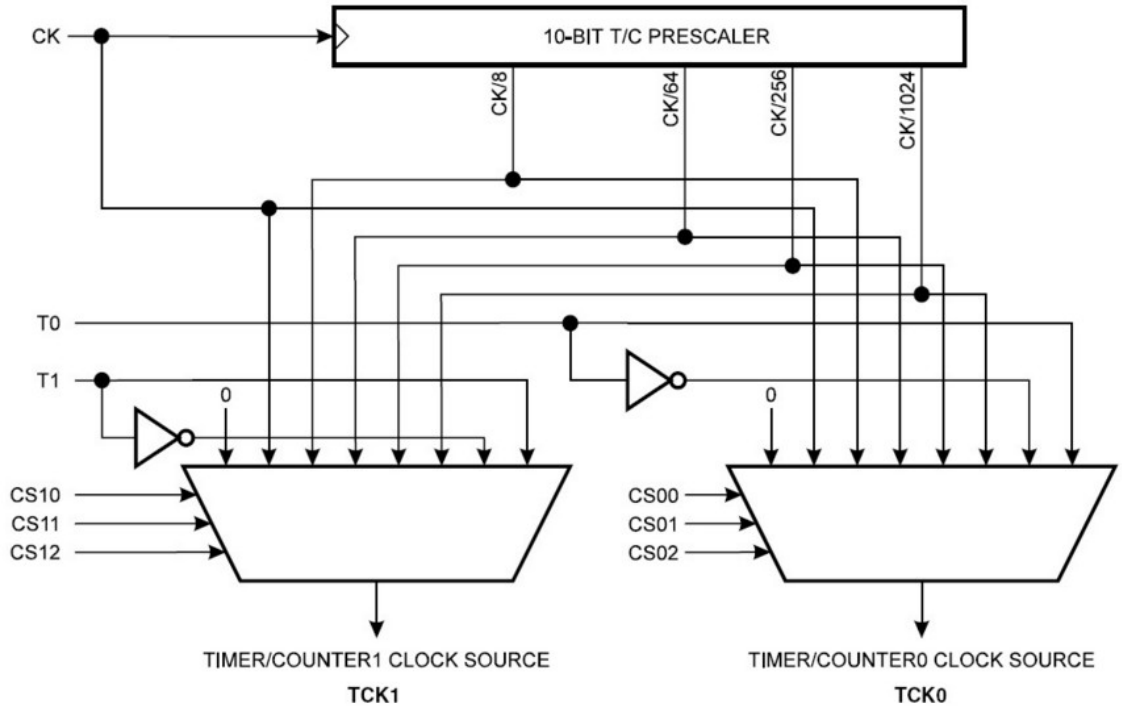
For this description of the hardware the 90S8515 was used. Newer chips like the Mega8515 may differ and have more or less internal hardware.

You will need to read the manufacturers data sheet for the processor you are using to learn about the special internal hardware available.

Timer / Counters

The AT90S8515 provides two general purpose Timer/Counters - one 8-bit T/C and one 16-bit T/C. The Timer/Counters have individual pre-scaling selection from the same 10-bit pre-scaling timer. Both Timer/Counters can either be used as a timer with an internal clock time base or as a counter with an external pin connection which triggers the counting.

Figure 28. Timer/Counter Prescaler



More about TIMERO 245

More about TIMER1 246

The WATCHDOG Timer 248

Almost all AVR chips have the ports B and D. The 40 or more pin devices also have ports A and C that also can be used for addressing an external RAM chip (XRAM 251). Since all ports are similar except that PORT B and PORT D have alternative functions, only these ports are described.

PORT B 248
PORT D 250

4.3 AVR Internal Registers

You can manipulate the internal register values directly from BASCOM. They are also reserved words. Each register acts like a memory location or program variable, except that the bits of each byte have a special meaning. The bits control how the internal hardware functions, or report the status of internal hardware functions. Read the data sheet to determine what each bit function is for.

The internal registers for the AVR90S8515 are : (other processors are similar, but vary)

Addr.	Register
$3F	SREG I T H S V N Z C
$3E	SPH SP15 SP14 SP13 SP12 SP11 SP10 SP9 SP8

$3D	SPL SP7 SP6 SP5 SP4 SP3 SP2 SP1 SP0
$3C	Reserved
$3B	GIMSK INT1 INT0 - - - - - -
$3A	GIFR INTF1 INTF0
$39	TIMSK TOIE1 OCIE1A OCIE1B - TICIE1 - TOIE0 -
$38	TIFR TOV1 OCF1A OCF1B -ICF1 -TOV0 -
$37	Reserved
$36	Reserved
$35	MCUCR SRE SRW SE SM ISC11 ISC10 ISC01 ISC00
$34	Reserved
$33	TCCR0 - - - - - CS02 CS01 CS00
$32	TCNT0 Timer/Counter0 (8 Bit)
$31	Reserved
$30	Reserved
$2F	TCCR1A COM1A1 COM1A0 COM1B1 COM1B0 - -PWM11 PWM10
$2E	TCCR1B ICNC1 ICES1 - - CTC1 CS12 CS11 CS10
$2D	TCNT1H Timer/Counter1 - Counter Register High Byte
$2C	TCNT1L Timer/Counter1 - Counter Register Low Byte
$2B	OCR1AH Timer/Counter1 - Output Compare Register A High Byte
$2A	OCR1AL Timer/Counter1 - Output Compare Register A Low Byte
$29	OCR1BH Timer/Counter1 - Output Compare Register B High Byte
$28	OCR1BL Timer/Counter1 - Output Compare Register B Low Byte
$27	Reserved
$26	Reserved
$25	ICR1H Timer/Counter1 - Input Capture Register High Byte
$24	ICR1L Timer/Counter1 - Input Capture Register Low Byte
$23	Reserved
$22	Reserved
$21	WDTCR - - - WDTOE WDE WDP2 WDP1 WDP0
$20	Reserved
$1F	Reserved - - - - - - - EEAR8
$1E	EEARL EEPROM Address Register Low Byte
$1D	EEDR EEPROM Data Register
$1C	EECR - - - - - EEMWE EEWE EERE
$1B	PORTA PORTA7 PORTA6 PORTA5 PORTA4 PORTA3 PORTA2 PORTA1 PORTA0
$1A	DDRA DDA7 DDA6 DDA5 DDA4 DDA3 DDA2 DDA1 DDA0
$19	PINA PINA7 PINA6 PINA5 PINA4 PINA3 PINA2 PINA1 PINA0
$18	PORTB PORTB7 PORTB6 PORTB5 PORTB4 PORTB3 PORTB2 PORTB1 PORTB0
$17	DDRB DDB7 DDB6 DDB5 DDB4 DDB3 DDB2 DDB1 DDB0
$16	PINB PINB7 PINB6 PINB5 PINB4 PINB3 PINB2 PINB1 PINB0
$15	PORTC PORTC7 PORTC6 PORTC5 PORTC4 PORTC3 PORTC2 PORTC1 PORTC0
$14	DDRC DDC7 DDC6 DDC5 DDC4 DDC3 DDC2 DDC1 DDC0
$13	PINC PINC7 PINC6 PINC5 PINC4 PINC3 PINC2 PINC1 PINC0
$12	PORTD PORTD7 PORTD6 PORTD5 PORTD4 PORTD3 PORTD2 PORTD1 PORTD0
$11	DDRD DDD7 DDD6 DDD5 DDD4 DDD3 DDD2 DDD1 DDD0

$10	PIND PIND7 PIND6 PIND5 PIND4 PIND3 PIND2 PIND1 PIND0
$0F	SPDR SPI Data Register
$0E	SPSR SPIF WCOL - - - - - -
$0D	SPCR SPIE SPE DORD MSTR CPOL CPHA SPR1 SPR0
$0C	UDR UART I/O Data Register
$0B	USR RXC TXC UDRE FE OR - - -
$0A	UCR RXCIE TXCIE UDRIE RXEN TXEN CHR9 RXB8 TXB8
$09	UBRR UART Baud Rate Register
$08	ACSR ACD - ACO ACI ACIE ACIC ACIS1 ACIS0
$00	Reserved

The registers and their addresses are defined in the xxx.DAT files which are placed in the BASCOM-AVR application directory.

The registers can be used as normal byte variables.

PORTB = 40 will place a value of 40 into port B.

Note that internal registers are reserved words. This means that they can't be dimensioned as BASCOM variables!

So you can't use the statement DIM SREG As Byte because SREG is an internal register.

You can however manipulate the register with the SREG = value statement, or var = SREG statement.

4.4 AVR Internal Hardware TIMER0

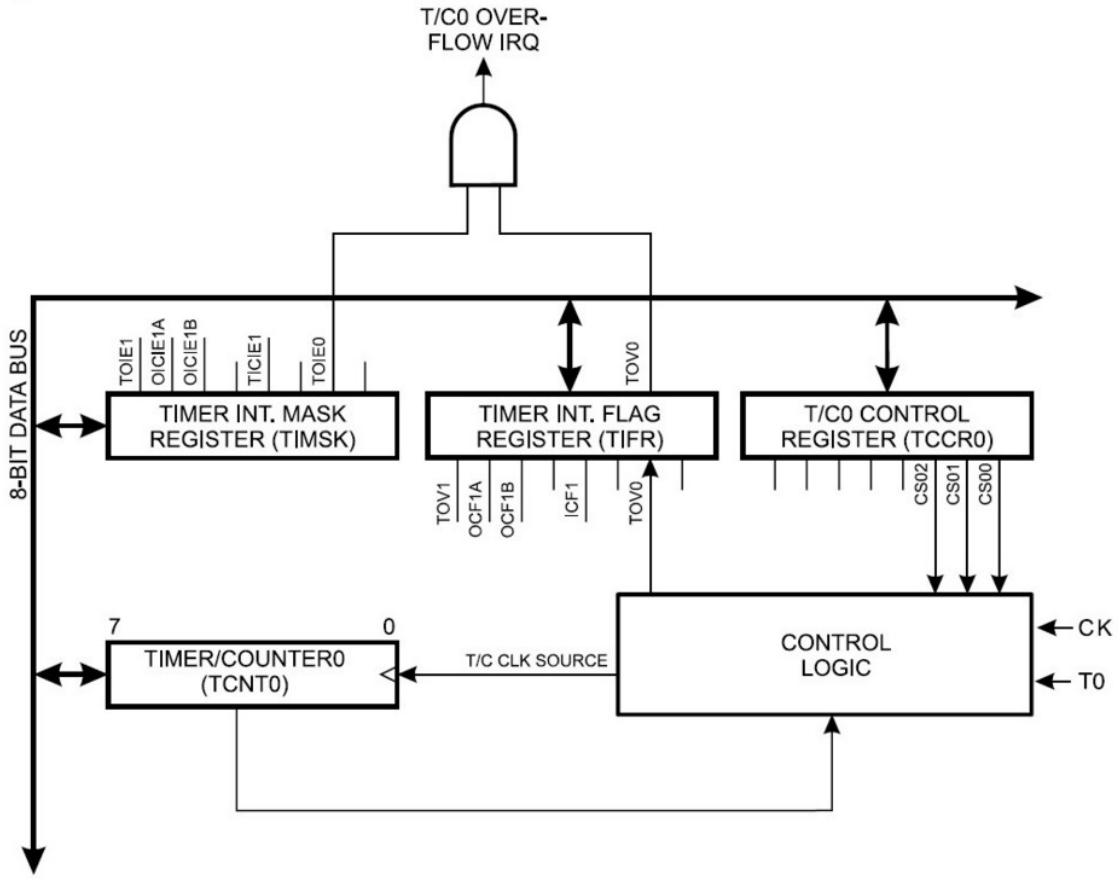
The 8-Bit Timer/Counter0

 The 90S8515 was used for this example. Other chips might have a somewhat different timer.
The 8-bit Timer/Counter0 can select its clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped (no clock).

The overflow status flag is found in the Timer/Counter Interrupt Flag Register - TIFR. Control signals are found in the Timer/Counter0 Control Register - TCCR0. The interrupt enable/disable settings for Timer/Counter0 are found in the Timer/Counter Interrupt Mask Register - TIMSK.

When Timer/Counter0 is externally clocked, the external signal is synchronized with the oscillator frequency of the CPU. To assure proper sampling of the external clock, the minimum time between two external clock transitions must be at least one internal CPU clock period. The external clock signal is sampled on the rising edge of the internal CPU clock.

Figure 29. Timer/Counter0 Block Diagram



The 8-bit Timer/Counter0 features both a high resolution and a high accuracy mode with lower pre-scaling values. Similarly, high pre-scaling values make the Timer/ Counter0 useful for lower speed functions or exact timing functions with infrequent actions.

4.5 AVR Internal Hardware TIMER1

The 16-Bit Timer/Counter1

The 90S8515 was used for the documentation. Other chips might have a somewhat different timer.

The 16-bit Timer/Counter1 can select its clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped (no clock).

The different status flags (overflow, compare match and capture event) and control signals are found in the Timer/Counter1 Control Registers - TCCR1A and TCCR1B.

The interrupt enable/disable settings for Timer/Counter1 are found in the Timer/ Counter Interrupt Mask Register - TIMSK.

When Timer/Counter1 is externally clocked, the external signal is synchronized with the oscillator frequency of the CPU. To assure proper sampling of the external clock, the minimum time between two external clock transitions must be at least one

internal CPU clock period.

The external clock signal is sampled on the rising edge of the internal CPU clock.

The 16-bit Timer/Counter1 features both a high resolution and a high accuracy usage with lower pre-scaling values.

Similarly, high pre-scaling values make the Timer/Counter1 useful for lower speed functions or exact timing functions with infrequent actions.

The Timer/Counter1 supports two Output Compare functions using the Output Compare Register 1 A and B -OCR1A and OCR1B as the data values to be compared to the Timer/Counter1 contents.

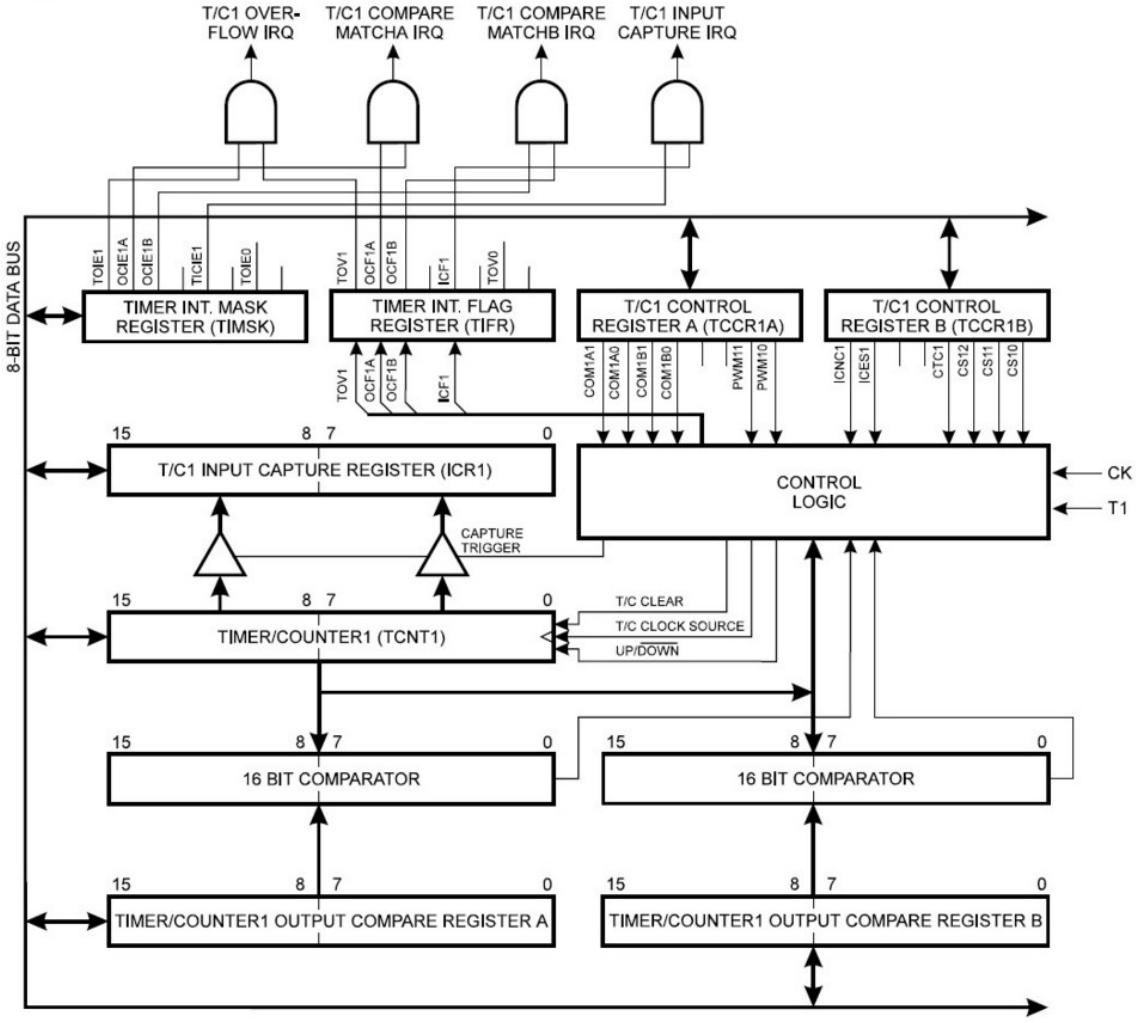
The Output Compare functions include optional clearing of the counter on compareA match, and can change the logic levels on the Output Compare pins on both compare matches.

Timer/Counter1 can also be used as a 8, 9 or 10-bit Pulse Width Modulator (PWM). In this mode the counter and the OCR1A/OCR1B registers serve as a dual glitch-free stand-alone PWM with centered pulses.

The Input Capture function of Timer/Counter1 provides a capture of the Timer/ Counter1 value to the Input Capture Register - ICR1, triggered by an external event on the Input Capture Pin - ICP. The actual capture event settings are defined by the Timer/Counter1 Control Register -TCCR1B.

In addition, the Analog Comparator can be set to trigger the Capture.

Figure 30. Timer/Counter1 Block Diagram



4.6 AVR Internal Hardware Watchdog timer

The Watchdog Timer

The Watchdog Timer is clocked from a separate on-chip oscillator which runs at approximately 1MHz. This is the typical value at VCC = 5V.

By controlling the Watchdog Timer pre-scaler, the Watchdog reset interval can be adjusted from 16K to 2,048K cycles (nominally 16 - 2048 ms). The BASCOM RESET WATCHDOG - instruction resets the Watchdog Timer.

Eight different clock cycle periods can be selected to determine the reset period.

If the reset period expires without another Watchdog reset, the AT90Sxxxx resets and program execution starts at the reset vector address.

4.7 AVR Internal Hardware Port B

Port B

Port B is an 8-bit bi-directional I/O port. Three data memory address locations are allocated for the Port B, one each for the Data Register - PORTB, $18($38), Data

Direction Register - DDRB, $17($37) and the Port B Input Pins - PINB, $16($36). The Port B Input Pins address is read only, while the Data Register and the Data Direction Register are read/write.

All port pins have individually selectable pull-up resistors. The Port B output buffers can sink 20mA and thus drive LED displays directly. When pins PB0 to PB7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated.

The Port B pins with alternate functions are shown in the following table:

When the pins are used for the alternate function the DDRB and PORTB register has to be set according to the alternate function description.

Port B Pins Alternate Functions

Port	Pin	Alternate Functions
PORTB.0	T0	(Timer/Counter 0 external counter input)
PORTB.1	T1	(Timer/Counter 1 external counter input)
PORTB.2	AIN0	(Analog comparator positive input)
PORTB.3	AIN1	(Analog comparator negative input)
PORTB.4	SS	(SPI Slave Select input)
PORTB.5	MOSI	(SPI Bus Master Output/Slave Input)
PORTB.6	MISO	(SPI Bus Master Input/Slave Output)
PORTB.7	SCK	(SPI Bus Serial Clock)

The Port B Input Pins address - PINB - is not a register, and this address enables access to the physical value on each Port B pin. When reading PORTB, the PORTB Data Latch is read, and when reading PINB, the logical values present on the pins are read.

PortB As General Digital I/O

All 8 bits in port B are equal when used as digital I/O pins. PORTB.X, General I/O pin: The DDBn bit in the DDRB register selects the direction of this pin, if DDBn is set (one), PBn is configured as an output pin. If DDBn is cleared (zero), PBn is configured as an input pin. If PORTBn is set (one) when the pin configured as an input pin, the MOS pull up resistor is activated.

To switch the pull up resistor off, the PORTBn has to be cleared (zero) or the pin has to be configured as an output pin.

DDBn Effects on Port B Pins

DDBn	PORTBn	I/O	Pull up	Comment
0	0	Input	No	Tri-state (Hi-Z)

0	1	Input	Yes	PBn will source current if ext. pulled low.
1	0	Output	No	Push-Pull Zero Output
1	1	Output	No	Push-Pull One Output

By default, the DDR and PORT registers are 0. CONFIG PORTx=OUTPUT will set the entire DDR register. CONFIG PINX.Y will also set the DDR register for a single bit/pin. When you need the pull up to be activated, you have to write to the PORT register.

4.8 AVR Internal Hardware Port D

Port D

Port D Pins Alternate Functions

Port	Pin	Alternate Function
PORTD.0	RDX	(UART Input line)
PORTD.1	TDX	(UART Output line)
PORTD.2	INT0	(External interrupt 0 input)
PORTD.3	INT1	(External interrupt 1 input)
PORTD.5	OC1A	(Timer/Counter1 Output compareA match output)
PORTD.6	WR	(Write strobe to external memory)
PORTD.7	RD	(Read strobe to external memory)

RD - PORTD, Bit 7
RD is the external data memory read control strobe.

WR - PORTD, Bit 6
WR is the external data memory write control strobe.

OC1- PORTD, Bit 5
Output compare match output: The PD5 pin can serve as an external output when the Timer/Counter1 com-pare matches.

The PD5 pin has to be configured as an out-put (DDD5 set (one)) to serve this f unction. See the Timer/Counter1 description for further details, and how to enable the output. The OC1 pin is also the output pin for the PWM mode timer function.

INT1 - PORTD, Bit 3
External Interrupt source 1: The PD3 pin can serve as an external interrupt source to the MCU. See the interrupt description for further details, and how to enable the source

INT0 - PORTD, Bit 2
INT0, External Interrupt source 0: The PD2 pin can serve as an external interrupt

source to the MCU. See the interrupt description for further details, and how to enable the source.

TXD - PORTD, Bit 1
Transmit Data (Data output pin for the UART). When the UART transmitter is enabled, this pin is configured as an output regardless of the value of DDRD1.

RXD - PORTD, Bit 0
Receive Data (Data input pin for the UART). When the UART receiver is enabled this pin is configured as an output regardless of the value of DDRD0. When the UART forces this pin to be an input, a logical one in PORTD0 will turn on the internal pull-up.

When pins TXD and RXD are not used for RS-232 they can be used as an input or output pin.

No PRINT, INPUT or other RS-232 statement may be used in that case.

The UCR register will by default not set bits 3 and 4 that enable the TXD and RXD pins for RS-232 communication. It is however reported that this not works for all chips. In this case you must clear the bits in the UCR register with the following statements:

RESET UCR.3
RESET UCR.4
or as an alernative : UCR=0

4.9 Adding XRAM with External Memory Interface

With ATMEGA AVR like ATMEGA128, ATMEGA1280 or older types like 90S8515 you can access external RAM (SRAM) or other peripherals through its External Memory Interface. Search in the Atmel ATMEGA datasheets for "External Memory Interface"

For ATXMEGA devices see App Note: AVR1312: Using the XMEGA External Bus Interface for details.

For example for an ATMEGA1280 the external memory interface consist of PORTA (multiplexed data and address low byte), PORTC (address high byte), and PORTG[2:0] (RD, WR and ALE).

ATMEGA1280 pin connections to SRAM device:
Port A = Multiplexed Address low byte (A**0**....A**7**) / Data (D0....D7) <--------> Direct connection to SRAM (D0...D7) and connected to Input D of octal latch (typically "74 x 573" or equivalent) to (A0.....A7) of SRAM chip
Port C = Address high byte (A**8**....A**15**) <--------> direct connection to for example SRM (A8....A15)
Port G Pin 0 = WR (Write strobe to external memory) <--------> direct connection to for example SRAM WR
Port G Pin 1 = RD (Read strobe to external memory) <--------> direct connection to for example SRAM RD
Port G Pin 2 = ALE (Address Latch Enable to external memory) <--------> Connected to G Input of octal latch (typically "74 x 573" like 74 x 573)

Example for 74HTC573 (TTL variant):

http://www.nxp.com/documents/data_sheet/74HC_HCT573.pdf

Address latch with octal latch:
The data bus and the low byte of the address bus is **multiplexed** on Port A. The ALE signal indicates when the address is present. This low byte must be stored by a latch until the memory access cycle is completed.

Schematics for connecting the ATMEGA with octal latch and sram can be found in:
- Atmel AVR Studio Help File (AVR tools user guide) search for: "external memory interface" and scroll down to **APPENDIX**. There you also find a list of 3.3 or 5V compatible SRAM's that can be used with ATMEGA's and there is a link to *STK503.pdf* which is part of the help file.
- The list of compatible SRAM devices can be also found here: http://www.atmel.com/images/stk503_ug.pdf (page 16)
- The datasheet of for example ATMEGA1280 also include a picture which show the connections between AVR, Octal Latch external SRAM device.

The data memory map for example for ATmega640/1280/1281/2560/2561:

Hex-Address:	
&H00 .... &H1F	32 Registers
&H20 .... &H5F	64 I/O Registers
&H60 .... &H1FF	416 external I/O Registers
&H200 .... &H21FF	Internal SRAM (8K in this case)
&H2200 .... &HFFFF	**External SRAM (XRAM)**

XRAM will use an area in the remaining address locations in the 64K address space (&HFFFF). This starts at the address following the internal SRAM.
Internal SRAM use the lowest 4,608/8,704 bytes, so when using 64KB (65,536 bytes) of XRAM, **60,478/56,832** Bytes of XRAM are **available**.
See datasheet of ATMEGA device for a way to use the complete 64KByte.

XRAM will be enabled by CONFIG XRAM 1161 (config XRAM is setting SRE bit of the atmega 1280 XMCRA Register)
The Pins of Port A, Port C and Port G from ATMEGA1280 are automatically enabled for XRAM and can not be used for other tasks by default if XRAM is enabled. The external memory address space can be divided in two sectors (upper and lower sector) with different wait-state bits. You can also release some Port C pins for other tasks (in the XMCRB Register**)** . See atmega 1280 datasheet for details.

See also: $XRAMSIZE 713 and $XRAMSTART 714

You can clear the XRAM for example with:

```
For N = Ramstart To Ramend                    'replace RamStart and RamEnd
with the real values
   Out N , 0                                  'zero or any value you like
Next
```

With XRAM, you should dim all your global variables with XRAM. Example: `Dim Var As Xram Byte`
This will leave the internal memory for the stacks and local created variables.
You can also use `$default Xram` when you do not want to add the XRAM to each DIM.

Example 1:

A real Example for using SRAM and another Bus-mode device is WIZ200WEB from Wiznet.

Here an ATMEGA128L, an external 32K SRAM and a W5300 Ethernet Chip is used. See also CONFIG TCPIP 1098

The data memory map for ATMEGA128:

Hex-Address:	
&H00 .... &H1F	32 Registers
&H20 .... &H5F	64 I/O Registers
&H60 .... &HFF	160 external I/O Registers
&H100 .... &H10FF	Internal SRAM (4K in this case)
&H1100	Start of External SRAM (XRAM)

So the config is following (&H8000 = 32kByte):

```
$xramstart = &H1100
$xramsize = &H8000
Config Xram = Enabled
```

The W5300 Chip from Wiznet is setup to use Base Address &H8000.

So the data memory map for ATMEGA128 is:

Hex-Address:	
&H00 .... &H1F	32 Registers
&H20 .... &H5F	64 I/O Registers
&H60 .... &HFF	160 external I/O Registers
&H100 .... &H10FF	Internal SRAM (4K in this case)
&H1100	Start of External SRAM (XRAM)
&H8000	Base Address of W5300 Chip from WIZNET
&H8400 .... &HFFFF	Not in use

See also CONFIG TCPIP 1098 for further details on W5300 Chip from Wiznet.

Example 2:

We use now an WIZ830mj module (which uses a W5300 Chip from Wiznet) on a board with an 64/128KByte SRAM in combination with ATMEGA1280:

Hex-Address:	
&H00 .... &H1F	32 Registers
&H20 .... &H5F	64 I/O Registers
&H60 .... &H1FF	416 external I/O Registers
&H200 .... &H21FF	Internal SRAM (8K in this case)
&H2200 .... &HFBFF	**External SRAM (XRAM) upper and lower**
&HFC00	Base Address of W5300 Chip over memory address selector

XRAM configuration here is:

```
$xramstart = &H2200
$xramsize = &HFBFF
Config Xram = Enabled
```

Writing to the first XRAM address is done by:

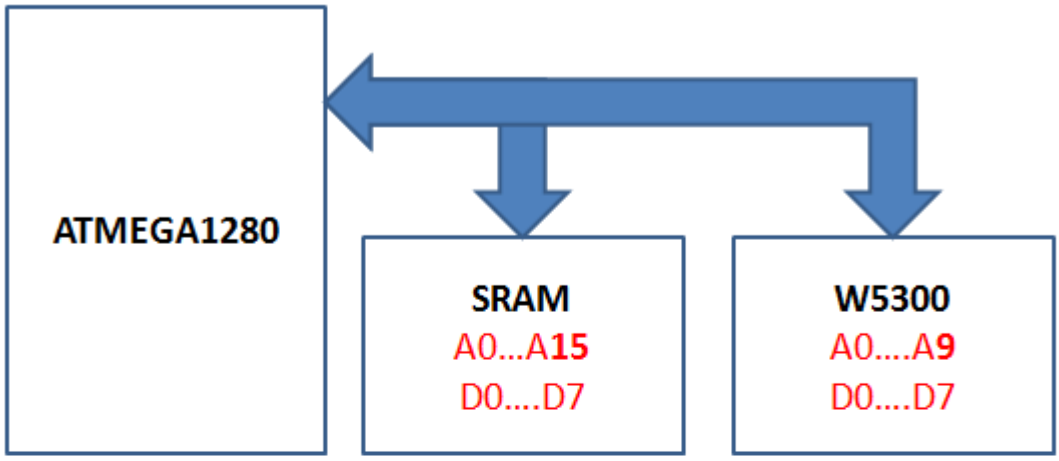
```
Out &H2200 , &H01
```

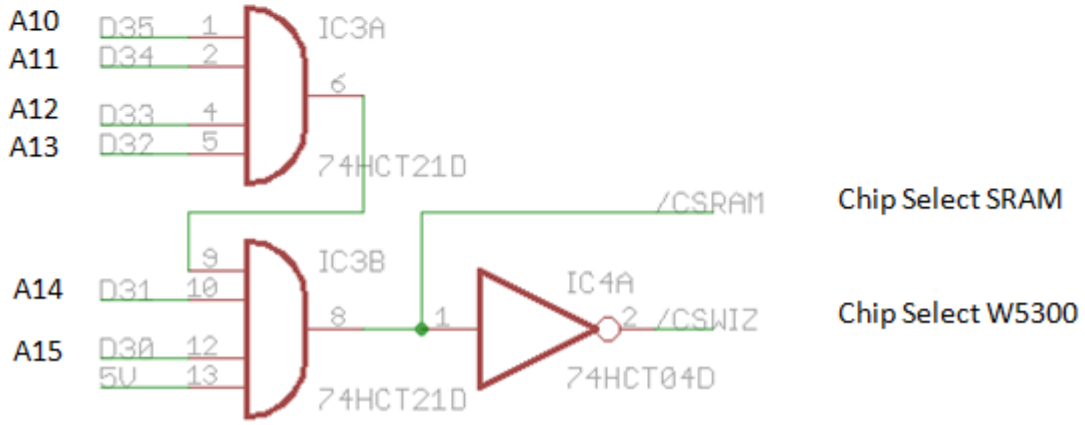
Reading from first XRAM is done by:

```
Var = Inp(&H2200)
```

The datasheet of W5300 say: "In the case of using an 8bit data bus width, ADDR[9:0] is used " so we have Address 0.....Address 9 but the SRAM need Address 0.....15.

Here an example of additional circuit between ATMEGA and W5300 and SRAM to solve the difference of address (A0...A15) for SRAM and (A0...A9) for W5300:





The Base Address for W5300 (WIZ830mj) in this case is **&HFC00**

Address Bit	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binary:	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Hex:	**FC00**															

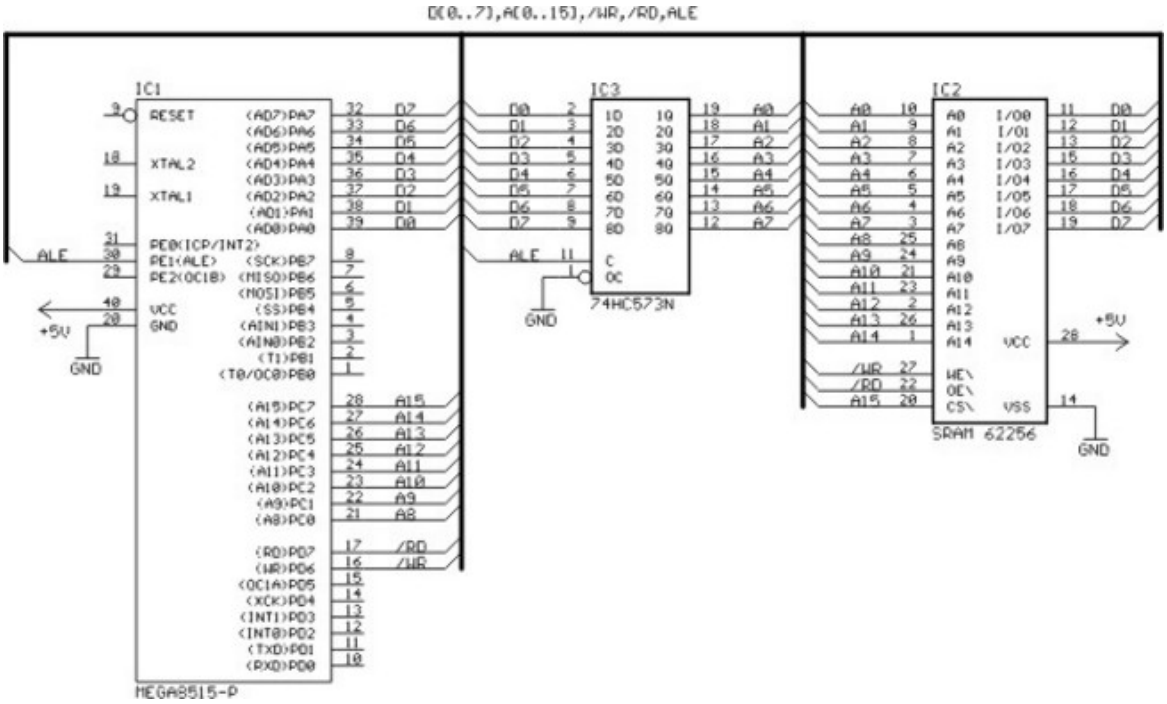
For older 90S8515 chips for example the maximum size of XRAM can be 64 Kbytes.
Example: The STK200 has a 62256 ram chip (32K x 8 bit).

Here is some info from the BASCOM user list :
If you do go with the external ram , be careful of the clock speed.
Using a 4 MHz crystal , will require a SRAM with 70 nS access time or less. Also the data latch (74HC573) will have to be from a faster
family such as a 74FHC573 if you go beyond 4 MHz.

You can also program an extra wait state, to use slower memory.

Here you will find a pdf file showing the STK200 schematics:
See Stk200_schematic.pdf for more information.

If you use a 32 KB SRAM, then connect the /CS signal to A15 which give to the range of &H0000 to &H7FFF, if you use a 64 KB SRAM, then
tie /CS to GND, so the RAM is selected all the time.



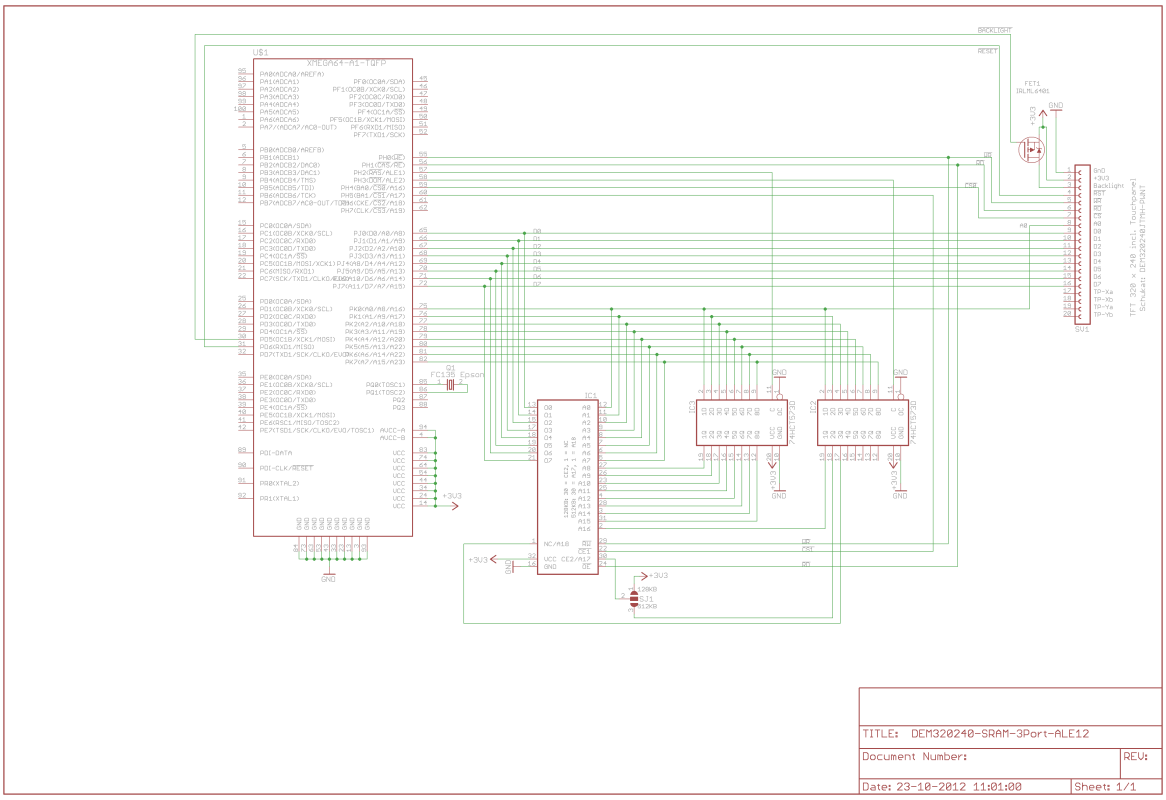
4.10 Adding XRAM to XMEGA using EBI

This information has been provided by Electronic Design Bitzer.
Some XMEGA processors have an EBI. The following circuit shows how to set up the EBI for 8 bit bus mode where the SRAM can be selected with a jumper.
128 KB SM621008VLLP70T : SRAM LLPow 3,3V 128Kx8 70ns TSOP32(I)
512 KB SM624008VLLP70M : SRAM LLPow 3,3V 512Kx8 70ns SOP32

The BASCOM setup code :

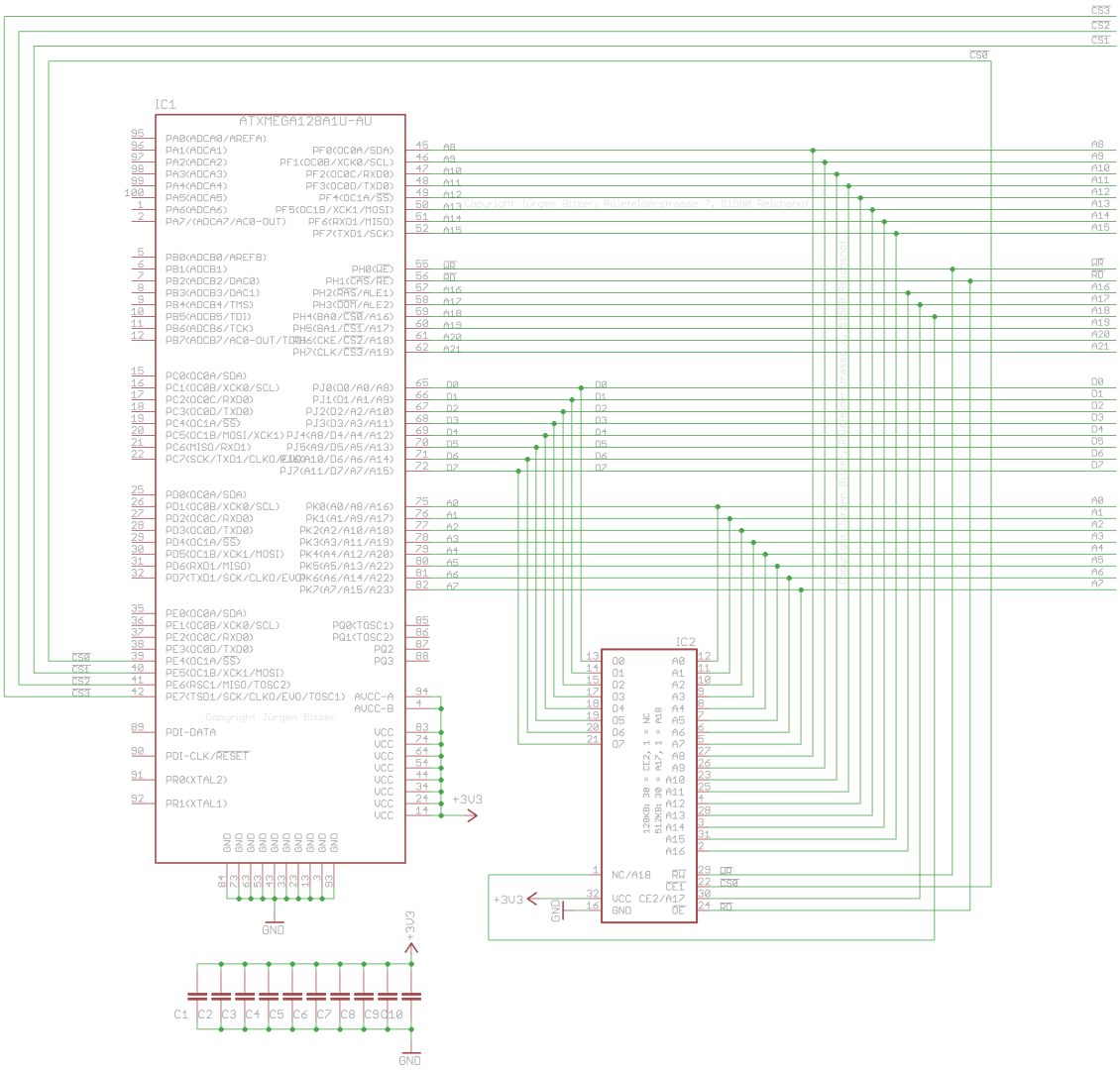
```
   ' All EBI-Ports must be set to OUTPUT
   ' All Ports, ACTIVE-LOW , must be set to 1 !!!
   ' All Ports, ACTIVE-HIGH, must be set to 0 !!!

   Porth_dirset = &B1111_1111 : Porth = &B1111_0011          'WR, RD, ALE1, ALE2, CS
   Portj_dirset = &B1111_1111 : Portj = &B1111_1111
   Portk_dirset = &B1111_1111 : Portk = &B1111_1111


Config Xram = 3port , Ale = Ale12 , Sdbus = 8 , Modesel0 = Sram , Adrsize0 = 256b ,
              Modesel1 = Sram , Adrsize1 = 128k , Waitstate1 = 1 , Baseadr1 = &H20
```

See also : CONFIG XRAM



4.11 Adding SRAM 4-port Non Multiplexed

The following information was contributed by Juergen Bitzer.



The EBI allows to use an SRAM in 4-port non multiplexed mode. This means that you need little parts but you loose 4 ports.

Example

```
$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = &H32
$swstack = &H32
$framesize = &H32
$xramstart = &H100000
$xramsize = &H080000

'------------------
' CPU:
' ATXMEGA128A1U-AU          : 2,23/100 Mouser  Muss -->A1U-AU<--
```

```
sein !!!
' ATXMEGA64A1U-AU
'------------------
' SRam:
' 512 KB AS6C4008-55PCN    : SRAM 4MB   2.7V-5.5V, 512KX8, PDIP32

' 512 KB AS6C4008-55SIN    : SRAM 4MB   2.7V-5.5V, 512KX8, SOP32
' 512 KB AS6C4008-55SIN    : SRAM 4MB   2.7V-5.5V, 512KX8, SOP32
'-----------------------------------------------------------------
---------------
'-----------------------------------------------------------------
---------------

' #### Four Port SRAM ####
' MODE SRAM 4Port direkt

' PortH.0 - Pin 55   /WR
' PortH.1 - Pin 56   /RD

' PortE.4 - Pin 39   /CS0 / A16 -> CS0: SRAM 512 KB
' PortE.5 - Pin 40   /CS1 / A17 -> CS1: unbenutzt
' PortE.6 - Pin 41   /CS2 / A18 -> CS2: unbenutzt
' PortE.7 - Pin 42   /CS3 / A19 -> CS3: unbenutzt

' PortJ.0 - Pin 65   D0   -> SRam
' PortJ.1 - Pin 66   D1   -> SRam
' PortJ.2 - Pin 67   D2   -> SRam
' PortJ.3 - Pin 68   D3   -> SRam
' PortJ.4 - Pin 69   D4   -> SRam
' PortJ.5 - Pin 70   D5   -> SRam
' PortJ.6 - Pin 71   D6   -> SRam
' PortJ.7 - Pin 72   D7   -> SRam

' PortK.0 - Pin 75   A0   -> SRam
' PortK.1 - Pin 76   A1   -> SRam
' PortK.2 - Pin 77   A2   -> SRam
' PortK.3 - Pin 78   A3   -> SRam
' PortK.4 - Pin 79   A4   -> SRam
' PortK.5 - Pin 80   A5   -> SRam
' PortK.6 - Pin 81   A6   -> SRam
' PortK.7 - Pin 82   A7   -> SRam

' PortF.0 - Pin 45   A8   -> SRam
' PortF.1 - Pin 46   A9   -> SRam
' PortF.2 - Pin 47   A10  -> SRam
' PortF.3 - Pin 48   A11  -> SRam
' PortF.4 - Pin 49   A12  -> SRam
' PortF.5 - Pin 40   A13  -> SRam
' PortF.6 - Pin 41   A14  -> SRam
```

```
' PortF.7 - Pin 42   A15  -> SRam

' PortH.2 - Pin 57   A16  -> SRam
' PortH.3 - Pin 58   A17  -> SRam
' PortH.4 - Pin 59   A18  -> SRam
' PortH.5 - Pin 60   A19 - unbenutzt
' PortH.6 - Pin 61   A20 - unbenutzt
' PortH.7 - Pin 62   A21 - unbenutzt

'-------------------------------------------------------------------
---------------
'----------generate a 32 MHz system clock by use of the PLL (2MHz
* 23 = 46MHz)

   Config Osc = Disabled , Extosc = Enabled

   'Set the Multiplication factor and select the clock Reference
for the PLL
   'Osc_pllctrl = &B00_0_10100
'2MHz clock Source and Multiplication factor = 23
   '                00        : 2 MHz internal OSC
   '                01        : Reerved
   '                10        : 32 MHz internal OSC
   '                11        : External Clock Source
   '                   1      : 0=PLL-Output devided by 1
|1=PLL-Output devided by 2
   '                     xxxxx: Multiplikation of PLL 1-31
   Osc_pllctrl = &B11_0_01000 : Const Mhz = 32              ' 32
MHz

   'enable PLL
   Set Osc_ctrl.4                                          'PLL
enable

   'configure the systemclock
   Config Sysclock = Pll                                   'use
PLL

'-------------------------------------------------------------------
---------------
'-------------------------------------------------------------------
---------------
   Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8

   Open "com1:" For Binary As #1

   ' Termninal initialisieren
   Printbin #1 , &H1B ; &H5B ; &H30 ; &H6D                 ' All
```

```
attributes off(normal)
   Printbin #1 , &H1B ; &H5B ; &H32 ; &H4A                    '
Bildschirm löschen
   Printbin #1 , &H1B ; &H5B ; &H48 ;                         '
Cursor Home
   Printbin #1 , &H1B ; &H5B ; &H3F ; &H32 ; &H35 ; &H68 ;    '
Cursor an

'-------------------------------------------------------------
---------------
   ' Einstellungen externer Speicher
   ' Alle EBI-Ports müssen auf OUTPUT
   ' ALLE Ports, die ATKIV-LOW sind müssen auf 1 gesetzt werden
!!!
   ' ALLE Ports, die ATKIV-HIGH sind müssen auf 0 gesetzt werden
!!!

   Print #1 , "Config Ports for external Adress / Data-Bus with
no ALE... ";

   Portj_dirset = &B1111_1111 : Portj = &B0000_0000           '
D0:7
   Portj_pin0ctrl = &B0_0_000_000
'Totem (PushPull)
   Portj_pin1ctrl = &B0_0_000_000
'Totem (PushPull)
   Portj_pin2ctrl = &B0_0_000_000
'Totem (PushPull)
   Portj_pin3ctrl = &B0_0_000_000
'Totem (PushPull)
   Portj_pin4ctrl = &B0_0_000_000
'Totem (PushPull)
   Portj_pin5ctrl = &B0_0_000_000
'Totem (PushPull)
   Portj_pin6ctrl = &B0_0_000_000
'Totem (PushPull)
   Portj_pin7ctrl = &B0_0_000_000
'Totem (PushPull)

   Portk_dirset = &B1111_1111 : Portk = &B1111_1111           '
A0:7
   Portk_pin0ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portk_pin1ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portk_pin2ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portk_pin3ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
```

```
   Portk_pin4ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portk_pin5ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portk_pin6ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portk_pin7ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper

'  PortX_pinYctrl = &B0_0_001_000 : X= Port A...   Y= Bit Nr. 0-7
   '                   X          :7  : 0= SlewRate normal, 1=
SlewRate limited
   '                     X        :6  : 0= IO normal, 1= IO
inverted
   '                       XXX    :5:3: 000 = Totem (PushPull)
   '                       XXX    :5:3: 001 = Totem + Buskeeper
   '                       XXX    :5:3: 010 = Totem + Pulldown on
Input
   '                       XXX    :5:3: 011 = Totem + PullUp on
Input
   '                       XXX    :5:3: 100 = Wired or
   '                       XXX    :5:3: 101 = Wired and
   '                       XXX    :5:3: 110 = Wired Or + PullDown
   '                       XXX    :5:3: 111 = Wired And + PullUp
   '                           XXX:2:0: 000 = Both edges trigger
port Interrupts / Events
   '                           XXX:2:0: 001 = Rising edge trigger
port Interrupts / Events
   '                           XXX:2:0: 010 = Falling edge trigger
port Interrupts / Events
   '                           XXX:2:0: 011 = Low Level trigger
port Interrupts / Events
   '                           XXX:2:0: 100 = Reserved
   '                           XXX:2:0: 101 = Reserved
   '                           XXX:2:0: 110 = Reserved
   '                           XXX:2:0: 111 = Input Buffer
Disabled (Only Port A to F) for use with ADC or AC


   Portf_dirset = &B1111_1111 : Portf = &B1111_1111         '
A8:15
   Portf_pin0ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portf_pin1ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portf_pin2ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portf_pin3ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
```

```
   Portf_pin4ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portf_pin5ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portf_pin6ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Portf_pin7ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper

   Porth_dirset = &B1111_1111 : Porth = &B1111_1111          ' WR,
RD , A16, A17, A18, A19, A20, A21
   Porth_pin0ctrl = &B0_0_000_000
'Totem (PushPull)
   Porth_pin1ctrl = &B0_0_000_000
'Totem (PushPull)
   Porth_pin2ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Porth_pin3ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Porth_pin4ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Porth_pin5ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Porth_pin6ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper
   Porth_pin7ctrl = &B0_0_001_000
'Totem (PushPull) + Buskeeper

   Porte_dirset = &B1111_1111 : Porte = &B1111_0000          '
CS3, CS2, CS1, CS0
   Porte_pin0ctrl = &B0_0_000_000
'Totem (PushPull) CS3
   Porte_pin1ctrl = &B0_0_000_000
'Totem (PushPull) CS2 SRAM
   Porte_pin2ctrl = &B0_0_000_000
'Totem (PushPull) CS1 SRAM
   Porte_pin3ctrl = &B0_0_000_000
'Totem (PushPull) CS0 TFT
   Porte_pin4ctrl = &B0_0_000_000
'Totem (PushPull)
   Porte_pin5ctrl = &B0_0_000_000
'Totem (PushPull)
   Porte_pin6ctrl = &B0_0_000_000
'Totem (PushPull)
   Porte_pin7ctrl = &B0_0_000_000
'Totem (PushPull)


   ' EBI-OUT legt die 4 ChipSelect's auf einen anderen Port,
```

```
damit die Adressleitungen
   ' A16 bis A21 auf Port-H frei wird.
   Portcfg_ebiout = &B0000_00_11
'              XXXX          : RESERVED
'                -> 00       : EBI Port3 adress output on PORT-F
0..7: SD: 4'h0, A[11:8] - SR or SR-LPC with SD on CS3: A[23:16] -
SR NoAle or ALE1: A[15:8]
'                   01       : EBI Port3 adress output on PORT-E
0..7: SD: 4'h0, A[11:8] - SR or SR-LPC with SD on CS3: A[23:16] -
SR NoAle or ALE1: A[15:8]
'                   10       : EBI Port3 adress output on PORT-F
4..7: SD: A[11:8]       - SR or SR-LPC with SD on CS3: A[19:16] -
SR NoAle or ALE1: ---
'                   11       : EBI Port3 adress output on PORT-E
4..7: SD: A[11:8]       - SR or SR-LPC with SD on CS3: A[19:16] -
SR NoAle or ALE1: ---

'                      00    : EBI CS-output on PORT-H 4..7
'                      01    : EBI CS-output on PORT-L 4..7
'                      10    : EBI CS-output on PORT-F 4..7
'                   ->  11   : EBI CS-output on PORT-E 4..7


   Ebi_ctrl = &B01_00_11_10                                    'SRAM
ALE12, 3Port
'                XX          : 00: 4 Bit Data Bus
'                XX      ->  : 01: 8 Bit Data Bus
'                XX          : 10: RESERVED
'                XX          : 11: RESERVED

'                   XX   ->  : 00: LPC-Mode: ALE1
'                   XX       : 01: LPC-Mode: RESERVED
'                   XX       : 10: LPC-Mode: ALE12
'                   XX       : 11: LPC-Mode: RESERVED

'                      XX    : 00: ALE1 - Adressbyte 0 and 1
multiplexed
'                      XX    : 01: ALE2 - Adressbyte 0 and 2
multiplexed
'                      XX    : 10: ALE12- Adressbyte 0,1 and 2
multiplexed
'                   ->  XX   : 11: NOALE- No adress multiplexing

'                         XX: 00: Externer Bus disabled
'                         XX: 01: 3 Port
Control-Bus # Data-Bus # A0:A7 und A8:A15 über ALE1 gemuxt
'                   ->    XX: 10: 4 Port
'                         XX: 11: 2 Port
```

```
'-----------------------------------------------------------------
--------------
'-----------------------------------------------------------------
--------------
'-----------------------------------------------------------------
--------------
'-----------------------------------------------------------------
--------------
   'ChipSelect0 für  512 KByte SRam

'                        xxxxxxxx xxxx----
'                  Bit: 23    bis    12          11 bis 0 nicht
verwendet
'                        |           |
   Ebi_cs0_baseaddr = &B00010000_00000000                    '
Start &H100000 = 1M
'                        00000000_00100000_00000000 = &H002000 =
8K
'                        00000000_00110000_00000000 = &H003000 =
12K
'                        00000000_01000000_00000000 = &H004000 =
16K
'                        00000000_10000000_00000000 = &H008000 =
32K
'                        00000001_00000000_00000000 = &H010000 =
64K
'                        00000010_00000000_00000000 = &H020000 =
128K
'                        00000100_00000000_00000000 = &H040000 =
256K
'                        00001000_00000000_00000000 = &H080000 =
512K
'                  ->    00010000_00000000_00000000 = &H100000 =
1M
'                        00100000_00000000_00000000 = &H800000 =
2M
'                        01000000_00000000_00000000 = &H800000 =
4M
'                        10000000_00000000_00000000 = &H800000 =
8M

   Ebi_cs0_ctrla = &B0_01011_01                                  '
Size &H080000 = 512 KB
'                     X           : RESERVED
'                       XXXXX     : AdressBlockSize
'                       00000     : 256 Byte
'                       00001     : 512 Byte
'                       00010     : 1 KByte
'                       00011     : 2 KByte
```

```
'                       00100     : 4 KByte
'                       00101     : 8 KByte
'                       00110     : 16 KByte
'                       00111     : 32 KByte
'                       01000     : 64 KByte
'                       01001     : 128 KByte
'                       01010     : 256 KByte
'                  ->   01011     : 512 KByte
'                       01100     : 1 MByte
'                       01101     : 2 MByte
'                       00110     : 4 MByte
'                       01111     : 8 MByte
'                       10000     : 16 MByte
'                             XX  : ChipSelectMode
'                             00  : disabled
'                       ->    01  : Enabled for SRAM
'                             10  : Enabled for SRAM LPC
(LowPinCount)
'                             11  : Enabled for SD-SRAM

   Ebi_cs0_ctrlb = &B00000_100                                ' je
nach Geschwindigkeit des SRam's
'                     XXXXX       : RESERVED
'                          XXX    : Waitstates
'                          000    : 0 Waitstates
'                          001    : 1 Waitstates
'                          010    : 2 Waitstates
'                          011    : 3 Waitstates
'                       ->  100   : 4 Waitstates
'                          101    : 5 Waitstates
'                          110    : 6 Waitstates
'                          111    : 7 Waitstates

'-------------------------------------------------------------------
---------------

' Nun kann das externe SRam genauso wie das interne SRam
angesprochen werden.
' Vorteil: erheblich schneller im Zugriff, wie DRam !!!
' So kann auch Hardware "memory-mapped" eingebunden werden
' oder ein ISA-Bus realisiert werden.

' Now you can use the external SRAM just like the internal SRAM.
' This is much faster like DRAM
' This way you can also map hardware and access registers as you
would do for SRAM

$xramstart = &H100000
$xramsize = &H080000
```

4.12 Attaching an LCD Display

A LCD display can be connected with two methods.

- By wiring the LCD-pins to the processor port pins. This is the pin mode. The advantage is that you can choose the pins and that they don't have to be on the same port. This can make your PCB design simple. The disadvantage is that more code is needed.
- By attaching the LCD-data pins to the data bus. This is convenient when you have an external RAM chip and will add only a little extra code.

The LCD-display can be connected in PIN mode as follows:

LCD DISPLAY	PORT	PIN
DB7	PORTB.7	14
DB6	PORTB.6	13
DB5	PORTB.5	12
DB4	PORTB.4	11
E	PORTB.3	6
RS	PORTB.2	4
RW	Ground	5
Vss	Ground	1
Vdd	+5 Volt	2
Vo	0-5 Volt	3

This leaves PORTB.1 and PORTB.0 and PORTD for other purposes.

You can change these pin settings from the Options LCD 148 menu.

BASCOM supports many statements to control the LCD-display.

For those who want to have more control of the example below shows how to use the internal BASCOM routines.

```
$ASM
  Ldi _temp1, 5         'load register R24 with value
  Rcall _Lcd_control    'it is a control value to control the display
  Ldi _temp1,65         'load register with new value (letter A)
  Rcall _Write_lcd      'write it to the LCD-display
$END ASM
```

Note that _lcd_control and _write_lcd are assembler subroutines which can be called from BASCOM.

See the manufacturer's details from your LCD display for the correct pin assignment.

4.13 Memory usage

SRAM

Every variable uses memory. Variables are stored in memory. This memory is also called SRAM (static ram).

The available memory depends on the chip. When you double click on the chip pinout, you can view the parameters of the used chip.

A special kind of memory are the registers in the AVR. Registers 0-31 have addresses 0-31.
Almost all registers are used by the compiler or might be used in the future.
Which registers are used depends on the program statements you use.

This brings us back to the SRAM.
No SRAM is used by the compiler other than the space needed for the software stack ($SWSTACK 705) and frame
($FRAMESIZE 641)
Some statements might use some SRAM. When this is the case it is mentioned in the help topic of that statement.

For example, CONFIG CLOCK 895 in user mode requires variables to hold the time. Variables like _sec , _min , _hour, _day , _month , _year.

Each 8 bits used occupy one byte. When you dimension 1 bit, you will also use 1 byte.
Each byte variable occupies one byte.
Each integer/word variable occupies two bytes.
Each Long, Dword or Single variable occupies four bytes.
Each double variable occupies 8 bytes.
Each string variable occupies at least 2 bytes.
A string with a length of 10 occupies 11 bytes.

 Strings need an additional byte (Null termination) to indicate the end of the string. That's why a string of 10 bytes occupies 11 bytes.

 With dimension of a bit you will occupy one byte.

Use bits or byte variables wherever you can to save memory. (not allowed for negative values)

See also DIM 1228

The software stack is used to store the addresses of LOCAL variables and for variables that are passed to SUB routines.

Each LOCAL variable and passed variable to a SUB/FUNCTION, requires two bytes to store the address (because it is a 16-Bit address = 2 bytes).
So when you have a SUB routine in your program that passes 10 variables, you need 10 * 2 = 20 bytes.
When you use 2 LOCAL variables in the SUB program that receives the 10 variables, you need additional 2 * 2 = 4 bytes.

See also DECLARE SUB 1221, DECLARE FUNCTION 1215

The software stack ($SWSTACK 705) size can be calculated by taking the maximum number of parameters in a SUB routine, adding the number of LOCAL variables and multiplying the result by 2. To be safe, add 4 more bytes for internally used LOCAL variables.

LOCAL variables are stored in a place that is named the Frame ($FRAMESIZE 641)

When you have a LOCAL STRING with a size of 40 bytes, and a LOCAL LONG, you need 41 + 4 bytes = 45 bytes of frame space.

When you use conversion routines such as STR 836, VAL 839, HEX 833, INPUT 1493 etc. that convert from numeric to string and vice versa, you also need a frame. Note that the use of the INPUT 1493 statement with a numeric variable, or the use of the PRINT 1501 or LCD 657 statement with a numeric variable, will also force you to reserve 24 bytes of frame space. This because these routines use the internal numeric<>string conversion routines.

 In fact, the compiler creates a buffer of 24 bytes that serves as scratchpad for temporary variables, and conversion buffer space. So the frame space should be **24 at minimum** ($FRAMESIZE 641 = 24). This 24 Byte start at the beginning of the Frame which act as the conversion buffer within the frame

For an ATXMEGA or ATMEGA you have usually enough SRAM so you can start with higher values of Stack and Frame.

With an ATTINY13 and 64Byte SRAM it is a challenge but also start with all stack defined and lower the Stack Values when your application program grows.

- Avoid to use SUB or FUNCTIONS (If you want to save SRAM space)
- If you use Functions like PRINT, LCD, INPUT and the FP num <> FORMAT(), String conversion you need to define the 24 Byte conversion buffer (at least 24Byte for Software Stack + FRAME together).

```
$hwstack = 30
$swstack = 0
$framesize = 24
```

In this case just 9 Bytes are left for global variables !

See also: $HWSTACK 648, $SWSTACK 705, $FRAMESIZE 641

XRAM

Some processors have an external memory interface. For example the ATMEGA128 has such an interface.
The additional memory is named XRAM memory (extended or external memory).
When you add 32 KB RAM, the first address will be 0.
But because the XRAM can only start after the internal SRAM, the lower memory locations of the XRAM will not be available for use. The processor will automatically use the SRAM if an address is accessed that is in range of the SRAM memory.
Thus adding 32KB of XRAM, will result in a total of 32 KB RAM.

With ATXMEGA you can add XRAM with the EBI (External Bus Interface). There is no problem to add for example
16 MByte of external SDRAM.
See CONFIG XRAM 1161

ERAM

Most AVR chips have internal EEPROM on board.
This EEPROM can be used to store and retrieve data.
In BASCOM, this data space is called ERAM.

An important difference is that an ERAM variable can only be written to a maximum of 100.000 times. So only assign an ERAM variable when it is required, and **never** use it in a loop or the ERAM will become unusable.
Always use the Brown out detection of the processor to prevent EEPROM corruption.

See also DIM 1228
For ATXMEGA see also CONFIG EEPROM 952

Constant code usage

Constants are stored in a constant table.
Each used constant in your program will end up in the constant table.

For example:

```
Print "ABCD"
Print "ABCD"
```

This example will only store one constant (ABCD).

```
Print "ABCD"
Print "ABC"
```

In this example, two constants will be stored because the strings differ.

Stack

See also: $HWSTACK 648, $SWSTACK 705, $FRAMESIZE 641

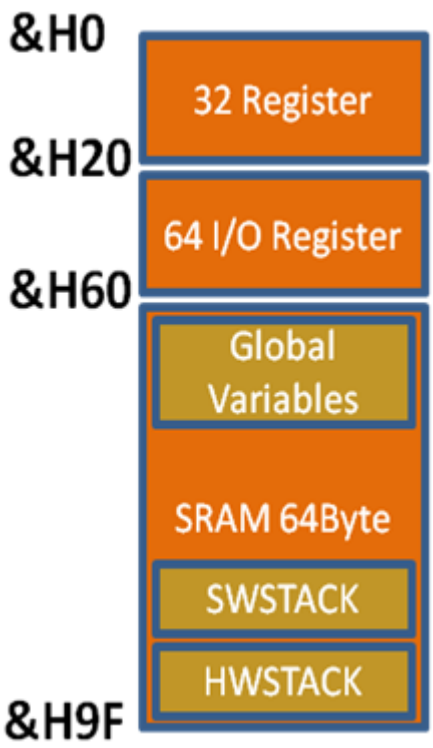
The Stack is a part of SRAM (Static RAM). In SRAM the compiler stores user dimensioned variables, as well as internal variables, but SRAM holds also Hardware Stack, Software Stack and Frame. The Variables always start at the lowest SRAM Address. After Reset all SRAM Bytes are 0 (and strings are "") so the SRAM memory is cleared after reset. With the $noramclear option you can turn this behavior off which means the SRAM is not cleared after reset.

The available SRAM depends on the Chip.

With ATTINY13 for example you have 64Byte of SRAM and you will find this information beside the user manual in the *.DAT file. You can also double click the chip in Chip Pinout to view the chip parameters.

The following you find in the *attiny13.dat* file: *SRAM = 64 ; SRAM size*

Global Variables start with the lowest SRAM Address and the Hardware Stack start with the highest SRAM Address.

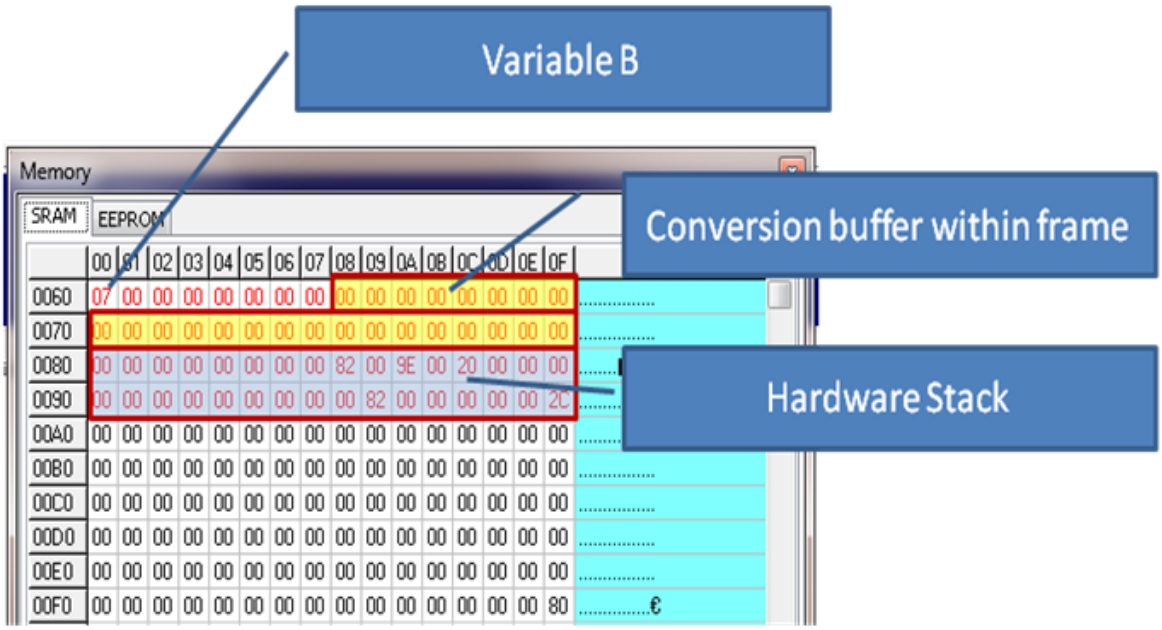


Example for using with Bascom-AVR Simulator:

```
$regfile = "attiny13.dat"
$crystal =  4000000
$hwstack = 30
$swstack = 0
$framesize = 24

Dim B As Byte
B = 5


Pcmsk = &B00000001 'PIN   Change   Int
ON   PCINT0    pin_change_isr
Set Gimsk. 5
Enable Interrupts

Do
!NOP
Loop

End 'end   program

pin_change_isr:
  B = 7
Return
```

With an ATTINY13 the SRAM is just 64Byte and it is easy to see which SRAM Bytes will be overwritten with Bascom AVR Simulator Memory Window.

Click on **M** to display the memory window.



Picture: SRAM of ATTINY13 when executing the above ATTINY13 example in Bascom Simulator

You can see the Hardware Stack (32 Byte) , Frame (24 Byte) and the Variable B.
For this example you do not really need a Frame so it could be also `$framesize = 0` for this example.

With ATXMEGA128A1 there is 8K Byte of SRAM available and you can find in the DAT file (*SRAM = 8192 ; SRAM size*)

The Values of Stack should be ALWAYS defined at the beginning of any BASCOM-AVR Program in the main project file. The best place is right after the $REGFILE 694 statement.

Example:

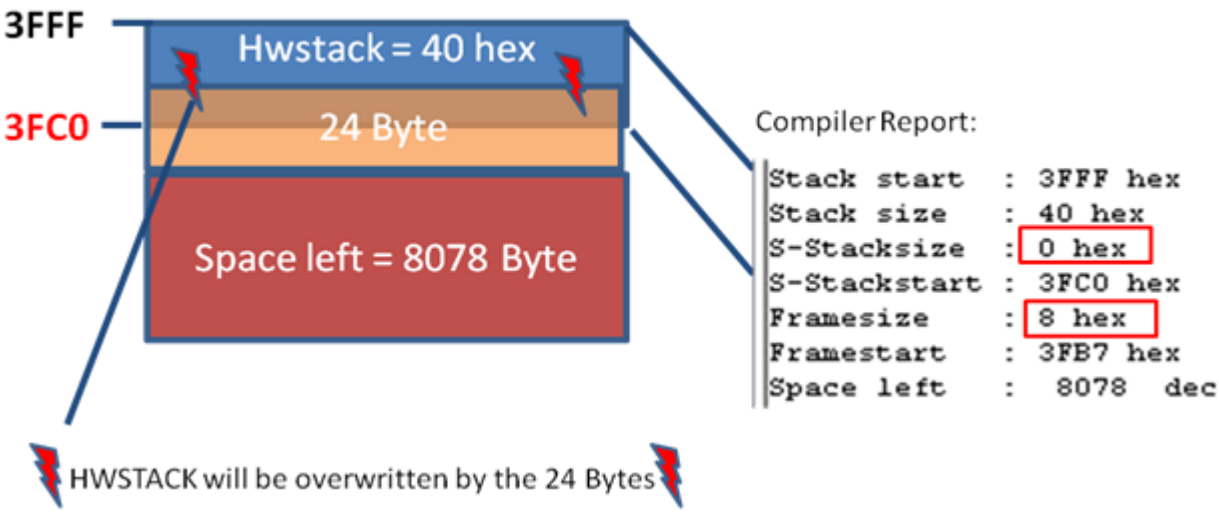
```
$hwstack = 32                                   ' default use 32 for the hardware stack
$swstack = 32                                   ' default use 32 for the SW stack
$framesize = 40                                 ' default use 40 for the frame space
```

The following example show what can happen when you define NO Stacks or Frame or when you define not enough Stack or Frame.

In this example we use: $hwstack = 64, $swstack = 0, $framesize = 8

As we know now Software Stack and FRAME together must be as absolute minimum 24 Byte (for the conversion buffer) so we force the overwriting of Hardware Stack which causes malfunction.

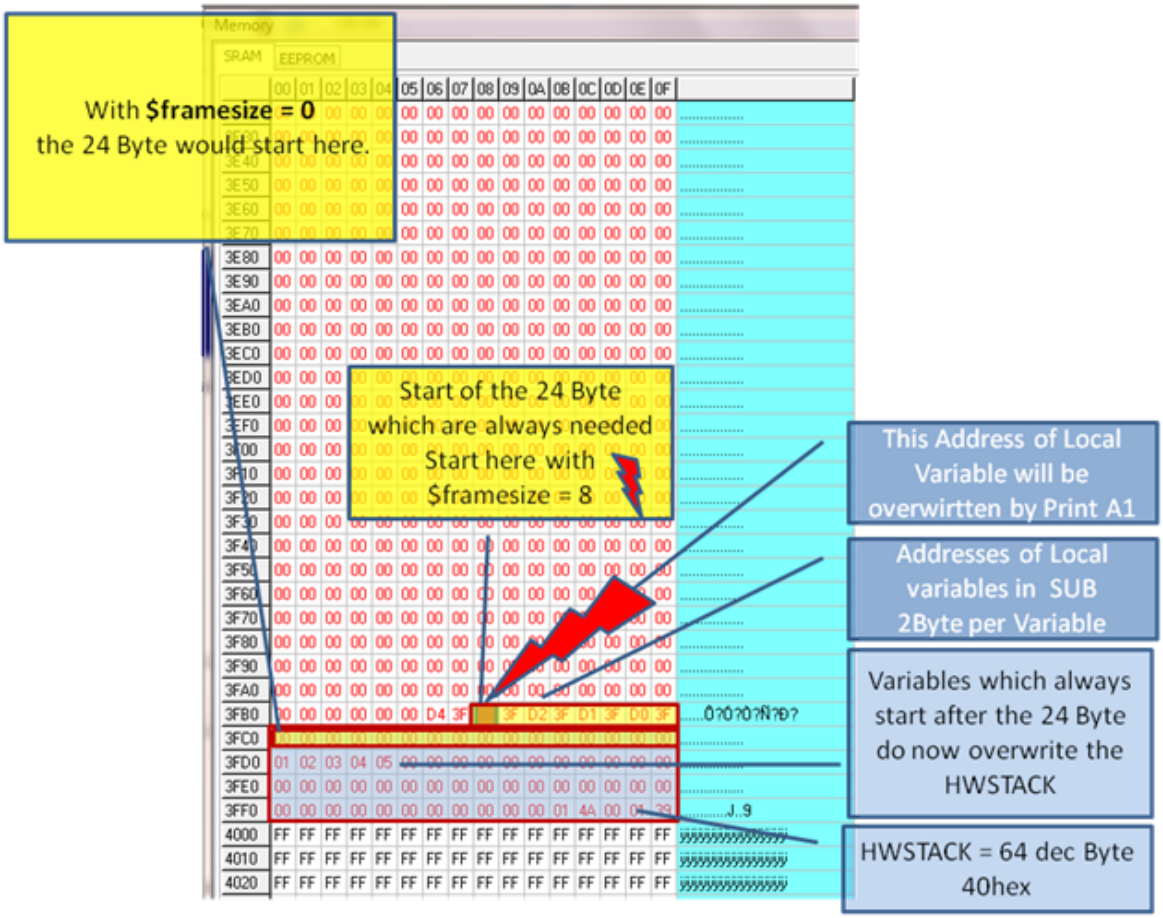
(Reminder: Don't start with the lowest values for Stack and Frame)



Picture : SRAM for example with$hwstack = 64, $swstack = 0, $framesize = 8

You can now imagine what could happen:

- Because of overwritten return address in Hardware Stack the micro is jumping to somewhere else and malfunction if forced.
- Functions like PRINT overwrite addresses of LOCAL Variables and here also will the micro jump to somewhere else and malfunction is forced.



Picture: Simulator Memory Windows for example with $hwstack = 64, $swstack = 0, $framesize = 8

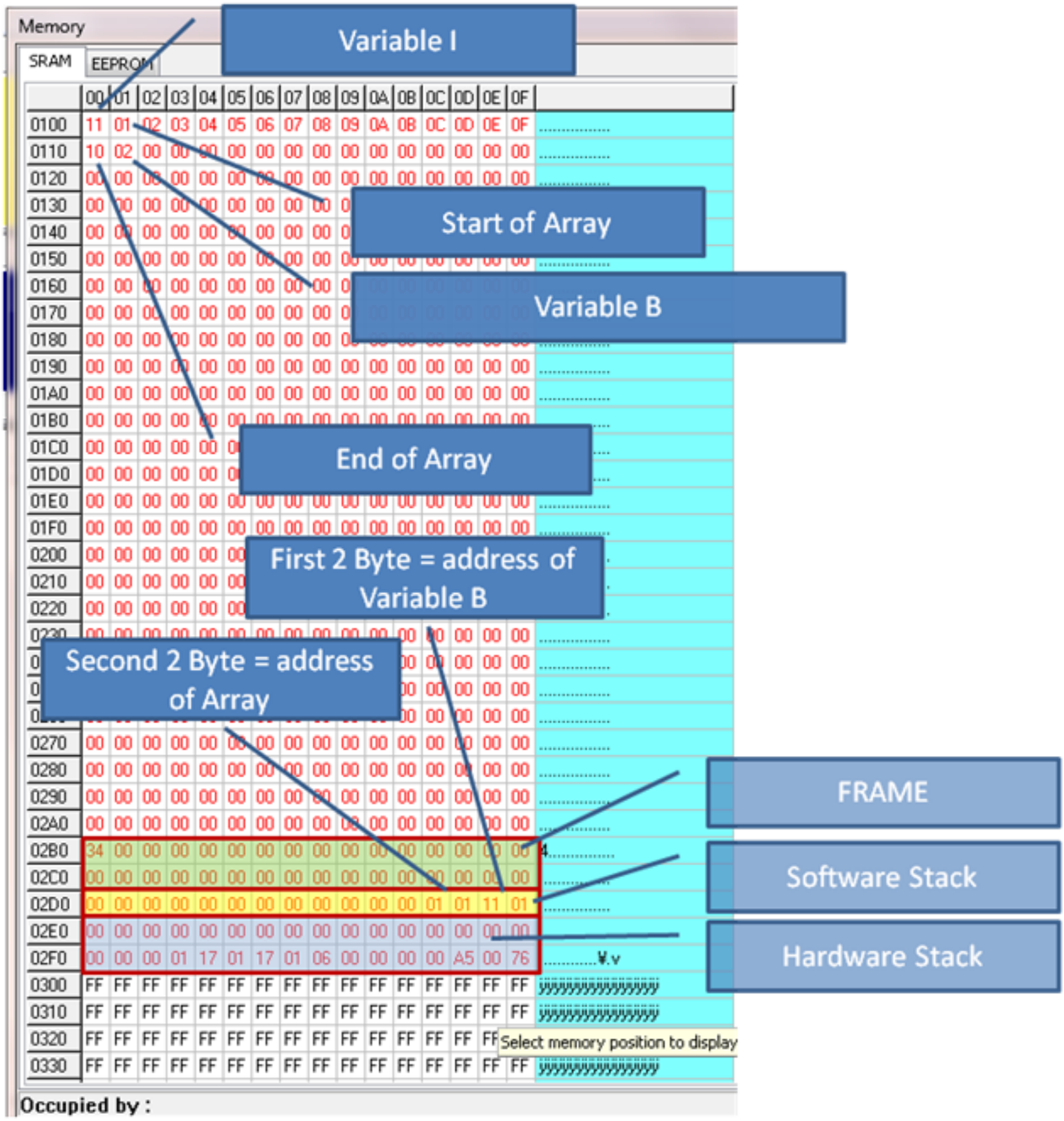
Now an example for passing an Array to a SUB:

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
$hwstack = 32
$swstack = 16
$framesize = 32


Dim I As Byte
I = 0


Dim Ar(16) As Byte

For I = 1 To 16
  Ar(i) = I
Next

Dim B As Byte
B = 2

Declare Sub Testarray(by_ref As Byte , Pass_array As Byte)


Call Testarray(b , Ar(1))
Print "ar(4) = " ; Ar(4)


End                                                 'end program


Sub Testarray(by_ref As Byte , Pass_array As Byte)        'start sub
  Print "b5=" ; Pass_array(5)                        'print passed variables
  Print "b6=" ; Pass_array(6)                        'print passed variables
End Sub
```

With this example we see the complete SRAM.
The SRAM start with the dimed variables. In this case it start with the variable I followed by the Array Ar of 16 Byte and in the end the variable **B**.

Because it is easier with the memory window of Bascom Simulator I choose multiple of 16 for Stack and Framesize.
We have here 2 Addresses stored in Software Stack. One address for the Array and one address for the variable **B**.

So passing an Array to a SUB just need 2 Bytes for the address in Stack which is the same size as for one Byte variable (here variable **B**).



Picture: Simulator Memory Window for example passing an Array to a SUB

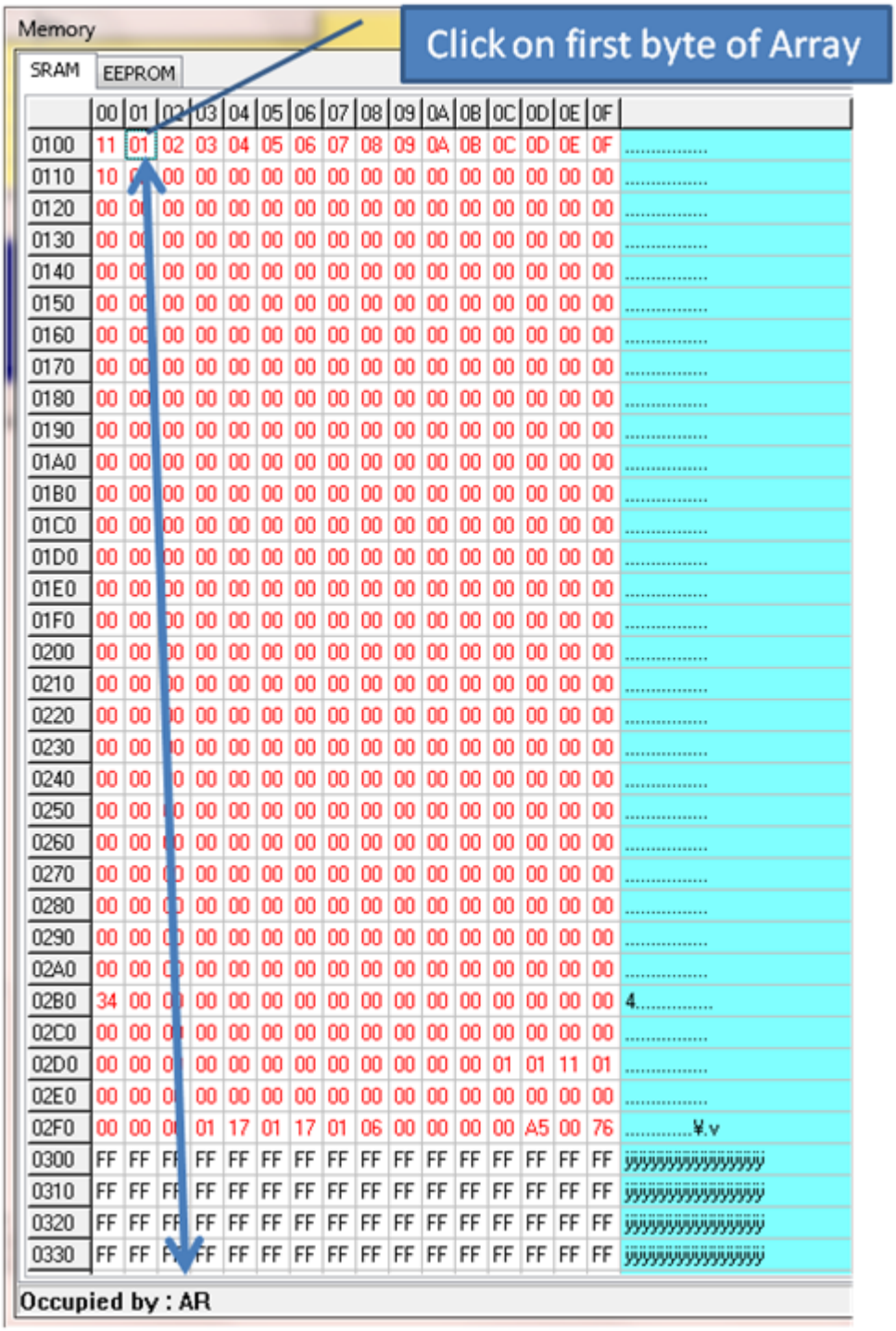
With this example you also see that especially with ATTINY and smaller ATMEGA it is not that complicated to see if other SRAM bytes will be overwritten by something and causes malfunction.
You have with the Simulator window the "big picture" of SRAM and STACK together.

As already written it is easier to use multiple of 16 for Hardware Stack, Software Stack and FRAME as a starting point because one line in Simulator Memory window is 16 Bytes.

How to see which Variables are stored on which SRAM Byte ?

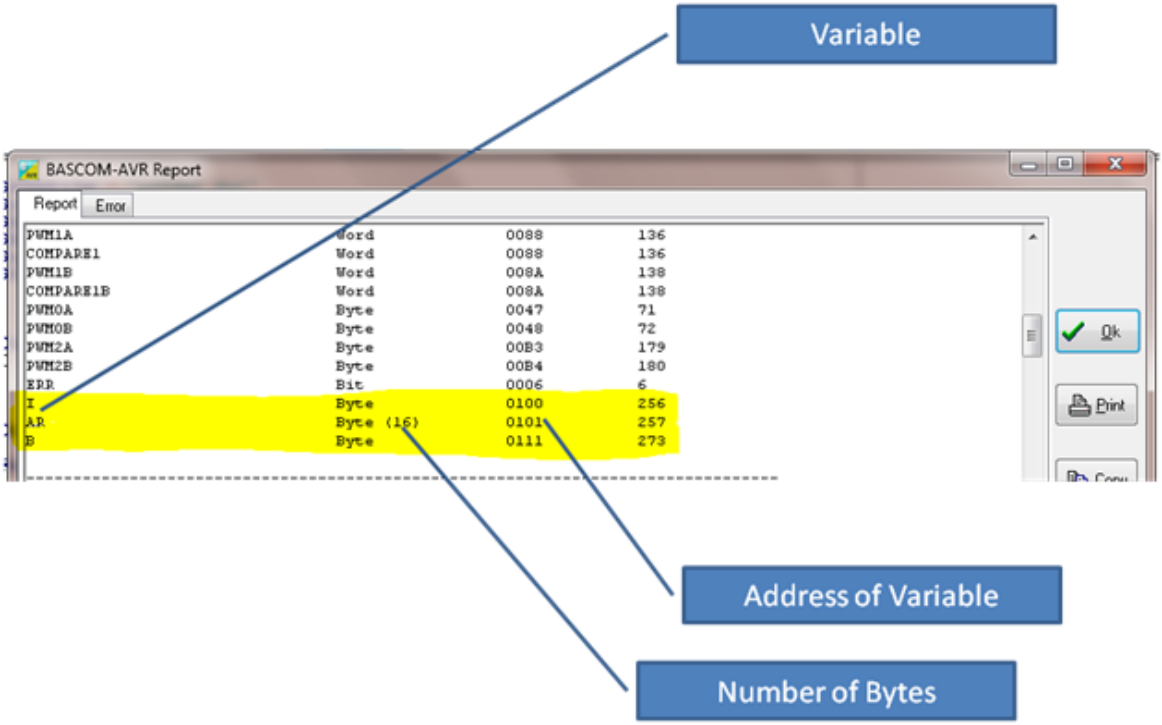
You can find out the stored variable with the Bascom-AVR Simulator Memory Window

by clicking on that byte.
Click on SRAM Bytes show the OCCUPIED BY in the footer of that window.
Only the first Byte of an Array will show the Name of the Array !



Picture: How to see which Variables are stored on which SRAM Byte

You can also find this information in the Compiler output report:
In this case under VARIABLES

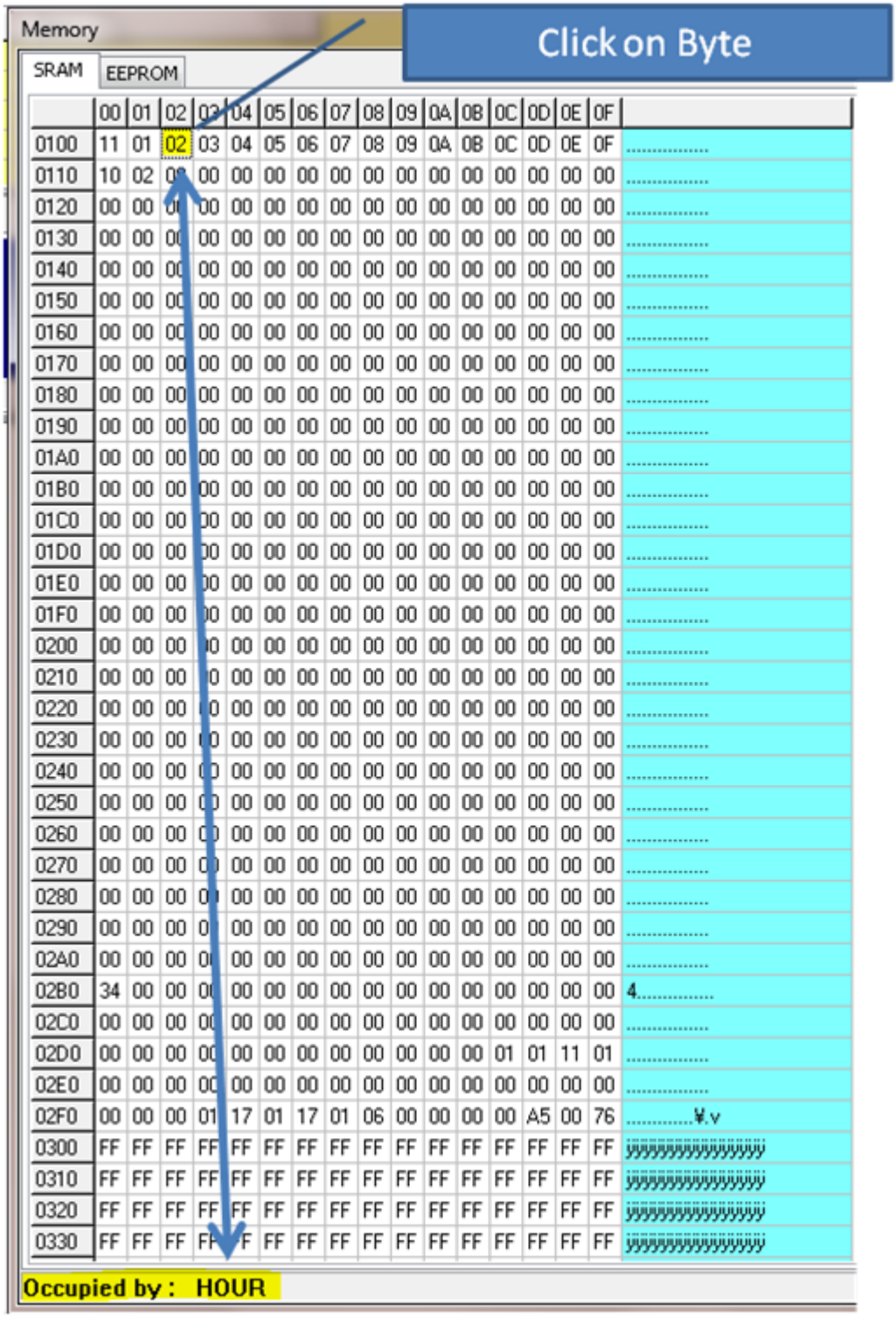


Picture: How to see which Variables are stored on which SRAM Address

The following small example is good for examining the Bascom-AVR internal variables like _sec, _min or _hour in Bascom-AVR Simulator Memory Window.

Config Clock = User for example create the internal variables for seconds (_sec), minutes (_min) ,hour (_hour) etc…. You can see this variables by clicking on the SRAM Byte and watch the footer of that Bascom-AVR Simulator Memory Window footer.

```
$regfile = "m88def.dat"
$hwstack = 48
$swstack = 80
$framesize = 80

Config  Clock = User

End 'end    program
```



Picture: Internal Variables in the Bascom-AVR Simulator Memory Window

See also: $HWSTACK 648, $SWSTACK 705, $FRAMESIZE 641

4.14 Statements and Hardware Resources

Some of the BASCOM statements and functions use a hardware resource.
This is a list of hardware resources and the statement/functions that use them.

USART0
$BAUD, BAUD

USART1
$BAUD1 , BAUD1,

USARTx
BUFSPACE, CLEAR, ECHO, WAITKEY, ISCHARWAITING, INKEY, INPUTBIN, INPUTHEX, INPUT, PRINT, PRINTBIN

TIMER0
DCF77 , READHITAG , GETRC5 , CONFIG SERVOS , TIME$, DATE$

TIMER1
DTMFOUT , RC5SEND, RC6SEND , SONYSEND.

TIMER2
TIME$, DATE$

ADC
GETADC

EEPROM
READEEPROM, WRITEEPROM

TWI
I2CINIT, I2CRECEIVE, I2CSEND, I2START I2CSTOP I2CRBYTE I2CWBYTE

SPI
SPIIN, SPIINIT, SPIMOVE, SPIOUT - SPI

CAN
CONFIG CANBUS, CONFIG CANMOB, CANBAUD, CANRESET, CANCLEARMOB, CANCLEARALLMOBS, CANSEND, CANRECEIVE, CANID, CANSELPAGE, CANGETINTS

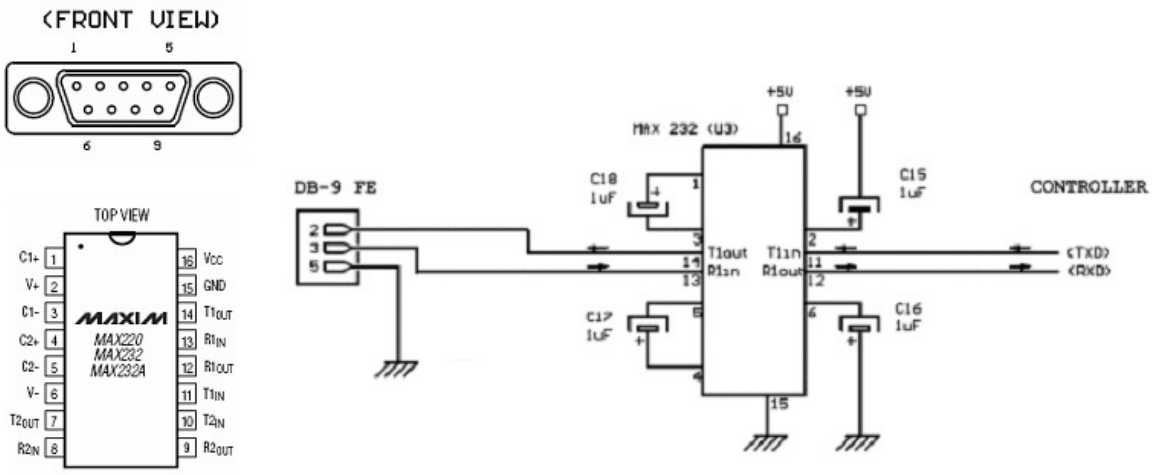
4.15 Using the UART

UART

A Universal Asynchronous Receiver and Transmitter (UART) can be used to send and receive data between two devices. More specific these devices can be PC-to-PC, PC-to-micro controller and micro controller-to-micro controller. The UART communicates using TTL voltages +5V and 0V or LVTTL depending on your micro controllers VCC voltage.

If you wish to connect to a PC you need to use RS232 protocol specifications. This means that the hardware communication is done with specific voltage levels. (+15V and -15V) This can be achieved by using a MAX232 level shifter.

The hardware is explained in this schematic:



The DB-9 connector has 9 pins but you only need to use 3 of them. Notice that the drawing above shows the FRONT VIEW thus remember that you are soldering on the other side. On most connectors the pin outs can also be found on the connector itself.

If your controller has no UART you can use a software UART see below. If your controller has one UART you connect controller pins TxD and RxD to TxD and RxD in the schematic above. If your controller has more than one UART you connect controller pins TxD0 and RxD0 to TxD and RxD in the schematic above.
You now need to initialize the program in your micro controller, open a new .bas file and add the following code in the beginning of your program.

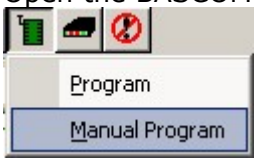
```
$regfile = "your micro here def.dat"
$crystal = 8000000
$baud = 19200
```

Make sure to define your micro controller after $regfile for example if you use the ATMega32

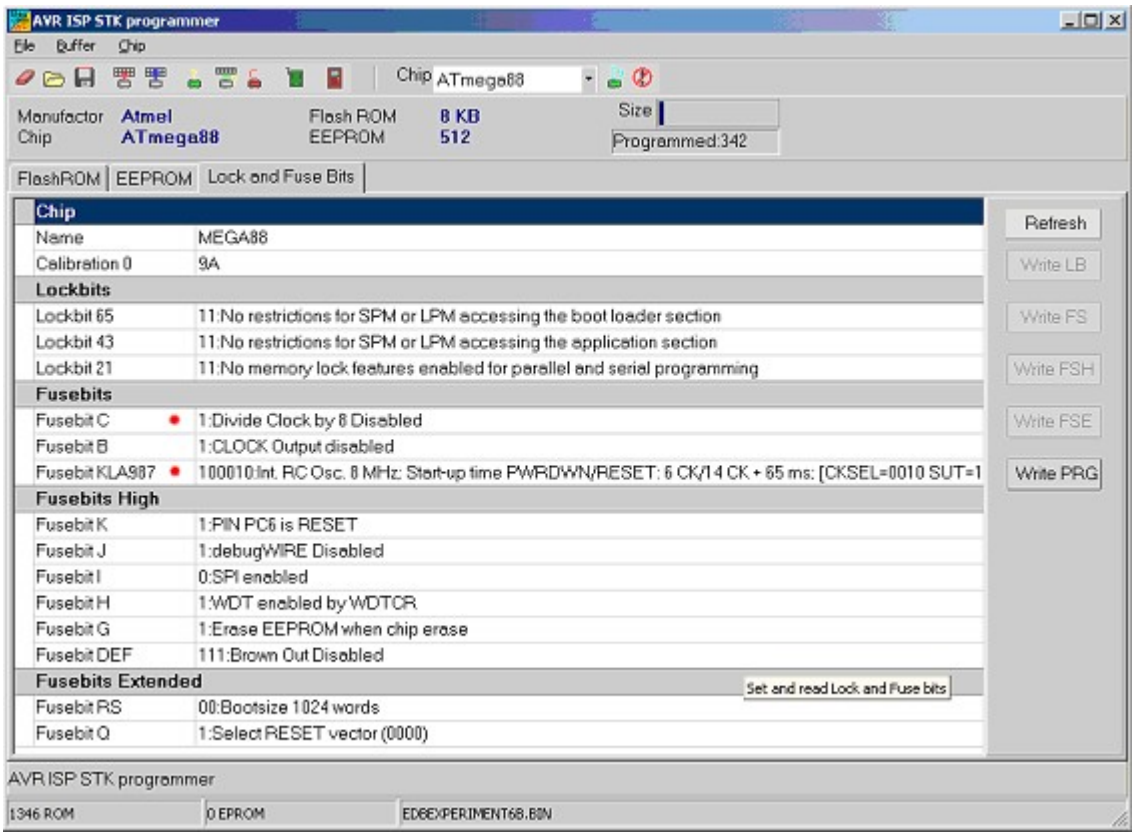
```
$regfile = "m32def.dat"
```

Some new chips can use an internal oscillator, also some chips are configured to use the internal oscillator by default. Using an internal oscillator means you do not need an external crystal.

Perform this step only if you have an internal oscillator.
Open the BASCOM-AVR programmer like this:



- Select the "Lock and Fuse Bits" tab and maximize the programmer window.
- Check if you see the following in the "Fusebit" section:
  "1:Divide Clock by 8 Disabled"
  and
  "Int. RC Osc. 8 MHz; Start-up time: X CK + X ms; [CKSEL=XXXX SUT=XX]"



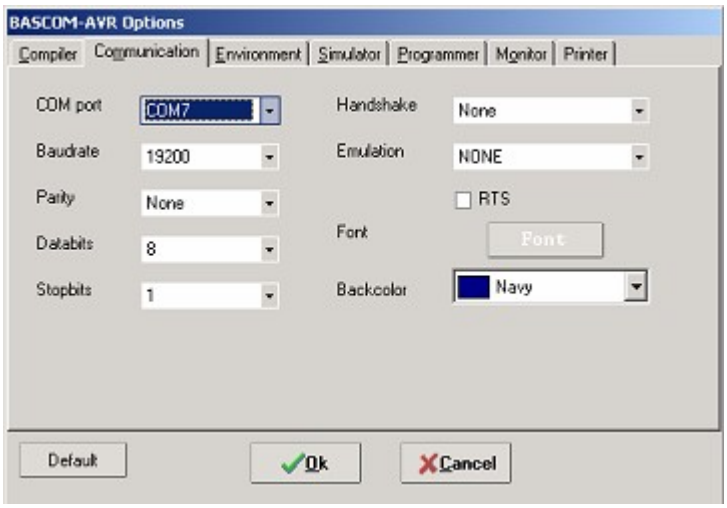
These options are not available for all AVR's, if you don't have the option do not change any fuse bits.

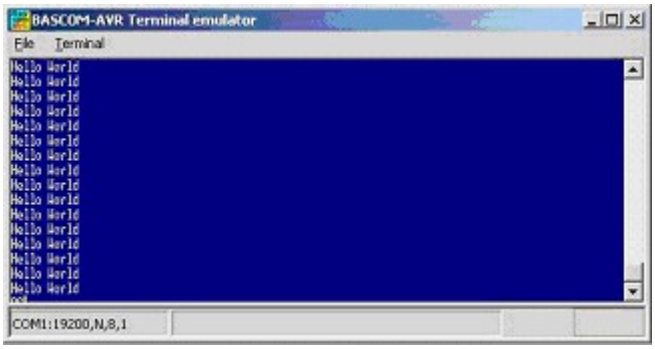
If these options are available, but in a wrong setting. Change the setting in the drop down box and click another Fuse section. Finally click the "Program FS" button. Click "Refresh" to see the actual setting.

Now connect a straight cable between the DB-9 connector, micro controller side and the PC side.
Program a test program into your micro controller, it should look like this:

```
$regfile = "m32def.dat"  'Define  your  own
$crystal = 8000000
$baud = 19200

Do
    Print "Hello      World"
   Waitms 25
Loop

End
```

Now open the BASCOM-AVR Terminal and set your connection settings by clicking "Terminal" -> "Settings" Select your computers COM port and select baud 19200, Parity none, Data bits 8, Stop bits 1, Handshake none, emulation none.





If you see the Hello World displayed in the BASCOM-AVR Terminal emulator window, your configuration is OK. Congratulations.

Example

You can also try this example with the BASCOM Terminal emulator, it shows you how to send and receive with various commands.

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200

Dim Akey As Byte  'Here we declare a byte variable

Print
Print "Hello, hit any alphanumerical key..."
Akey = Waitkey()  'Waitkey waits untill a char is received from the UART
Print Akey

Wait 1
Print
Print "Thanks!, as you could see the controller prints a number"
Print "but not the key you pressed."

Wait 1
Print
Print "Now try the enter key..."
Akey = Waitkey()
Akey = Waitkey()
Print Akey

Print
Print "The number you see is the ASCII value of the key you pressed."
```

```
Print "We need to convert the number back to the key..."
Print     'Notice what this line does
Print "Please try an alphanumerical key again..."
Akey = Waitkey()
Print Chr(akey) 'Notice what this does
Print "That's fine!"

Wait 1
Print
Print "For a lot of functions, just one key is not enough..."
Print "Now type your name and hit enter to confirm"

Dim Inputstring As String * 12          'Declare a string variable here


Do
Akey = Waitkey()
If Akey = 13 Then Goto Thanks           'On enter key goto thanks
   Inputstring = Inputstring + Chr(akey) 'Assign the string
Loop

Thanks:
Print "Thank you " ; Inputstring ; " !"             'Notice what ; does

Wait 1
Print
Print "Take a look at the program code and try to understand"
Print "how this program works. Also press F1 at the statements"
Print
Print "If you understand everything continue to the next experiment"

End
```

ASCII

As you could have seen in the previous example we use the PRINT statement to send something to the UART. Actually we do not send just text. We send ASCII characters. ASCII means American Standard Code for Information Interchange. Basically ASCII is a list of 127 characters.

ASCII Table (Incomplete)

Decimal	Hex	Binary	Value	
000	000	00000000	NUL	(Null char.)
008	008	00001000	BS	(Backspace)
009	009	00001001	HT	(Horizontal Tab)
010	00A	00001010	LF	(Line Feed)
012	00C	00001100	FF	(Form Feed)
013	00D	00001101	CR	(Carriage Return)
048	030	00110000	0	
049	031	00110001	1	
052	034	00110100	4	
065	041	01000001	A	
066	042	01000010	B	
067	043	01000011	C	

You can find a complete ASCII table here 600

CARRIAGE RETURN (CR) AND LINE FEED (LF)

In the previous example you can also see that a second print statement always prints the printed text to the following line. This is caused by the fact that the print statement always adds the CR and LF characters.

Basically if we state:
Print "ABC"
We send 65 66 67 13 10 to the UART. (In binary format)

The carriage return character (13) returns the cursor back to column position 0 of the current line. The line feed (10) moves the cursor to the next line.

Print "ABC" ;
When we type a semicolon (;) at the end of the line...
Bascom does not send a carriage return/line feed, so you can print another text after the ABC on the same line.

Print "ABC" ; Chr(13) ;
This would send only ABC CR. The next print would overwrite the ABC.

OVERVIEW

Here are some other commands that you can use for UART communications:

Waitkey( )
Waitkey will until a character is received in the serial buffer.

Ischarwaiting( )
Returns 1 when a character is waiting in the hardware UART buffer.

Inkey( )
Inkey returns the ASCII value of the first character in the serial input buffer.

Print
Sends a variable or non-variable string to the UART

ANOTHER EXAMPLE

This example shows how to use Ischarwaiting to test if there is a key pressed. And if there is, reac to a variable.

```
'Print "Press B key to start"
Dim Serialcharwaiting As Byte, Serialchar As Byte

Serialcharwaiting = Ischarwaiting()    'Check if B or b pressed then goto
If Serialcharwaiting = 1 Then
   Serialchar = Inkey()
   If Serialchar = 66 Or Serialchar = 98 Then
      Goto MyRoutine
   End If
End If

Goto Main

Myroutine:
'Statements

Main:
'Statements
End
```

BUFFERING SERIAL DATA

If you wish to send and receive data at high speed, you need to use serial input and

serial output buffers. This buffering is implemented in BASCOM-AVR and can only be used for hardware UART's.

To configure a UART to use buffers, you need to use the Config statement.

```
Config Serialout = Buffered , Size = 20
and/or
Config Serialin = Buffered , Size = 20
```

More information can be found in BASCOM-Help. Search topic = "config serialin" 1051. There is also a sample program "RS232BUFFER.BAS" in the samples folder if you wish a demonstration of the buffering.

SOFTWARE UART

The previous examples used the hardware UART. That means the compiler uses the internal UART registers and internal hardware (RxD(0) and TxD(0)) of the AVR. If you don't have a hardware UART you can also use a software UART.

The Bascom compiler makes it easy to "create" additional UART's. Bascom creates software UART's on virtually every port pin.

Remember that a software UART is not as robust as a hardware UART, thus you can get timing problems if you have lots of interrupts in your program.

For this example we use micro controller pins portc.1 and portc.2.
Connect portc.1 to TxD and portc.2 to RxD see the schematic above.

Change the $regfile and program this example:

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200

Dim B As Byte
Waitms 100

'Open a TRANSMIT channel for output
Open "comc.1:19200,8,n,1" For Output As #1
Print #1 , "serial output"

'Now open a RECEIVE channel for input
Open "comc.2:19200,8,n,1" For Input As #2
'Since there is no relation between the input and output pin
'there is NO ECHO while keys are typed

Print #1 , "Press any alpha numerical key"

'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
   'Store in byte
   B = Inkey(#2)
   'When the value > 0 we got something
   If B > 0 Then
      Print #1 , Chr(b)                          'Print the character
   End If
Loop
Close #2                                          'Close the channels
Close #1
```

```
End
```

After you have programmed the controller and you connected the serial cable, open the terminal emulator by clicking on  in Bascom.
You should see the program asking for an alphanumerical input, and it should print the input back to the terminal.

4.16 Using a BOOTLOADER

With booting we mean starting up like when you power a PC, after a reset.
When you reset the processor your user code will be executed.
A boot loader is a small program which task it is to re-program the processor. That is : in our case. A boot loader could do security checks or have other functions but when we say boot loader it always refer to some code that will reprogram the processor.

A boot loader typically runs just after a reset. But it could be initiated on any possible way. The recommended way is to use the statement : RESET MICRO.

A normal AVR and Xmega have a fuse that can be set to select the reset vector. It can point to address 0 which is the normal start address. Or it can point to an address at the end of flash memory.
When the fuse points to the boot vector, a jump is executed to this memory. And the boot code that is located at this memory is executed. Typically it will receive data and will flash/reprogram the memory from address 0 and up.
After that, the code at address 0 is executed and the normal user code runs.
The $LOADER directive is used to place the program code at the boot address at the end of memory. The exact address depends on other fuse settings and the processor used.
The normal user code always end up in location 0 and up.

The new Xtiny/MegaX and AVRX processors work different. A reset will always execute address 0. But when you want to program a boot loader, you use a Boot fuse to tell how big the boot loader is. A value of 1 will reserve a size of 512 bytes. A value of 2 will reserve 1024 bytes, etc. So you are flexible in selecting the size of the boot loader.
After the boot block your normal code starts. But now you see the difference : the normal user code runs at a higher address. So we must tell the compiler that all code must be relocated.
We use the $ROMSTART directive for that.

A boot loader does not have a $ROMSTART directive.
While the boot fuse uses blocks of 512 bytes, the flash page size can be different. Do not confuse this. AVRX processors typically have flash page size of 512 bytes.

For the AVRX series we include here a sample boot loader.

```
'-----------------------------------------------------------------
'                          (c) 1995-2025, MCS
'                        Bootloader-AVRX.bas
'-----------------------------------------------------------------
'  This sample demonstrates how you can write your own bootloader
'  in BASCOM BASIC for the AVR DB/DA series
```

```
' The AVRX has a different memory lay out and a different NVM
controller
' Normal AVR starts with normal application code and the boot
code is placed at the end of memory.
' AVRX starts with BOOT memory. So we always start in BOOT mode.
' With the BOOTEND fuse we can set the size of the BOOT area.
' Different processors have different page sizes. For DA/DB
series the pages size is 512 bytes
'  A value of 1 will give a size of 512 bytes, a value of 2 gives
1024 bytes, etc.
'  In this example we use less than 1536 bytes so we set the fuse
to 3.
' Notice that the fuse page size can differ from the flash page
size ! In this case the sizes are equal.

' After the optional boot space there is the APPLICATION CODE
' And after the APPLICATION CODE there is the APPLICATION DATA
' When you dont want a boot loader you set the bootend fuse to 0
which is the default.
'  your app will use the boot and application code
' When you want a boot loader, you determine the size of the boot
loader and then
'  set the fuse to the proper size
' With the MCS boot loader, the code simply checks if the #123
data is received.
' If so, it starts the loading. If not it continues.
' It is similar to the normal AVR boot loading. The normal AVR
boot starts after the normal space.
' # There is one important difference. With normal AVR all the
code start at &H0000.
'   For the loader we then use the $LOADER directive to place the
code at the proper address
' For XTINY/AVRX the boot loader starts at &H0000 thus is
considered a normal application without specific switches
' Your normal code must now be located AFTER the bootloader. This
means you need to instruct the compiler to place the code
' at a different address. We use $ROMSTART for this purpose.
' Remember that AVR has word address. Which means that each
address uses 2 bytes of memory.
'
'-------------------------------------------------------------
-

$regfile ="AVRX128da28.dat"
$crystal = 24000000
$hwstack = 40
$swstack = 40
$framesize = 40
```

```
'declare subs and functions
Declare Sub Erasepage()
Declare Sub Protected_write_spm(byreg R16 As Byte)

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz
'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1
'define the baud rate and port/usart
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

Const Pagesize_bts = 512                                        '
page size in bytes is 512 bytes
Const Fword = Pagesize_bts / 2                                  '
page size in words is 256
Const Numpages = 256                                            '
number of page is 256
Const Key_ctrla_spm = &H9D                                      ' ccp
key write protect for SPM NVM_CTRLA
Const Key_ctrlb_io = &HD8                                       ' ccp
key write protect for IO  NVM_CTRLB
' NVM Commands
Const Nocmd = 0                                                 'no
command
Const Noop = 1                                                  'no
operation
Const Flwr = 2
'flash write enable
Const Flper = 8
'flash erase enable

'flash constants
Const Maxwordbit = 8
Const Maxword =(2 ^ Maxwordbit) * 2                             '512
Const Maxwordshift = Maxwordbit + 1                             '9

'It is VERY IMPORTANT that the baud rate match the one of the
boot loader
'do not try to use buffered com since we do not use interrupts
'you could however use interrupts but it occupies more space

'Possible return codes of the PC bootloader.exe
' -6005    Cancel requested
' -6006    Fatal time out
' -6007    Unrecoverable event during protocol
' -6008    Too many errors during protocol
' -6009    Block sequence error in Xmodem
' -6016    Session aborted
```

```
'since this is a boot loader we do not want a vector table
'we reduce the vector table to 0
$reduceivr

'Dim the used variables
Dim Z As Dword                                              '
this is the Z pointer word
Dim Vl As Byte , Vh As Byte                                 '
these bytes are used for the data values
Dim Wrd As Word , Page As Word                              '
these vars contain the page and word address

Dim Bstatus As Byte , Bretries As Byte , Bblock As Byte ,
Bblocklocal As Byte
Dim Bcsum1 As Byte , Bcsum2 As Byte , Buf(128) As Byte , Csum As
Byte
Dim J As Byte , Wptr As Word


'We start with receiving a file. The PC must send this binary
file

'some constants used in serial com
Const Cnak = &H15
Const Cack = &H06

$timeout = 400000                                           ' we
use a timeout
'When you get LOADER errors during the upload, increase the
timeout value
'for example at 16 Mhz, use 200000

Bretries = 5                                                ' we
try 5 times
Do
   Bstatus = Waitkey()                                      '
wait for the loader to send a byte
   Print Chr(bstatus);                                      '
echo back

   If Bstatus = 123 Then                                    ' did
we received value 123 ?
      Goto Loader                                           '
jump into boot loader
   End If
   Decr Bretries
Loop Until Bretries = 0
```

```
'if we arrive here, there was not boot character received. we
simply continue
Goto Application_start_noload                               '
goto the normal code


'this is the loader routine. It is a Xmodem-checksum reception
routine
Loader:
   Do
      Bstatus = Waitkey()                                   '
flush the data
   Loop Until Bstatus = 0

'just like the normal AVR loader we need to erase  a page first
'but since we write beyond the loader we need to set the page to
the proper value which is essential the same as the FUSE BOOT
SIZE value !
   page=3 'remember each page uses 512 bytes, this code is less
than 1536 so the normal app start on page 3
   Erasepage
'erase it

   Bretries = 10                                            '
number of retries
   Do
      Bblocklocal = 1
      Csum = 0                                              '
checksum is 0 when we start
      Print Chr(cnak);                                      '
first time send a nack
      Do

         Bstatus = Waitkey()                                '
wait for status byte

         Select Case Bstatus
            Case 1:                                         '
start of heading, PC is ready to send
               Csum = 1                                     '
checksum is 1
               Bblock = Waitkey() : Csum = Csum + Bblock    ' get
block
               Bcsum1 = Waitkey() : Csum = Csum + Bcsum1    ' get
checksum first byte
               For J = 1 To 128                             ' get
128 bytes
                  Buf(j) = Waitkey() : Csum = Csum + Buf(j)
               Next
```

```
               Bcsum2 = Waitkey()                          ' get
second checksum byte
               If Bblocklocal = Bblock Then                ' are
the blocks the same?
                  If Bcsum2 = Csum Then                    ' is
the checksum the same?
                     Gosub Writepage                       ' yes
go write the page
                     Print Chr(cack);                      '
acknowledge
                     Incr Bblocklocal                      '
increase local block count
                  Else                                     ' no
match so send nak
                     Print Chr(cnak);
                  End If
               Else
                  Print Chr(cnak);                         '
blocks do not match
               End If
            Case 4:
' end of transmission , file is transmitted
            '  Waitms 100                                   '
OPTIONAL REMARK THIS IF THE DTR SIGNAL ARRIVES TO EARLY
               Print Chr(cack);                            '
send ack and ready
               Waitms 20
               Goto Application_start                      '
start new program
            Case &H18:                                     ' PC
aborts transmission
               Goto Application_start                      '
ready
            Case 123 : Exit Do                             ' was
probably still in the buffer
            Case Else
               Exit Do                                     ' no
valid data
         End Select
      Loop
      If Bretries > 0 Then                                 '
attempte left?
         Waitms 1000
         Decr Bretries                                     '
decrease attempts
      Else
         Exit Do                                           '
reset chip
      End If
```

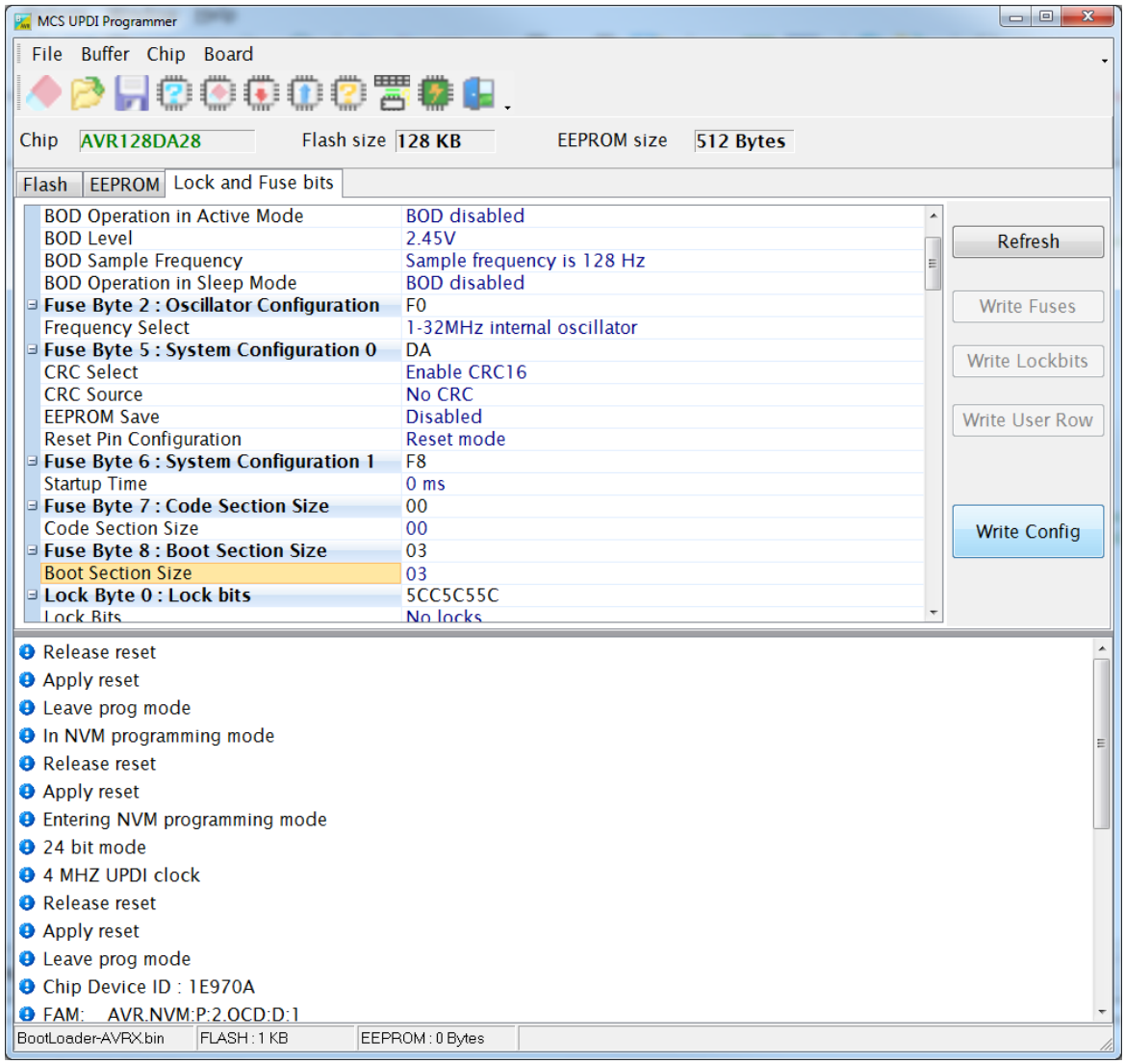
```
   Loop

Application_start:
   Protected_write_SPM nocmd                                    '
clear current command
   Cpu_ccp = key_CTRLB_io
   Rstctrl_swrr = 1                                             '
perform a soft reset

'no code between the label above and below !

Application_start_noload:
   r23=&B111                                                    ' set
value
   Cpu_ccp = key_CTRLB_io                                       '
configuration change protect
   Nvmctrl_ctrlb = R23                                          '
enable boot section read protect, appdatawrite protect, appcode
protect
   'the addres to go to is a word address, since our loader is
less than 1536 bytes, this is 600 in hex and divided by 2 for
words gives 300
   Goto &H300                                                   '
normal app code

'this sub routine will write the pages
Writepage:
   For J = 1 To 128 step 2                                      ' get
128 bytes
      Z = Page                                                  ' make
equal to page
      Shift Z , Left , Maxwordshift                             '
shift to proper place
      Z = Z + Wrd                                               ' add
word
      r0= Buf(j)                                                '
r0:r1 point to the DATA for SPM
      R1 = Buf(j + 1)
      ! lds r30,{Z}                                              ; Z
points to address
      ! lds r31,{Z+1}

      #if _romsize > 65536
         !  lds r24,{Z+2}
         !  sts rampz,r24                                        '
we need to set rampz also for 128KB chips
      #endif
      !spm                                                      ;
this actual executes the instruction
```

```
      wrd=wrd+2                                                ' we
write word data so increase by 2
   Next

   if wrd=maxword then                                        ' page
is full
      wrd=0                                                   '
reset word counter for next page
      page=page+1                                             '
increase page
      Erasepage                                                '
erase so we can write
   end if
return

'erase a page based on PAGE value
'RAMPZ ZH  ZL (bit0 for low high byte selection)
'          7 bits for word address : 128 word address
'          1 bit from ZH -> 256 word address
'  leves 7 bit for PAGE address
'zzzz,ppppppppw,wwwwwwwx
Sub Erasepage()
   Z = Page                                                    '
make equal to page
   Shift Z , Left , Maxwordshift                               '
shift to proper place

   bitwait NVMCTRL_STATUS.0,reset                              '
make sure flash is not busy
   Protected_write_SPM nocmd                                   '
clear current command
   Protected_write_SPM flper                                   '
erase page enable
   'dummy write needed, how dum

   ! lds r30,{Z}
   ! lds r31,{Z+1}
   #if _romsize > 65536
      !  lds r24,{Z+2}
      !  sts rampz,r24                                         '
we need to set rampz also for 128KB chips
   #endif
   ! clr r0                                                   ;
data must be 0 for the dummy write
   ! clr r1

   !spm                                                       ;
execute dummy write
```

```
   bitwait NVMCTRL_STATUS.0,reset                                '
make sure flash is not busy
   Protected_write_SPM nocmd                                     '
clear current command
   Protected_write_SPM flwr                                      '
enable write to flash
end sub

sub Protected_write_SPM(byreg r16 as byte)
   CPU_CCP= key_CTRLA_spm
'change protect key
   NVMCTRL_CTRLA=r16
'write to SPM register
end sub

'this loader takes less than 1536 bytes so the BOOT FUSE is set
to 3.
'do NOT FORGET that your normal app must use $ROMSTART=&H300 in
that case  : halve of the bytes size of the loader

'one other thing : the reset pin works in UPDI mode by default.
so your chip will not reset when you do not change the fuse
'but there are other ways of reset such as soft reset, bod and
wd.
```

You compile the code and from the report you can find that the code size is 1214 bytes. Because the boot fuse block is 512, it means the value must be : 1214/512=2.31 We round up so end with 3.

We now run the MCS UPDI programmer and select the fuses TAB



The boot section size must be set to 3 as shown above.
You can also use the WRITE CONFIG button to write a CONFIG FUSE line in your code. Just make sure the editor is set to a new line. in this case we end up with :

Config Fuses = Off , Lock = Off , Fuse0 = &H00 , FUSE1 = &H20 , FUSE2 = &HF0 , FUSE5 = &HDA , FUSE6 = &HF8 , FUSE7 = &H00 , **FUSE8 = &H03** , UROW0 = &H00 , UROW31 = &H31

All user row values with a non default value are included too.

When you compile with CONFIG FUSES=OFF, nothing will happen. Only when you change the FUSES=OFF to FUSES=ON, the fuses will be programmed when you auto program the chip.
And when you change LOCK=OFF to LOCK=ON, the processor will be locked too. The lock will make sure the data can not be read any longer. only the UNLOCK chip option from the programmer can unlock the chip. An operation that will also erase the chip. So typically writing LOCK bytes is something you do when all is tested.

Once we have programmed our boot loader and set the boot fuse to the correct size, we can use the MCS Boot loader to load new code.
Do not forget to chose the MCS Boot loader.

The only thing you need to remember : include the $ROMSTART=&H300 line in your NORMAL code.
Most examples you find do not have this directive.

Instead of the MCS Bootloader you can also make a boot loader that use Blue Tooth, Wifi, or an SD card.
The mechanism is always : at boot check some condition, when met, perform the update.

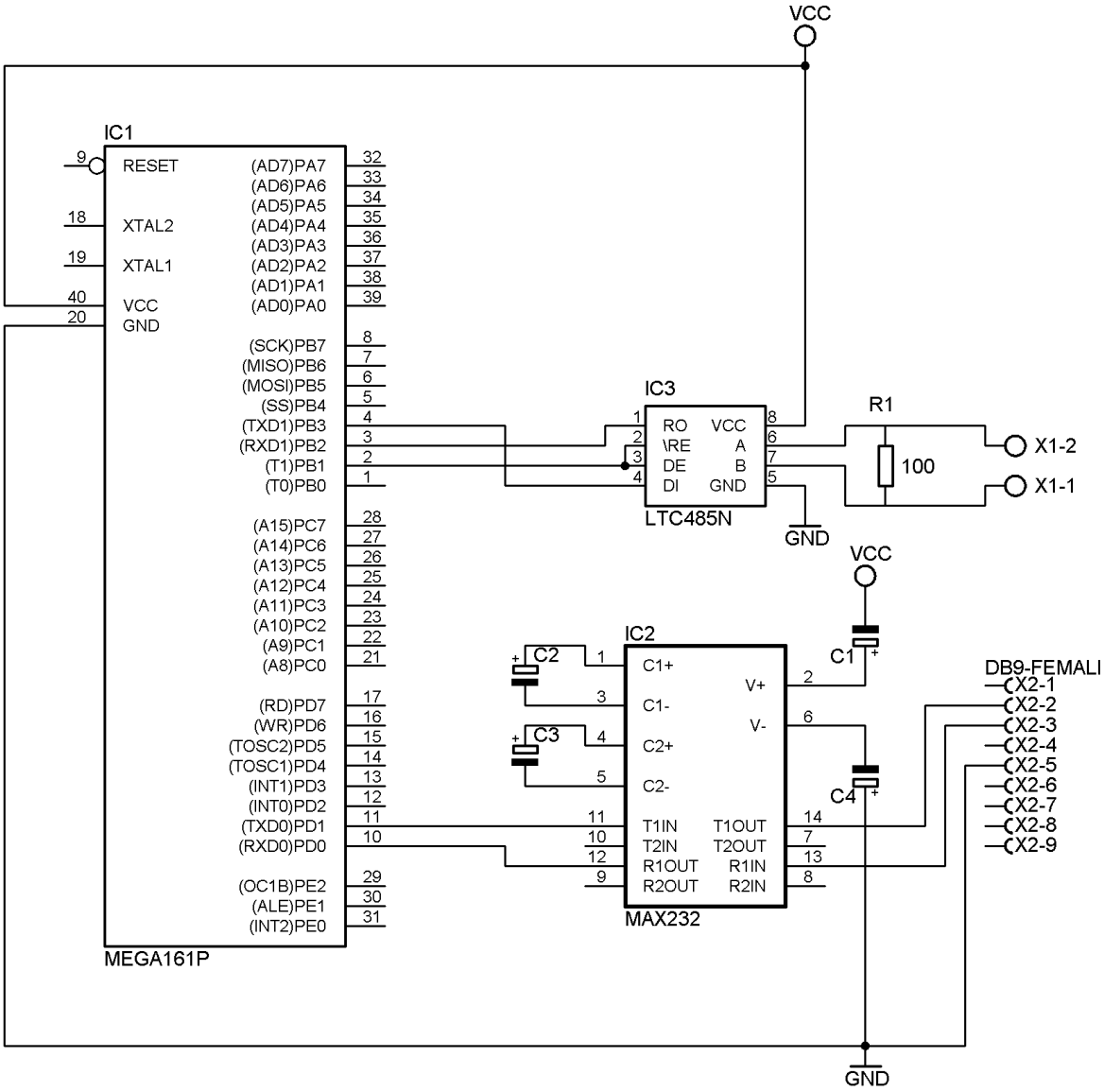
The actual boot loader will remain in the processor.
When you have suitable hardware like an external EEPROM with the size of the FLASH memory, you could update the firmware in your normal code. The boot code then would copy from EEPROM when required.

A boot loader could use decryption so your firmware can not be easily disassembled.

4.17 USING RS485

RS485

RS485 is used for serial communication and well suited for transmission over large distances.
Similar to RS232 we need a level shifter.



The sample above uses a MEGA161 or MEGA162 which has 2 UARTS. This way you can have both a RS232 and RS485 interface.
The RS232 is used for debugging.
In order to test you need 2 or more similar circuits. One circuit would be the master. The other(s) would be a slave.
The same hardware is used to test the MODBUS protocol. The bus need to be terminated at both ends with a resistor. 100 ohm is a typical used value.
The GND of both circuits may not be connected ! Only connect point A and B from both circuits. For industrial usage it is best to use an optical isolated level shifter.

Simple MASTER sample

```
$regfile = "m162def.dat"                    ' specify the used micro
$crystal = 8000000
$baud = 19200                               ' use baud rate
$hwstack = 42                               ' default use 32 for the hardware stack
$swstack = 40                               ' default use 10 for the SW stack
$framesize = 40                             ' default use 40 for the frame space

$lib "modbus.lbx"
Config Print1 = Portb.1 , Mode = Set        ' use portb.1 for the direction
```

```
Rs485dir Alias Portb.1
Config Rs485dir = Output
Rs485dir = 0   ' go to receive mode
Portc.0 = 1    ' a switch is connected to pinc.0 so activate pull up resistor
'           TX    RX
' COM0   PD.1   PD.0   monitor
' COM1   PB.3   PB.2   rs485
'          PB.1          data direction rs485


Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits =
8 , Clockpol = 0
Config Com2 = 9600 , Synchrone = 0 , Parity = Even , Stopbits = 1 , Databits = 8 ,
Clockpol = 0  ' MUST MATCH THE SLAVE

'use OPEN/CLOSE for using the second UART
Open "COM2:" For Binary As #1

'dimension some variables
Dim B As Byte
Dim W As Word
Dim L As Long

W = &H4567   ' set some values
L = &H12345678


Print "RS-485 MODBUS master"
Do
  If Pinc.0 = 0 Then                              ' test button
     Waitms 500                                  ' delay since we want to send just 1
frame
     Print "send request to slave/server"          ' to debug terminal
    ' Print #1 , Makemodbus(2 , 3 , 8 , 2);              'slave 2, function 3, start
address 8, 2 bytes
    ' Print #1 , Makemodbus(2 , 6 , 8 , W);              'slave 2, function 6, address
8  , value of w
      Print #1 , Makemodbus(b , 16 , 8 , L);             'send a long
  End If
  If Ischarwaiting(#1) <> 0 Then  'did we got something back?
    B = Waitkey(#1)  ' yes so get it
    Print Hex(b) ; ",";  ' print it
  End If
Loop
```

A slave would simply listen to data, and once enough data received, send it back.

4.18 Using the I2C protocol

I²C bus

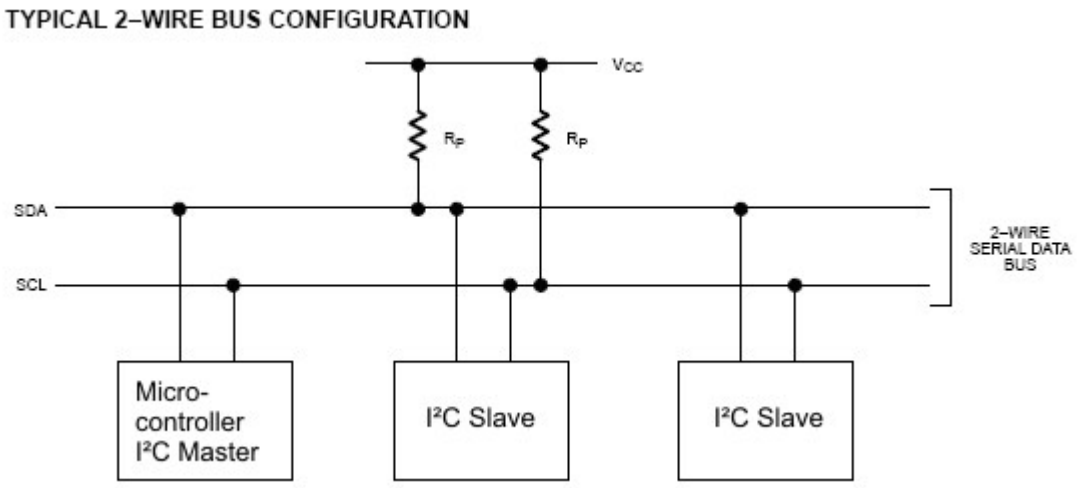
I²C bus is an abbreviation for Inter Integrated Circuit bus or "I-Squared-C".
Some manufacturer call it TWI (Two-Wire-Interface) which is technically the same as I2C.

There is also SMBus. The I²C bus and the SMBus™ are essentially compatible with each other. Normally devices, both masters and slaves, are freely interchangeable between both buses. Both buses feature addressable slaves (although specific address allocations can vary between the two).
The buses operate at the same speed, up to 100kHz, but the I²C bus has both 400kHz and 2MHz
versions. Complete compatibility between I2C and SMBus is ensured only below 100kHz.

I²C is a serial and synchronous bus protocol. In standard applications hardware and timing are often the same. The way data is treated on the I²C bus is to be defined by the manufacturer of the I²C master and slave chips.

In a simple I²C system there can only be one master, but multiple slaves. The difference between master and slave is that the master generates the clock pulse. The master also defines when communication should occur. For bus timing it is important that the slowest slave should still be able to follow the master's clock. In other words the bus should be as fast as the slowest slave.

A typical hardware configuration is shown in the figure below:



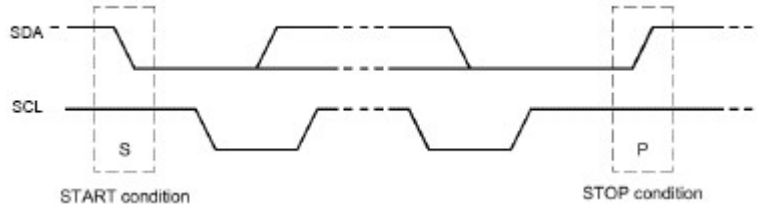
Note that more slave chips can be connected to the SDA and SCL lines, normally Rp has a value of 1kOHM.
The clock generated by the master is called Serial Clock (SCL) and the data is called Serial Data (SDA).

Always check if the pull-up resistors are connected !

In most applications the micro controller is the I²C Master. Slave chips can be Real Time Clocks and Temperature sensors. For example the DS1307 and the DS1624 from http://www.maximintegrated.com .

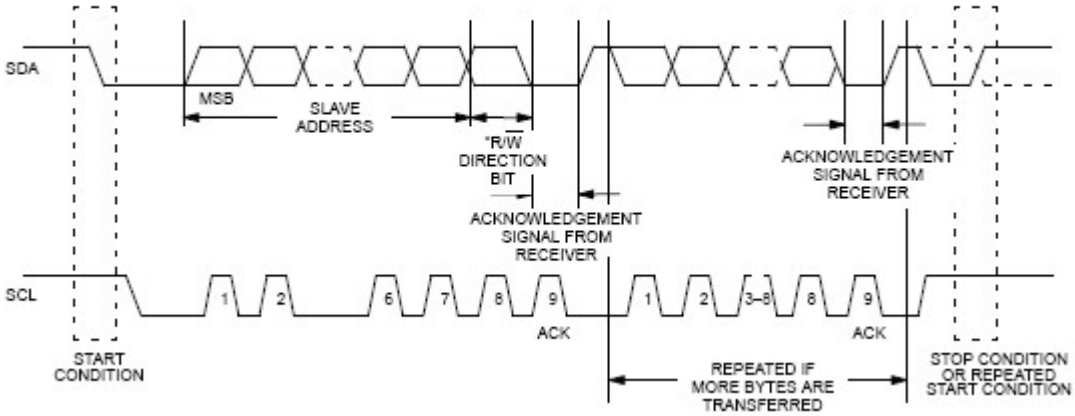
Of course you can also create your own I2C slaves by programming an ATTINY or ATMEGA . See CONFIG I2CSLAVE 978
In that case there is AVR Master to AVR Slave communication.

LOGIC BUS LEVELS AND CONDITIONS



Data can only occur after the master generates a **start condition.** A start condition is a high-to-low transition of
the SDA line while SCL remains high. After each data transfer a **stop condition** is generated. A stop condition is a low-to-high transition of the SDA line while SCL remains high.

DATA TRANSFER ON 2–WIRE SERIAL BUS



As said a data transfer can occur after a **start condition** of the master. The length of data sent over I²C is always 8 bit this includes a read/write direction bit, so you can effectively send 7 bits every time.
The most significant bit MSB is always passed first on the bus.

If the master writes to the bus the R/W bit = 0 and if the master reads the R/W bit = 1.

After the R/W bit the master should generate one clock period for an acknowledgement ACK.

Each receiving chip that is addressed is obliged to generate an acknowledge after the reception of each byte. A chip that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse.

After an acknowledge there can be a stop condition, if the master wishes to leave the bus idle. Or a repeated start condition. A repeated start is the same as a start condition.

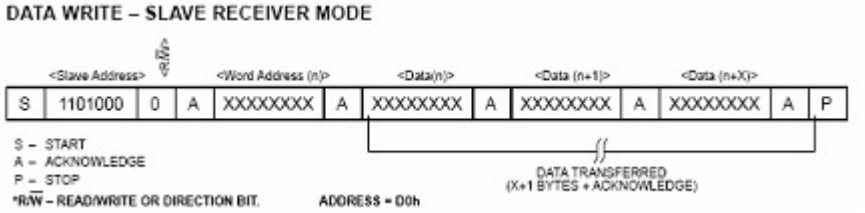
When the master reads from a slave it should acknowledge after each byte received. There are two reasons for the master not to acknowledge. The master sends a not acknowledge if data was not received correctly or if the master wishes the stop receiving.

In other words if the master wishes to stop receiving, it sends a not acknowledge after the last received byte.

The master can stop any communication on the bus **at any time** by sending a stop condition.

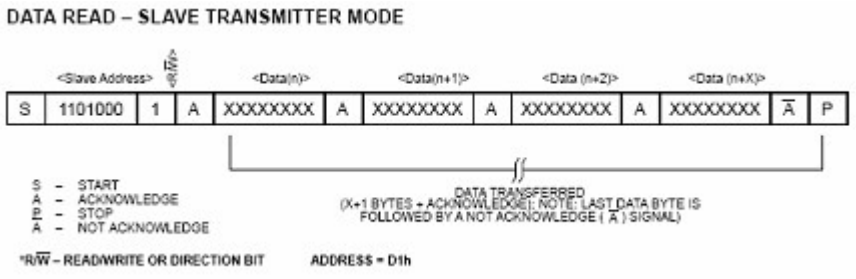
BUS ADRESSING

Let's say we have a slave chip with the address &B1101000 and that the master wishes to write to that slave, the slave would then be in receiver mode, like this:



You can see here that the master always generates the start condition, then the master sends the address of the slave and a "0" for R/W. After that the master sends a command or word address. The function of that command or word address can be found in the data sheet of the slave addressed.

After that the master can send the data desired and stop the transfer with a stop condition.



Again the start condition and the slave address, only this time the master sends "1" for the R/W bit. The slave can then begin to send after the acknowledge. If the master wishes to stop receiving it should send a not acknowledge.

OVERVIEW of Routines

Config Sda = Portx.x
Configures a port pin for use as serial data SDA.

Config Scl = Portx.x
Configures a port pin for use as serial clock SCL.

I2cinit
Initializes the SCL and SDA pins.

I2cstart
Sends the start condition.

I2cstop
Sends the stop condition.

I2cwbyte
Writes one byte to an I²Cslave.

I2crbyte
Reads one byte from an I²Cslave.

I2csend

Writes a number of bytes to an I²Cslave.

I2creceive
Reads a number of bytes from an I²Cslave.

I2C write and read:

A typical **I2C write** to send one byte of data looks like this:

```
I2cstart
I2cwbyte      I2c_address_of_slave
I2cwbyte    Byte_to_send
I2cstop
```

(**I2cstart** generates the start condition on the I2C bus were all devices are listen to. After this we send the Slave address of the device we want to send a byte to. The I2C slave with this address will send out a Ack where all other do nothing. Now you can start to send a byte (or more bytes) to this Slave address. After this an **I2cstop** release the bus.)

A typical **I2C read** to read one byte of data looks like this:

```
I2cstart
I2cwbyte      I2c_address_of_slave
I2crbyte      Databyte_to_read , Nack
I2cstop
```

(**Nack** indicates that the master do not want to read more bytes)

A typical **I2C read** to read one byte of data looks like this:

```
I2cstart
I2cwbyte      I2c_address_of_slave
I2crbyte      Databyte_to_read , Ack
I2crbyte      Databyte_to_read , Nack
I2cstop
```

(**Ack** indicates that the master want to read more bytes from the slave and with the last byte to read the master indicate this with **Nack**)

I2C Software vs. Hardware Routines

By default BASCOM will use software routines when you use I2C statements. This because when the first AVR chips were introduced, there was no TWI yet. Atmel named it TWI because Philips is the inventor of I2C. But TWI is the same as I2C.

So BASCOM allows you to use I2C on every AVR chip. Most newer AVR chips have build in hardware support for I2C. With the I2C_TWI 1753 lib you can use the hardware TWI which has advantages since it require less code.

To force BASCOM to use the hardware TWI, you need to insert the following statement into your code:

$LIB "I2C_TWI.LBX"

You also need to choose the correct SCL and SDA pins with the CONFIG SCL and CONFIG SDA statements.
The TWI will save code but the disadvantage is that you can only use the fixed SCL and SDA pins.


You only should include this library for a normal(older) AVR processor. Like

Mega88. Do not include for Xmega, Xtiny, MegaX or AVRX.

XMEGA

When using XMEGA, there is a difference : here you are supposed to use the hardware TWI. So that is a default. The reason is that Xmega has up to 4 different TWI-buses. There is no need to include a library. The only requirement is that you dimension a byte : Dim Twi_start As Byte
To force to the software solution, use $FORCESOFTI2C 635

XTINY/MEGAX/AVRX

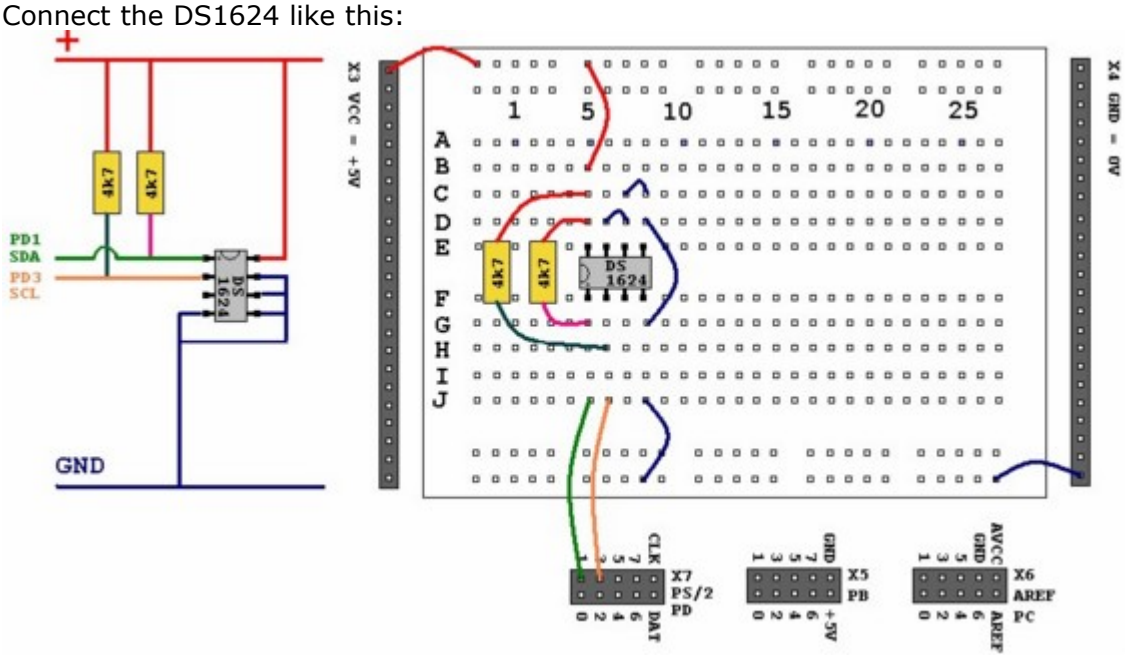
These are all new AVR processors. The have a similar architecture as Xmega. For this reason, you are supposed to use the TWI hardware.
There is no need to include a library. The only requirement is that you dimension a byte : Dim Twi_start As Byte

See also:

Using USI (Universal Serial Interface) 321, Config TWI 1116, CONFIG TWISLAVE 1123, I2C_TWI 1753, $FORCESOFTI2C 635
I2CSEND 1305 , I2CSTART 1306 , I2CSTOP 1306 , I2CRBYTE 1306 , I2CWBYTE 1306 , I2C_TWI Library for using TWI 1753

EXAMPLE with Software Routines

This example shows you how to setup and read the temperature from a DS1624 temperature sensor.
Connect the DS1624 like this:



Then program this sample into your micro controller and connect your micro controller to the serial port of your PC.

```
$regfile = "m88def.dat"                    'Define the chip you use
$crystal = 8000000                                  'Define speed
$hwstack = 40
$swstack = 30
```

```
$framesize = 40

$baud = 19200                                 'Define UART BAUD rate


' Declare RAM for temperature storage
Dim I2ctemp As Byte                           'Storage for the temperature

' We use here the software emulated I2C routines
' Configure pins we want to use for the I²C bus
Config Scl = Portd.1                          'Is serial clock SCL
Config Sda = Portd.3                          'Is serial data SDA
I2cinit


' Declare constants - I2C chip addresses
Const Ds1624wr = &B10010000                   'DS1624 Sensor write
Const Ds1624rd = &B10010001                   'DS1624 Sensor read

' This section initializes the DS1624
    I2cstart                                  'Sends start condition
    I2cwbyte Ds1624wr                         'Sends the address

'byte with r/w 0

'Access the CONFIG register (&HAC address byte)
    I2cwbyte &HAC
'Set continuous conversion (&H00 command byte)
    I2cwbyte &H00
    I2cstop                                   'Sends stop condition
    Waitms 25             'We have to wait some time after a stop

    I2cstart
    I2cwbyte Ds1624wr
'Start conversion (&HEE command byte)
    I2cwbyte &HEE
    I2cstop
    Waitms 25
'End of initialization

Print                                         'Print empty line


Do

    'Get the current temperature
    I2cstart
    I2cwbyte Ds1624wr
    I2cwbyte &HAA         'Read temperature (&HAA command byte)
    I2cstart
    I2cwbyte Ds1624rd     'The chip will give register contents
'Temperature is stored as 12,5 but the ,5 first
    I2crbyte I2ctemp , Ack
'So you'll have to read twice... first the ,5
    I2crbyte I2ctemp , Nack
'And then the 12... we don't store the ,5
    I2cstop
                                                                  'That's why we read twice.

'We give NACK if the last byte is read

    'Finally we print
Print "Temperature:      " ; Str(i2ctemp) ; "    degrees" ; Chr(13) ;

    Waitms 25

Loop
End
```

You should be able to read the temperature in your terminal emulator.
Note that the used command bytes in this example can be found in DS1624 temperature sensor data sheet.

Example which use I2C Master hardware in AVR

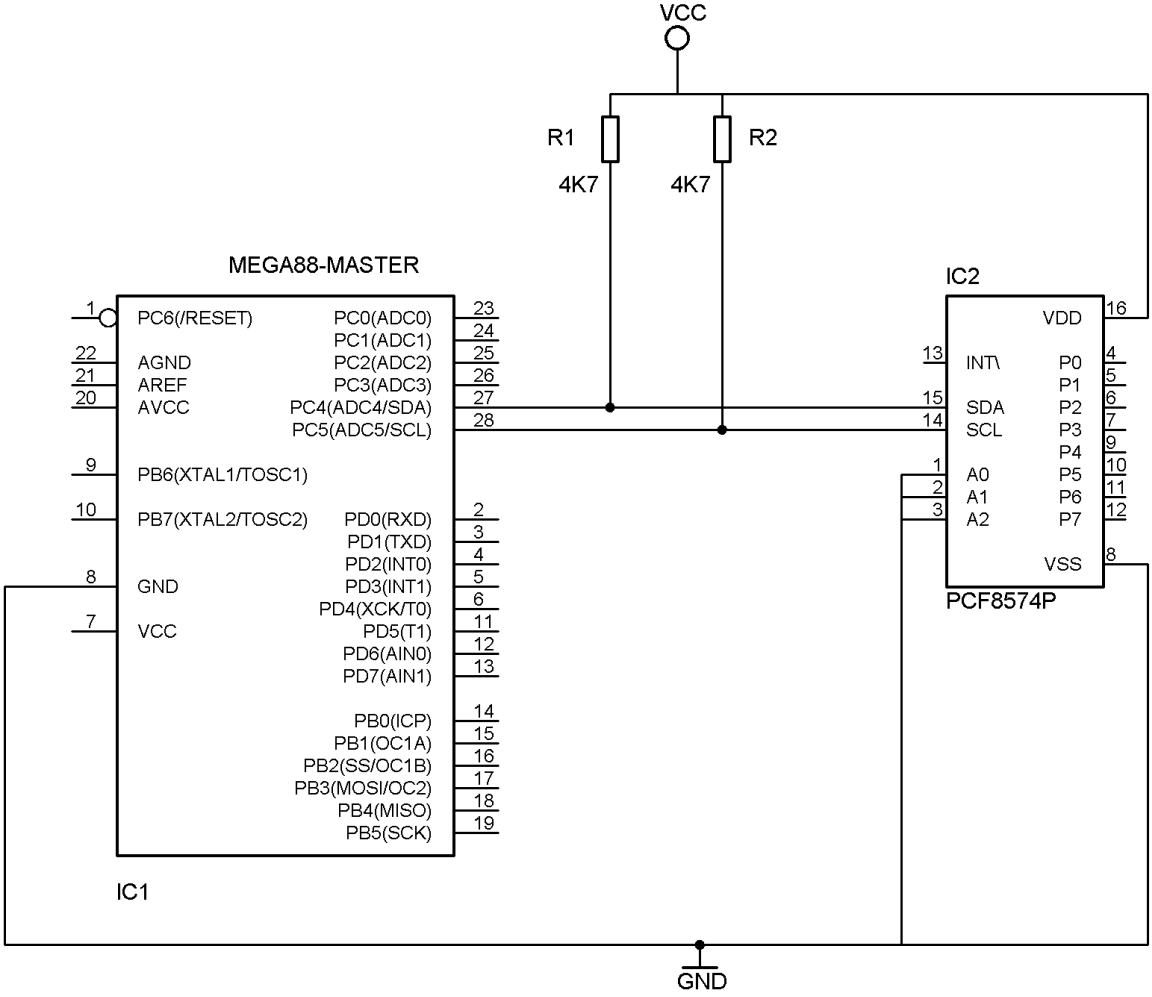
See here: CONFIG TWI 1116

I2C Practice (Tips&Tricks)

The design below shows how to implement an I2C-bus. The circuit is using a Mega88 as a master.
The TWI bus is used. While you can use any pin for software mode I2C, when a micro has TWI hardware build in, it is advised to use the TWI hardware.

R1 and R2 are 4K7 pull up resistors.

There are many I2C slave chips available. The example shows the PCF8574. With the additional TWI slave library you can make your own slave chips.



How to calculate Pull Up Resistor

The maximum of bus capacitance is 400pF (which is independent of bus speed 100KHz or 400KHz).
Here is a good article which describe how to calculate the Pull Up Resistor:
http://www.edn.com/design/analog/4371297/Design-calculations-for-robust-I2C-communications

Using AVR interal pull-up resistor (with Hardware Routines)

It is recommended to use external pull-up resistors !
For testing you could use also the AVR interal pull-up resistors

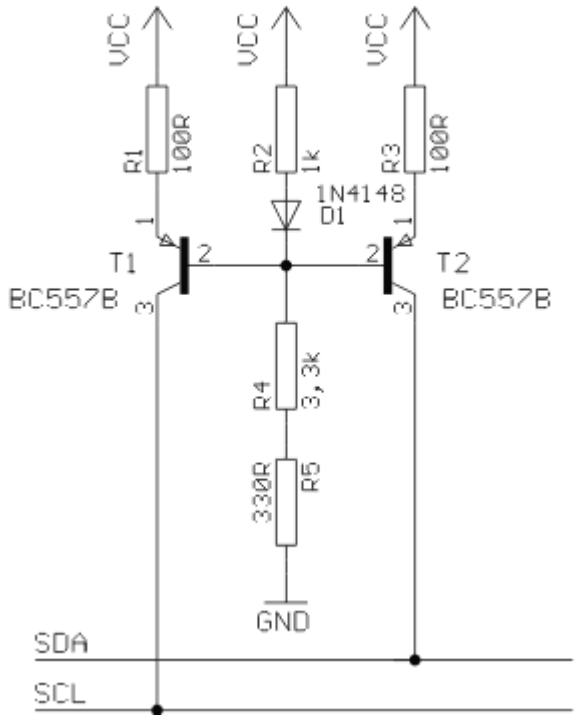
See example where Portc.4 and Portc.5 is SDA and SCL (the pull-up needs to be set

after i2cinit):

```
i2cinit
Portc.4 = 1
Portc.5 = 1
```

Active Termination of I2C

The following information was submitted by Detlef Queck:
Many people have problems over and over with I2C(TWI) Termination. Use 4,7k or 10 k pull up? How long can the SCL, SDA line be when used with pull ups etc, etc.
You can simplify this confusing problem. Here is a Schematic for an active Termination of I2C and TWI. We have used this Schematic for over 10 years, and have had no problems with it. The I2C (TWI) lines can be up to 80cm (400KHz) without any problem when the Terminator is at the end of the lines.



How to handle longer cable length between I2C Master and Slaves or Multi-drop Configurations

The I2C-bus capacitance limit of 400 pF restricts practical communication distances.
You can extend the use of the I2C in systems
with more devices and / or longer bus lengths with P82B715 or P82B96.

P82B96

- Isolates capacitance allowing 400 pF on Sx/Sy side and **4000 pF** on Tx/Ty side
- 400 kHz operation over at least 20 meters of wire (see AN10148)
- Create Multi-drop configurations
- Supply voltage range of 2 V to 15 V with I2C-bus logic levels on Sx/Sy side independent of supply voltage
- Splits I2C-bus signal into pairs of forward/reverse Tx/Rx, Ty/Ry signals for interface with opto-electrical isolators and similar devices that need unidirectional input and output signal paths.

P82B715

- Increase the total connected capacitance of an I2C-bus system to around **3000 pF** and drive signals over long cables to approximately 50m
- Multi-drop distribution of I2C-bus signals using low cost twisted-pair cables

I2C Multiplexing, Switch and Voltage Level translation between different I2C busses

Some specialized devices only have one I2C or SMBus address and sometimes several identical devices are needed in the same system. The multiplexers and switches split the I2C bus into several sub-branches and allow the I2C master to select and address one of multiple identical devices, in order to resolve address conflict issues. An example is PCA9544A or PCA9546A (which also llows voltage level translation between 1.8 V, 2.5 V, 3.3 V and 5 V buses).

Your I2C (TWI) connection is not working (Tips&Tricks):

Checklist:
- Is the configured I2C clock frequency matching the frequency of the connected chip
- Check if you have pull-up resistors on SDA and SCL (and if the pull-up resistors are working)
- Do you have the right SDA and SCL pins conected ?
- connect also GND to have the same potential
- You can use the **Err** variable to check which I2C function is not working. When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.
- How about the voltage levels on both chips (do not connect 3.3V systems to 5V systems without voltage adapter)

- Is the system you are connecting the I2C to using a 7 Bit address or 8 Bit address (8-bit addresses include the read/write bit) ?
Then you can try with shift left:

```
' you can simply do this;    &HC4 is an example address
const    someI2caddress= &H4C * 2
'   this   would   shift   the   address   to   the   left.
```

- It is important that you specify the proper crystal frequency. Otherwise it will result in a wrong TWI clock frequency

- With following lib you do not use the software emulated TWI (I2C). You use the hardware I2C (for the AVR's that have an hardware I2C)

```
  $lib "i2c_twi.lbx"                                    ' we do not use software emulated I2C but the TWI
```

- By default BASCOM will use software routines for I2C.
- Do you have the right I2C read address ?

Here an example I2C write address which Bascom expects:

```
&B01000000 = &H40
```

Read address would be for this example:

```
&b01000001 = &h41
```

- In case of using TWI (I2C) Slave: Are you using the right library for your used chip ?

With the I2C TWI Slave add-on library you get both libraries:

- **i2cslave.lib** and **i2cslave.lbx** : This library is used for AVR's which have no

hardware TWI/I2C interface like for example ATTINY2313 or ATTINY13. In this case TIMER0 and INT0 is used for SDA and SCL (Timer0 Pin = SCL, INT0 Pin = SDA). Only AVR' with TIMER0 and INT0 on the same port can use this library like for example ATTINY2313 or ATTINY13. The i2cslave.lib file contains the ASM source. The i2cslave.lbx file contains the compiled ASM source. See **CONFIG I2CSLAVE** below.

- **i2c_TWI-slave.LBX** : This library can be used when an AVR have an TWI/I2C hardware interface like for example ATMEGA8, ATMEGA644P or ATMEGA128. In this case the hardware SDA and SCL pin's of the AVR will be used (with ATMEGA8: SCL is PORTC.5 and SDA is PORTC.4). This library will be used when USERACK = OFF. When USERACK =ON then **i2c_TWI-slave-acknack.LBX** will be used. See also Config TWISLAVE 1123

Operation at 400 kHz

Fast- mode devices can only be operated at 400 kHz clock frequency if no standard-mode devices (100KHz) are on the bus.

You can use an I2C Scanner to find I2C devices:

You basically use the **Err** variable. When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.
So 0 means we have found a I2C Slave with that address.

```
'-----------------------------------------------------------------------
'                          (c) 1995-2025 MCS
'                              i2cscan.bas
'purpose : scan all i2c addresses to find slave chips
'use this sample in combination with twi-slave.bas
'Micro:  Mega88
'-----------------------------------------------------------------------
$regfile = "M88def.dat"                                     ' the used chip
$crystal = 8000000                                          ' frequency used
$baud = 19200                                               ' baud rate
$hwstack = 40
$swstack = 30
$framesize = 40

Dim B As Byte

'we use the TWI pins of the Mega88
$lib "i2c_twi.lbx"                                          ' we do not use software
emulated I2C but the TWI


Config Scl = Portc.5                                        ' we need to provide the SCL
pin name
Config Sda = Portc.4                                        ' we need to provide the SDA
pin name
I2cinit
Config Twi = 100000                                         ' wanted clock frequency when
using $lib "i2c_twi.lbx"
'will set TWBR and TWSR
'Twbr = 12 'bit rate register
'Twsr = 0 'pre scaler bits

Print "Scan start"
For B = 0 To 254 Step 2                                     'for all odd addresses
  I2cstart                                                  'send start
  I2cwbyte B                                                'send address
  If Err = 0 Then                                           'we got an ack
     Print "Slave at : " ; B ; " hex : " ; Hex(b) ; " bin : " ; Bin(b)
  End If
  I2cstop                                                   'free bus
Next
Print "End Scan"
End
```

I2C Slave Library

See I2C TWI Slave 1831

I2C Slave LIB - how to Send/Receive more than 1 Byte for chips that do not have hardware I2C ?

Using following config:

```
Config  I2cslave = &H34 , Int = Int0 , Timer = Timer0
```

When you want to receive/send multiple bytes, you need to keep track of them. You can do this with a byte counter. this counter you would need to reset when the slave is addressed.
To do this the lib need to be altered:

- open i2cslave.lib with notepad
- look for label : **I2c_adr_ack:**

Then add this line :
rcall i2c_master_addressed

-then save and add this label to your code:

```
I2c_master_addressed:
    Br = 0                                                  'clear the byte counter
   Bw = 0
Return
```

in your code where the bytes are passed you can increase them.
The BR you increase when a byte is read, the BW you increase when a byte is passed.
for example:

```
I2c_master_has_data:
   Incr Bw
    Myarray( bw) = _a1
Return
```

Using ATXMEGA I2C with Software Routines (then you can choose the SDA and SCL Pins)

ATXMEGA have usually enough I2C interfaces. But nevertheless there is a possibility to use the I2C software routines and you can use any Pin you want as SDA and SCL.

Following the ATXMEGA Master and below the ATMEGA328P I2C Slave which was tested with the ATXMEGA Master in I2C Software Mode:

Master

```
' Using ATXMEGA with software I2C routines to use also pins which are no hardware SDA/SCL pins
'   Needed Library:   $lib  "i2c.lbx"
'   The $forcesofti2c directive force the ATXMEGA to use software I2c/TWI Library

' The hardware for this example is XMEGA-A3BU XPlained board from Atmel
'  Don't forget the pull-ups on SDA/SCL pin !
'  Bascom Version 2.0.7.6 or higher needed

$regfile = "XM256A3BUDEF.DAT"
$crystal = 32000000                                         '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80

$forcesofti2c                                               ' with this the software I2C/TWI commands are used when inlcuding i2c.lbx
```

```
$lib "i2c.lbx"                                          ' override the normal xmega
i2c lib

Config Osc = Enabled , 32mhzosc = Enabled
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Portr.0 = Output
Led0 Alias Portr.0                                      'LED 0 (XMEGA-A3BU XPlained
board from Atmel )

Config Portr.1 = Output
Led1 Alias Portr.1                                      'LED 1 (XMEGA-A3BU XPlained
board from Atmel )

Dim B As Byte

'We use here Virtual port 0
Config Vport0 = B                                       ' 'map portB to virtual port0

Config Scl = Port0 . 1                                  ' Pin to use as SCL (The
hardware pin is Pinb.1)
Config Sda = Port0 . 0                                  ' Pin to use as SDA (The
hardware pin is Pinb.0)
I2cinit                                                 ' Bring the Pin's in the
proper state

Do


 Waitms 500
 Set Led1
 Reset Led0
 Waitms 500
 Reset Led1
 Set Led0

 Incr B

 I2cstart
 I2cwbyte &H24                                          ' address of I2C Slave
 I2cwbyte B                                             ' databyte to send to slave
 I2cstop

Loop

End                                                     'end program
```

Slave (for ATXMEGA using Soft I2C Master)

```
' I2C Slave Example for using with ATXMEGA
' ATMEGA328P running @ 3.3 Volt !

' Terminal output of this example when used with XMEGA_ise_soft_i2c.bas:

' (

ATXMEGA using Software TWI/I2C <------> ATMEGA 328P Bascom-AVR @ 3.3V...
>>> 180
>>> 181
>>> 182
>>> 183
>>> 184
>>> 185
>>> 186
>>> 187
>>> 188
>>> 189
>>> 190
>>> 191


' )

$regfile = "m328pdef.dat"
$crystal = 12e6                                         '16MHz
```

```
$hwstack = 80
$swstack = 80
$framesize = 160


'CONFIG TWI SLAVE
Config   Twislave = &H24 ,   Btr = 1 ,     Bitrate = 100000 ,   Gencall = 1
'  With  the  CONFIG  BTR,  you  specify  how  many  bytes  the  master  will  read.


Dim  Receive As Byte
Dim S As Byte

Enable Interrupts


Config Com1 = 19200 ,   Synchrone = 0 ,   Parity = None ,   Stopbits = 1 ,   Databits = 8 ,
Clockpol = 0

Wait 3

Print
Print "ATXMEGA  using  Software  TWI/I2C  <------>  ATMEGA  328P  Bascom-AVR  @  3.3V..."

Do
   nop
Loop

End                                                                    'end   program

'--------------------------------I2C-----------------------------------------

'Master  sent  stop  or  repeated  start
Twi_stop_rstart_received:

Return

'We  were  addressed  and  master  will  send  data
Twi_addressed_goread:

Return


'We  were  addressed  and  master  will  read  data
Twi_addressed_gowrite:

Return

'this  label  is  called  when  the  master  sends  data  and  the  slave  has  received  the  byte
'the  variable  TWI  holds  the  received  value
'Twi_btw  is  the  BYTE  NUMBER
Twi_gotdata:
    Receive = Twi                                                      'lesen
  Print ">>>  " ;  Twi                                                 'Print  what  we  have  received
(ONLY FOR TESTING)
Return

'this  label  is  called  when  the  master  receives  data  and  needs  a  byte
'the  variable  twi_btr  is  a  byte  variable  that  holds  the  index  of  the  needed  byte
'so  when  sending  multiple  bytes  from  an  array,  twi_btr  can  be  used  for  the  index
'twi_btr  is  the  BYTE  NUMBER
Twi_master_needs_byte:

 Return

'when  the  mast  has  all  bytes  received  this  label  will  be  called
Twi_master_need_nomore_byte:
Return
'-----------------------------------------------------------------------------
```

4.19 Using the 1 WIRE protocol

The 1-wire protocol was invented by Dallas Semiconductors and needs only 1 wire for two-way communication. You also need power and ground of course.

This topic is written by Göte Haluza. He tested the new 1-wire search routines and is building a weather station.

Dallas Semiconductor (DS) 1-wire. This is a brief description of DS 1-wire bus when

used in combination with BASCOM. For more detailed explanations about the 1-wire bus, please go to http://www.maxim-ic.com. Using BASCOM makes the world a lot easier. This paper will approach the subject from a "BASCOM-user-point-of-view".

1-wire-net is a serial communication protocol, used by DS devices. The bus could be implemented in two basic ways :

With 2 wires, then DQ and ground is used on the device. Power is supplied on the DQ line, which is +5V, and used to charge a capacitor in the DS device. This power is used by the device for its internal needs during communication, which makes DQ go low for short periods of time. This bus is called the 1-wire bus.

With 3 wires, when +5V is supplied to the VDD line of the device, and DQ + ground as above. This bus is called the 2-wire bus.

So, the ground line is "not counted" by DS. But hereafter we use DS naming conventions.

How it works. (1-wire)

The normal state of the bus is DQ=high. Through DQ the device gets its power, and performs the tasks it is designed for.

When the host (your micro controller (uC)) wants something to happen with the 1-wire bus, it issues a reset-command. That is a very simple electric function that happens then; the DQ goes active low for a time (480uS on original DS 1-wire bus). This put the DS-devices in reset mode; then (they) send a presence pulse, and then (they) listen to the host.

The presence pulse is simply an active low, this time issued by the device(s).

Now, the host cannot know what is on the bus, it is only aware of that at least 1 DS device is attached on the bus.

All communication on the 1-wire bus is initialized by the host, and issued by time-slots of active-low on a normally high line (DQ), issued by the device, which is sending at the moment. The devices(s) internal capacitor supplies its power needs during the low-time.

How do you work with 1-wire-bus

Thereafter, you can read a device, and write to it. If you know you only have 1 sensor attached, or if you want to address all sensors, you can start with a "Skip Rom" - command. This means; take no notice about the IDs of the sensors - skip that part of the communication.

When you made a 1-wire-reset, all devices of the bus are listening. If you chose to address only one of them, the rest of them will not listen again before you have made a new 1-wire-reset on the bus.

I do not describe BASCOM commands in this text - they are pretty much self-explanatory. But the uC has to write the commands to the bus - and thereafter read the answer. What you have to write as a command depends on devices you are using - and what you want to do with it. Every DS chip has a data sheet, which you can find at http://www.dalsemi.com/datasheets/pdfindex.html. There you can find out all about the actual devices command structure.

There are some things to have in mind when deciding which of the bus-types to use.

The commands, from BASCOM, are the same in both cases. So this is not a problem.

The +5V power-supply on the VDD when using a 2-wire bus has to be from a separate power supply, according to DS. But it still works with taking the power from the same source as for the processor, directly on the stabilizing transistor. I have not got it to work taking power directly from the processor pin.

Some devices consume some more power during special operations. The DS1820 consumes a lot of power during the operation "Convert Temperature". Because the sensors knows how they are powered (it is also possible to get this information from the devices) some operations, as "Convert T" takes different amount of time for the sensor to execute. The command "Convert T" as example, takes ~200mS on 2-wire, but ~700mS on 1-wire. This has to be considered during programming.

And that power also has to be supplied somehow.

If you use 2-wire, you don't have to read further in this part. You can do simultaneously "Convert T" on all the devices you attach on the bus. And save time. This command is the most power-consuming command, possible to execute on several devices, I am aware of.

If you use 1-wire, there are things to think about. It is about not consuming more power than you feed. And how to feed power? That depends on the devices (their consumption) and what you are doing with them (their consumption in a specific operation).

Short, not-so-accurate description of power needs, not reflecting on cable lengths.

Only the processor pin as power supplier, will work < 5 sensors. (AVR, 1-wire-functions use an internal pull-up. 8051 not yet tested). Don't even think of simultaneous commands on multiple sensors.

With +5V through a 4K7 resistor, to the DQ-line, 70 sensors are tested. But, take care, cause issuing "Convert T" simultaneously, would cause that to give false readings. About ~15 sensors is the maximum amount of usable devices, which simultaneously performs some action. This approach DS refers to as "pull-up resistor".

With this in mind, a bus with up to 70 devices has been successfully powered this way.

The resistor mentioned, 4K7, could be of smaller value. DS says minimum 1K5, I have tested down to 500 ohm - below that the bus is not usable any more. (AVR). Lowering the resistor feeds more power - and makes the bus more noise resistant. But, the resistor minimum value is naturally also depending on the uC-pin electric capabilities. Stay at 4K7 - which is standard recommendation.

DS recommends yet another approach, called "strong pull-up" which (short) works via a MOS-FET transistor, feeding the DQ lines with enough power, still on 1-wire, during power-consuming tasks. This is not tested, but should naturally work. Because this functionality is really a limited one; BASCOM has no special support for that. But anyway, we tell you about it, just in case you wonder. Strong pull-up has to use one uC pin extra - to drive the MOS-FET.

Cable lengths (this section is only for some limitation understanding)

For short runs up to 30 meters, cable selection for use on the 1-Wire bus is less critical. Even flat modular phone cable works with limited numbers of 1-Wire devices. However, the longer the 1-Wire bus, the more pronounced cable effects become, and therefore greater importance is placed on cable selection.

For longer distances, DS recommends twisted-pair-cable (CAT5).

DS standard examples show 100 meters cable lengths, so they say, that's no problem. They also show examples with 300m cabling, and I think I have seen something with 600-meter bus (but I cant find it again).

Noise and CRC

The longer cable and the noisier environment, the more false readings will be made. The devices are equipped with a CRC-generator - the LSByte of the sending is always a checksum. Look in program examples to learn how to re-calculate this checksum in your uC. AND, if you notice that there are false readings - do something about your cables. (Shield, lower resistor)

Transfer speed

On the original 1-wire bus, DS says the transfer speed is about 14Kbits /second. And, if that was not enough, some devices has an overdrive option. That multiplies the speed by 10. This is issued by making the communication-time-slots smaller (from 60 uS to 6uS) which naturally will make the devices more sensitive, and CRC-error will probably occur more often. But, if that is not an issue, ~140Kbit is a reachable speed to the devices. So, whatever you thought before, it is FAST.

The BASCOM scanning of the bus is finds about 50 devices / second , and reading a specific sensors value to a uC should be about 13 devices / second.

Topology

Of the 1w-net - that is an issue we will not cover so much. Star-net, bus-net? It seems like you can mix that. It is a bus-net, but not so sensitive about that.

The benefit of the 1-wire bus

Each device is individual - and you can communicate with it over the media of 2 wires. Still, you can address one individual device, if you like. Get its value. There are 64 ^ 2 unique identifications-numbers.

Naturally, if lot of cables are unwanted, this is a big benefit. And you only occupy 1 processor pin.

DS supplies with different types of devices, which all are made for interfacing an uC - directly. No extra hardware. There are sensors, so you can get knowledge about the real world, and there are also potentiometers and relays, so you can do something about it. On the very same bus.

And the Ibutton approach from DS (ever heard of it?) is based on 1wire technology. Maybe something to pick up.

BASCOM let you use an uC with 1wire-devices so easy, that (since now) that also has to count as a benefit - maybe one of the largest. ;-)

The disadvantages of the 1-wire bus

So far as I know, DS is the only manufacturer of sensors for the bus. Some people think their devices are expensive. And, until now, it was really difficult to communicate with the devices. Particularly when using the benefit of several devices on one bus. Still some people say that the 1w-bus is slow - but I don't think so.

Göte Haluza
System engineer

4.20 Using the SPI protocol

General description of the SPI

The SPI allows high-speed synchronous data transfer between the AVR and peripheral devices or between several AVR devices. On most parts the SPI has a second purpose where it is used for In System Programming (ISP).

The interconnection between two SPI devices always happens between a master device and a slave device. Compared to some peripheral devices like sensors which can only run in slave mode, the SPI of the AVR can be configured for both master and slave mode.

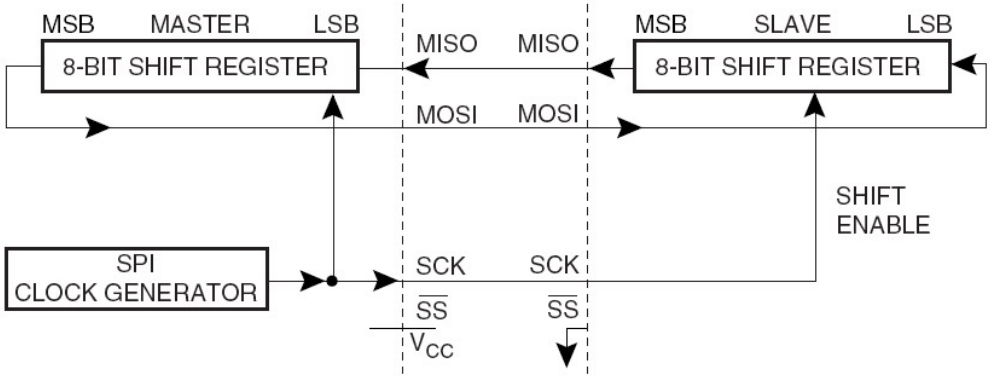
The mode the AVR is running in is specified by the settings of the master bit (MSTR) in the SPI control register (SPCR).

Special considerations about the /SS pin have to be taken into account. This will be described later in the section "Multi Slave Systems - /SS pin Functionality".

The master is the active part in this system and has to provide the clock signal a serial data transmission is based on. The slave is not capable of generating the clock signal and thus can not get active on its own.

The slave just sends and receives data if the master generates the necessary clock signal. The master however generates the clock signal only while sending data. That means that the master has to send data to the slave to read data from the slave.

Figure 61. SPI Master-Slave Interconnection



Data transmission between Master and Slave

The interaction between a master and a slave AVR is shown in Figure 1. Two identical SPI units are displayed. The left unit is configured as master while the right unit is

configured as slave. The MISO, MOSI and SCK lines are connected with the corresponding lines of the other part.

The mode in which a part is running determines if they are input or output signal lines. Because a bit is shifted from the master to the slave and from the slave to the master simultaneously in one clock cycle both 8-bit shift registers can be considered as one 16-bit circular shift register. This means that after eight SCK clock pulses the data between master and slave will be exchanged.

The system is single buffered in the transmit direction and double buffered in the receive direction. This influences the data handling in the following ways:

1. New bytes to be sent can not be written to the data register (SPDR) / shift register before the entire shift cycle is completed.
2. Received bytes are written to the Receive Buffer immediately after the transmission is completed.
3. The Receive Buffer has to be read before the next transmission is completed or data will be lost.
4. Reading the SPDR will return the data of the Receive Buffer.

After a transfer is completed the SPI Interrupt Flag (SPIF) will be set in the SPI Status Register (SPSR). This will cause the corresponding interrupt to be executed if this interrupt and the global interrupts are enabled. Setting the SPI Interrupt Enable (SPIE) bit in the SPCR enables the interrupt of the SPI while setting the I bit in the SREG enables the global interrupts.

Pins of the SPI

The SPI consists of four different signal lines. These lines are the shift clock (SCK), the Master Out Slave In line (MOSI), the Master In Slave Out line (MISO) and the active low Slave Select line (/SS). When the SPI is enabled, the data direction of the MOSI, MISO, SCK and /SS pins are overridden according to the following table.

Table 1. SPI Pin Overrides

Pin Direction Overrides	Master SPI Mode Direction Overrides	Slave SPI Modes
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
SS	User Defined	Input

This table shows that just the input pins are automatically configured. The output pins have to be initialized manually by software. The reason for this is to avoid damages e.g. through driver contention.

Multi Slave Systems - /SS pin Functionality

The Slave Select (/SS) pin plays a central role in the SPI configuration. Depending on the mode the part is running in and the configuration of this pin, it can be used to activate or deactivate the devices. The /SS pin can be compared with a chip select pin which has some extra features. In master mode, the /SS pin must be held high to ensure master SPI operation if this pin is configured as an input pin. A low level will switch the SPI into slave mode and the hardware of the SPI will perform the following actions:

1. The master bit (MSTR) in the SPI Control Register (SPCR) is cleared and the SPI system becomes a slave. The direction of the pins will be switched according to Table 1.

2. The SPI Interrupt Flag (SPIF) in the SPI Status Register (SPSR) will be set. If the SPI interrupt and the global interrupts are enabled the interrupt routine will be executed. This can be useful in systems with more than one master to avoid that two masters are accessing the SPI bus at the same time. If the /SS pin is configured as output pin it can be used as a general purpose output pin which does not affect the SPI system.

Note: In cases where the AVR is configured for master mode and it can not be ensured that the /SS pin will stay high between two transmissions, the status of the MSTR bit has to be checked before a new byte is written. Once the MSTR bit has been cleared by a low level on the /SS line, it must be set by the application to re-enable SPI master mode.

In slave mode the /SS pin is always an input. When /SS is held low, the SPI is activated and MISO becomes output if configured so by the user. All other pins are inputs. When /SS is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data.

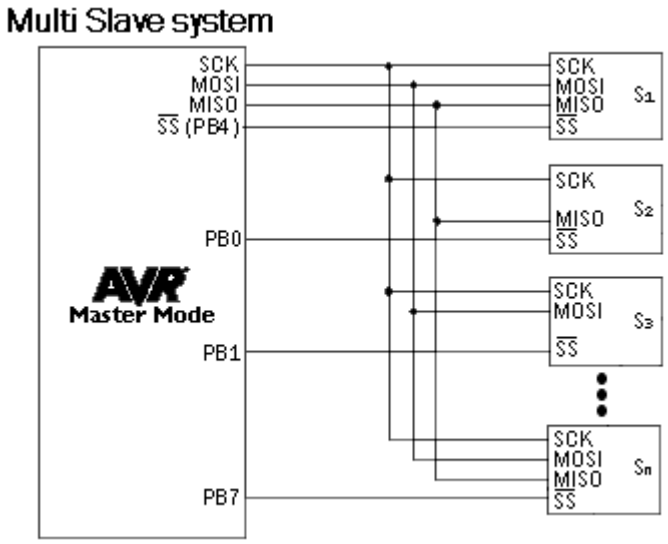
Table 2 shows an overview of the /SS Pin Functionality.

Note: In slave mode, the SPI logic will be reset once the /SS pin is brought high. If the /SS pin is brought high during a transmission, the SPI will stop sending and receiving immediately and both data received and data sent must be considered as lost.

TABLE 2. Overview of SS pin.

Mode	/SS Config	/SS Pin level	Description
Slave	Always input	High	Slave deactivated
		Low	Slave activated
Master	Input	High	Master activated
		Low	Master deactivated
	Output	High	Master activated
		Low	

As shown in Table 2, the /SS pin in slave mode is always an input pin. A low level activates the SPI of the device while a high level causes its deactivation. A Single Master Multiple Slave System with an AVR configured in master mode and /SS configured as output pin is shown in Figure 2. The amount of slaves, which can be connected to this AVR is only limited by the number of I/O pins to generate the slave select signals.



The ability to connect several devices to the same SPI-bus is based on the fact that only one master and only one slave is active at the same time. The MISO, MOSI and SCK lines of all the other slaves are tri stated (configured as input pins of a high impedance with no pull up resistors enabled). A false implementation (e.g. if two slaves are activated at the same time) can cause a driver contention which can lead to a CMOS latch up state and must be avoided. Resistances of 1 to 10 k ohms in series with the pins of the SPI can be used to prevent the system from latching up. However this affects the maximum usable data rate, depending on the loading capacitance on the SPI pins.

Unidirectional SPI devices require just the clock line and one of the data lines. If the device is using the MISO line or the MOSI line depends on its purpose. Simple sensors for instance are just sending data (see S2 in Figure 2), while an external DAC usually just receives data (see S3 in Figure 2).

SPI Timing

The SPI has four modes of operation, 0 through 3. These modes essentially control the way data is clocked in or out of an SPI device. The configuration is done by two bits in the SPI control register (SPCR). The clock polarity is specified by the CPOL control bit, which selects an active high or active low clock. The clock phase (CPHA) control bit selects one of the two fundamentally different transfer formats. To ensure a proper communication between master and slave both devices have to run in the same mode. This can require a reconfiguration of the master to match the requirements of different peripheral slaves.

The settings of CPOL and CPHA specify the different SPI modes, shown in Table 3. Because this is no standard and specified different in other literature, the configuration of the SPI has to be done carefully.

Table 3. SPI Mode configuration

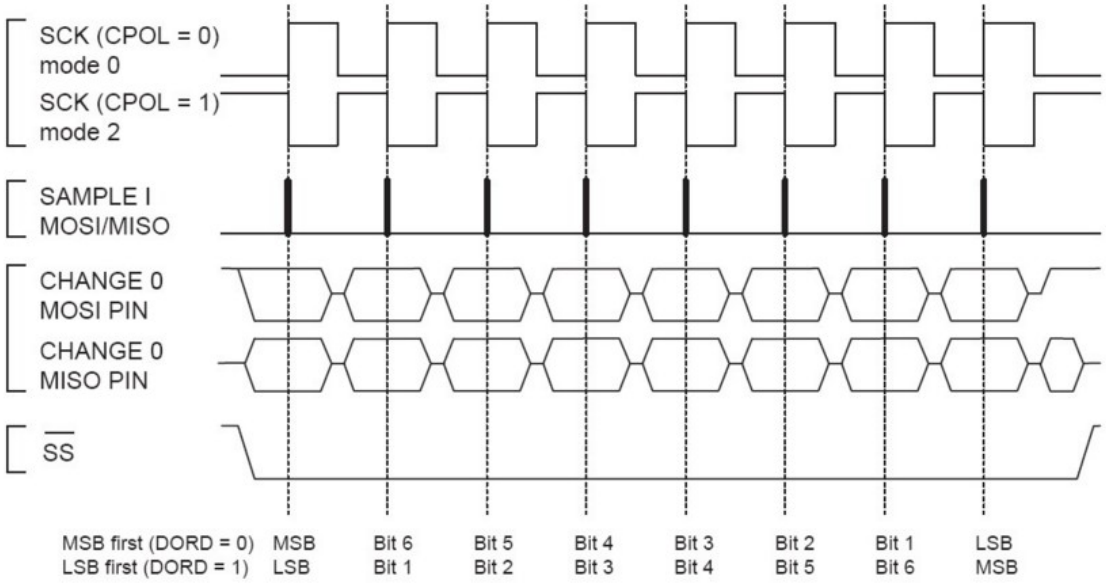
SPI Mode	CPOL	CPHA	Shift SCK edge	Capture SCK edge
0	0	0	Falling	Rising
1	0	1	Rising	Falling
2	1	0	Rising	Falling
3	1	1	Falling	Rising

The clock polarity has no significant effect on the transfer format. Switching this bit causes the clock signal to be inverted (active high becomes active low and idle low becomes idle high). The settings of the clock phase, how-ever, selects one of the two different transfer timings, which are described closer in the next two chapters. Since the MOSI and MISO lines of the master and the slave are directly connected to each other, the diagrams show the timing of both devices, master and slave. The /SS line is
the slave select input of the slave. The /SS pin of the master is not shown in the diagrams. It has to be inactive by a high level on this pin (if configured as input pin) or by configuring it as an output pin.

A.) CPHA = 0 and CPOL = 0 (Mode 0) and CPHA = 0 and CPOL = 1 (Mode 1)

The timing of a SPI transfer where CPHA is zero is shown in Figure 3. Two wave forms are shown for the SCK signal -one for CPOL equals zero and another for CPOL equals one.

Figure 62. SPI Transfer Format with CPHA = 0



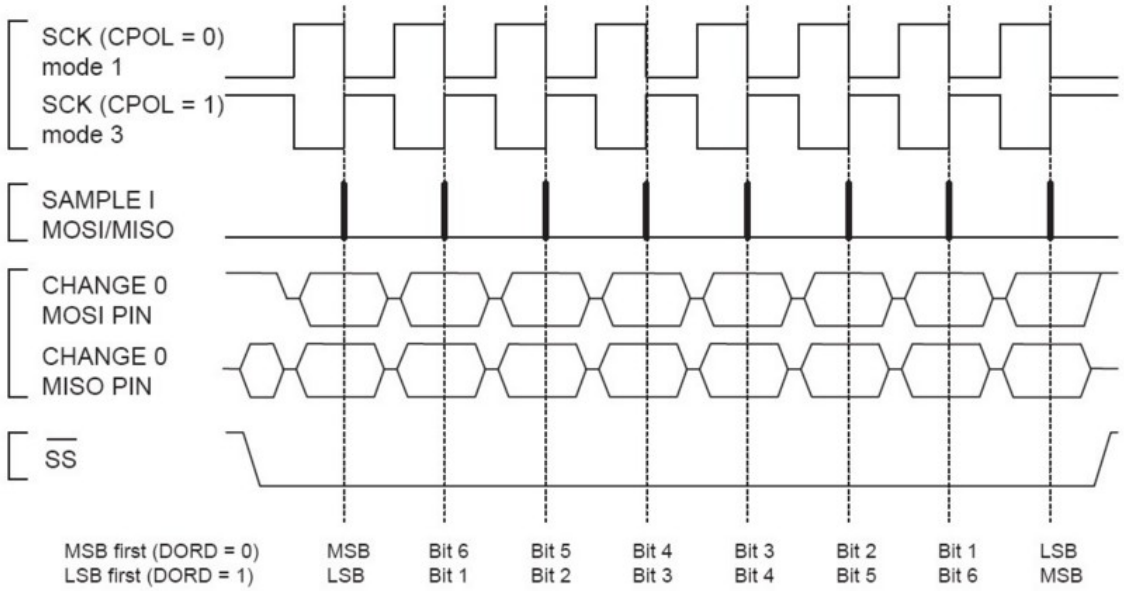
When the SPI is configured as a slave, the transmission starts with the falling edge of the /SS line. This activates the SPI of the slave and the MSB of the byte stored in its data register (SPDR) is output on the MISO line. The actual transfer is started by a software write to the SPDR of the master. This causes the clock signal to be generated. In cases where the CPHA equals zero, the SCK signal remains zero for the first half of the first SCK cycle. This ensures that the data is stable on the input lines of both the master and the slave. The data on the input lines is read with the edge of the SCK line from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one). The edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one) causes the data to be shifted one bit further so that the next bit is output on the MOSI and MISO lines.

After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to the receive buffer.

B.) CPHA = 1 and CPOL = 0 (Mode 2) and CPHA = 1 and CPOL = 1 (Mode 3)

The timing of a SPI transfer where CPHA is one is shown in Figure 4. Two wave forms are shown for the SCK signal -one for CPOL equals zero and another for CPOL equals one.

Figure 63. SPI Transfer Format with CPHA = 1



Like in the previous cases the falling edge of the /SS lines selects and activates the slave. Compared to the previous cases, where CPHA equals zero, the transmission is not started and the MSB is not output by the slave at this stage. The actual transfer is started by a software write to the SPDR of the master what causes the clock signal to be generated. The first edge of the SCK signal from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one) causes both the master and the slave to output the MSB of the byte in the SPDR.

As shown in Figure 4, there is no delay of half a SCK-cycle like in Mode 0 and 1. The SCK line changes its level immediately at the beginning of the first SCK-cycle. The data on the input lines is read with the edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one).

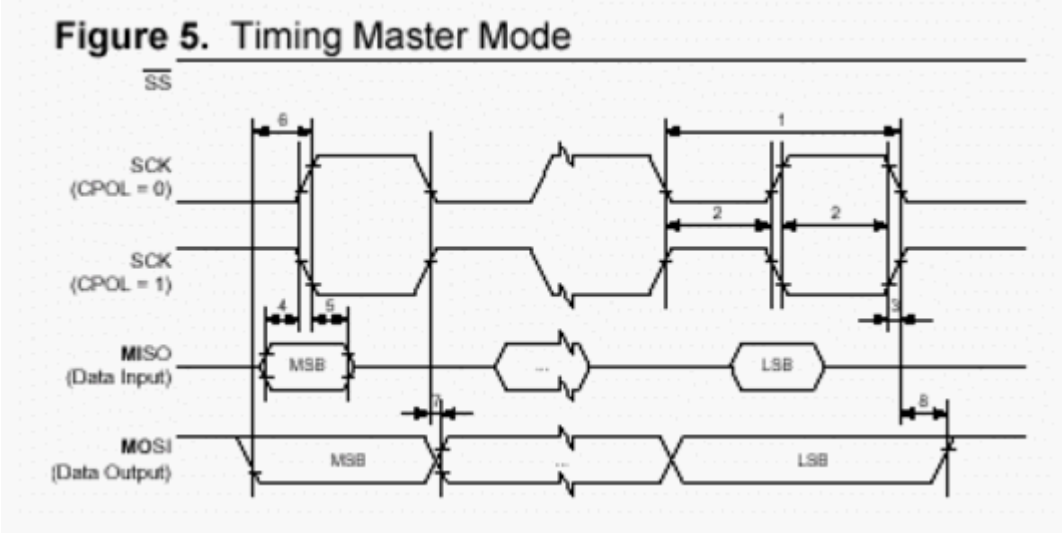
Mode	**Leading Edge**	**Trailing Edge**
0 CPOL=0, CPHA=0	Rising, Sample	Falling, Setup
1 CPOL=0, CPHA=1	Rising, Setup	Falling, Sample
2 CPOL=1, CPHA=0	Falling, Sample	Rising, Setup
3 CPOL=1, CPHA=1	Falling, Setup	Rising, Sample

After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to the receive buffer.

Considerations for high speed transmissions

Parts which run at higher system clock frequencies and SPI modules capable of running at speed grades up to half the system clock require a more specific timing to match the needs of both the sender and receiver. The following two diagrams show the timing of the AVR in master and in slave mode for the SPI Modes 0 and 1. The

exact values of the displayed times vary between the different pars and are not an issue in this application note. However the functionality of all parts is in principle the same so that the following considerations apply to all parts.



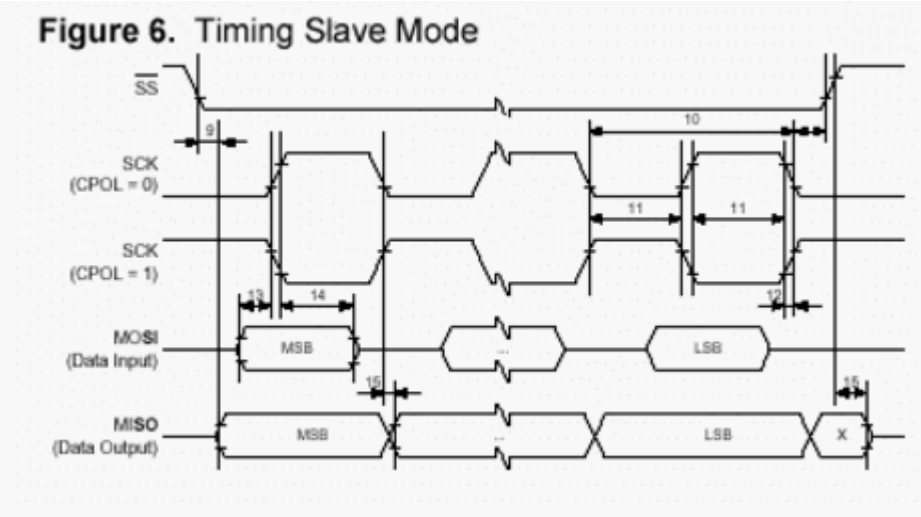
The minimum timing of the clock signal is given by the times "1" and "2". The value "1" specifies the SCK period while the value "2" specifies the high / low times of the clock signal. The maximum rise and fall time of the SCK signal is specified by the time "3". These are the first timings of the AVR to check if they match the requirements of the slave.

The Setup time "4" and Hold time "5" are important times because they specify the requirements the AVR has on the interface of the slave. These times determine how long before the clock edge the slave has to have valid output data ready and how long after the clock edge this data has to be valid.

If the Setup and Hold time are long enough the slave suits to the requirements of the AVR but does the AVR suit to the requirements of the slave?

The time "6" (Out to SCK) specifies the minimum time the AVR has valid output data ready before the clock edge occurs. This time can be compared to the Setup time "4" of the slave.

The time "7" (SCK to Out) specifies the maximum time after which the AVR outputs the next data bit while the time "8" (SCK to Out high) the minimum time specifies during which the last data bit is valid on the MOSI line after the SCK was set back to its idle state.



In principle the timings are the same in slave mode like previously described in master mode. Because of the switching of the roles between master and slave the requirements on the timing are inverted as well. The minimum times of the master mode are now maximum times and vice versa.

SPI Transmission Conflicts

A write collision occurs if the SPDR is written while a transfer is in progress. Since this register is just single buffered in the transmit direction, writing to SPDR causes data to be written directly into the SPI shift register. Because this write operation would corrupt the data of the current transfer, a write-collision error in generated by setting the WCOL bit in the SPSR. The write operation will not be executed in this case and the transfer continues undisturbed. A write collision is generally a slave error because a slave has no control over when a master will initiate a transfer. A master, however, knows when a transfer is in progress. Thus a master should not generate write collision errors, although the SPI logic can detect these errors in a master as well as in a slave mode.

When you set the SPI option from the Options, Compiler, SPI menu SPCR will be set to 01010100 which means ; enable SPI, master mode, CPOL = 1

When you want to control the various options with the hardware SPI you can use the CONFIG SPI 1061 statement.

See also: config SPI 1061, Config SPIx 1068, SPISLAVE 1831, SPIINIT 1514, SPIOUT 1518, SPIIN 1513, Using USI (Universal Serial Interface) 321

4.21 Using USI (Universal Serial Interface)

The Universal Serial Interface (USI) is a multi purpose hardware resource which provide the basics hardware for various serial communications and is faster and reliable then implementing it in software.
You mainly find the USI on ATTINY devices but also for example on ATMEGA169.

USI Features:
- Two-wire Synchronous Data Transfer
- Three-wire Synchronous Data Transfer
- Data Received Interrupt
- Wakeup from Idle Mode

The USI can be used in Two wire mode and in three wire mode:
- 2 wire mode --> I2C/TWI
- 3 wire mode --> SPI

The USI handle only the low level communication. High level communication for example for 2 wire mode (I2C) like address setting, message interpreting or preparing of data needs to be handled by software in the main loop.
There are Application Notes from Atmel available:

AVR312: Using the USI module as a I2C slave

AVR310: Using the USI module as a I2C master

The 3 wire mode (SPI) is easier to implement and therefore shown here as an example.
The Slave Select (SS) needs to be implemented in software if needed.
The USI Pin names are: DI, DO and USCK.

AVR319: Using the USI module for SPI communication

See also:

Using the SPI protocol 314, SPISLAVE 1831, Using I2C Protocol 297, confiig TWISLAVE 1123, I2C TWI Slave 1831 , USI as TWI Slave 1138

Following an example how to use an ATTINY as an SPI Master and another example show an SPI Slave over USI.

Example (SPI Master with USI):

1. Configure the port pin's:

```
'---------Using ATTINY as SPI MASTER over USI-----------------------------------
Config Portb.2 = Output                                 'USCK ----> SCK (Slave)
Config Portb.1 = Output                                 'DO ----> SDI (Slave)
Config Portb.0 = Input                                  'DI ----> SDO (Slave)
Set Portb.0
Sdo Alias Pinb.0                                        'Pullup
```

2. Configure the Slave Select

```
Config Portb.3 = Output                                 'Slave Select (SS) ----> SEL (Slave)
Set Portb.3
Sel Alias Portb.3
```

3. Configure the 3 wire mode

```
Set Usicr.usiwm0                                        'Three-wire mode. Uses DO, DI, and USCK pins.

'The Data Output (DO) pin overrides the corresponding bit in the PORTA
'register. However, the corresponding DDRA bit still controls the data direction.
'When the port pin is set as input the pin pull-up is controlled by the PORTA bit.
'The Data Input (DI) and Serial Clock (USCK) pins do not affect the normal port
'operation. When operating as master, clock pulses are software generated by
'toggling the PORTA register, while the data direction is set to output. The
'USITC bit in the USICR Register can be used for this purpose.
```

4. Function for send or receive a byte over USI (SPI Master mode)

```
Const     Usi_clk_low = &B0001_0001
Const     Usi_clk_high = &B0001_0011

'Wirte or read a byte over USI in SPI Master Mode
Function   Usi_byte(usi_out As Byte) As Byte
     Local I As Byte
       Usidr = Usi_out                                    'Byte to write over USI
        For I = 1 To 8
             Usicr = Usi_clk_low                          'Toggle the USI Clock to
send or receive the single bits over USI (8 Bit)
             Usicr =     Usi_clk_high
        Next
       Usi_byte = Usidr                                   'Byte received over USI
End Function
```

5. call the function to send/receive a byte

```
Reset  Sel
      Usi_return =    Usi_byte( my_byte)
Set  Sel
```

Example (SPI Slave with USI):

The following example show how to use an USI of ATTINY85 as SPI SLAVE.
(you will also find the SPI Master for this USI of ATTINY85 as SPI SLAVE example)

ATXMEGA (SPI Master) <-----SPI------> (SPI Slave over USI) ATTIN85

1. First we configure the USI in Three-wire Mode
2. Setup the USI Overflow Interrupt
3. And wait until the USI Oveflow Interrupt is fired
4. Then we read the USI Data-Register and clear the USI Interrupt flag

```
' Using USI as an SPI slave with Attiny85
' The ATTINY85 work with 3.3 V so we can direct connect it to an ATXMEGA
' Following you find also a SPI configuration with an XMEGA as SPI Master which I have
tested with this SPI Slave

' (
Config Spid = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk128 , Data_order = Msb , Ss =
Auto
'SS = Auto set the Slave Select (SS) automatically before a print #X or input #X command
(including initialization of the pin)
'Master SPI clock = 32MHz/Clk128 = 250KHz
Open "SPID" For Binary As #12
' )

$regfile = "ATtiny85.DAT"
$crystal = 8000000                                        'internal crystal
$hwstack = 32
$swstack = 10
$framesize = 30

Dim B As Byte
Dim   Usi_data_ready As Bit

Config Portb.1 = Output                                   'DO ---> MISO of ATXMEGA
(PD6)

Config Portb.2 = Output                                   'USCK ---> SCK of ATXMEGA
(PD7)
Set Portb.2                                               'enable Pullup

Config Portb.0 = Input                                    'DI ---> MOSI of ATXMEGA
(PD5)
Set Portb.0                                               'enable Pullup

'We do not use Slave Select in this example but this would be the configuration
Config Portb.4 = Input                                    'Slave Select
Set Portb.4                                               ' enable Pullup
Ss Alias Pinb.4


Config Portb.3 = Output                                   'Serial Debug output
```

```
Open "comb.3:9600,8,n,1" For Output As #1
Print #1 , "serial output"

'Init USI as SPI Slave in USICR = USI Control Register
Set Usicr.usiwm0                                    'Three-wire mode. Uses DO, DI,
and USCK pins.
Set Usicr.usics1                                    'Clock Source: External,
positive edge ; External, both edges
Set Usicr.usioie                                    'USI Counter Overflow
Interrupt Enable


On Usi_ovf Usi_overflow_int
Enable Usi_ovf
Enable Interrupts


Do
  If Usi_data_ready = 1 Then
    Reset Usi_data_ready
    Print #1 , B                                    'print the received byte over
debug output
  End If
Loop

End                                                 'end program

' After eight clock pulses (i.e., 16 clock edges) the 4-Bit USI counter will generate an
overflow interrupt
' A USI Overflow Int can also wakeup the Attiny from Idle mode if needed
Usi_overflow_int:
    Set Usi_data_ready
    B = Usidr
    Usisr = &B01_000000                             'Reset Overflow Flag and reset
4-Bit USI counter
Return
```

SPI Master for the ATTIN85 as SPI Slave over USI:

```
'This is the SPI MASTER for the ATTINY85 with USI in SPI Slave Mode

$regfile = "xm256a3bdef.dat"
$crystal = 32000000                                 '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80


Config Osc = Disabled , 32mhzosc = Enabled
Config Sysclock = 32mhz                             '--> 32MHz

'configure the priority
Config Priority = Static , Vector = Application , Lo = Enabled , Med = Enabled
Enable Interrupts

Config Com7 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8
Waitms 2
Open "COM7:" For Binary As #1
Print #1 ,
Print #1 , "------------SPI MASTER-Slave Test----------------"


'We use Port D for SPI
Config Pind.7 = Output
Config Pind.6 = Input
Config Pind.5 = Output
Config Pind.4 = Output
'Bit7 = SCK = Output ------> USCK (ATTINY85) (PinB.2)
'Bit6 = MISO = Input ------> DO (ATTINY85) (PinB.1)
'Bit5 = MOSI = Output ------> DI (ATTINY85) (PinB.0)
'Bit4 = SS = Output ------> SS (ATTINY85) (PinB.4)
Slave_select Alias Portd.4
Set Slave_select

Portd_pin4ctrl = Bits(3 , 4)                        ' Enalbe Pullup


Dim Bspivar As Byte
Dim Spi_send_byte As Byte
Dim Spi_receive_byte As Byte
```

```
Dim      Spi_master_want_send  As  Byte


'SPI,    Master|Slave    ,    MODE,    clock    division
Config   Spid  =  Hard  ,    Master  =  Yes  ,  Mode  =  0  ,     Clockdiv  =   Clk128  ,     Data_order  =  Msb  ,  Ss  =
Auto
'SS  =  Auto  set  the  Slave  Select  (SS)  automatically  before  a  print  #X  or  input  #X  command
(including      initialization      of      the      pin)
'Master  SPI  clock  =  32MHz/Clk128  =  250KHz
Open  "SPID"  For  Binary  As  #12

Main:

Do
    Wait  3                                                          'Every   3   seconds

    Incr     Spi_send_byte
    Print  #1  ,  "Spi_send_byte    =    " ;      Spi_send_byte
    'SEND TO SLAVE
    Print  #12  ,   Spi_send_byte                                    'SEND one BYTE TO SLAVE

    Waitms  10

    Input  #12  ,        Spi_receive_byte
    Print  #1  ,        Spi_receive_byte
Loop


End                                                                  'end     program

'there   is   NO   CLOSE   for   SPI


'
```

4.22 Power Up

At power up all ports are in Tri-state and can serve as input pins.

When you want to use the ports (pins) as output, you must set the data direction first with the statement : CONFIG PORTB = OUTPUT

Individual bits can also be set to be used as input or output.

For example : DDRB = &B00001111 , will set a value of 15 to the data direction register of PORTB.

PORTB.0 to PORTB.3 (the lower 4 bits) can be used as outputs because they are set high. The upper four bits (PORTB.4 to PORTB.7), can be used for input because they are set low.

You can also set the direction of a port pin with the statement :

CONFIG PINB.0 = OUTPUT | INPUT

The internal RAM is cleared at power up or when a reset occurs. Use $NORAMCLEAR to disable this feature.

You may use $INITMICRO 656 to set a port level and direction immediately on startup.

4.23 Reference Designs

4.23.1 EM4095 RFID Reader

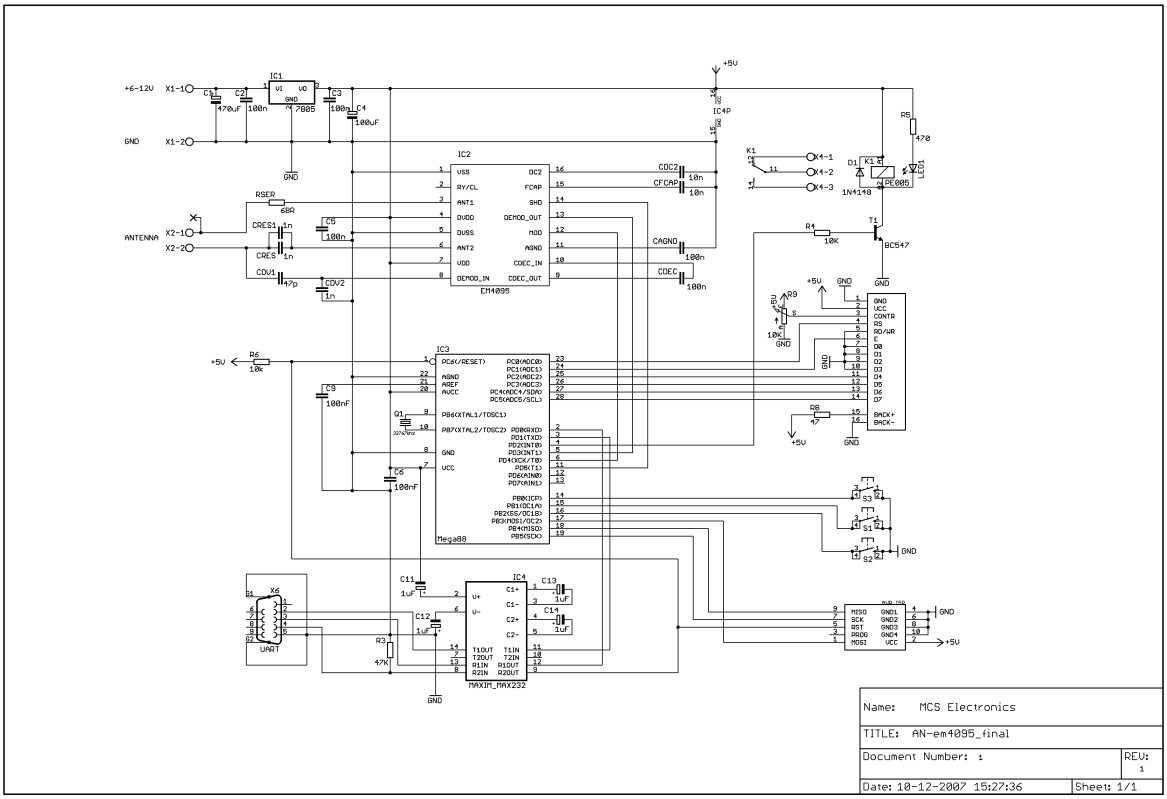
Introduction

RFID technology is an exciting technology. The EM4095 chip allows us to create a reader with little code or processor resources.
A complete KIT is available from the web shop at www.mcselec.com

This topic describes the reference design.
The data sheets you can download from:

EM4095 (chip) , EM4102 (transponder)

The circuit

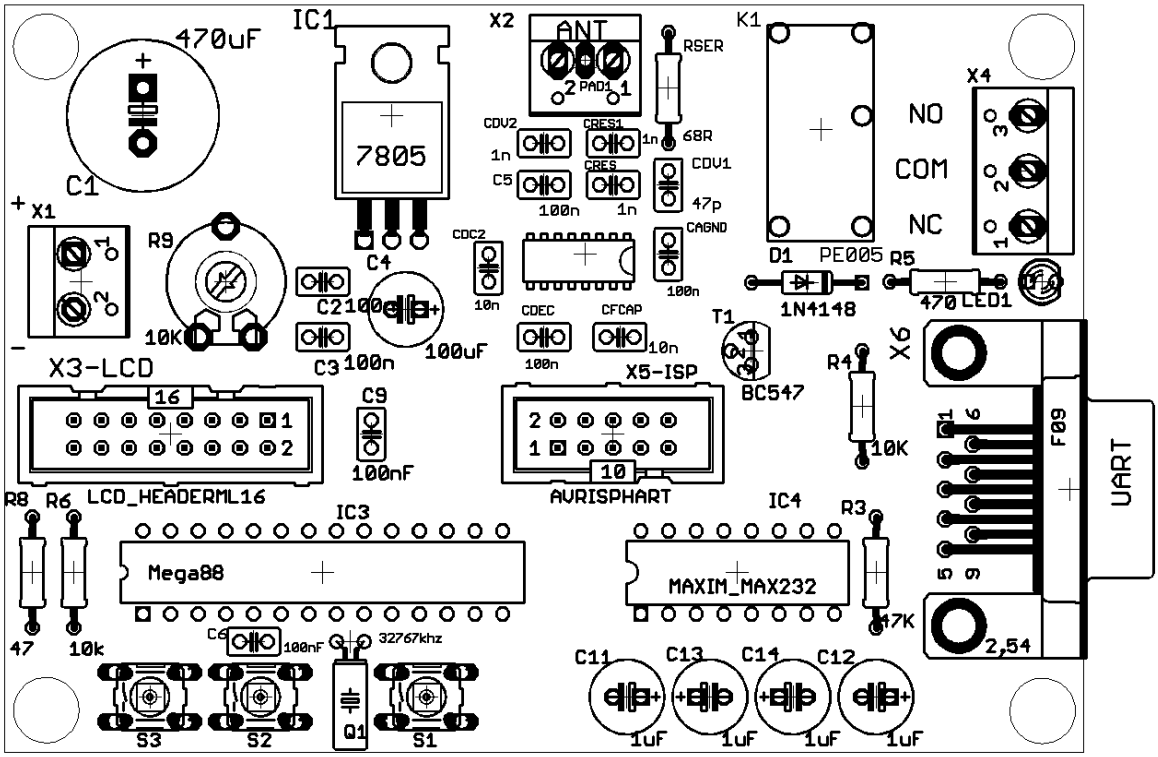


As you can see from the data sheets, the EM4095 needs little external hardware. A coil, capacitors that tune the coil for 125 KHz, are basically all that you need. IC1 is a voltage regulator that regulates the input voltage to 5V. (you can operate it from a 9V battery). The capacitors stabilize the output voltage. The DEMOD output of the EM4095 is connected to the microprocessor and the pin is used in input mode. The MOD and SHD pins are connected to micro pins that are used in output mode.

The micro(mega88) has a small 32 KHz crystal so the soft clock can be used. There are 3 switches that can be used for menu input, and there is a relay that can be used to activate a door opener. Parallel on the relay there is a LED for a visible indication. IC4 is a serial interface buffer so we can connect the PCB to our computer for logging and programming. The Mega88 is delivered with a Boot loader and thus can be serial programmed with the MCS Boot loader. That is why pin 4 of X6 (DTR) is connected via IC4(pin 8-9) to the reset pin of the micro(pin 1).
Further there is a standard 10-pins ISP programmer connector for the USB-ISP or

STK200, and an LCD connector for an optional LCD display.

The PCB



Part list

Component	Value
C1	470uF/25V
C2,C3,C5,C6,C9,CDEC,CAGND	100nF (104)
C4	100uF/16V
CRES1,CRES, CDV2	1nF(102)
CDV1	47pF
CDC2,CFCAP	10nF(103)
C11,C12,C13,C14	1uF/16V
RSER	68
R4,R6	10K
R5	470
R8	47
R3	47K
R9	1K-10K pot
IC1	7805
IC2	EM4095
IC3	ATMEGA88
IC4	MAX232
20 pin IC feet, 16 pin IC feet	
X1,X2	2-pin header
X3	16 pin boxed header
X4	3-pin header
X5	10-pin boxed header

X6	DB-9 female connector
T1	BC547
D1	1N4148
LED1	3 mm LED, red
K1	Relay, 5V
S1,S2,S3	switch
Q1	32768 Hz crystal
Antenna	
M3x6 bolt and nut	
4 rubber feet	

Building the PCB

As usually we start with the components that have the lowest height. And normally we would solder all passive components first, and insert/solder the active components last. This to prevent damage to the active components(IC). But since the EM4095 is only available in SMD, we need to solder this chip first. Make sure the chip is lined out right and that pin 1 matches the small dot on the chip which is an indication for pin 1. Then solder pin 1 and 16 so the chip can not be moved anymore. Now solder the remaining pins. Use an iron with a small tip. When you use too much solder, and two feet are soldered together do not panic. Just finish soldering and when ready, use some copper braid to remove the solder between the 2 feet. This works best when you lay the braid over the 2 pins, then push the solder iron to the braid so it will heat up. Then after some seconds, add some solder which will get sucked into the braid. This will in turn suck the other solder into the braid. While it does not seem logical to add solder, it will conduct the heat better. But since the used SMD chip is relatively large there should not be any problem.
Now mount and solder the following components :

- RSER (68 ohm)
- R3 (47K)
- R4,R6 (10 K)
- R5 (470)
- R8 (47 for LCD)
- D1 (diode 1N4148). The black line must match the line on the PCB(Kathode)
- C2,C3,C5,C6,C9,CDEC,CAGND (100 nF)
- CRES1,CRES , CDV2 (1nF)
- CDV1 (47pF)
- CDC2,CFCAP (10nF)
- 28 pins IC feet for the Mega88 and 16 pins IC feet for the MAX232
- Bend the wires of IC1 and mount IC1 with the bolt and nut
- Bend the wires of the crystal and mount Q1
- S1,S2,S3 (switches)
- LED1. The square pad matches the longest wire of the LED(Anode)
- R9 (potmeter for LCD contrast)
- T1(transistor BC547)
- Boxed header X5 and X3. Notice the gap in the middle which must match with the PCB
- X6 (DB9-female connector)
- K1 (relay)
- C11,C12,C13,C14 (1uF/16V)
- C4 (100uF/16V)
- X1,X2 (2 pins screw connectors)
- X4 (3 pin screw connector)
- C1 (470 uF/25V)
- 4 rubber feet

Operation

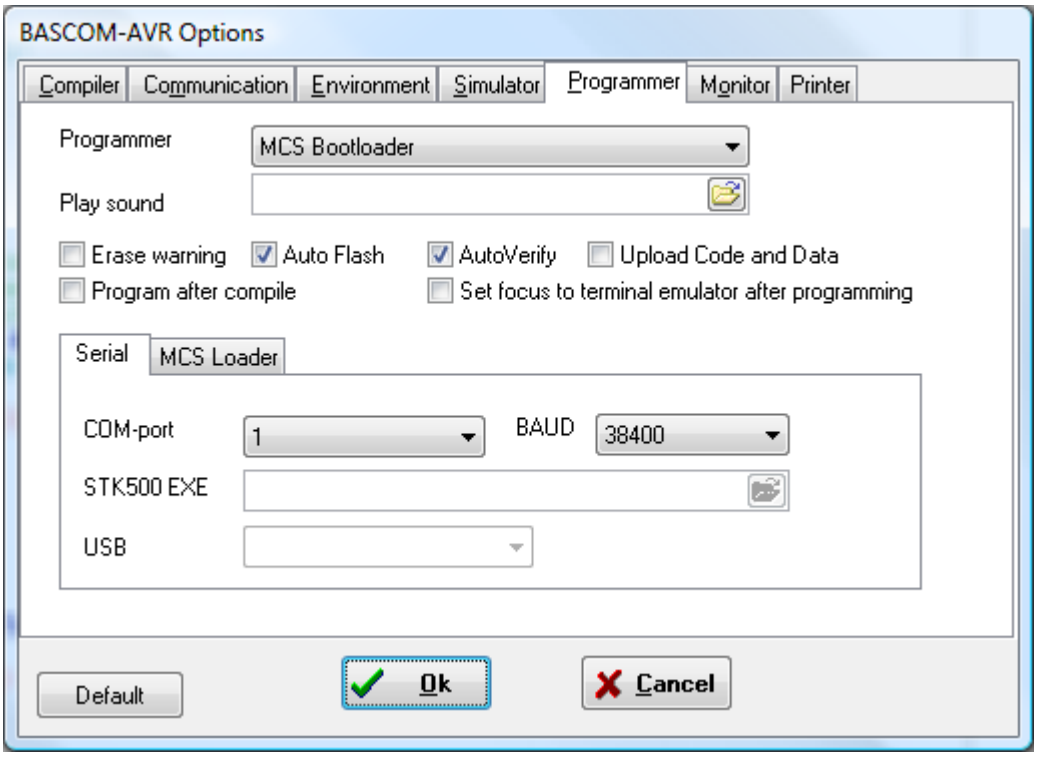
Now the PCB is ready. Make sure there are no solder drops on the PCB. You can measure with an Ohm-meter if there is a short circuit.
Measure pin 1 and pin 2 of IC1 (the voltage input) and pin 3 and pin 2 of IC1 (the voltage output).
When everything is ok, insert the MAX232 and the MEGA88.
You can connect the battery cord to header X1. The red wire is the plus. Since the circuit is not for beginners, there is no reverse polarity protection. While the 7805 does not mind a short circuit, the C1 elco might not like it.
Connect the battery and measure with a Volt meter if IC1 actual outputs 5V. If not, check the input voltage, and for a possible shortcut.

Connect the antenna to connector X2. The PCB is now ready for use. When you have the LCD display, connect it to the LCD header and adjust the variable resistor R9 so you can see square blocks.

Since the chip has a boot loader, you can serial program the device. We made a simple AN that can be used as a door opener. It has simple menu, and we can add new tags. When a valid tag is held in front of the antenna, it will activate the relay for 2 seconds. The LED will be turned on as well.
Compile the program **AN_READHITAG_EM4095.BAS** and select the **MCS Boot Loader programmer**. Connect a serial cable to X6 and press F4 to program.

You need a normal straight cable.

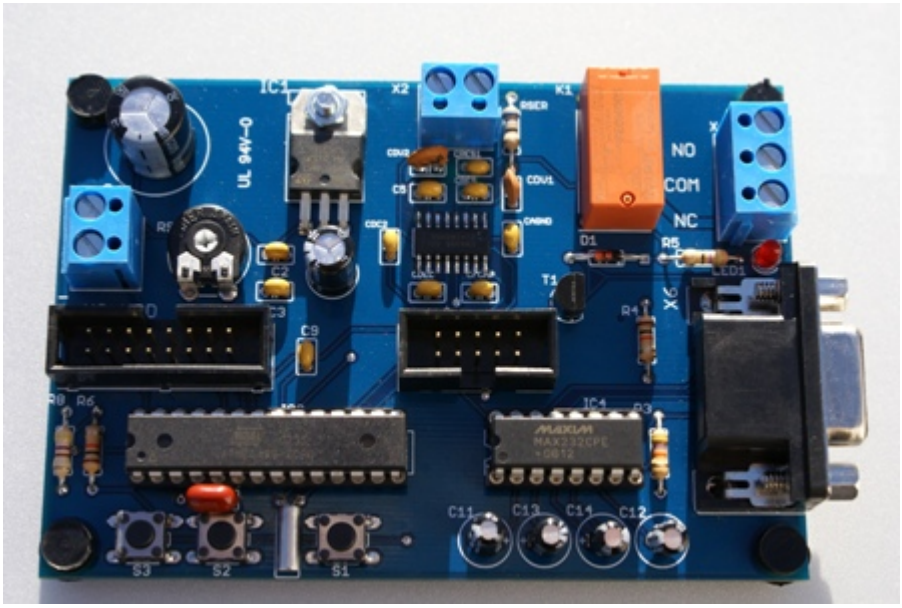
When you did not used the **MCS Bootloader** before, check the COM port settings and make sure the BAUD is set to 38400 as in the following screenshot:



You also need to set 'RESET via DTR' on the 'MCS Loader' TAB.
Now the program will start and show some info on the LCD. Each time you hold a RFID tag before the antenna/coil, the TAG ID will be shown.
When you press S3, you can store an RFID. Press S3, and then hold the TAG before the coil. When there is room , or the tag is new, it will be stored. Otherwise it will be ignored. The TAG ID is also stored in EEPROM.
Now when you hold the tag before the coil, the relay is activated for 2 seconds.
The AN is very simple and you can change and extend it easily.
One nice idea from Gerhard : use one TAG as a master tag to be able to add/remove tags.

Security

To make the code more secure you could add a delay so that a valid tag must be received twice, so after the valid TAG, wait 1 second, and then start a new measurement and check if the TAG is valid again.
This will prevent where a bit generator could be used to generate all possible codes. With 64 bit times a second, it would take ages before it would work.
The other hack would be to listen with a long range 125 KHz antenna, and recording all bits. A long range scanner would be very hard to make. It would be easier to open the door with a crowbar.
When you open your door with this device, make sure you have a backup option like a key in case there is no power. Also, when the door is opened by a magnetic door opener, make sure it has the right quality for the entrance you want to protect.



AN Code

```
'-----------------------------------------------------------------------------
'                   (c) 1995-2025 MCS Electronics
'  This sample will read a HITAG chip based on the EM4095 chip
'  Consult EM4102 and EM4095 datasheets for more info
'-----------------------------------------------------------------------------
'  The EM4095 was implemented after an idea of Gerhard Günzel
'  Gerhard provided the hardware and did research at the coil and capacitors.
'  The EM4095 is much simpler to use than the HTRC110. It need less pins.
'  A reference design with all parts is available from MCS
'-----------------------------------------------------------------------------
$regfile = "M88def.dat"
$baud = 19200
$crystal = 8000000
$hwstack = 40
$swstack = 40
$framesize = 40


Declare Function Havetag(b As Byte ) As Byte

'Make SHD and MOD low
_md Alias Portd.4
Config _md = Output
_md = 0

_shd Alias Portd.5
Config _shd = Output
_shd = 0

Relay Alias Portd.2
Config Relay = Output

S3 Alias Pinb.0
S2 Alias Pinb.2
S1 Alias Pinb.1
Portb = &B111                                               ' these are all input p

Config Clock = Soft                                         'we use a clock
Config Date = Dmy , Separator = -
```

```
Enable Interrupts                                          ' the clock and RFID co
Date$ = "15-12-07"                                         ' just a special date t
Time$ = "00:00:00"

'Config Lcd Sets The Portpins Of The Lcd
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5
Config Lcd = 16 * 2                                        '16*2 type LCD screen
Cls
            Lcd " EM4095 sample"
Lowerline : Lcd "MCS Electronics"

Dim Tags(5) As Byte                                        'make sure the array is
Dim J As Byte , Idx As Byte
Dim Eramdum As Eram Byte                                   ' do not use first posi
Dim Etagcount As Eram Byte                                 ' number of stored tags
Dim Etags(100) As Eram Byte                                'room for 20 tags
Dim Stags(100) As Byte                                     'since we have enough S
Dim Btags As Byte , Tmp1 As Byte , Tmp2 As Byte
Dim K As Byte , Tel As Byte , M As Byte

Config Hitag = 64 , Type = Em4095 , Demod = Pind.3 , Int = @int1
Print "EM4095 sample"


'you could use the PCINT option too, but you must mask all pins out so it will only
' Pcmsk2 = &B0000_0100
' On Pcint2 Checkints
' Enable Pcint2
On Int1 Checkints Nosave                                   'we use the INT1 pin al
Config Int1 = Change                                       'we have to config so t
Enable Interrupts                                          'as last we have to ena


'read eeprom and store in sram
'when the program starts we read the EEPROM and store it in SRAM
For Idx = 1 To 100                                         'for all stored tags
   Stags(idx) = Etags(idx)
   Print Hex(stags(idx)) ; ",";
Next

Btags = Etagcount                                          ' get number of stored
If Btags = 255 Then                                        ' an empty cell is  FF
   Print "No tags stored yet"
   Btags = 0 : Etagcount = Btags                           ' reset and write to ee
Else                                                       ' we have some tags
   For J = 1 To Btags
       Tmp2 = J * 5                                        'end
       Tmp1 = Tmp2 - 4                                     'start
       Print "RFID ; " ; J                                 ' just for debug
       For Idx = Tmp1 To Tmp2
         Print Hex(stags(idx)) ; ",";
       Next
       Print
   Next
End If

Do
   Print "Check..."
   Upperline : Lcd Time$ ; " Detect"
   If Readhitag(tags(1)) = 1 Then                          'this will enable INT1
      Lowerline
      For J = 1 To 5
         Print Hex(tags(j)) ; ",";
         Lcd Hex(tags(j)) ; ","
```

```
        Next
        M = Havetag(tags(1))                                    'check if we have this
        If M > 0 Then
           Print "Valid TAG ;" ; M
           Relay = 1                                            'turn on relay
           Waitms 2000                                          'wait 2 secs
           Relay = 0                                            'relay off
        End If
        Print
    Else
       Print "Nothing"
    End If
    If S3 = 0 Then                                              'user pressed button 3
       Print "Button 3"
       Cls : Lcd "Add RFID"
       Do
         If Readhitag(tags(1)) = 1 Then                         'this will enable INT1
            If Havetag(tags(1)) = 0 Then                        'we do not have it yet
               If Btags < 20 Then                                  'will it fit?
                  Incr Btags                                       'add one
                  Etagcount = Btags
                  Idx = Btags * 5                                  'offset
                  Idx = Idx - 4
                  Lowerline
                  For J = 1 To 5
                    Lcd Hex(tags(j)) ; ","
                    Stags(idx) = Tags(j)
                    Etags(idx) = Tags(j)
                    Incr Idx
                  Next
                  Cls
                  Lcd "TAG stored" : Waitms 1000
               End If
            End If
            Exit Do
         End If
       Loop
    End If
    If S2 = 0 Then
       Print "Button 2"
    End If
    If S1 = 0 Then
       Print "Button 1"
    End If

    Waitms 500
  Loop



  'check to see if a tag is stored already
  'return 0 if not stored
  'return value 1-20 if stored
  Function Havetag(b As Byte ) As Byte
    Print "Check if we have TAG : ";
    For K = 1 To 5
       Print Hex(b(k)) ; ","
    Next


    For K = 1 To 20
      Tmp2 = K * 5                                              'end addres
      Tmp1 = Tmp2 - 4                                           'start
```

```
    Tel = 0
    For Idx = Tmp1 To Tmp2
       Incr Tel
       If Stags(idx) <> B(tel) Then                    'if they do not match
          Exit For                                     'exit and try next
       End If
    Next

    If Tel = 5 Then                                    'if we did found 5 matc
       Print "We have one"
       Havetag = K                                     'set index
       Exit Function
    End If
  Next
  Havetag = 0                                          'assume we have nothing

End Function



Checkints:
 Call _checkhitag                                      'in case you have used
Return
```

Tips and Tricks

The oscillator frequency must be 125 KHz. You can measure this with an oscilloscope. It is possible that you need to remove a few windings of the antenna coil to get an exact 125 KHz. This will result in a higher distance that you can use for the tags.

Part V

5 Chips

5.1 AVR

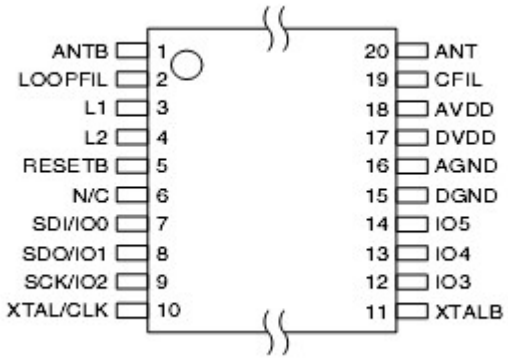
This topic describes some general hardware related problems that were found by users.

Unexpected brown out
- Processors with analog ports (used for A/D) are connected to AVCC and not VCC. This can cause the brown out detection to trigger. For example this is true for the Mega1284 PORTC. Using the ports to switch a small load would trigger the brown out while using a different port, powered from VCC would not give this problem.

Errata
The Errata you will find in the data sheet of the processor. It contains information about bugs in the hardware. Some times there are work around's, and some times there is no solution. It is a good idea to read the Errata BEFORE you begin to use the processor for a new design.

5.2 AT86RF401

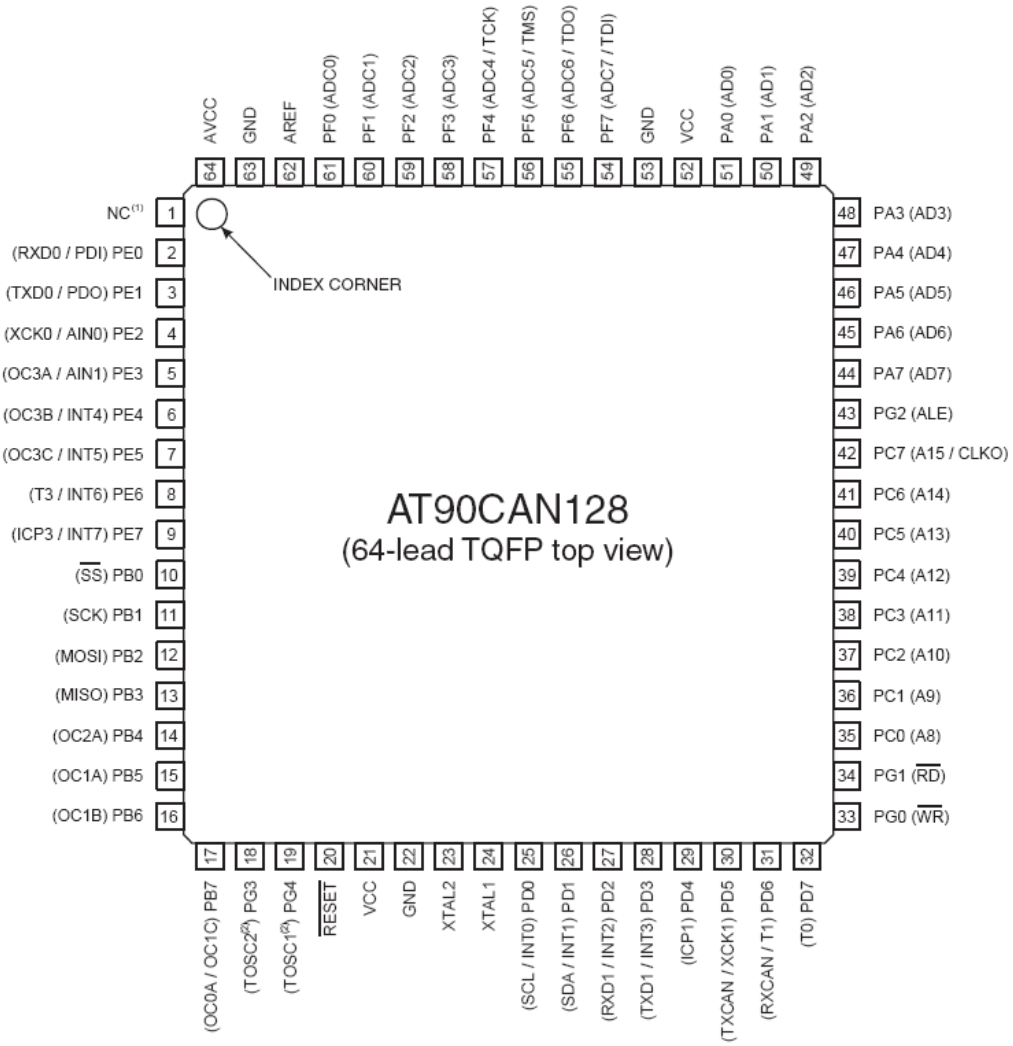
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.3 AT90

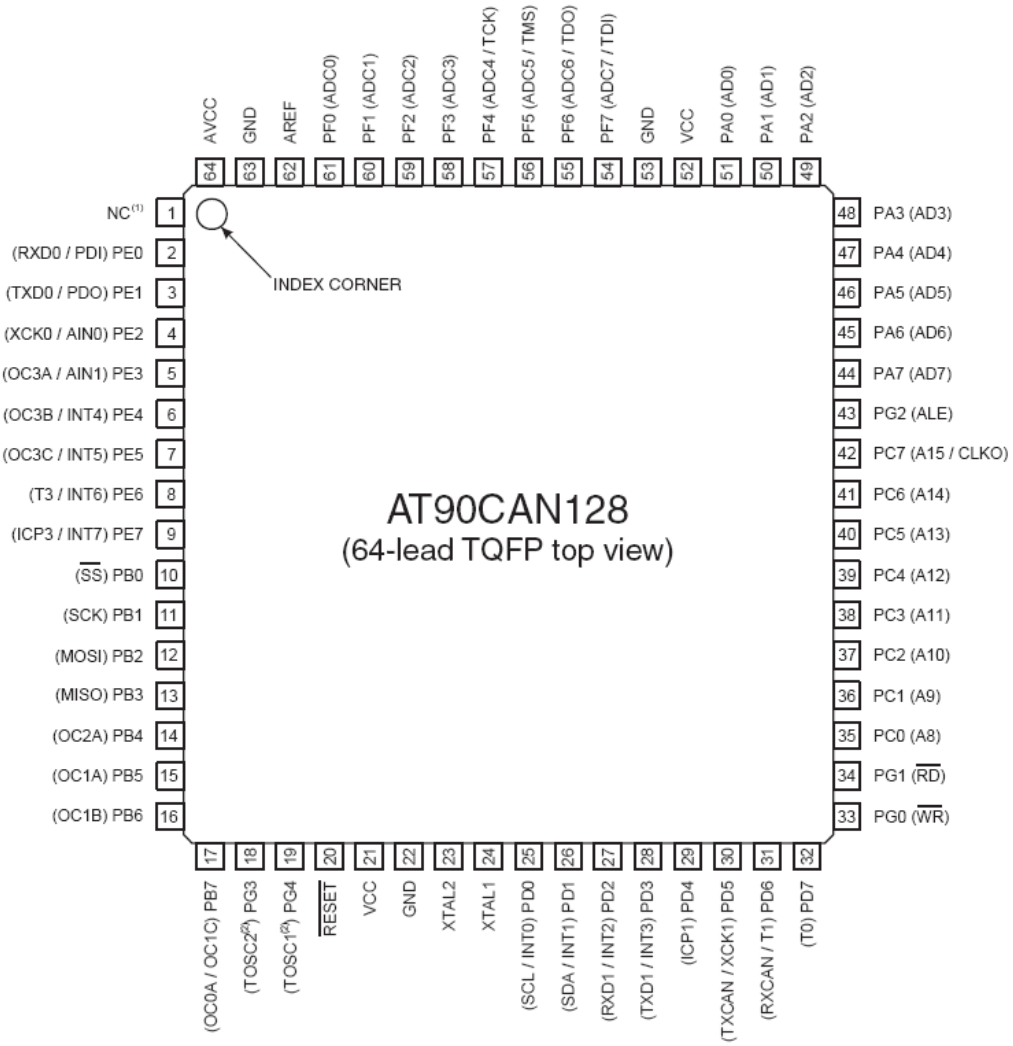
5.3.1 AT90CAN32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



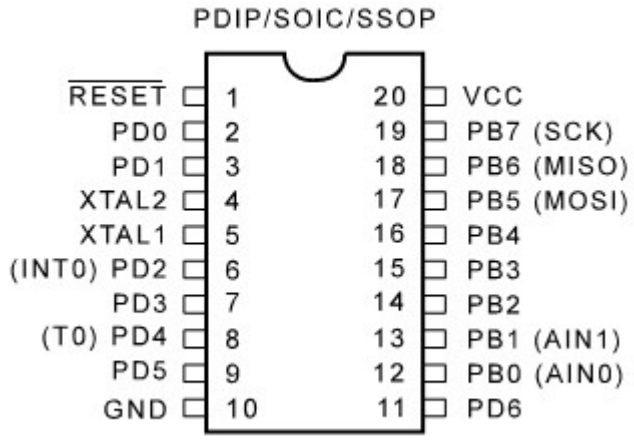
5.3.2 AT90CAN128

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



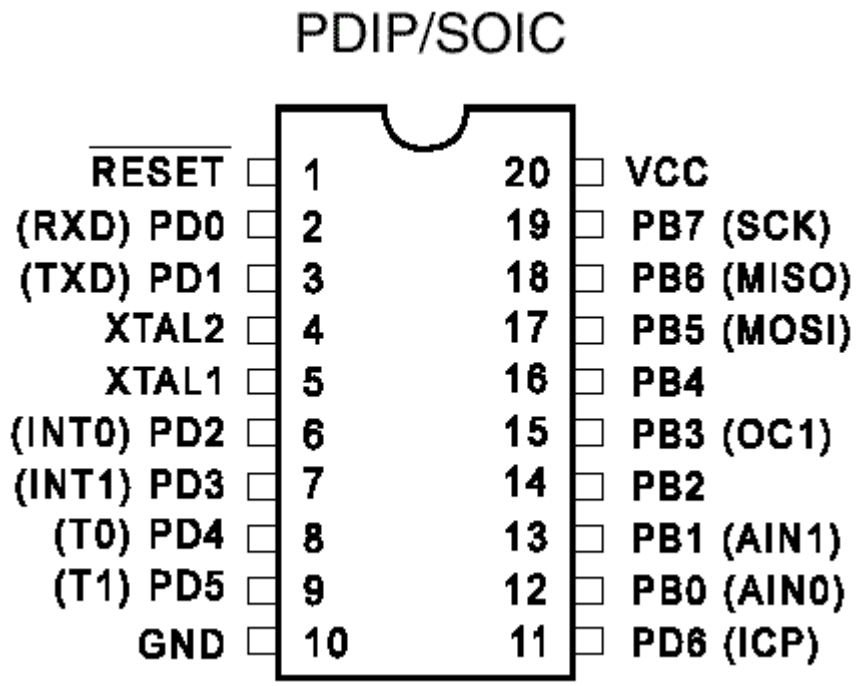
5.3.3 AT90S1200

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.3.4 AT90S2313

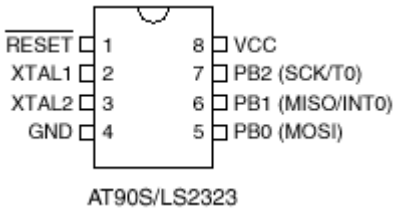
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The ATTiny2313 should be used for new designs.

5.3.5 AT90S2323

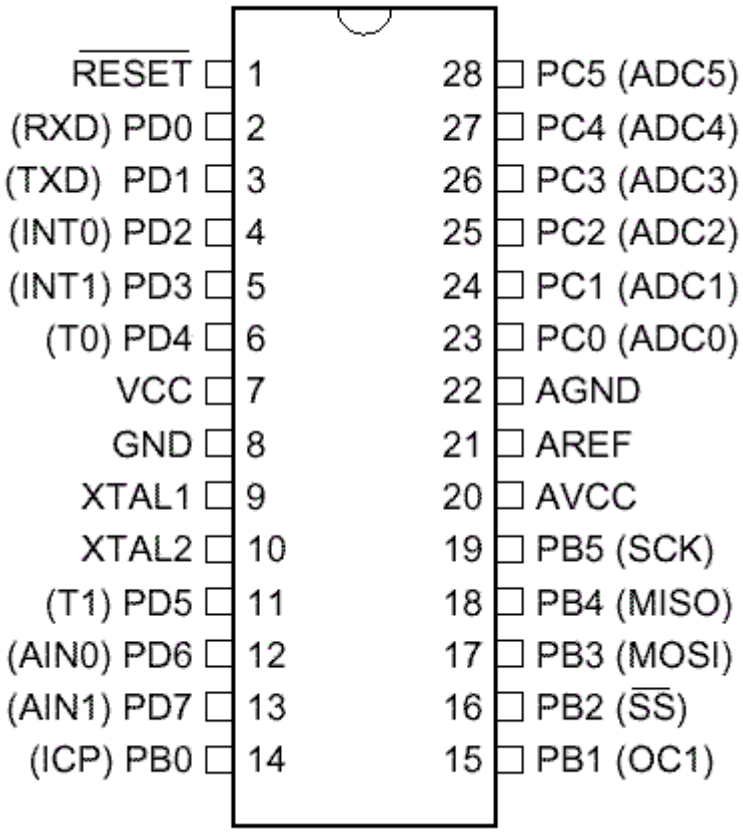
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



AT90S/LS2323

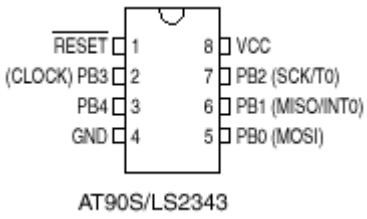
5.3.6 AT90S2333

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.3.7 AT90S2343

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



AT90S/LS2343

[tip from Martin Verschuren]

When using the AT90S2343 with BASCOM-AVR 1.11.6.4 and the STK200. Programming must be done with jumper ext-clk.

The BASCOM build in programmer will detect a Tiny22, which seems to have the same ID string as the 2343 (Atmel source) so no wonder.

By using the internal clock RCEN=0, then the jumper of the STK200 must be on int.

clk after programming.

Don't leave this away, some AT90S2343 will not correctly startup.

In your own project notice that you have to pull up the clk pin(2) at power up else it won't work. (I just looked for it for a day to get this problem solved:-)

Note : the at90s2343 and tiny22 have the same chip ID. In BASCOM you need to choose the tiny22 even if you use the 2343.

I note from MCS : only the AT23LS43-1 has the internal oscillator programmed by default! All other 2343 chips need an external clock signal. Tip: use a AT90S2313 and connect X2 to the clock input of the 2343.

[tip from David Chambers]

Using the AT90S2343 with BASCOM 1.11.7.3 the DT006 hardware there are no problems with programming the chip ie no special jumper conditions to enable programming. However it is best to remove links connecting ports to the DT006 LED’s before programming. If access to PB3 and PB4 is desired then jumpers J11 & J12 must be installed with pins 2 and 3 linked in both cases. Note that PB3 and PB4 are each connected to a momentary pushbutton on the DT006 board. These can be used to check contact closure functions, so bear this in mind when writing code for contact monitoring.

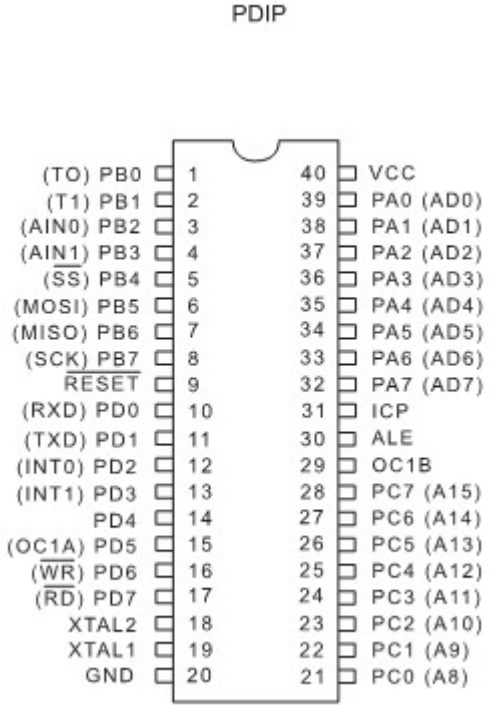
The current ATMEL data sheet specifies that all versions –1, -4 and –10 are supplied with a fuse bit set for the internal clock that operates at approximately 1Mhz. If using the internal clock make sure to enter 1000000 under Options\Compiler\Communication\frequency.

A great little chip with minimal external components. Only the resistor and capacitor required for RESET during power up.

Note that the LED’s on the DT006 are not connected to the same programmed port pins when changing the chip type. This is because the special functions assigned ports varies between the 8pin, 20 pin and 28 pin products eg the MOSI, MISI and SCK functions are assigned to PB0, PB1 and PB2 for an 8 pin processor and PB5, PB6 and PB7 for a 20 pin processor. The result is that for a given program the LED’s that respond are different.

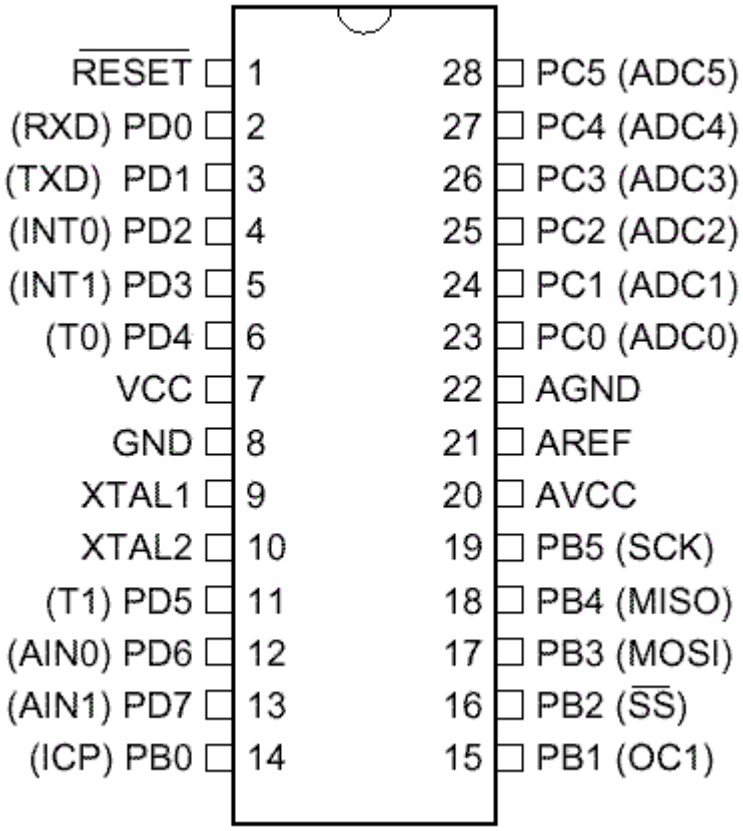
5.3.8 AT90S4414

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



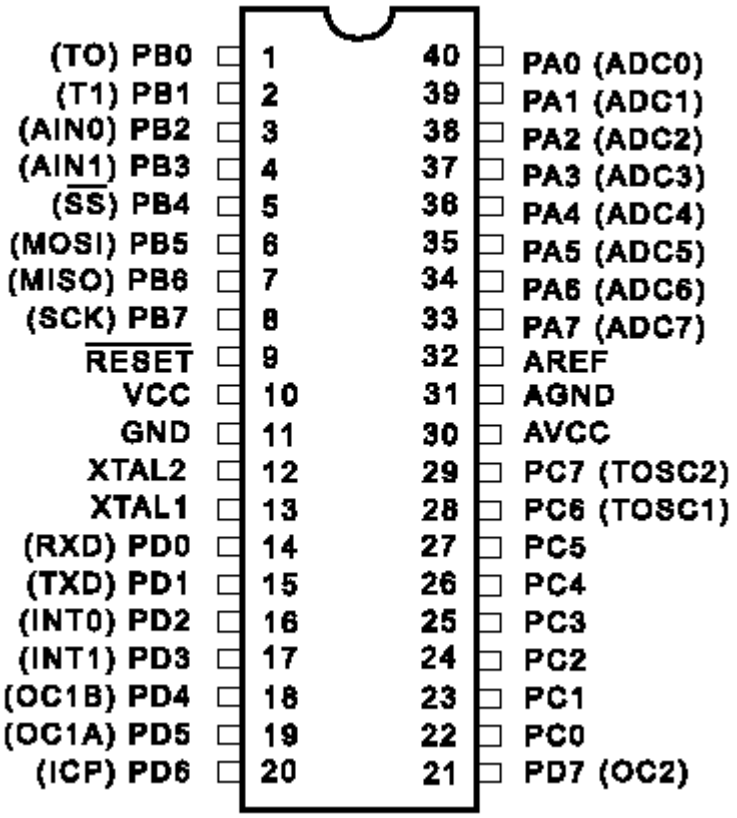
5.3.9 AT90S4433

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



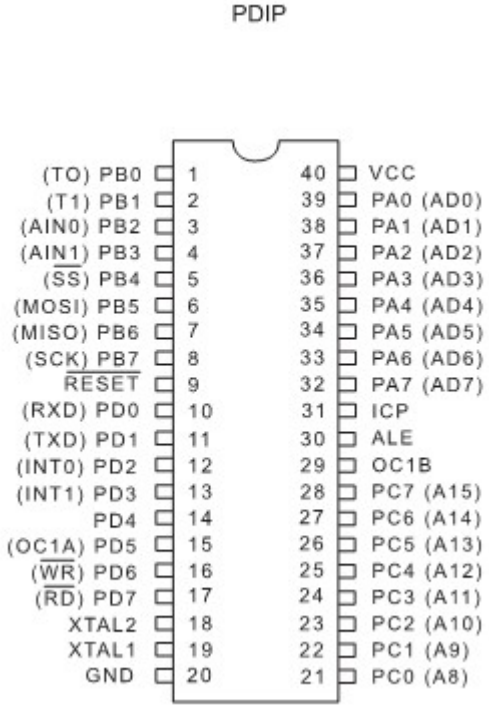
5.3.10 AT90S4434

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



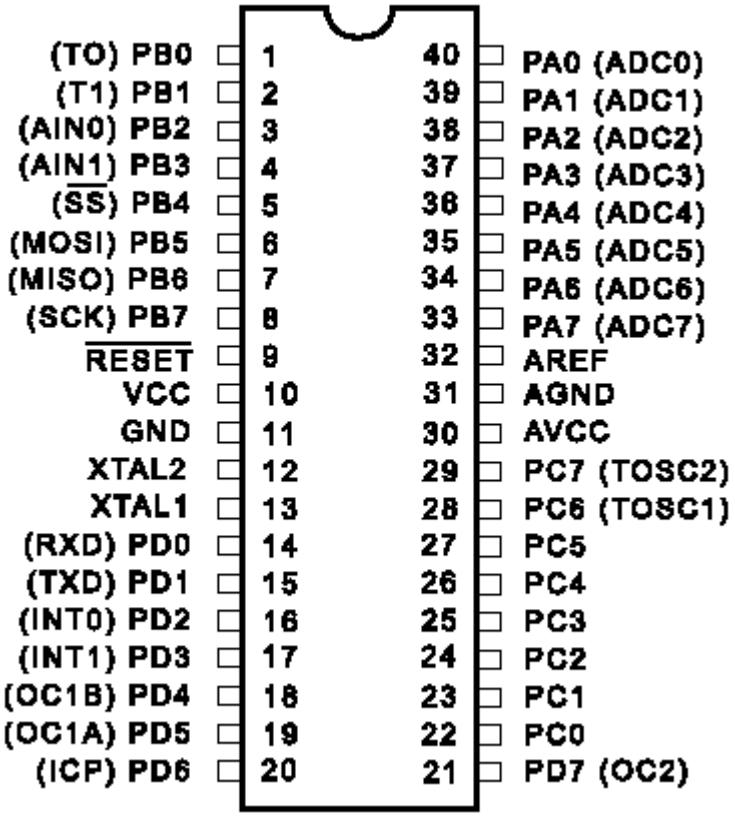
5.3.11 AT90S8515

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



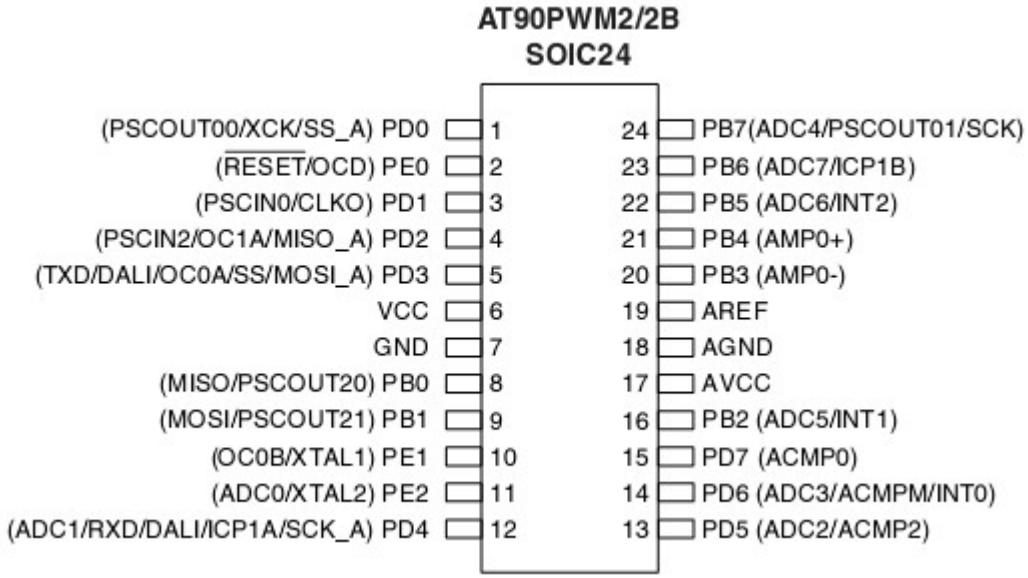
5.3.12 AT90S8535

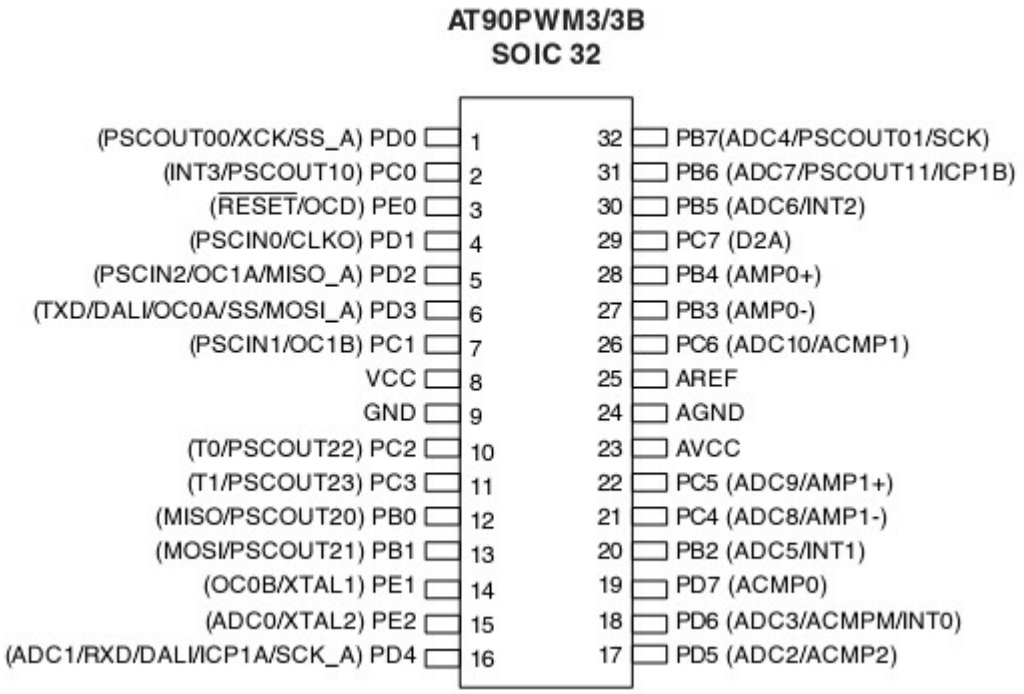
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.3.13 AT90PWM2-3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

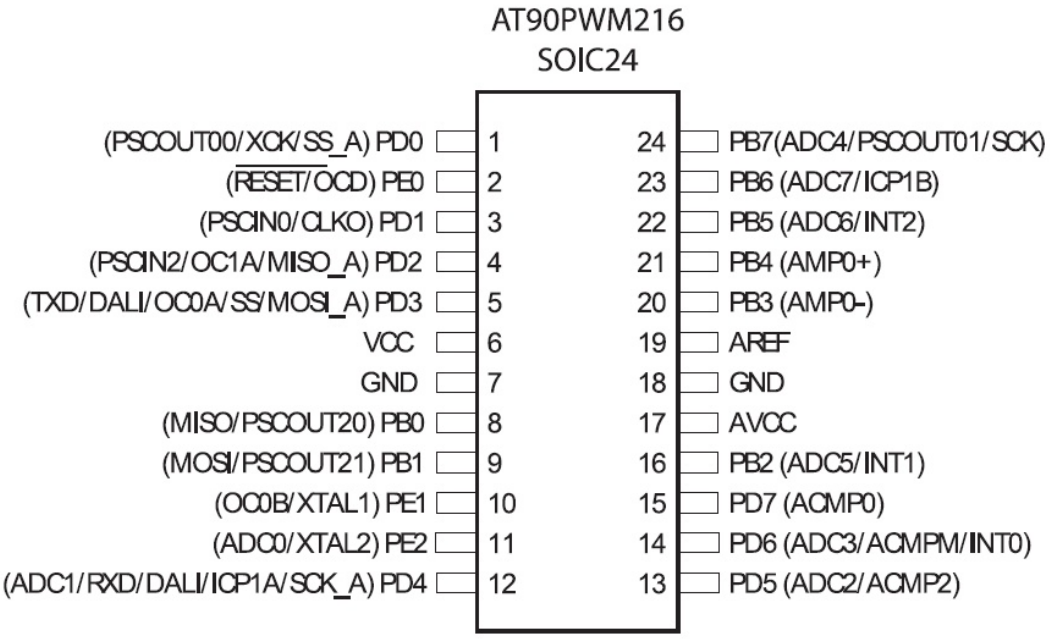




5.3.14 AT90PWM216

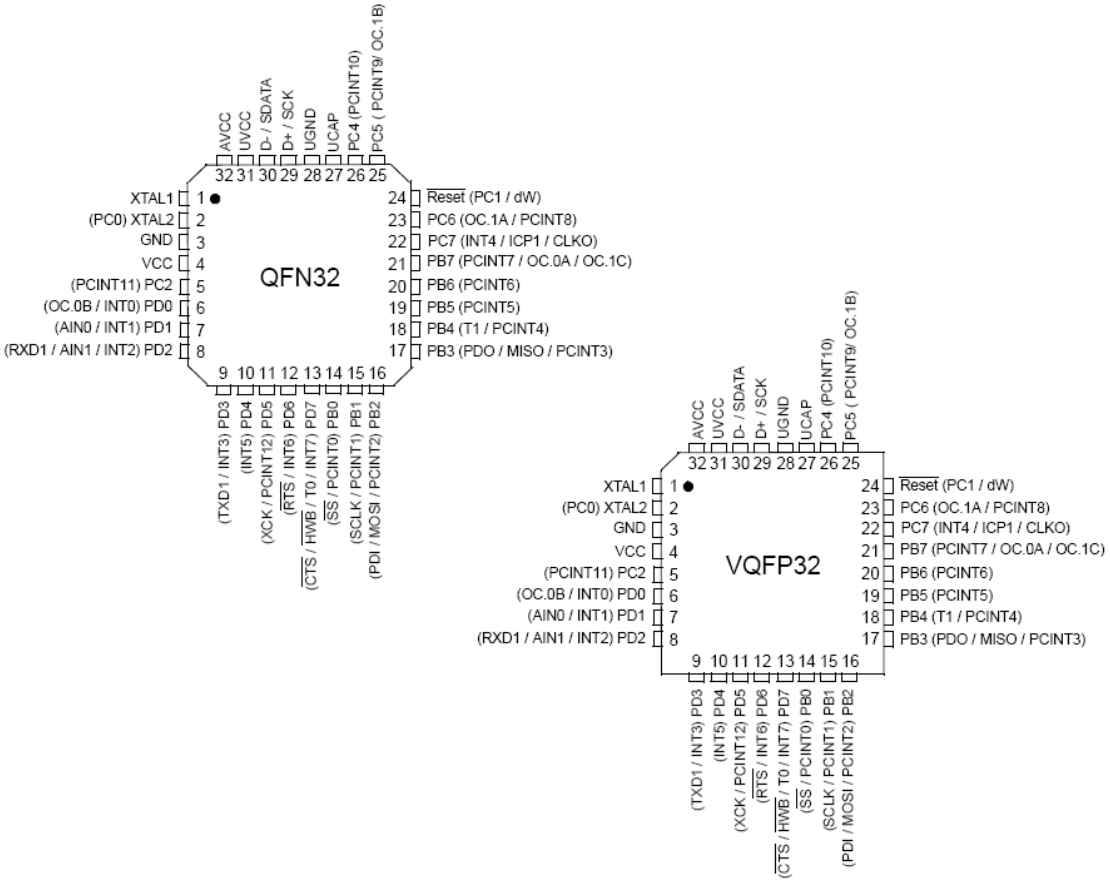
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 3-1. SOIC 24-pin Package



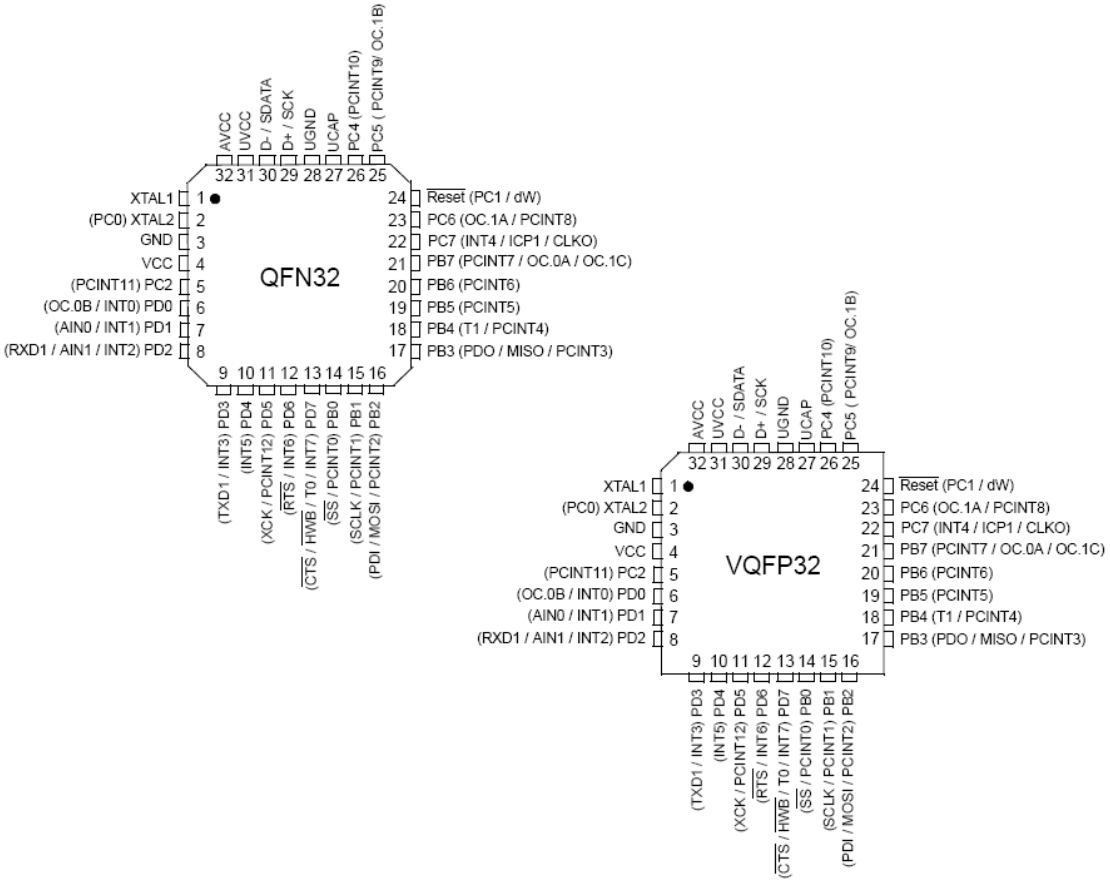
5.3.15 AT90US82

The USB82 is supported by the optional USB Add On. PORTC.4 is used to sense the power of the USB bus.

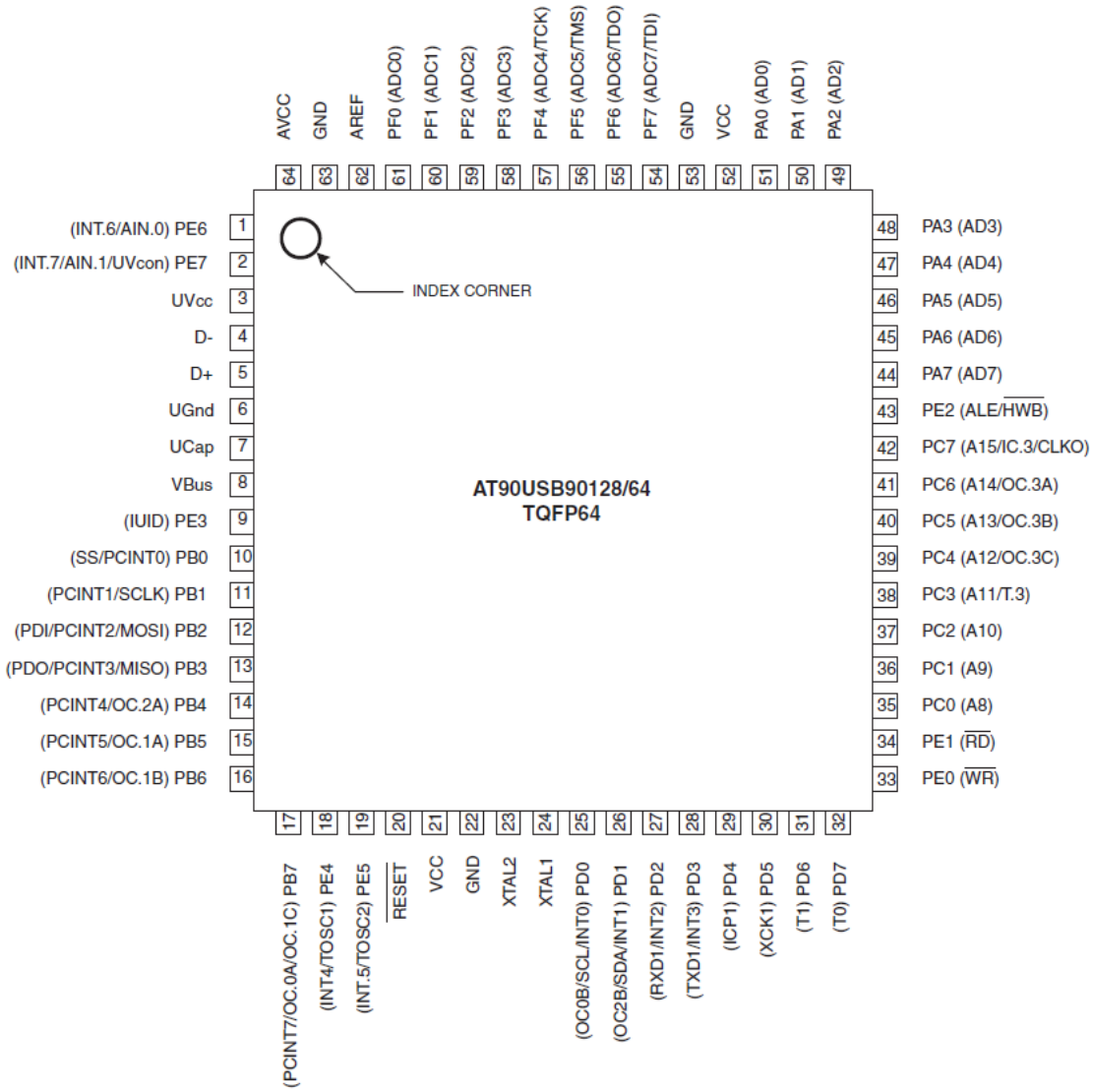


5.3.16 AT90USB162

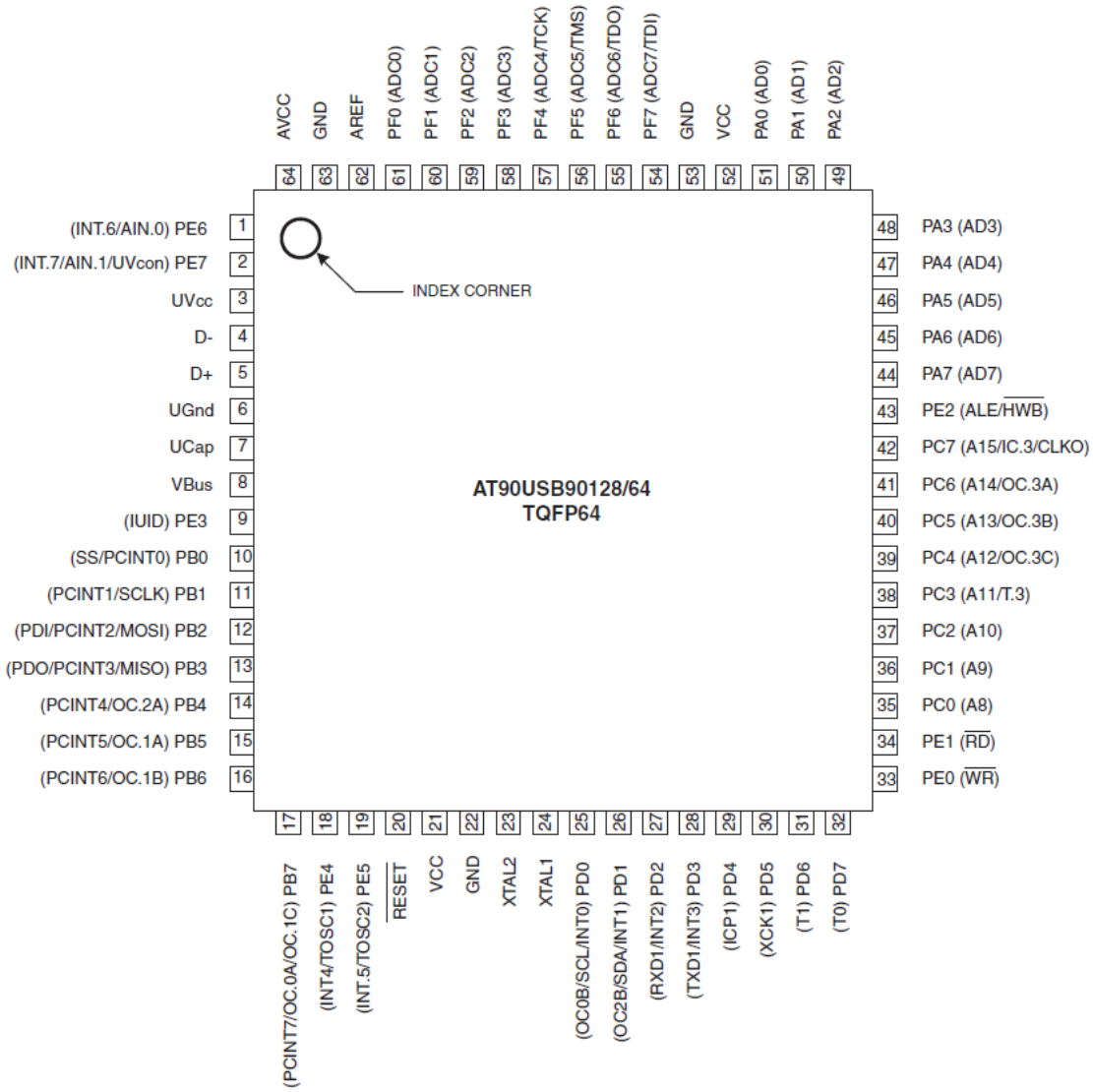
The USB162 is supported by the optional USB Add On. PORTC.4 is used to sense the power of the USB bus.



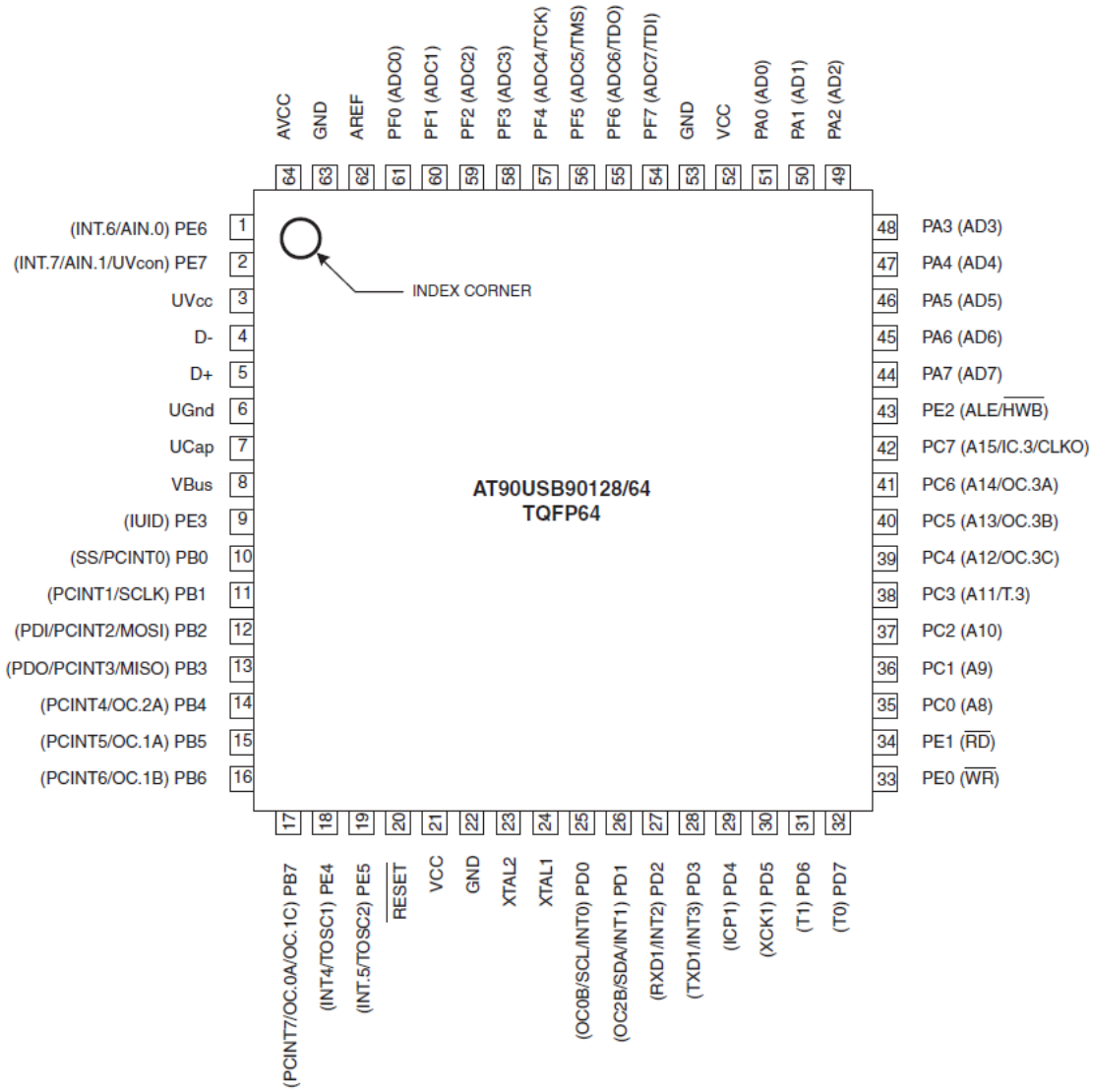
5.3.17 AT90USB646



5.3.18 AT90USB1286



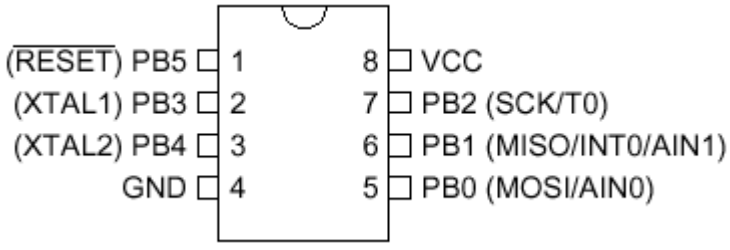
5.3.19 AT90USB1287



5.4 ATTINY

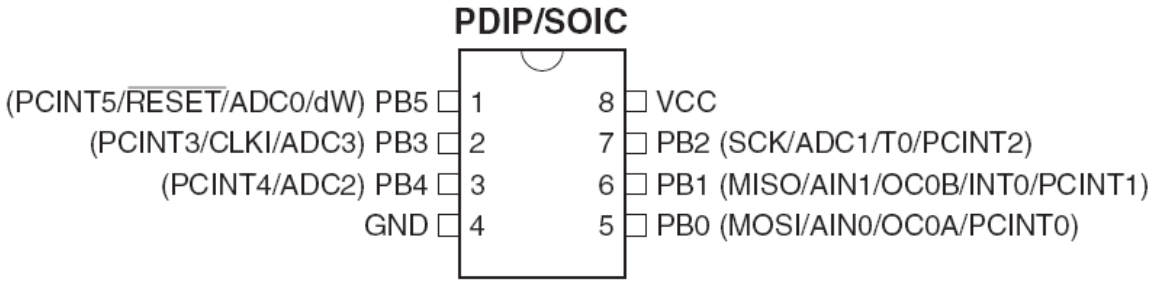
5.4.1 ATTINY12

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



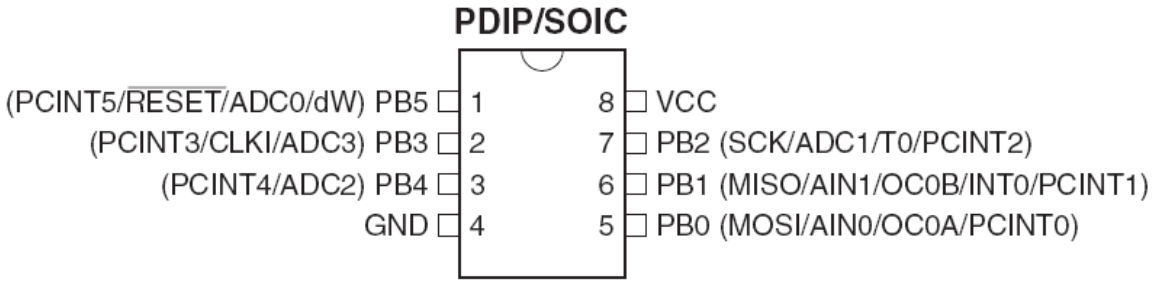
5.4.2 ATTINY13

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



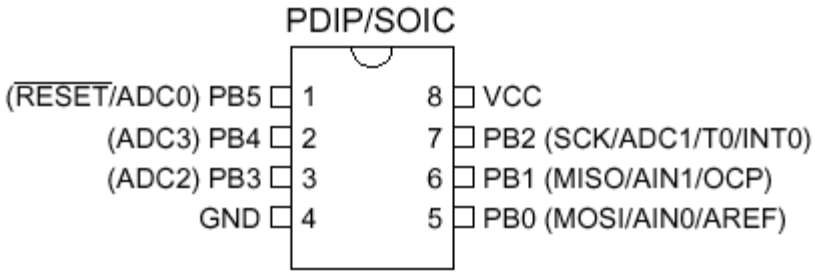
5.4.3 ATTINY13A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.4 ATTINY15

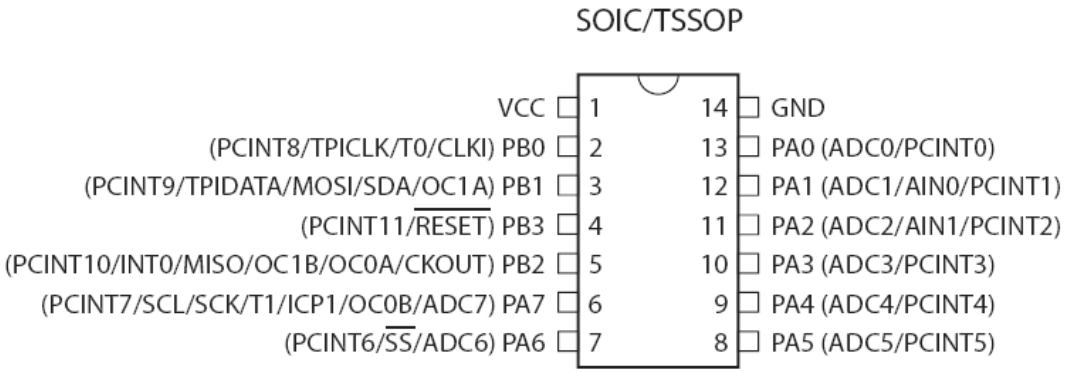
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.5 ATTINY20

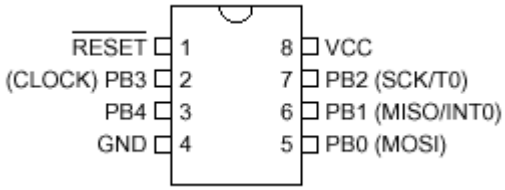
The ATTINY20 is a 14 pins AVR chip. It has NO EEPROM. It also does not have a UART.
The TWI slave interface is not compatible with TWI found in other AVR chips.
The chip has a PDI programming interface and does not support ISP or JTAG.
The watchdog is also different compared to other AVR chips. It is using a CCP register which is similar as the Xmega.

The processor also only has 16 registers (R16-R31) and is missing registers R0-R15. This does not make the chip a good choice for using with BASCOM since BASCOM uses the lower registers as well.



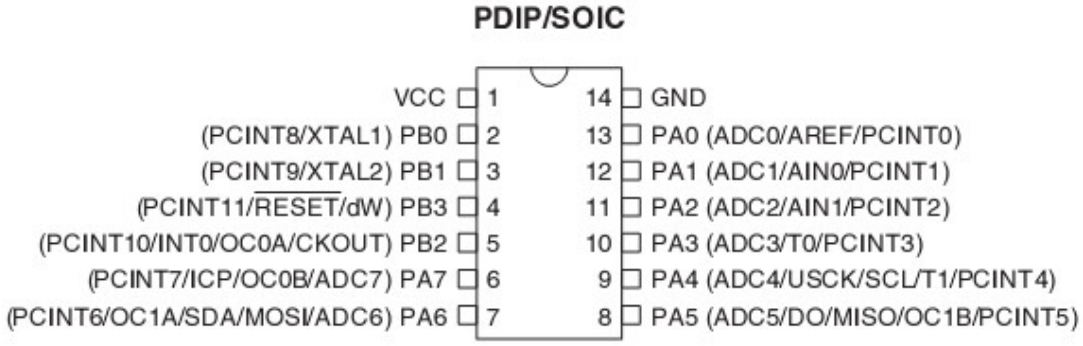
5.4.6 ATTINY22

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.7 ATTINY24

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

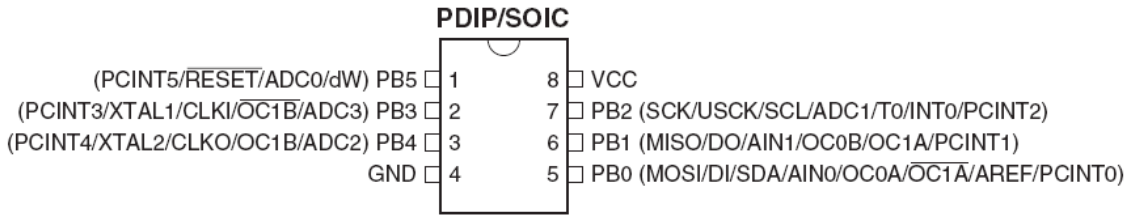
HWMUL=0 ; this chip does not have hardware multiplication

Some users reported that the HWMUL did work. Some batches might support the HW

MUL, but since we found chips that did not, the value is set to 0. You can change it at your own risk.

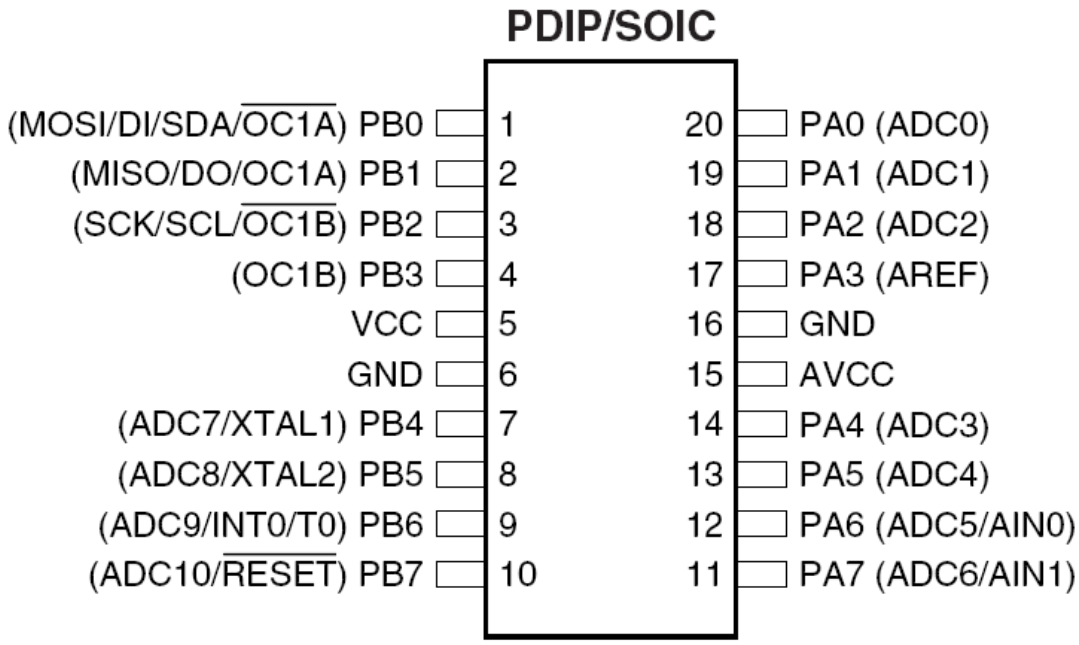
5.4.8 ATTINY25

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

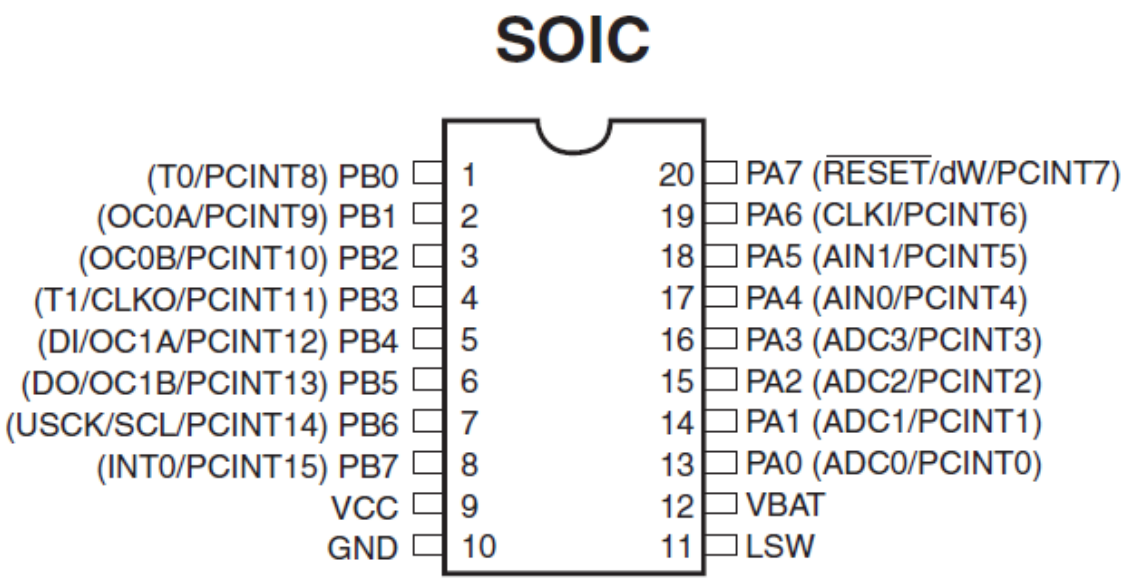


5.4.9 ATTINY26

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

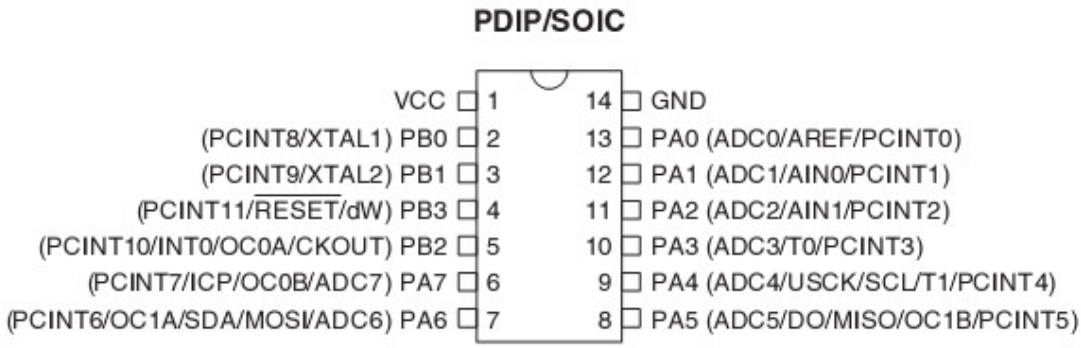


5.4.10 ATTINY43U



5.4.11 ATTINY44

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



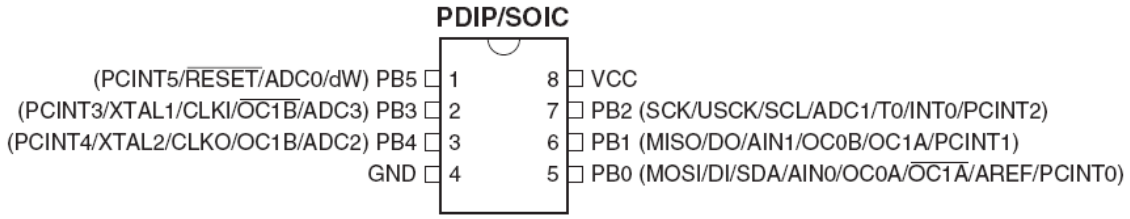
The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

HWMUL=0 ; this chip does not have hardware multiplication

Some users reported that the HWMUL did work. Some batches might support the HW MUL, but since we found chips that did not, the value is set to 0. You can change it at your own risk.

5.4.12 ATTINY45

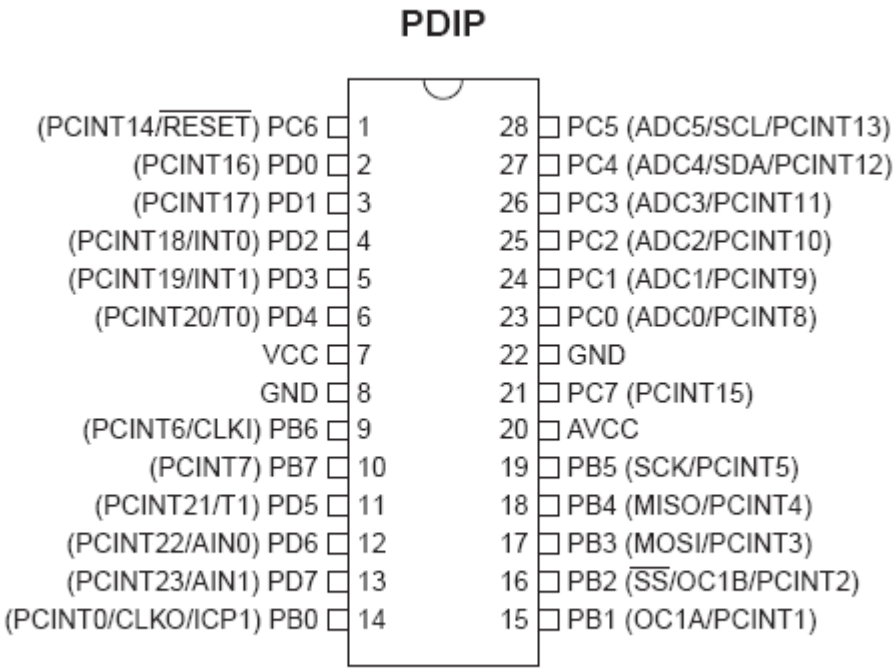
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.13 ATTINY48

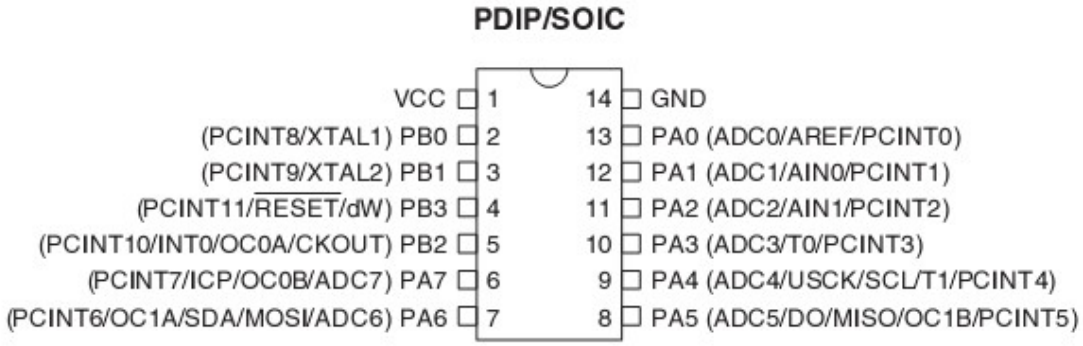
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Notice that the TINY48 is NOT the same as the MEGA48. The TINY48 does not have a UART.



5.4.14 ATTINY84

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



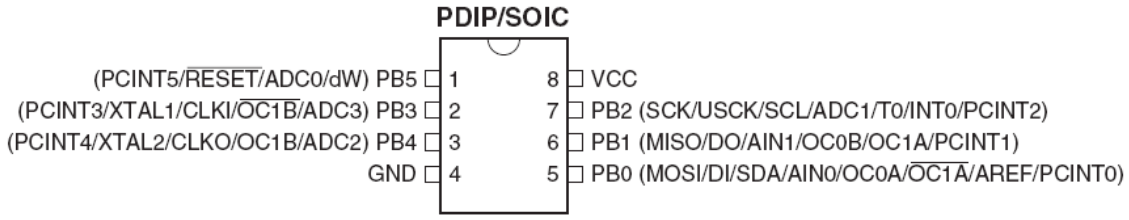
The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

HWMUL=0 ; this chip does not have hardware multiplication

Some users reported that the HWMUL did work. Some batches might support the HW MUL, but since we found chips that did not, the value is set to 0. You can change it at your own risk.

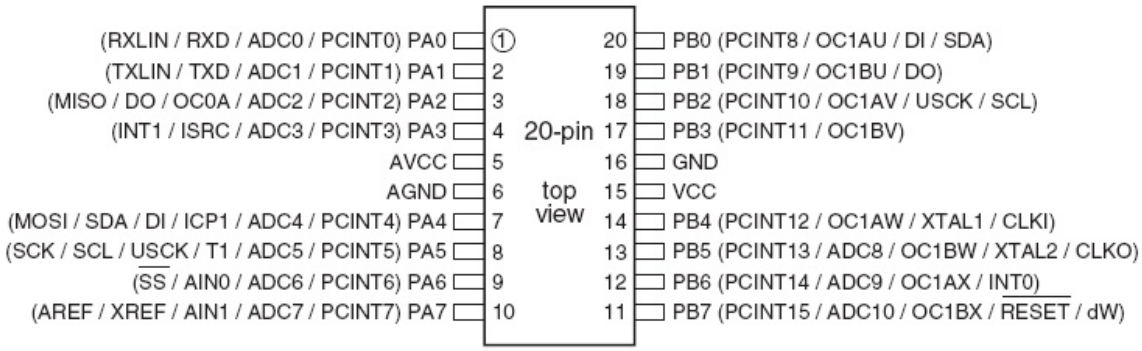
5.4.15 ATTINY85

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.16 ATTINY87

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

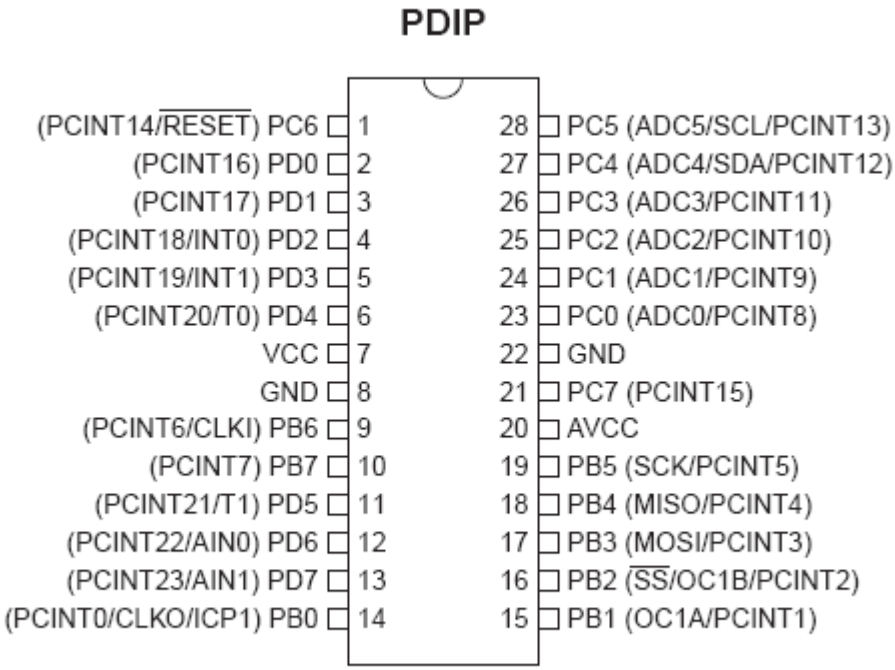


The TINY167/87 have a special LIN/UART. In version 2077 this UART is supported in normal mode. Buffered input/output is not supported.

5.4.17 ATTINY88

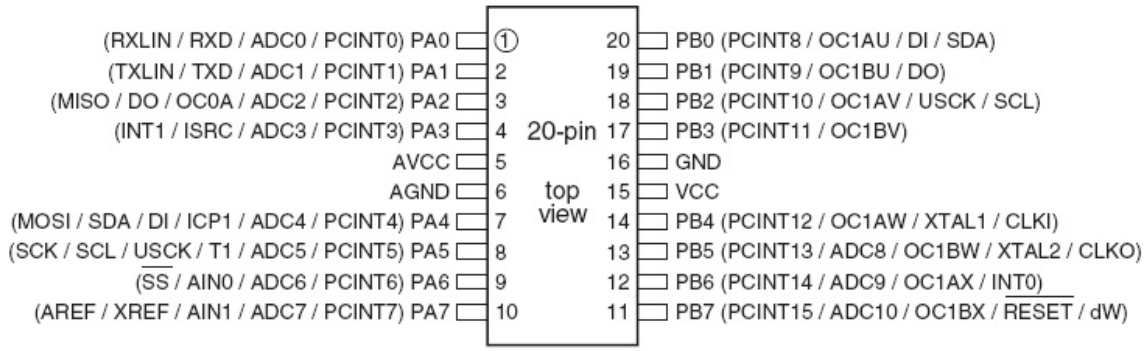
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Notice that the TINY88 is NOT the same as the MEGA88. The TINY88 does not have a UART.



5.4.18 ATTINY167

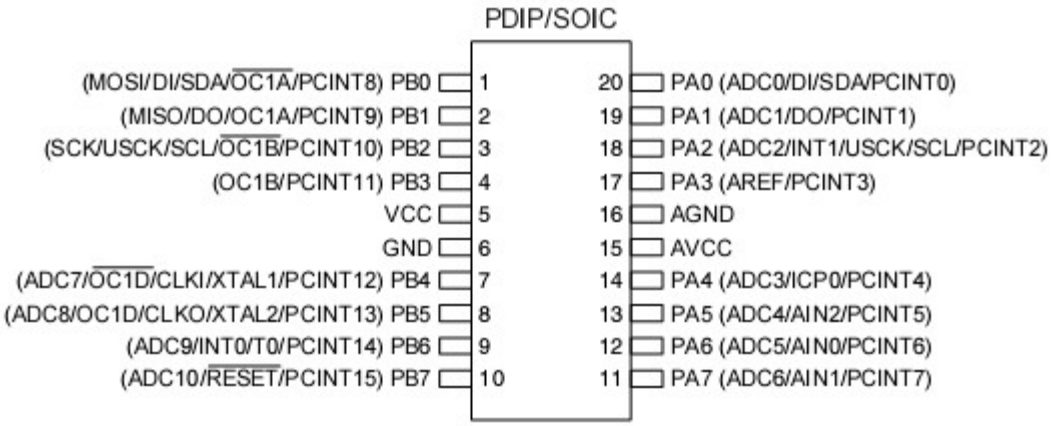
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The TINY167/87 have a special LIN/UART. In version 2077 this UART is supported in normal mode. Buffered input/output is not supported.

5.4.19 ATTINY261

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.20 ATTINY441

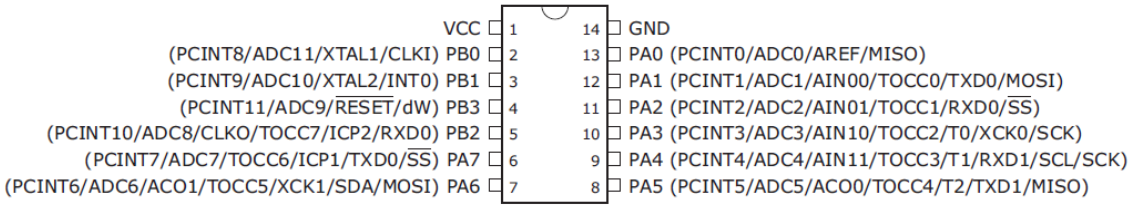
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

The Tiny441 has 256 bytes of EEPROM memory.
Usually this means that the high byte EEPROM address register is missing since it has no purpose.
The tiny441 however has an EEARH register. And it seems to cause problems.
The data sheet says :

Devices with 256 bytes of EEPROM, or less, do not require a high address registers (EEARH). In such devices the high
address register is therefore left out but, for compatibility issues, the remaining register is still referred to as the low byte
of the EEPROM address register (EEARL).
Devices that to do not fill an entire address byte, i.e. devices with an EEPROM size not equal to 256, implement readonly
bits in the unused locations. Unused bits are located in the most significant end of the address register and they
always read zero.

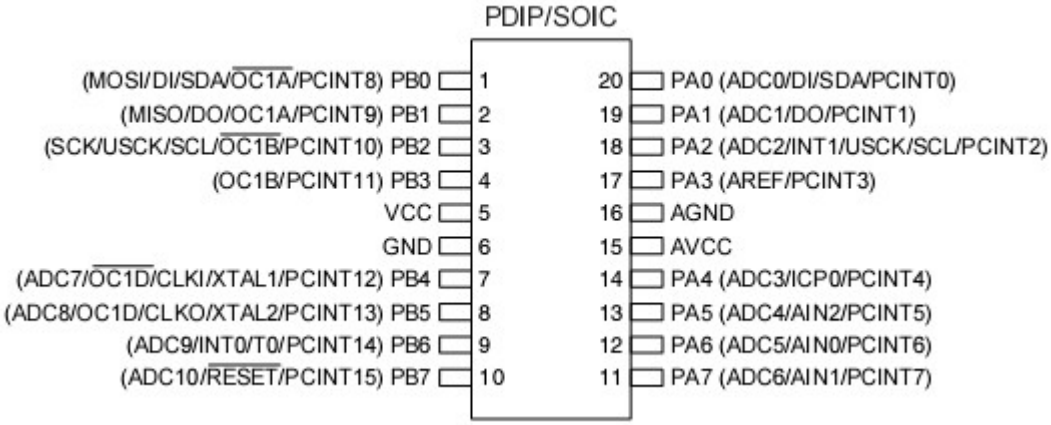
The bascom write/read EEPROM code checks if the EEPROM size > 256. If that is the case, the EEARH is addressed.
But in this case this register is not touched.
When you have problems, set EEARH register to 0 in your code.
While we could always write this register, it is a waste of code.
When having problems contact support.

Figure 1-1. Pinout in 14-pin SOIC.



5.4.21 ATTINY461

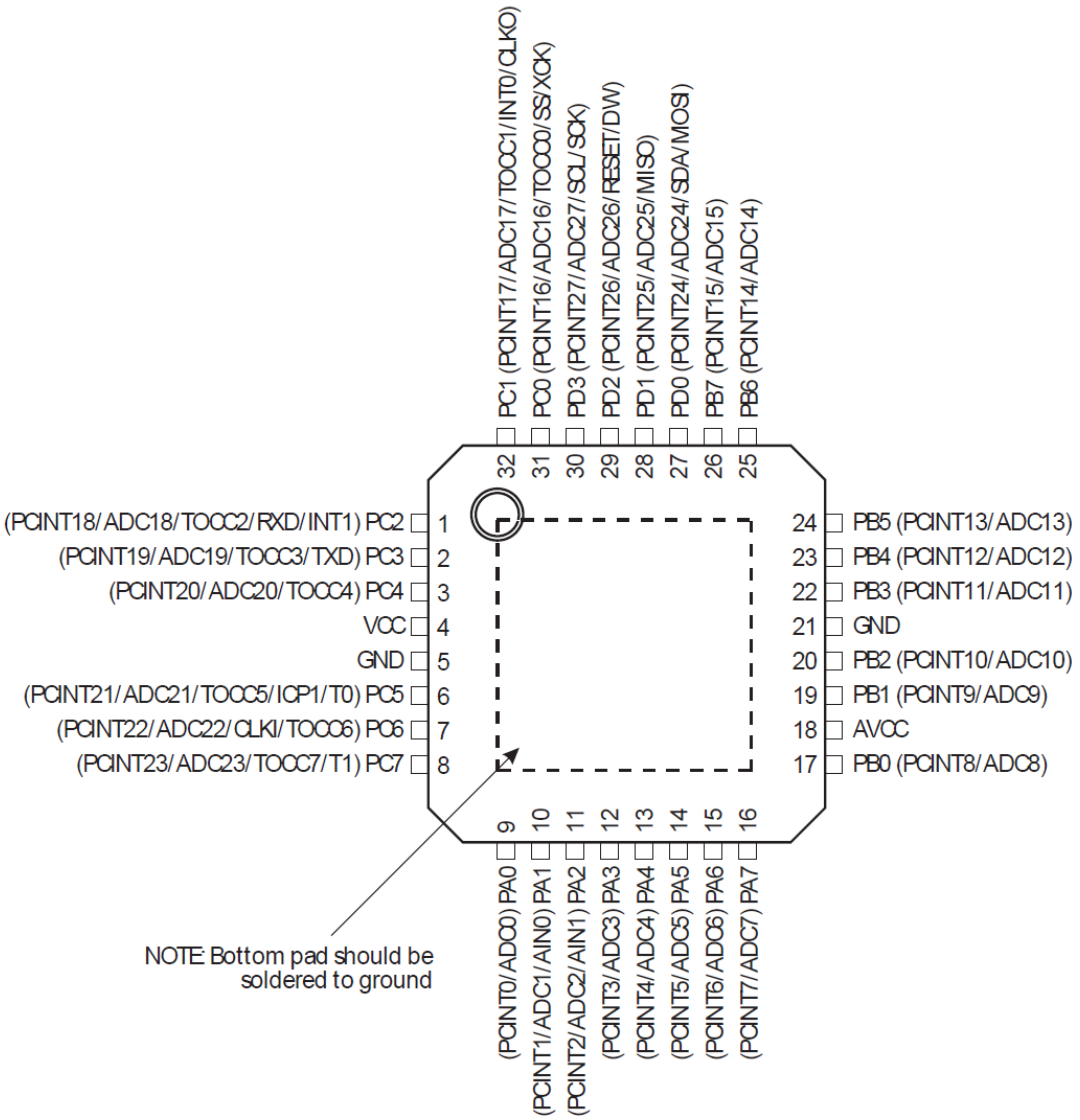
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The processor has only one PCINT interrupt. But there are two PCINT interrupt masks to serve all the PCINTx pins. Most processors have their own interrupt for each PCINT mask register so you have better control over the different pins which caused the interrupt.
Since there is only one interrupt, the ENABLE 1250 and DISABLE 1240 statements, set/reset both the PCIE0 and PCIE1 flags in the GIMSK register. You still have to set the PCMSK0 and PCMSK1 registers to specify which bits can cause a PCINT interrupt.

5.4.22 ATTINY828

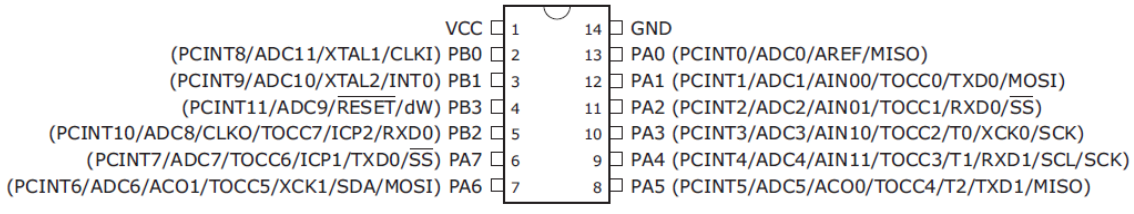
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.23 ATTINY841

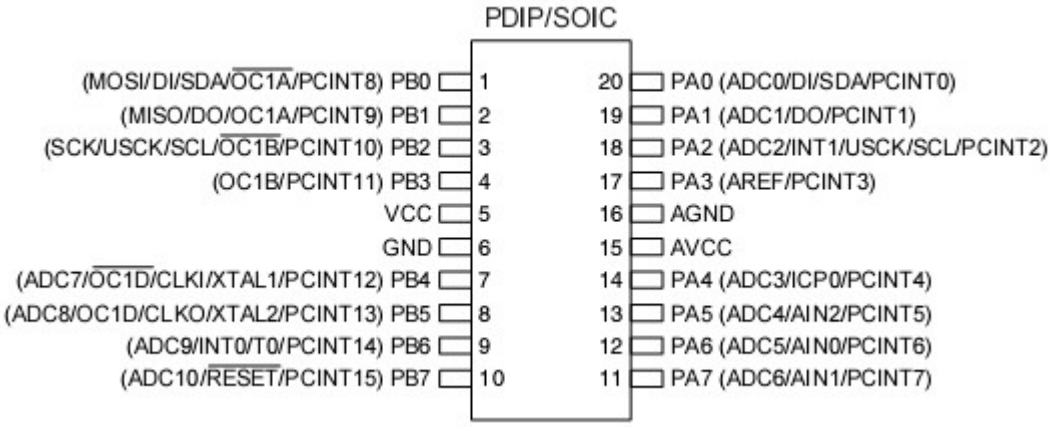
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 1-1. Pinout in 14-pin SOIC.



5.4.24 ATTINY861

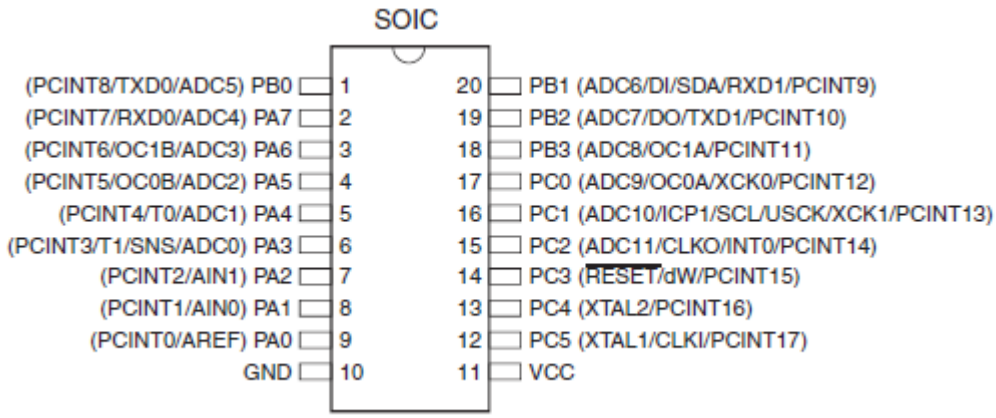
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The processor has only one PCINT interrupt. But there are two PCINT interrupt masks to serve all the PCINTx pins. Most processors have their own interrupt for each PCINT mask register so you have better control over the different pins which caused the interrupt.
Since there is only one interrupt, the ENABLE 1250 and DISABLE 1240 statements, set/reset both the PCIE0 and PCIE1 flags in the GIMSK register. You still have to set the PCMSK0 and PCMSK1 registers to specify which bits can cause a PCINT interrupt.

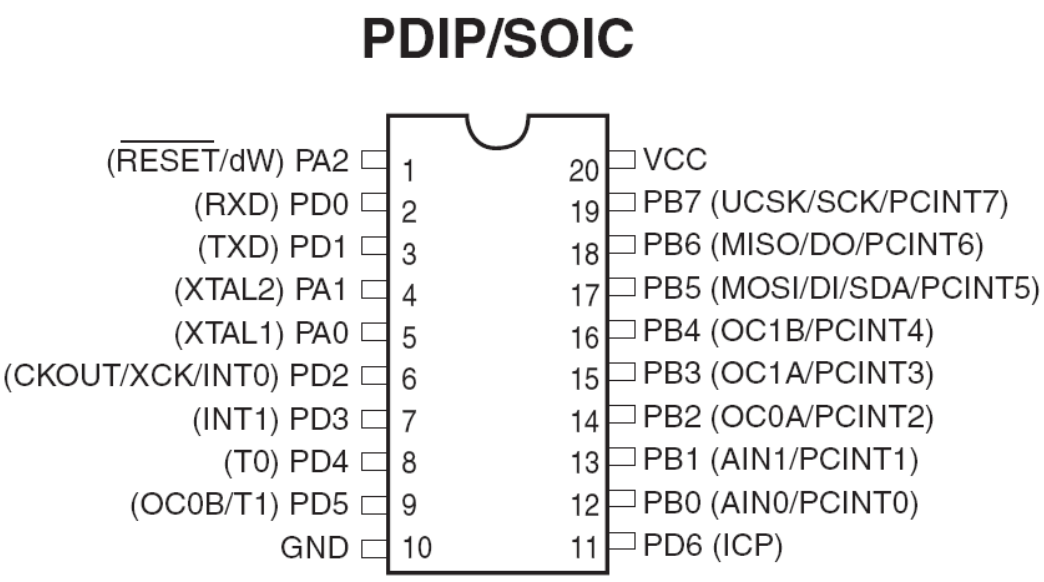
5.4.25 ATTINY1634

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.26 ATTINY2313

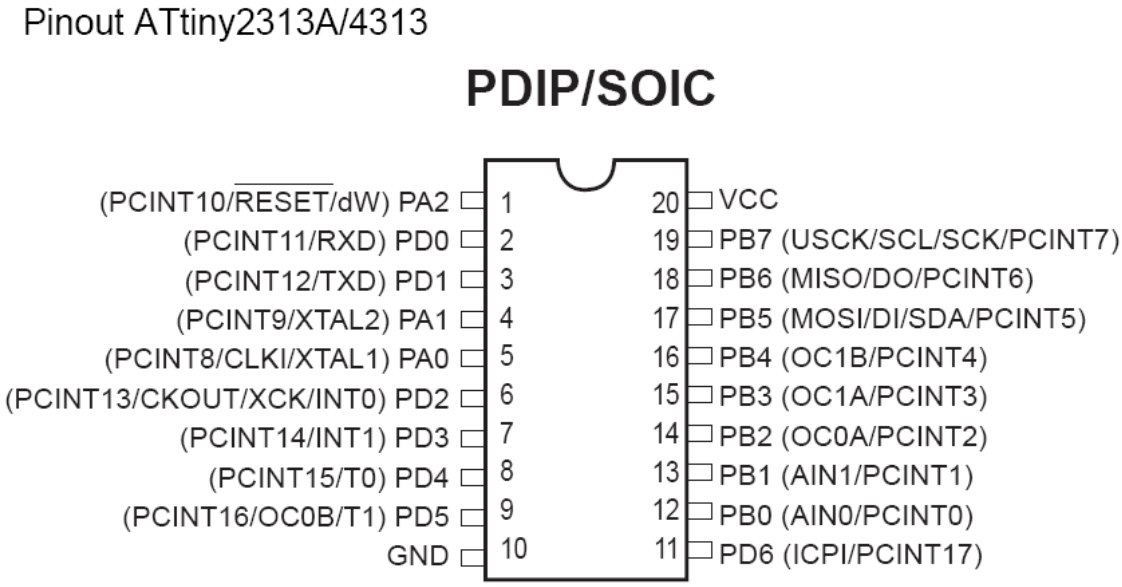
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The tiny2313 has an internal oscillator that can run at various frequencies. The 4 MHz seems not to work precise. when using the UART for serial communication you can get wrong output. You can best use the 8 MHz internal oscillator , or tweak the UBRR register. For example, UBRR=UBRR+1
That worked for 4 Mhz, at 19200 baud.

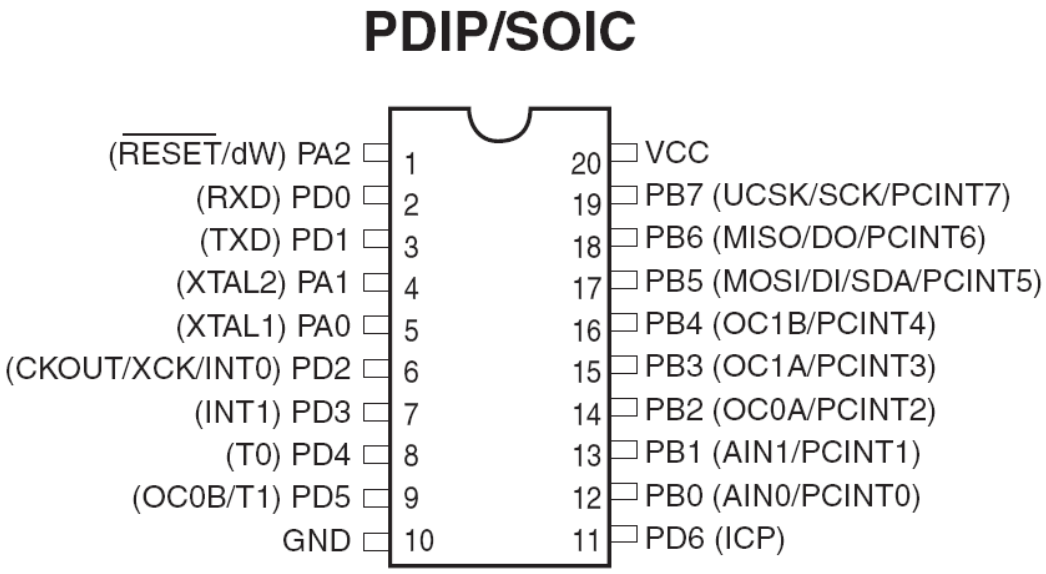
5.4.27 ATTINY2313A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.4.28 ATTINY4313

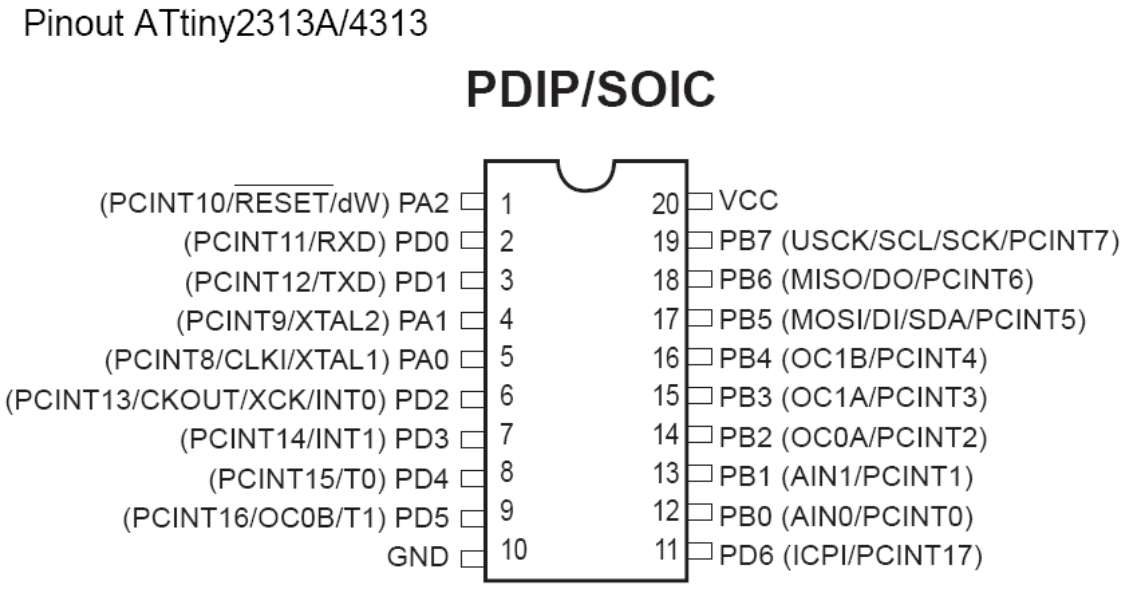
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The tiny4313 has an internal oscillator that can run at various frequencies. The 4 MHz seems not to work precise. when using the UART for serial communication you can get wrong output. You can best use the 8 MHz internal oscillator , or tweak the UBRR register. For example, UBRR=UBRR+1
That worked for 4 Mhz, at 19200 baud.

5.4.29 ATTINY4313A

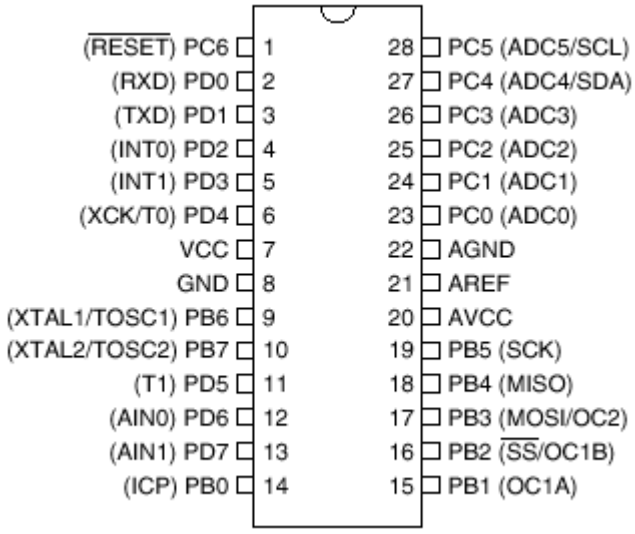
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5 ATMEGA

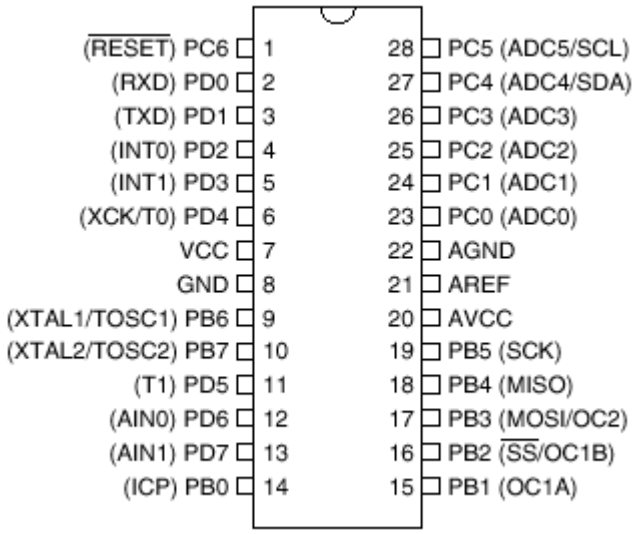
5.5.1 ATMEGA8

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



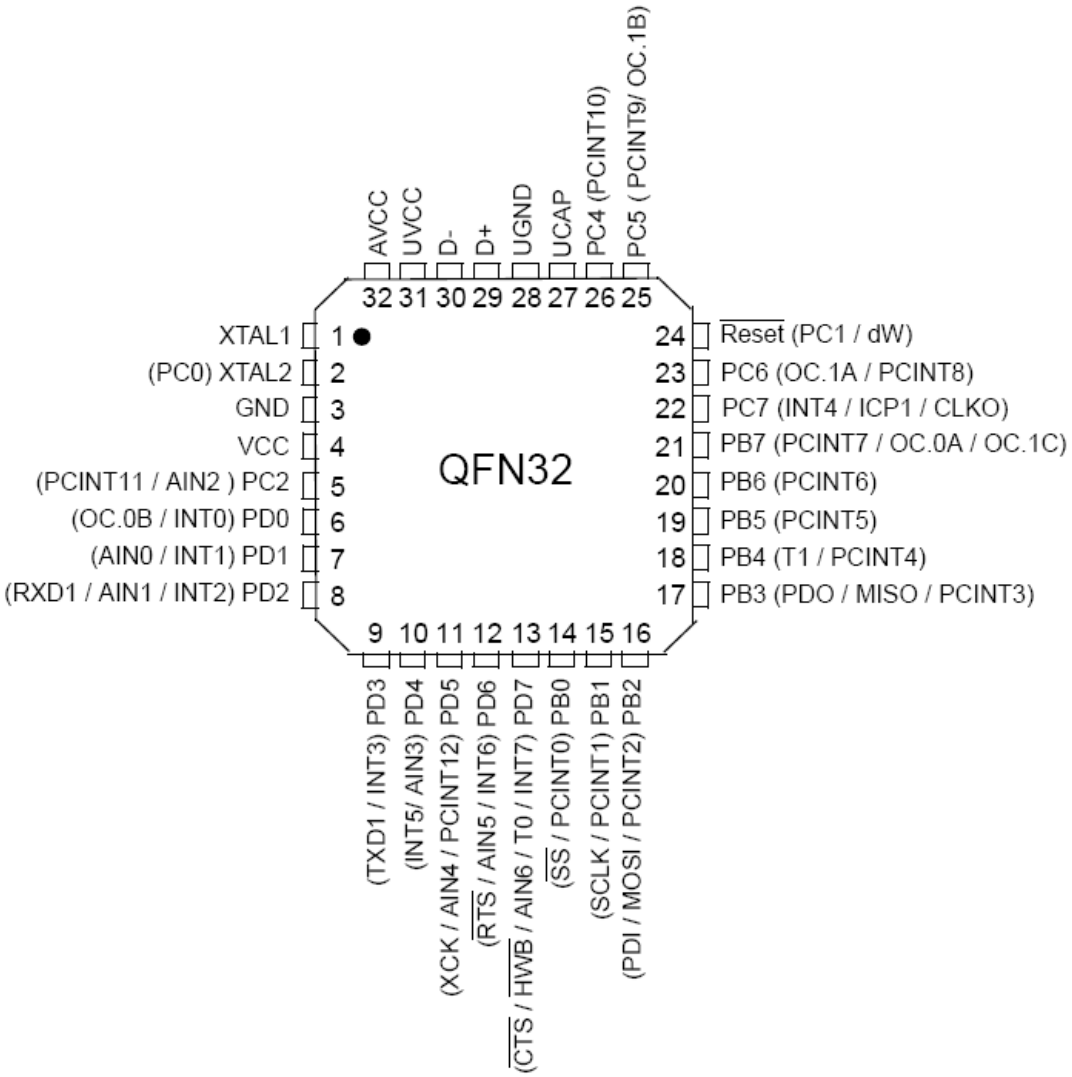
5.5.2 ATMEGA8A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



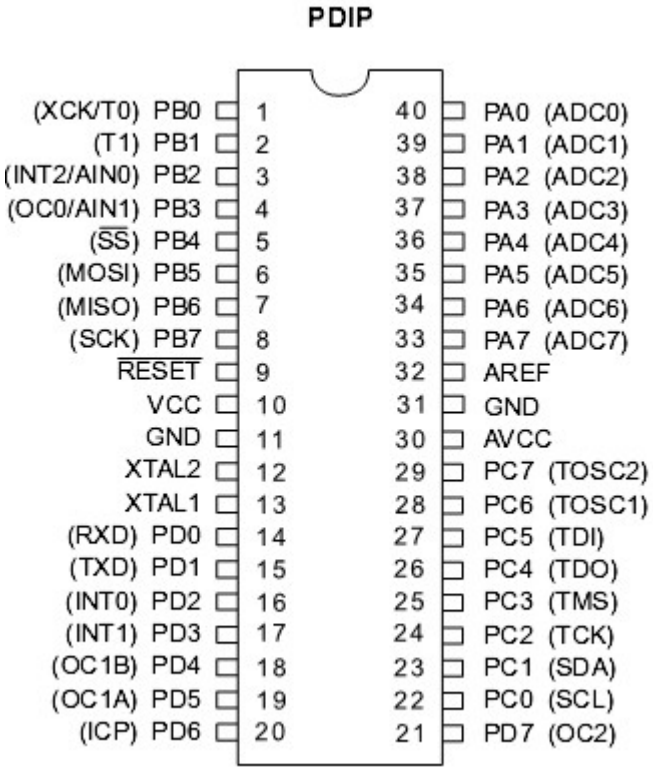
5.5.3 ATMEGA8U2

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



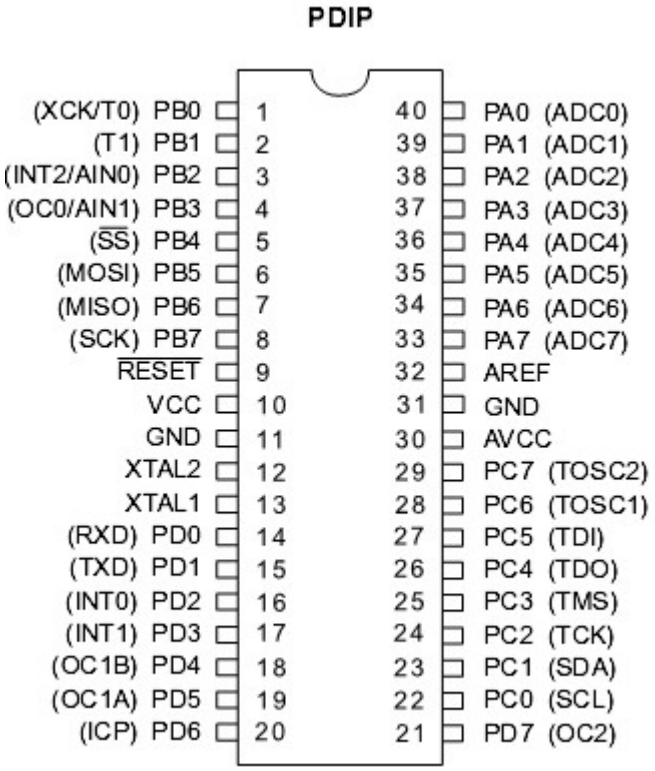
5.5.4 ATMEGA16

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



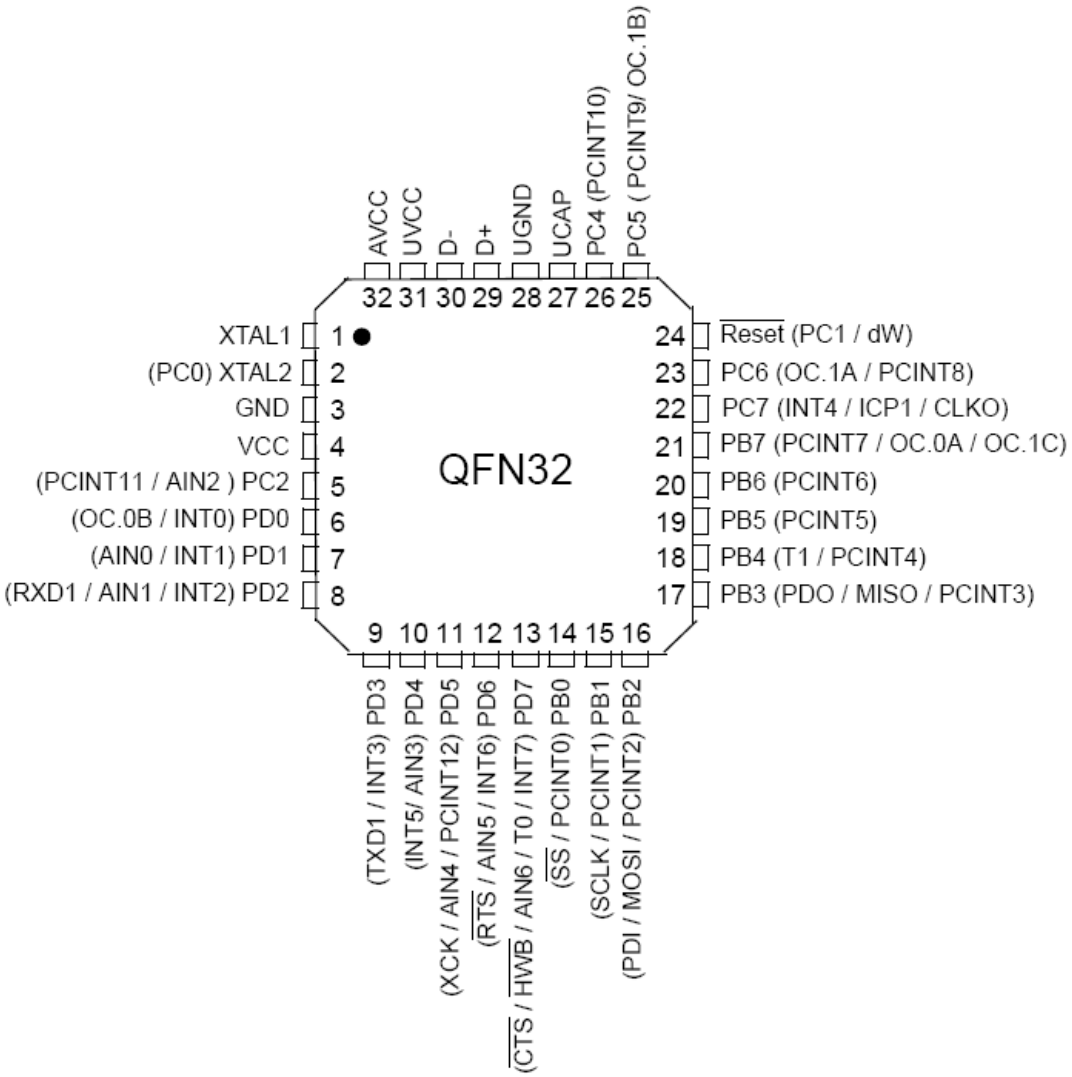
5.5.5 ATMEGA16A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



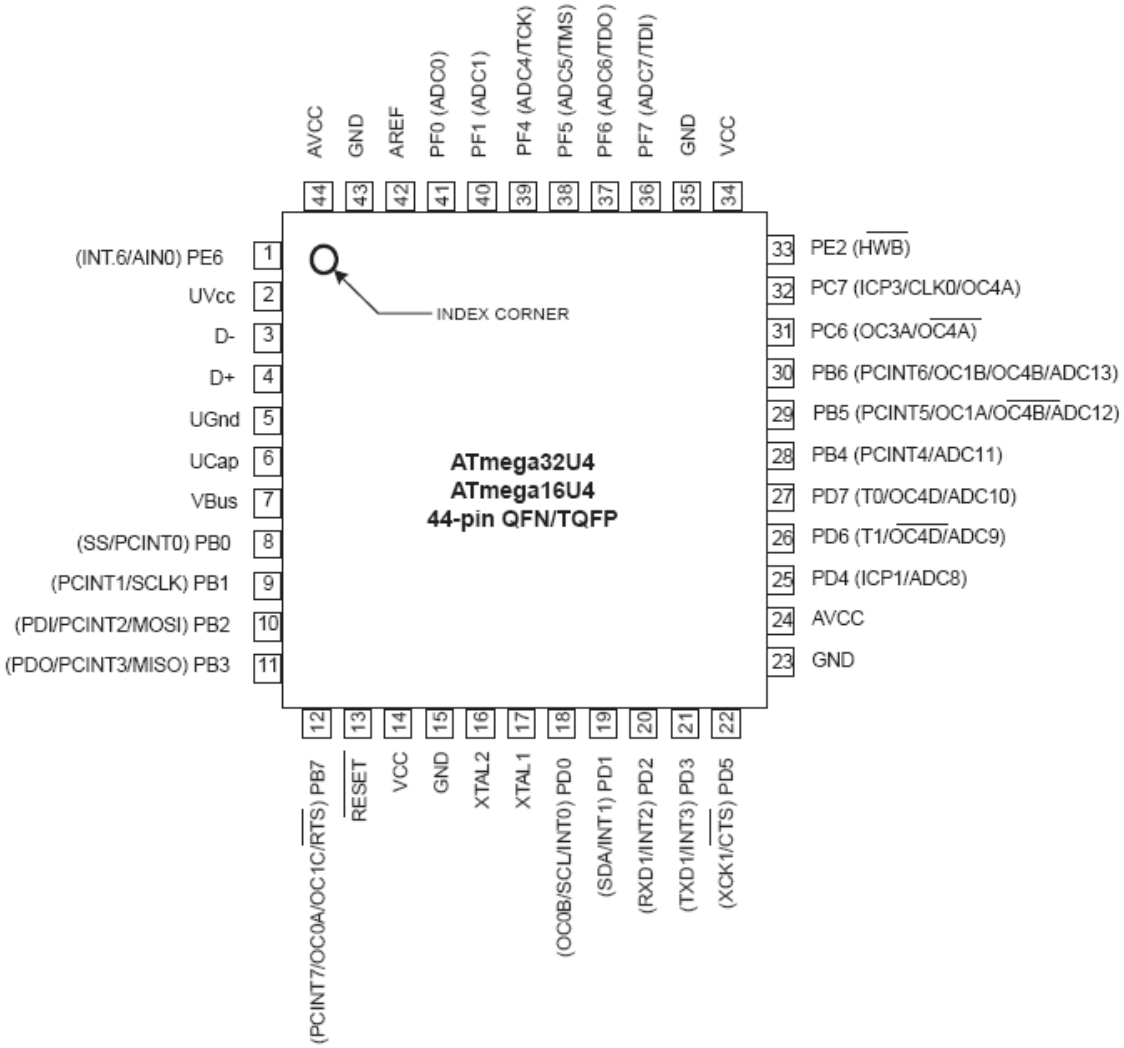
5.5.6 ATMEGA16U2

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



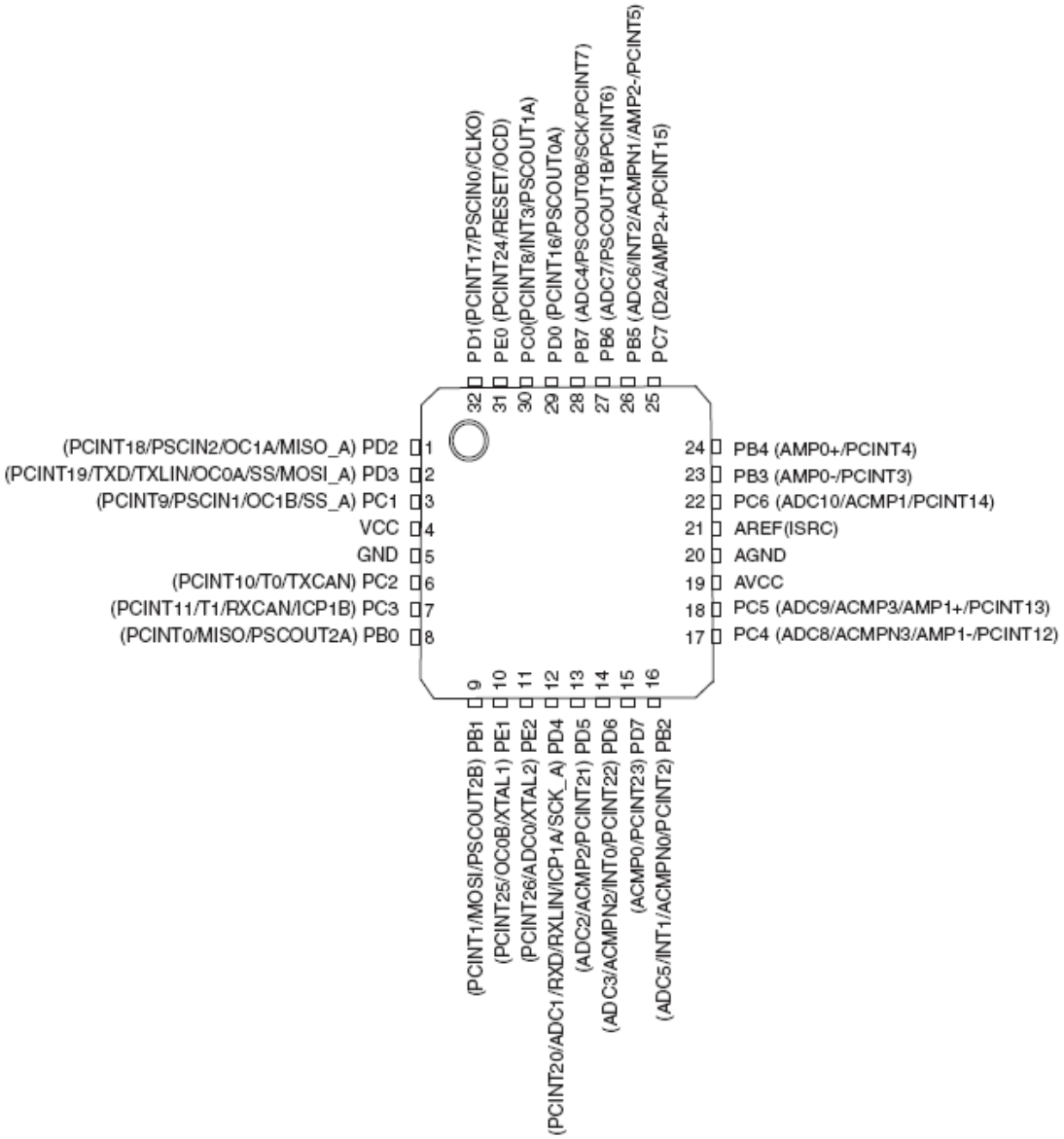
5.5.7 ATMEGA16U4

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



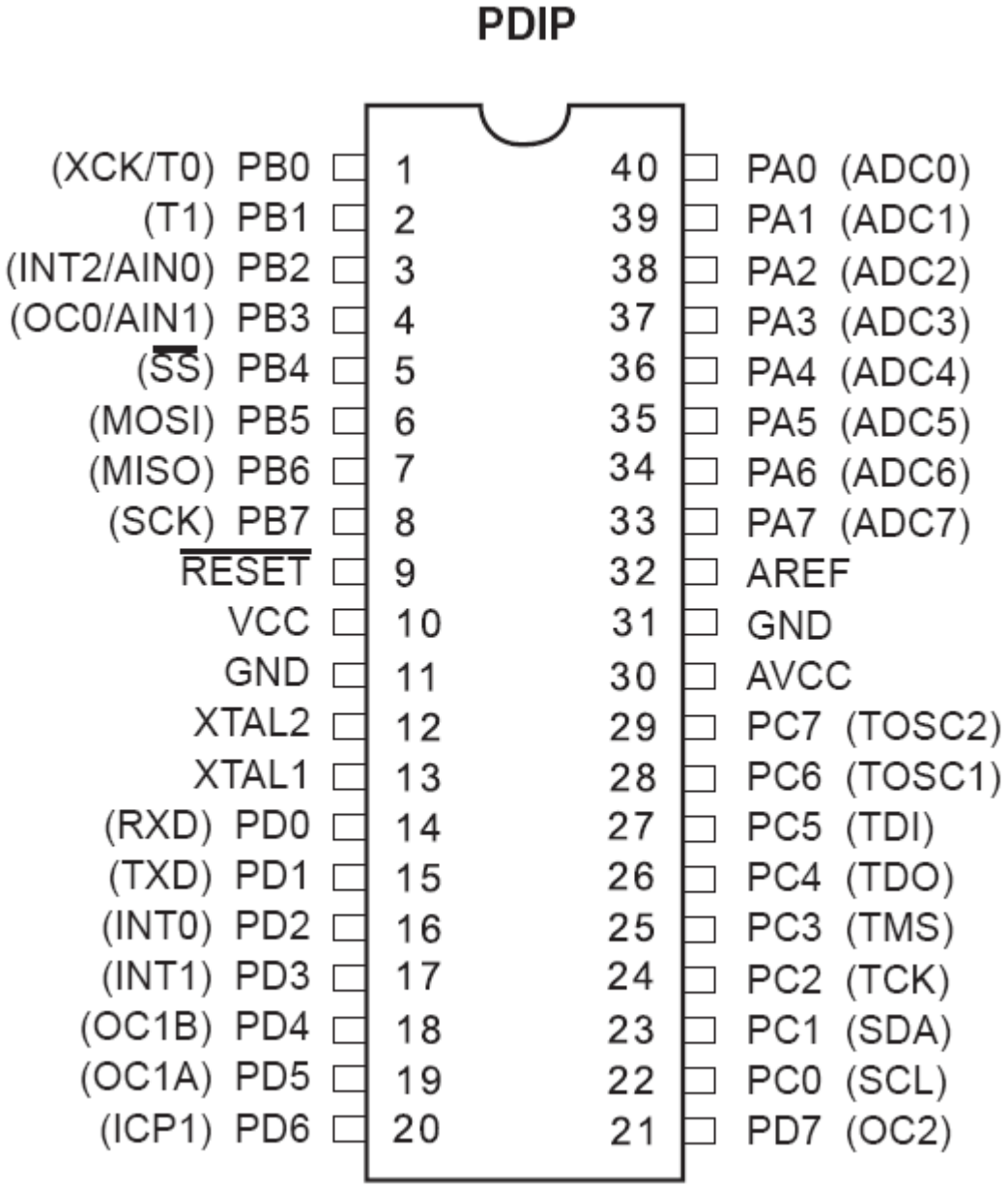
5.5.8 ATMEGA16M1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



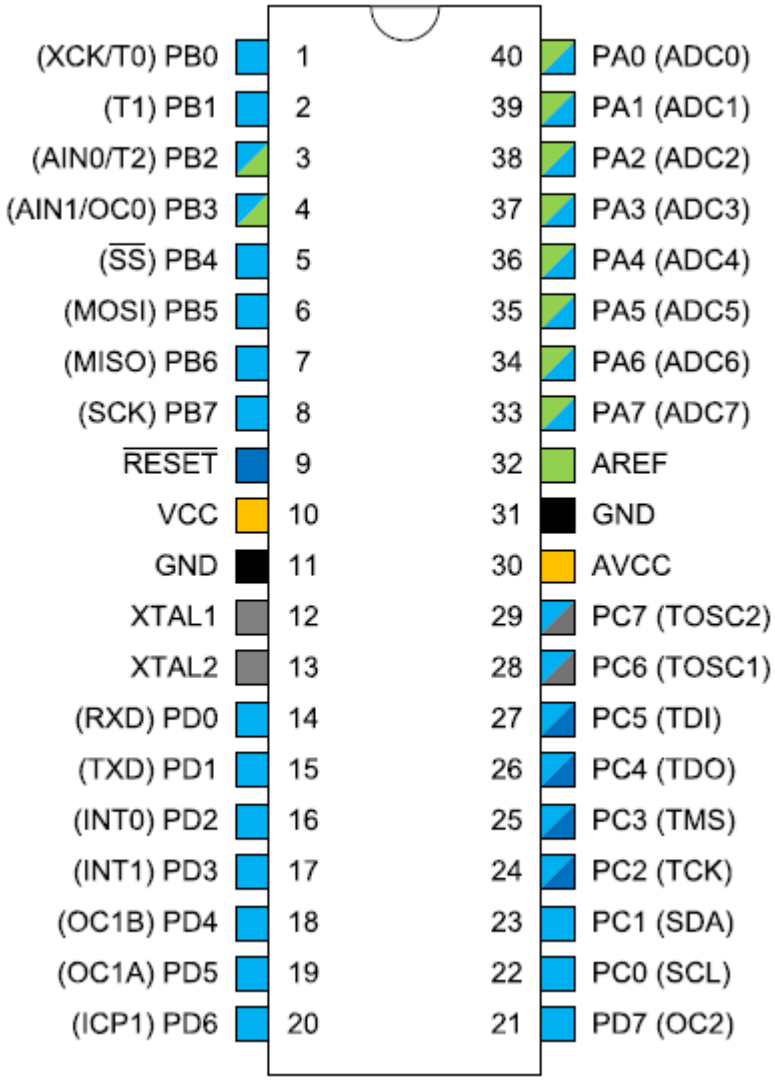
5.5.9 ATMEGA32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



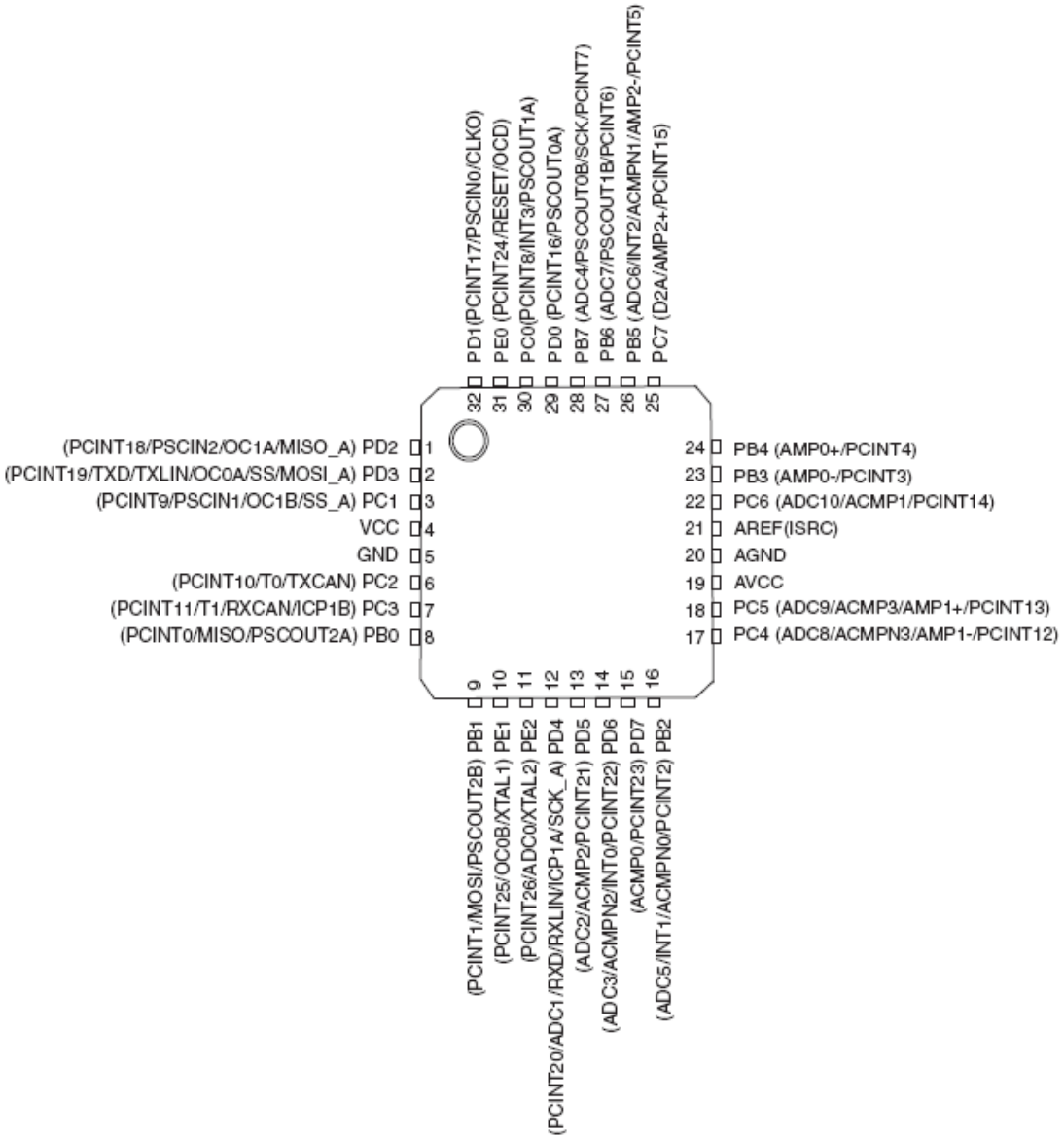
5.5.10 ATMEGA32A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



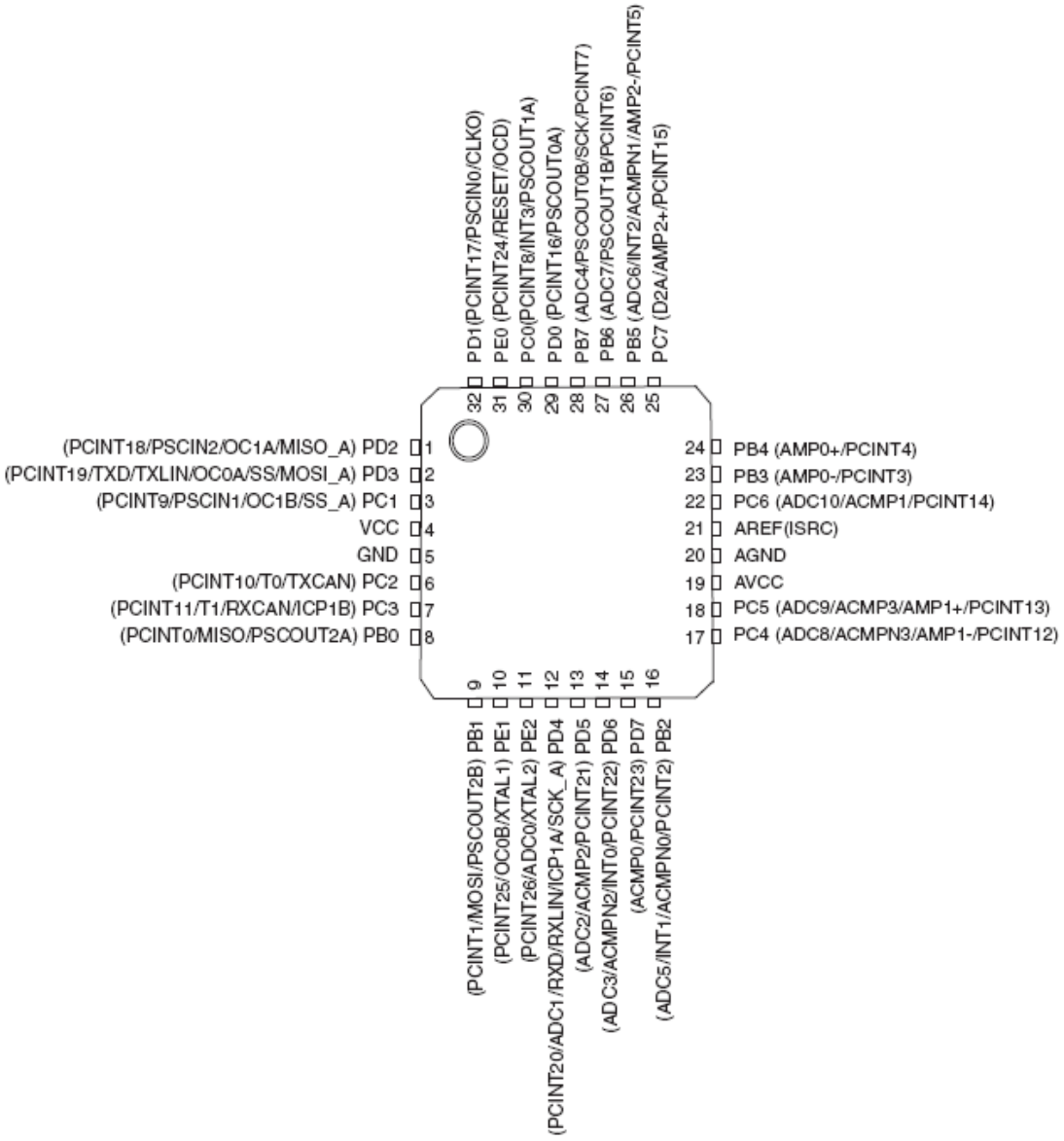
5.5.11 ATMEGA32C1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.12 ATMEGA32M1

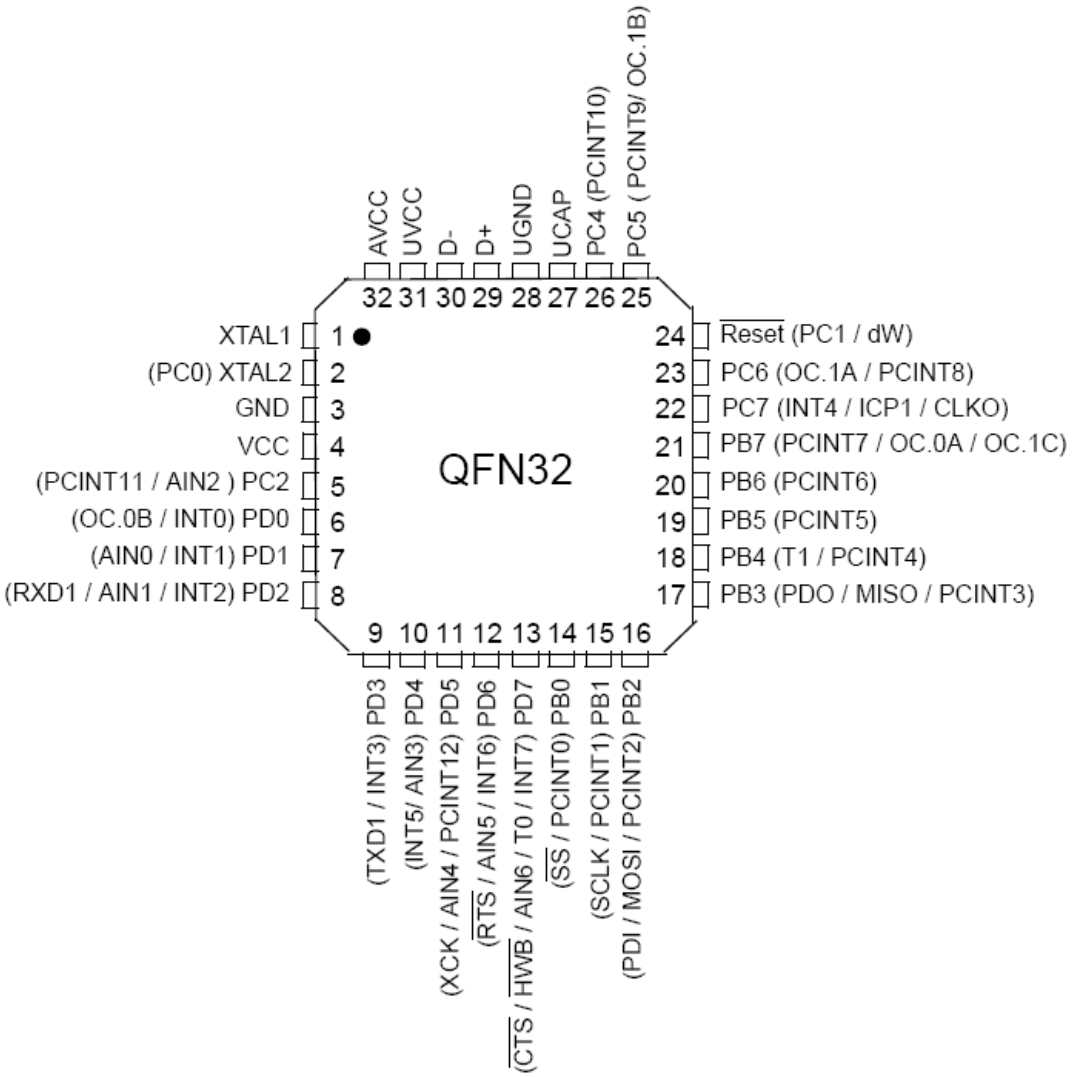
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



- FOR ISP programming, notice that pin PD.2, PD.3 and PD4 are used.

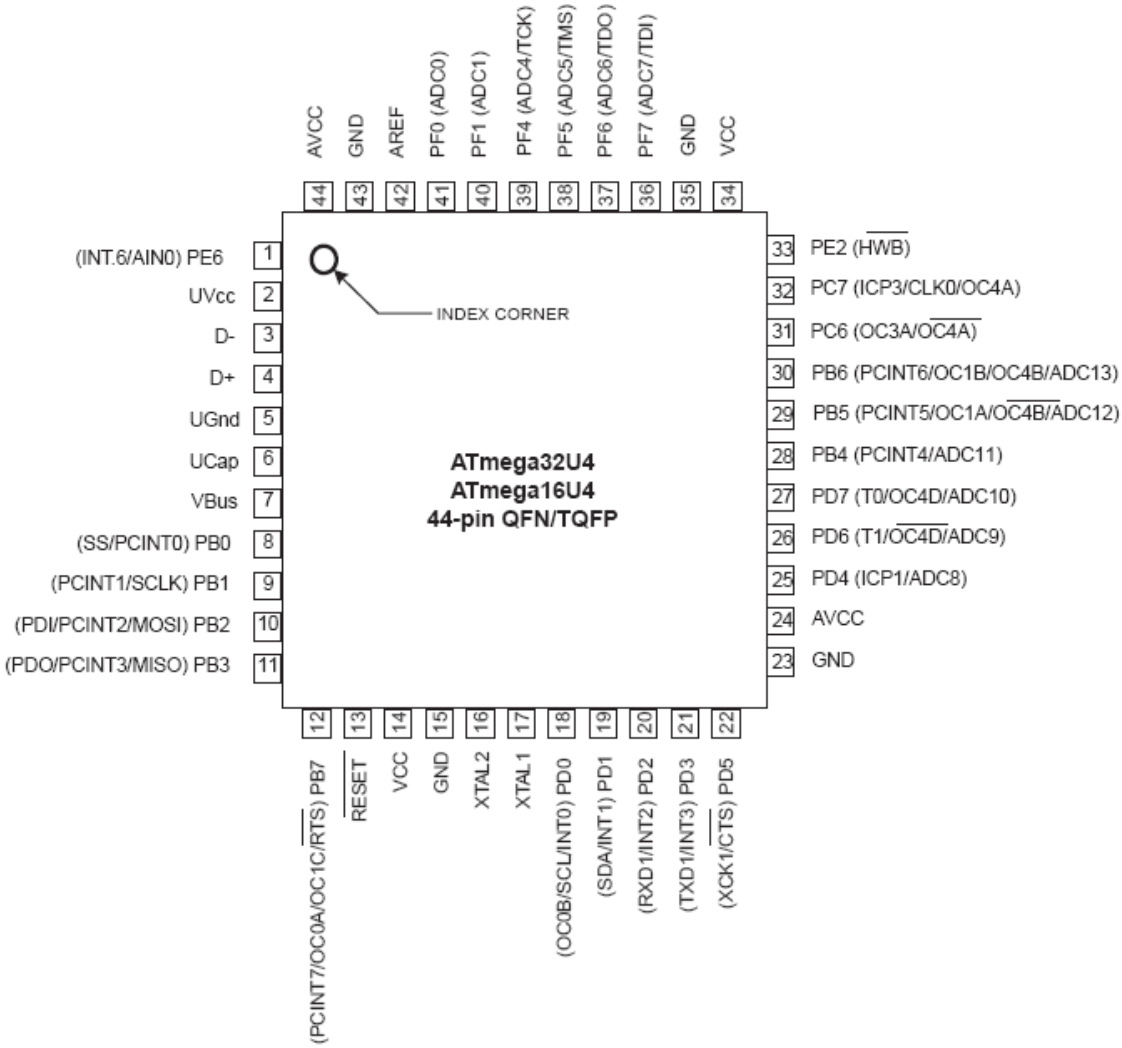
5.5.13 ATMEGA32U2

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



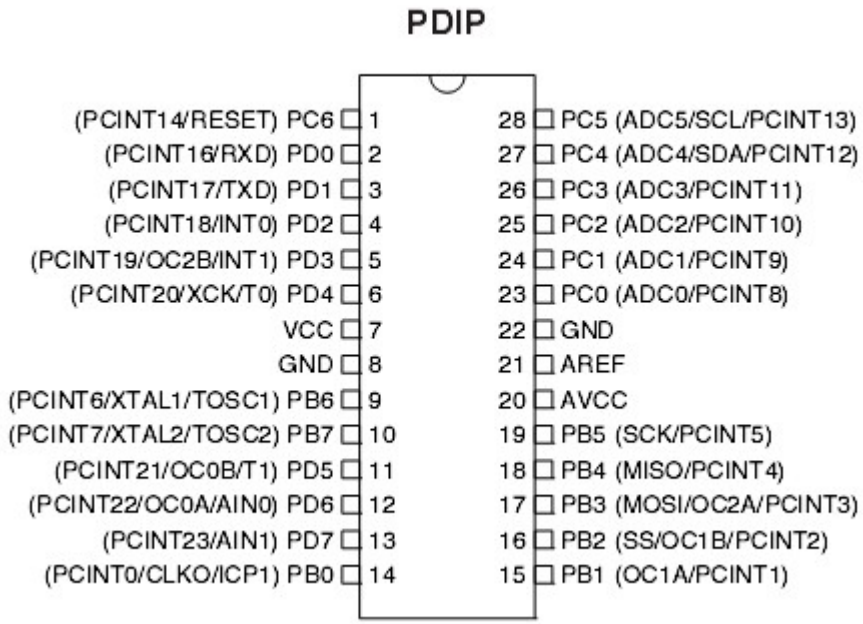
5.5.14 ATMEGA32U4

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



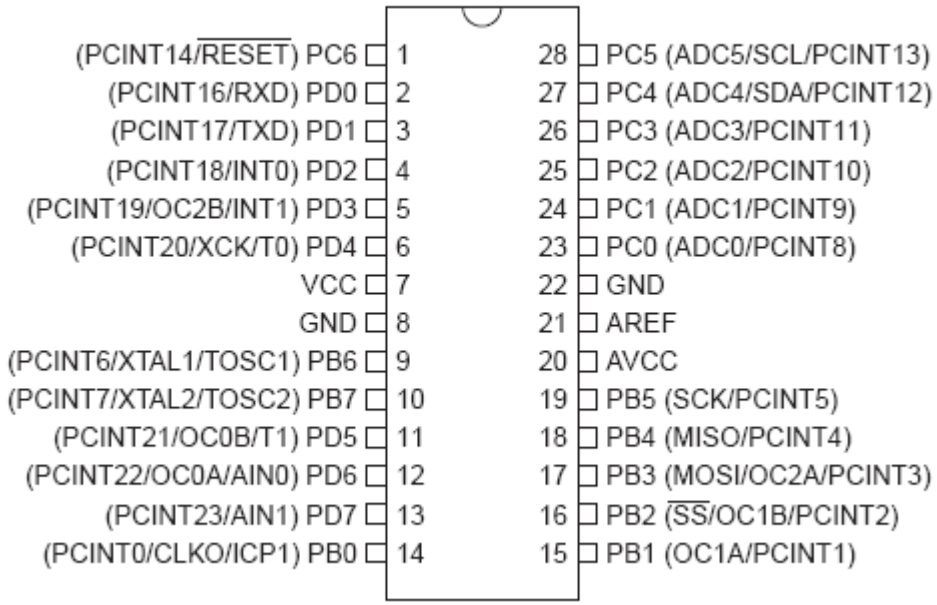
5.5.15 ATMEGA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



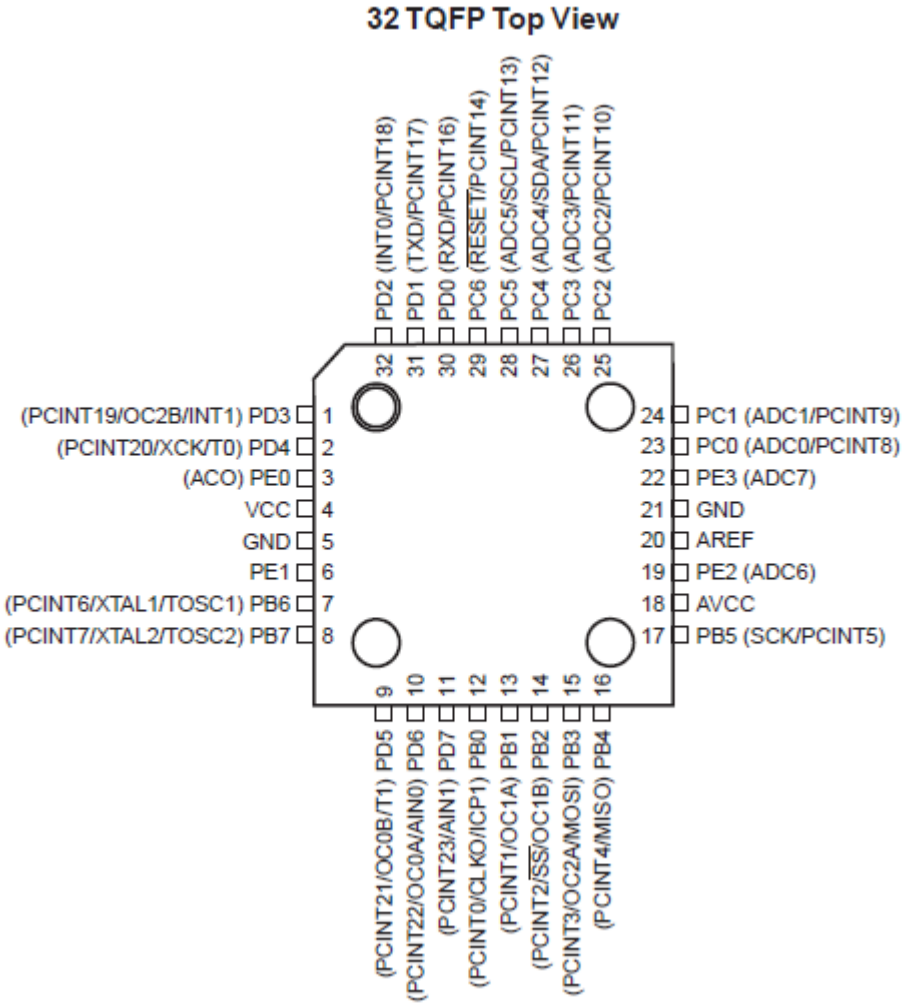
5.5.16 ATMEGA48P-ATMEGA48PA

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.17 ATMEGA48PB

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

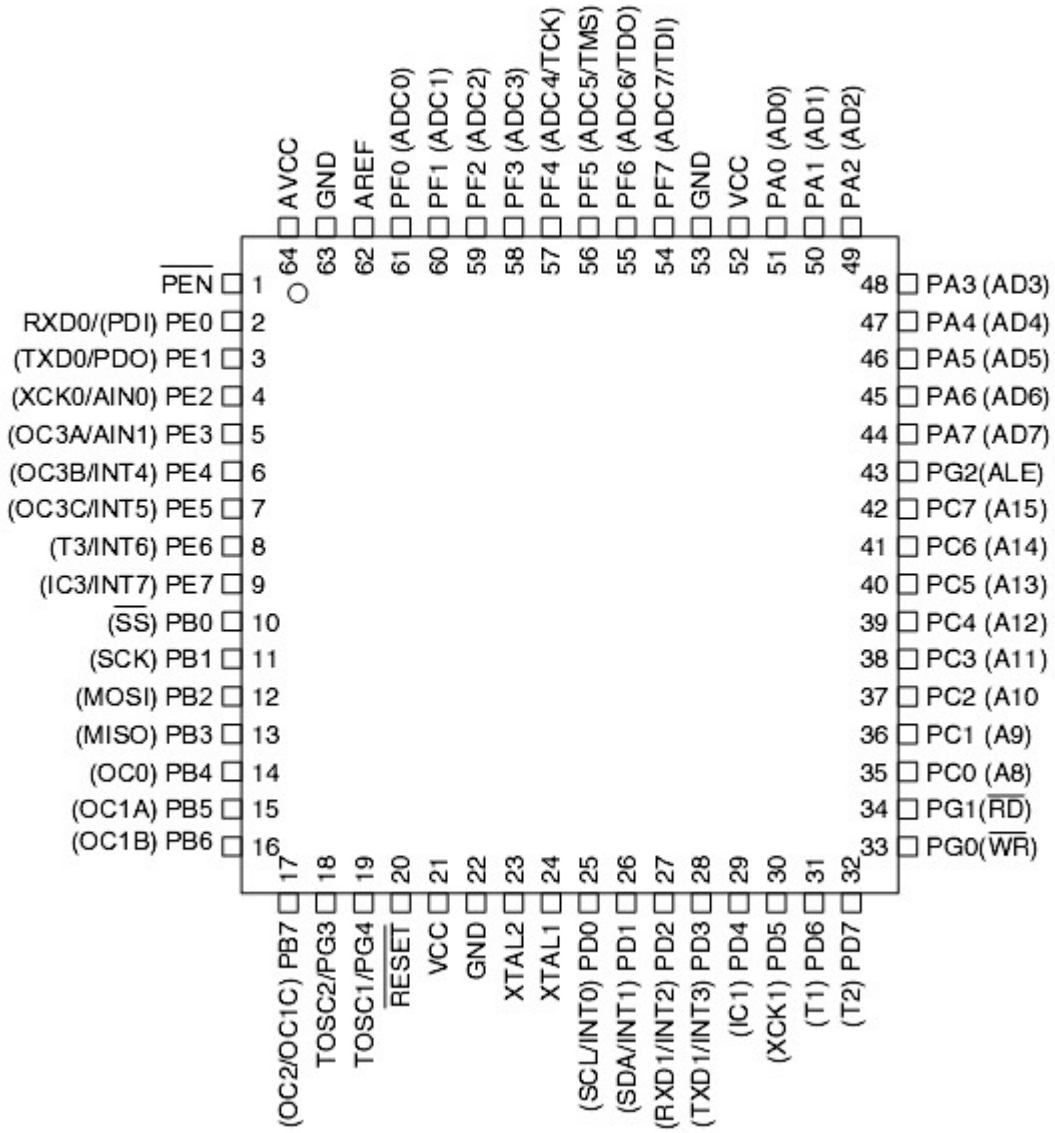


NO DIP version available

5.5.18 ATMEGA64

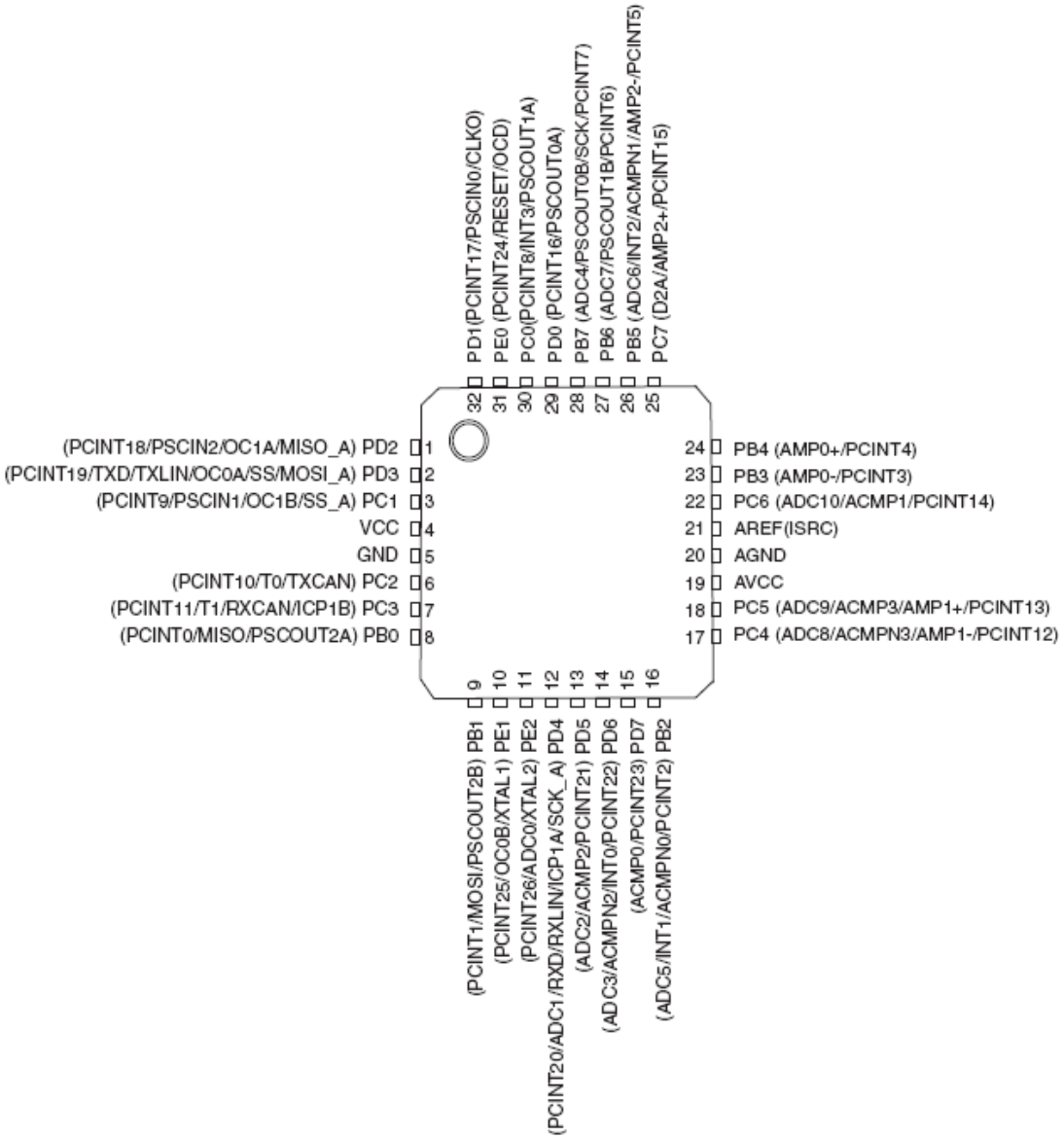
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 1. Pinout ATmega64



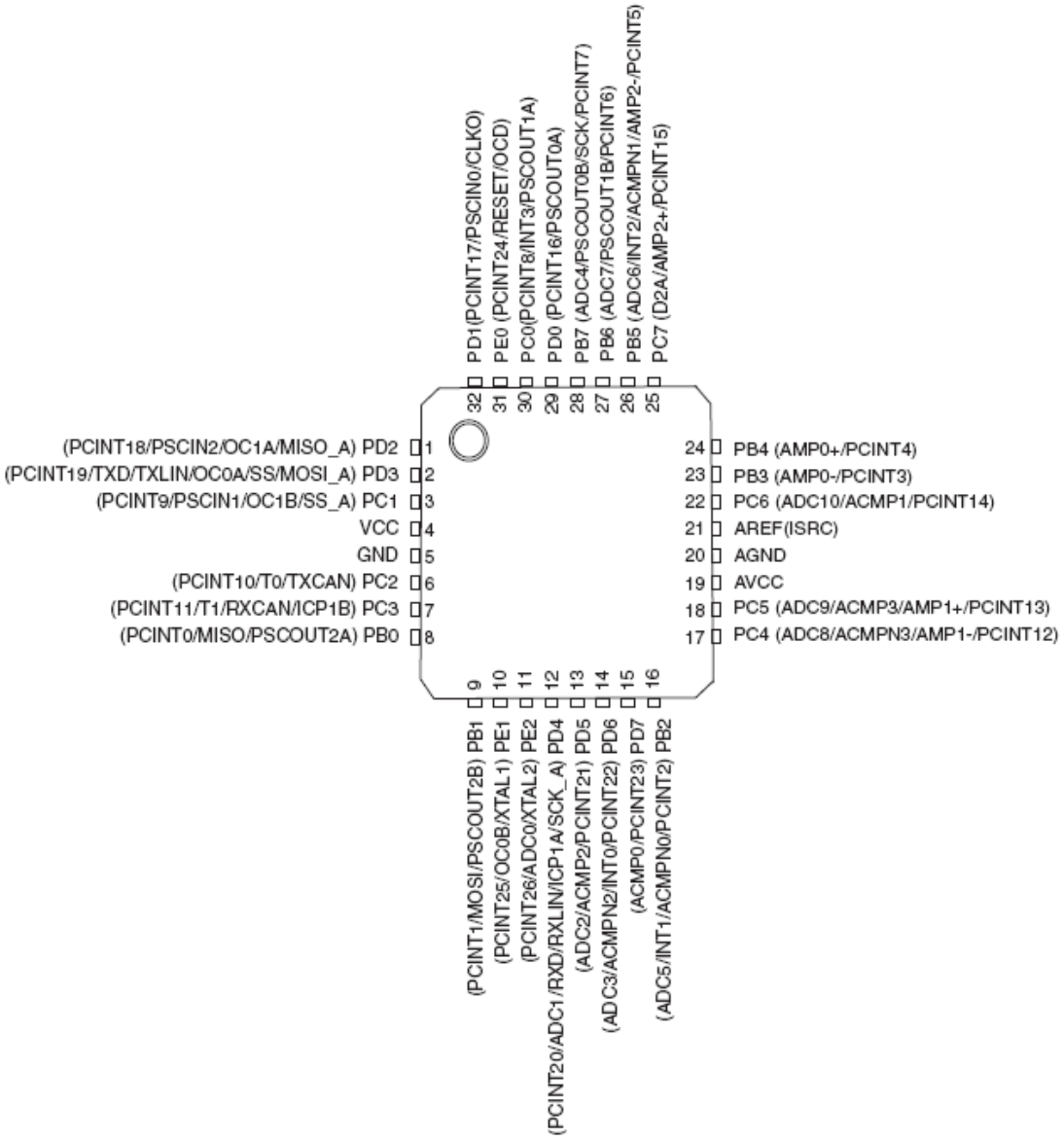
5.5.19 ATMEGA64C1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



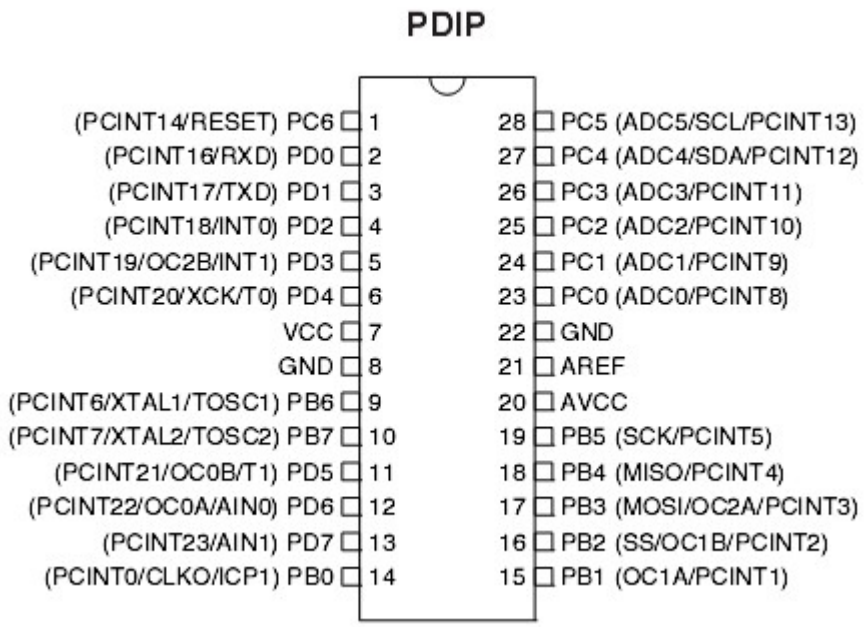
5.5.20 ATMEGA64M1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



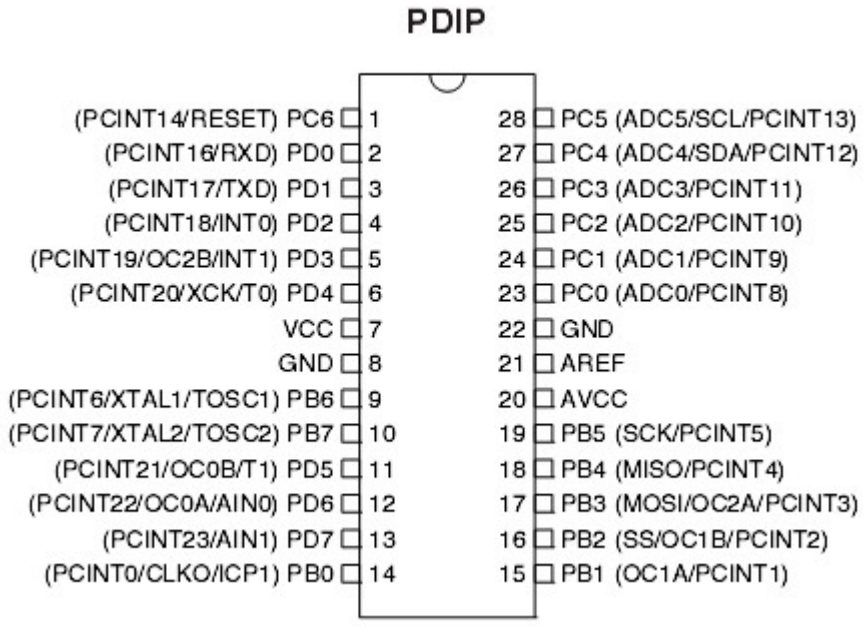
5.5.21 ATMEGA88

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



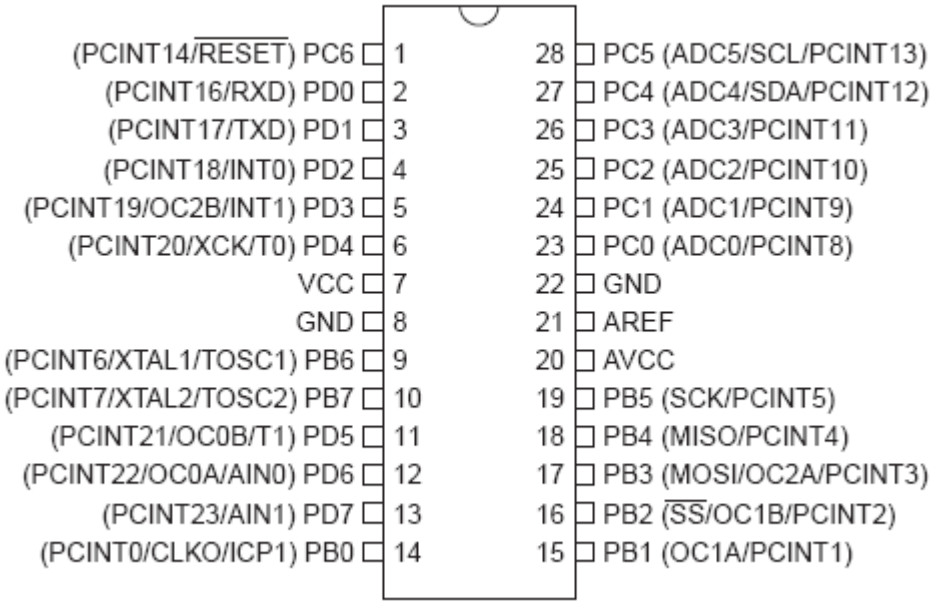
5.5.22 ATMEGA88A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



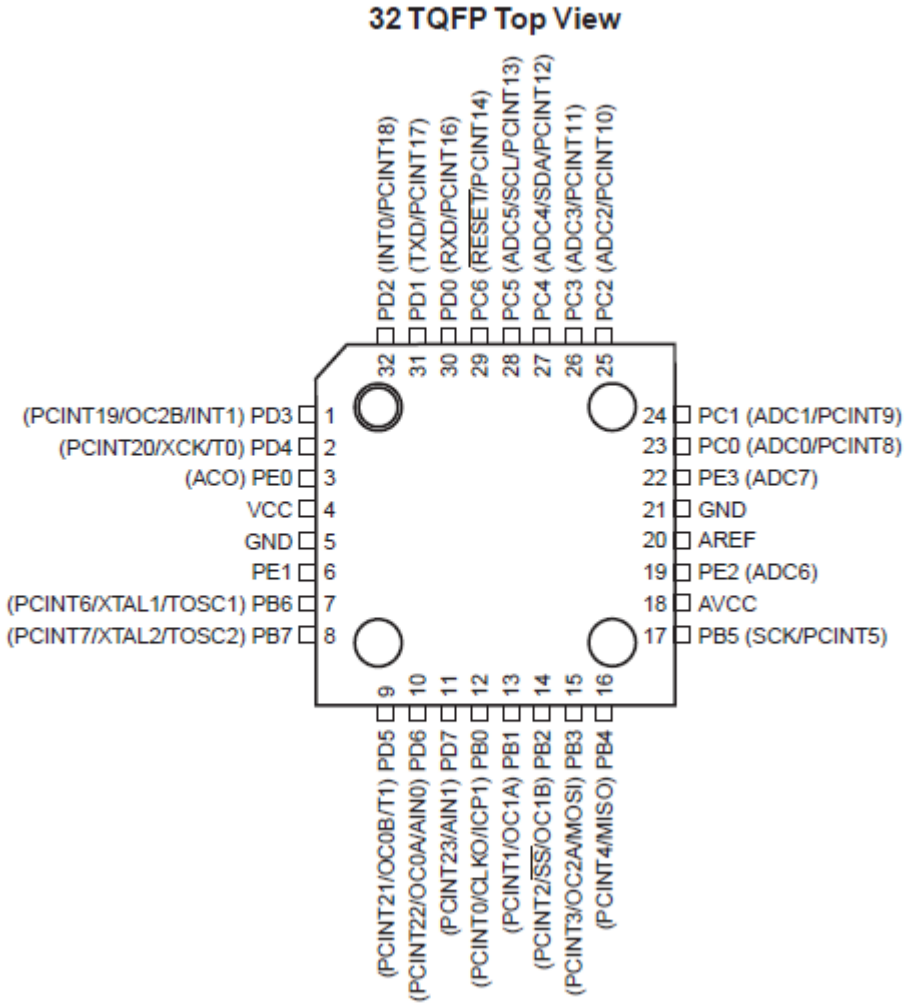
5.5.23 ATMEGA88P-ATMEGA88PA

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.24 ATMEGA88PB

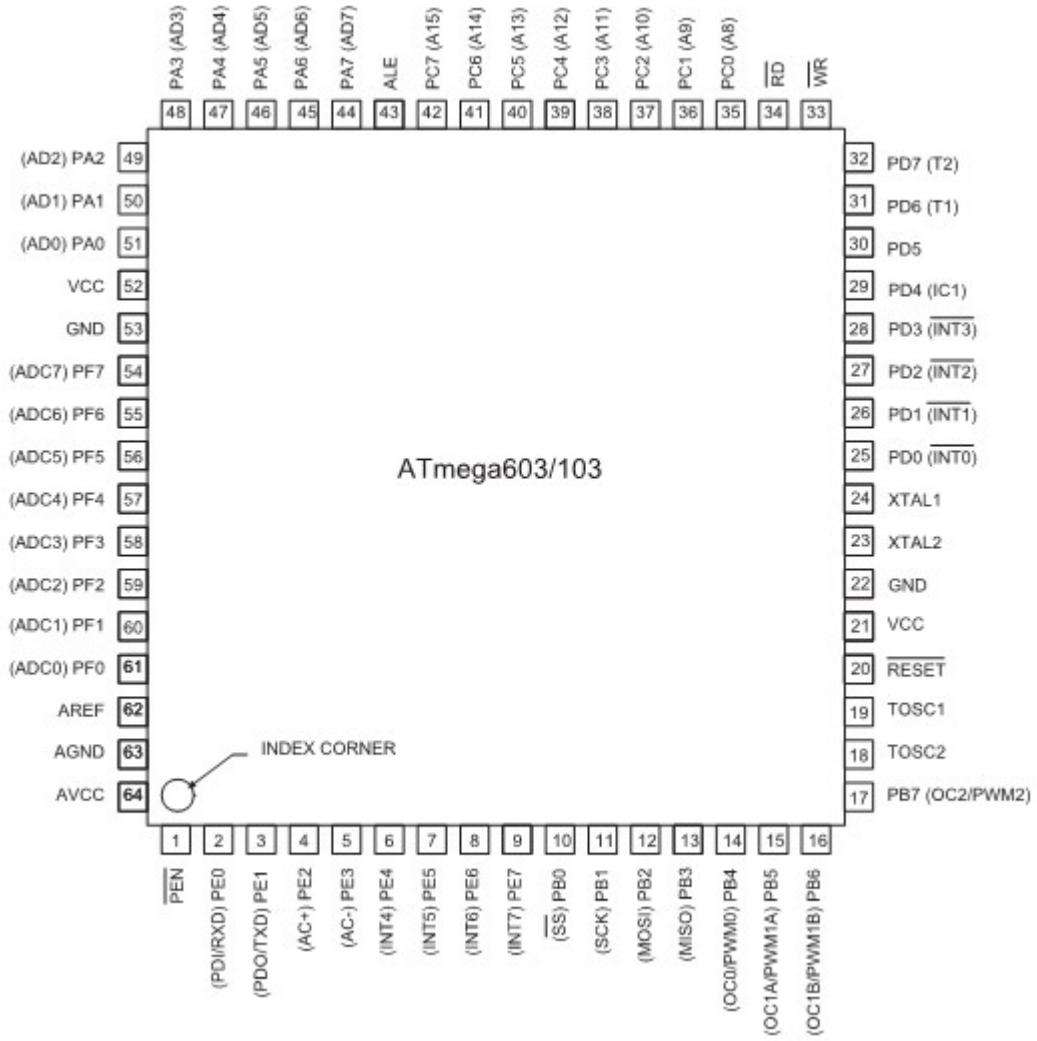
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



NO DIP version available

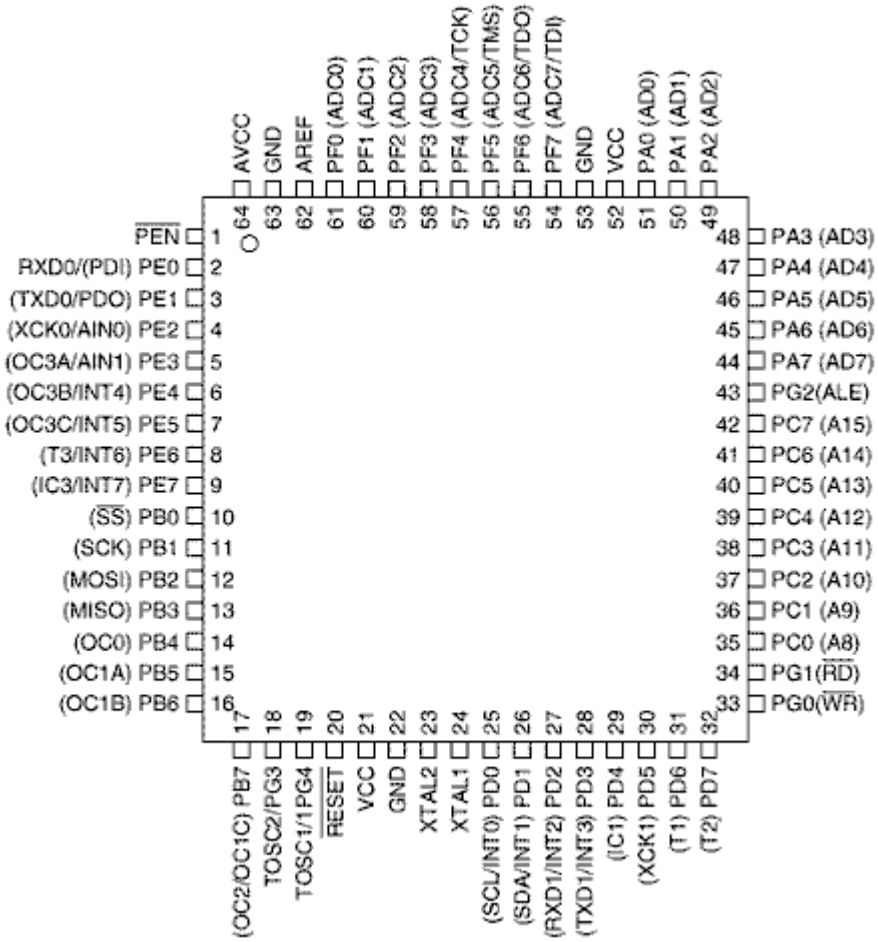
5.5.25 ATMEGA103

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.26 ATMEGA128

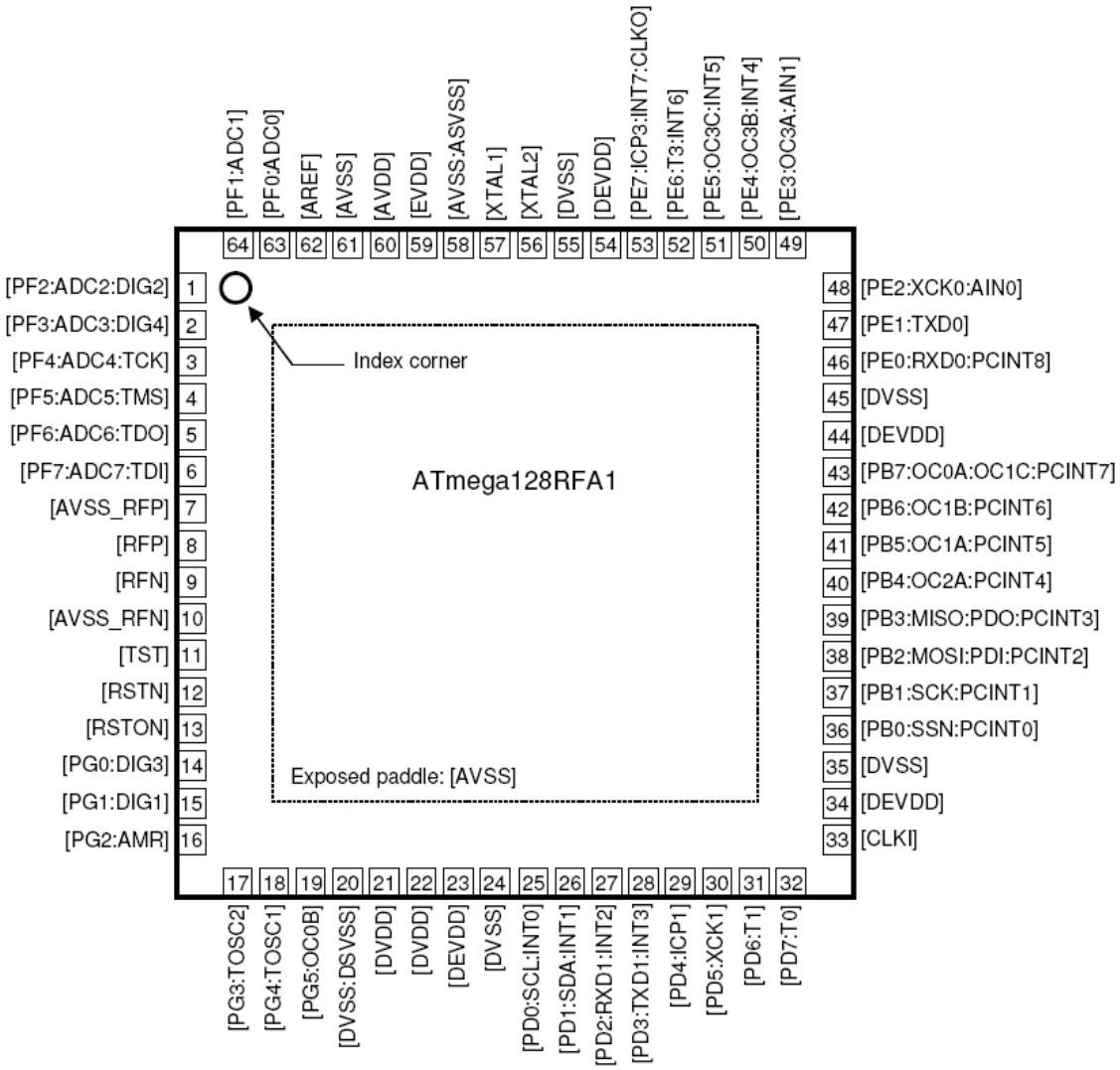
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



- When using XRAM and IDLE, the micro need the CONFIG XRAM after returing from the power down mode.
- The register mapping for PORTF is not in sequence. Which means that PINF, DDRF and PORTF are not placed in memory after each other. This has some impact on some of the functions that use the ability. 1wire and soft i2c for example. But also serin will not work. It is best to use PORTF for normal digital tasks.

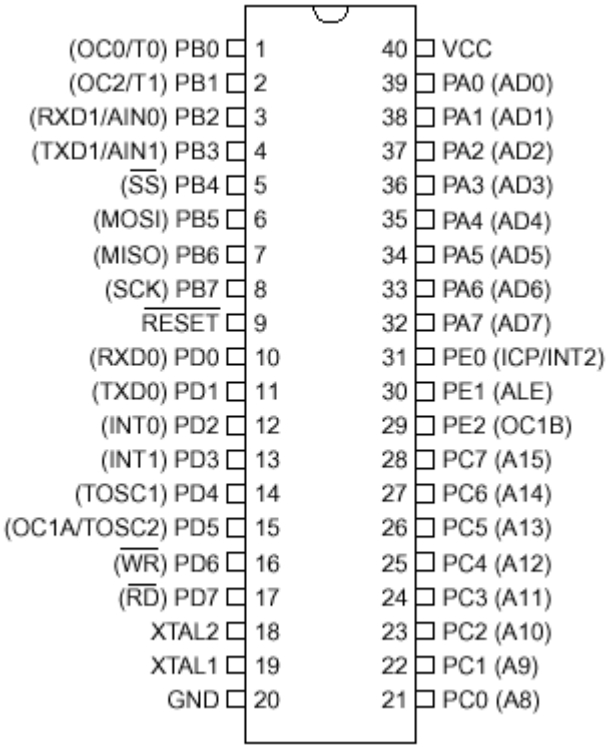
5.5.27 ATMEGA128RFA1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



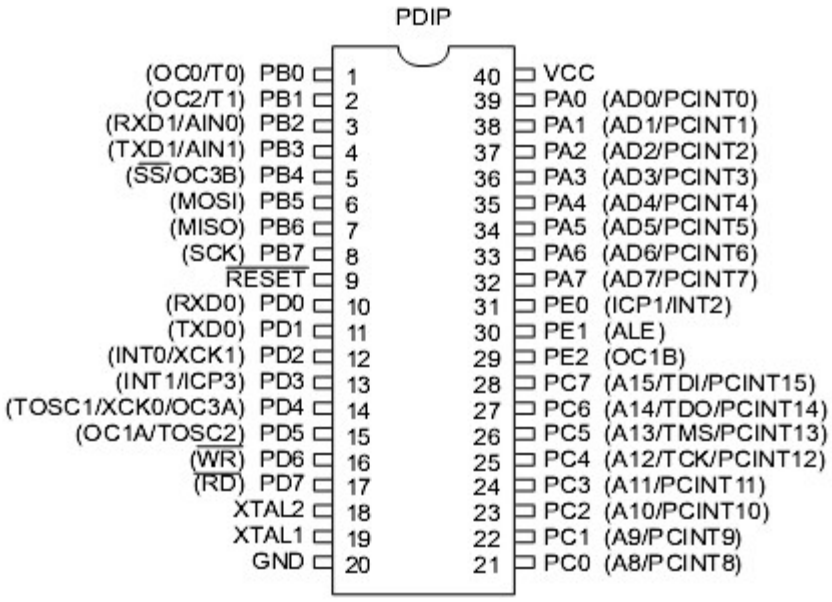
5.5.28 ATMEGA161

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.29 ATMEGA162

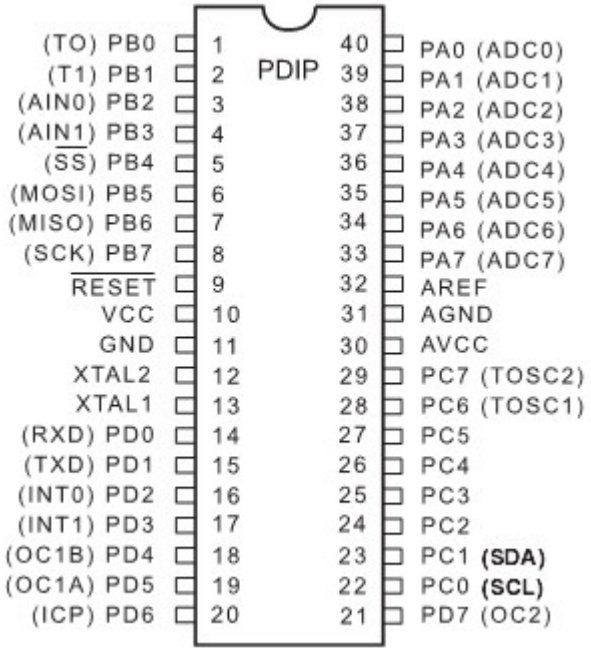
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The M162 has a clock-16 divider enabled by default. See the M162.bas sample file

5.5.30 ATMEGA163

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The M163 by default uses the internal clock running at 1 MHz

When you have problems with timing set the right fuse bit A987= 0101. This will solve this problem.

I have just found a small difference in PortB when using the Mega163 in place of a 8535. The difference is in regard to PortB.4 - PortB.7 when not used as a SPI

interface. The four upper bits of PortB are shared with the hardware SPI unit.

If the SPI is configured in SLAVE mode (DEFAULT) the MOSI , SCK , /SS

Are configured as inputs, Regardless of the DDRB setting !

The /SS (slave select) pin also has restrictions on it when using it as a general input.- see data sheet ATmega163 - p57.

This sample allows you to use the upper nibble of PortB as outputs.

Portb = &B0000_0000

DDRB = &B1111_0000 'set upper bits for output.

Spcr = &B0001_0000 ' set SPI to Master and Disable.

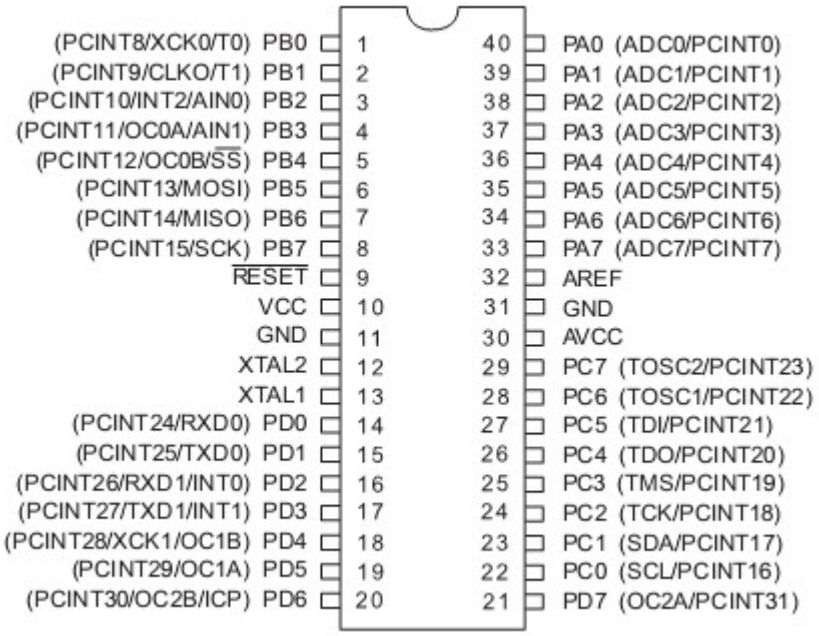
If The SPCR register is not set for Master, you cannot set the pins for

Output.

5.5.31 ATMEGA164P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

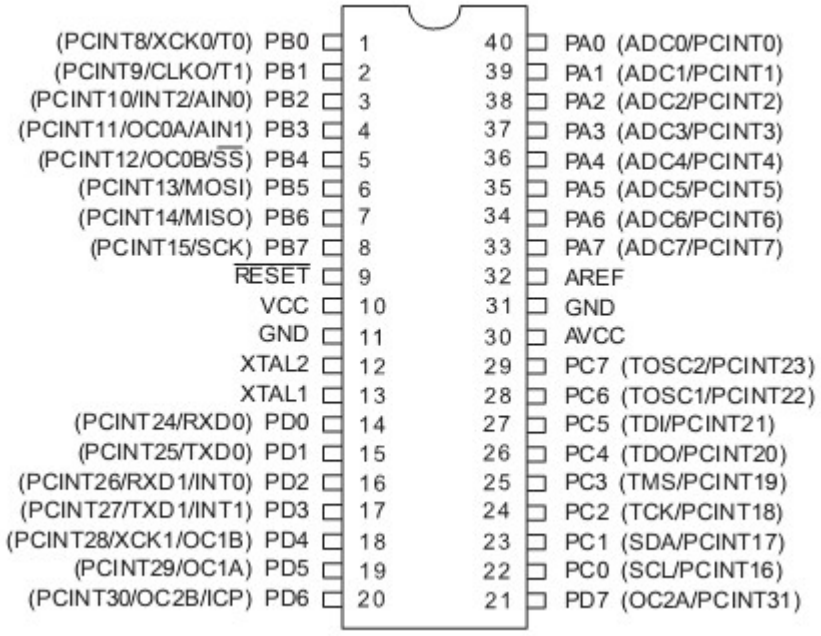
While the data sheet might make you believe this processor has a TIMER3, there is NO TIMER3 in the MEGA164.
You need a MEGA1284 when you need a TIMER3.



5.5.32 ATMEGA164PA

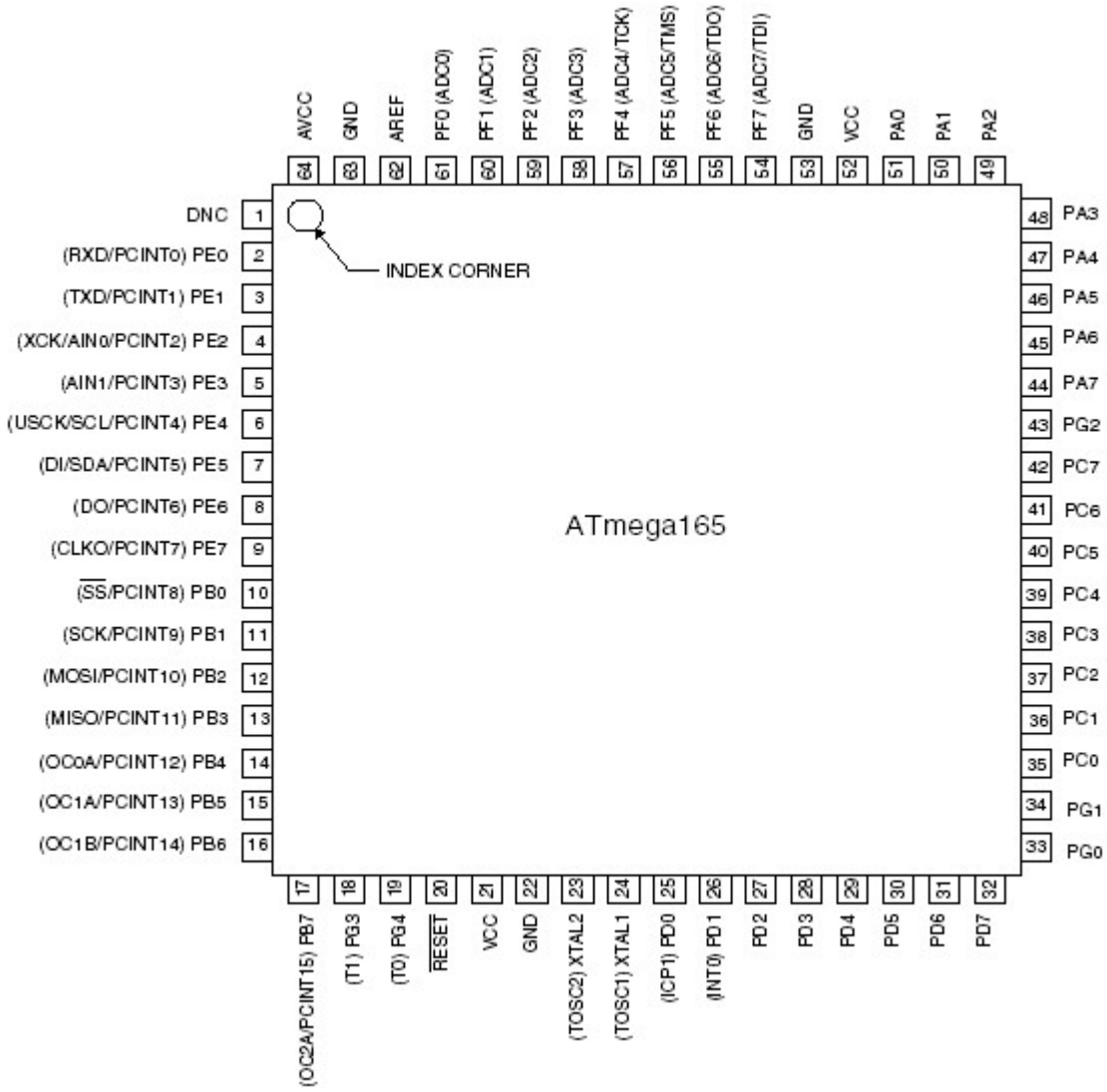
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

While the data sheet might make you believe this processor has a TIMER3, there is NO TIMER3 in the MEGA164.
You need a MEGA1284 when you need a TIMER3.



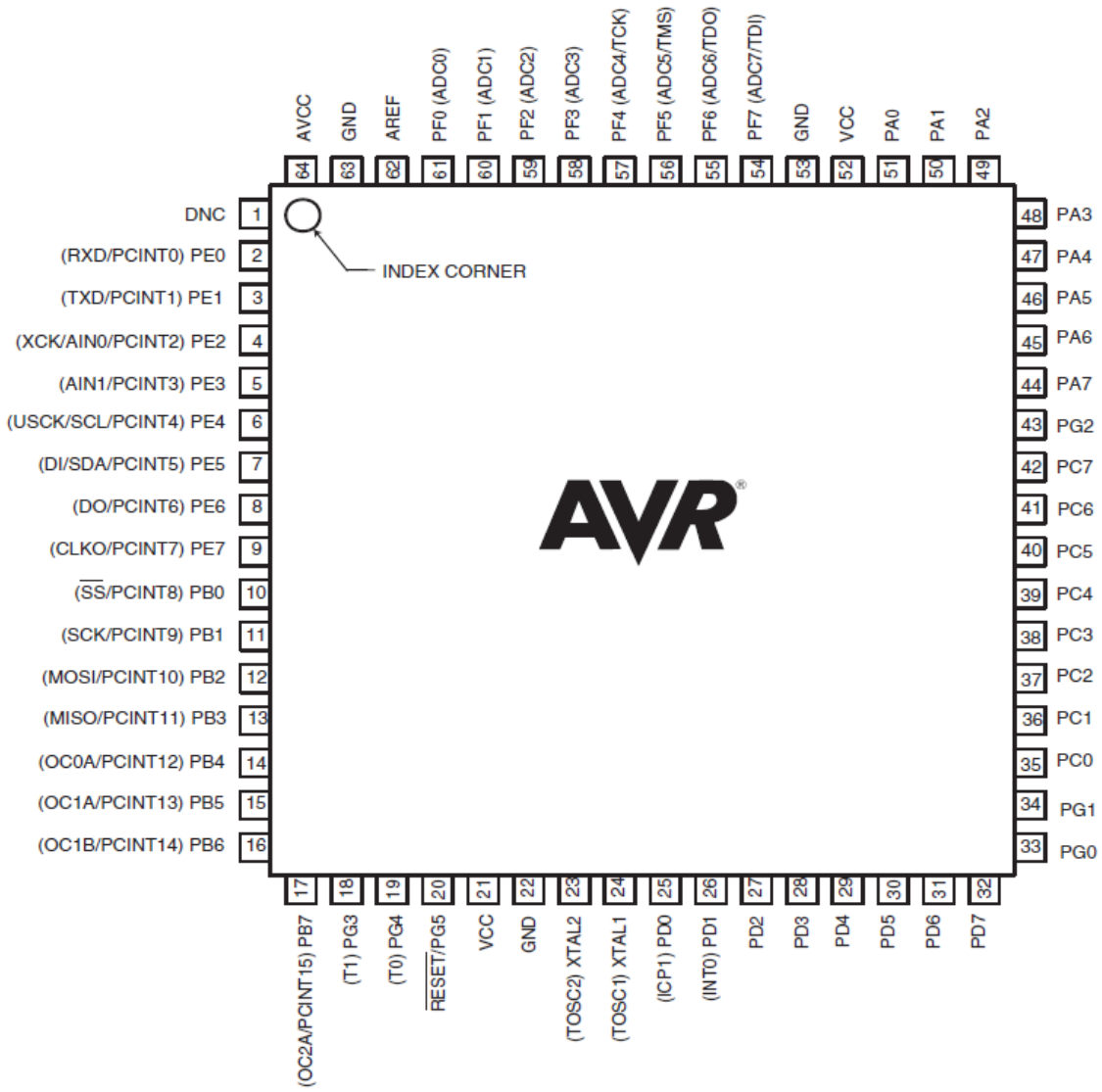
5.5.33 ATMEGA165

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



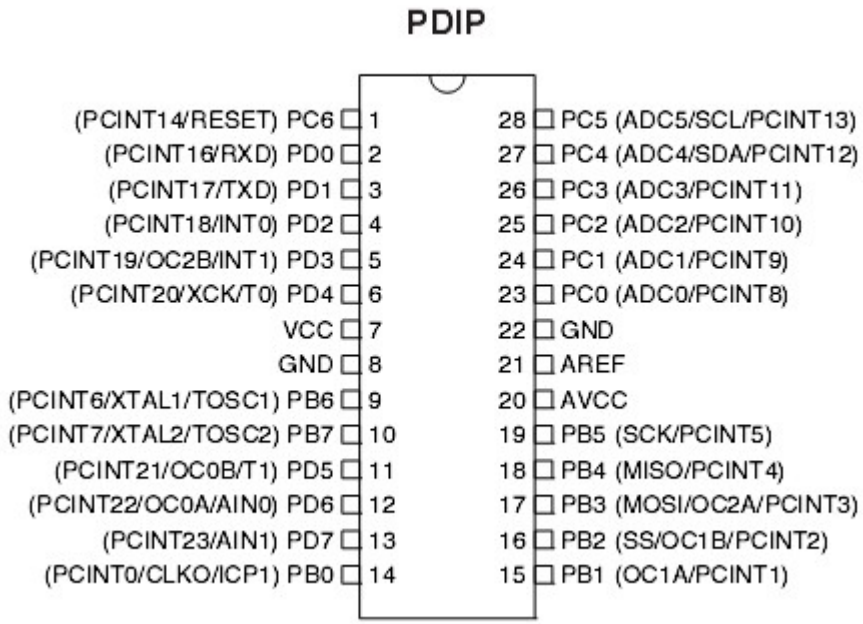
5.5.34 ATMEGA165A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



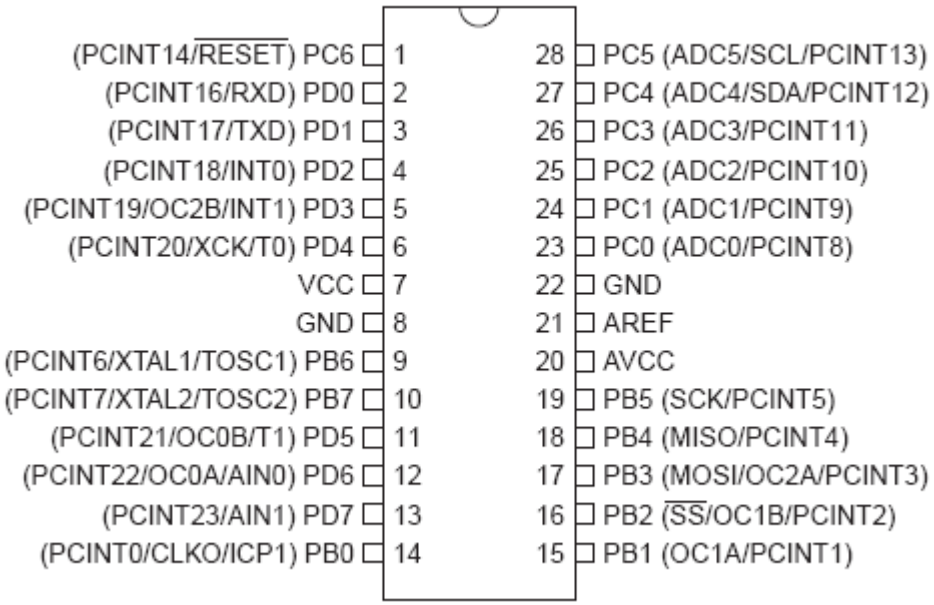
5.5.35 ATMEGA168

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



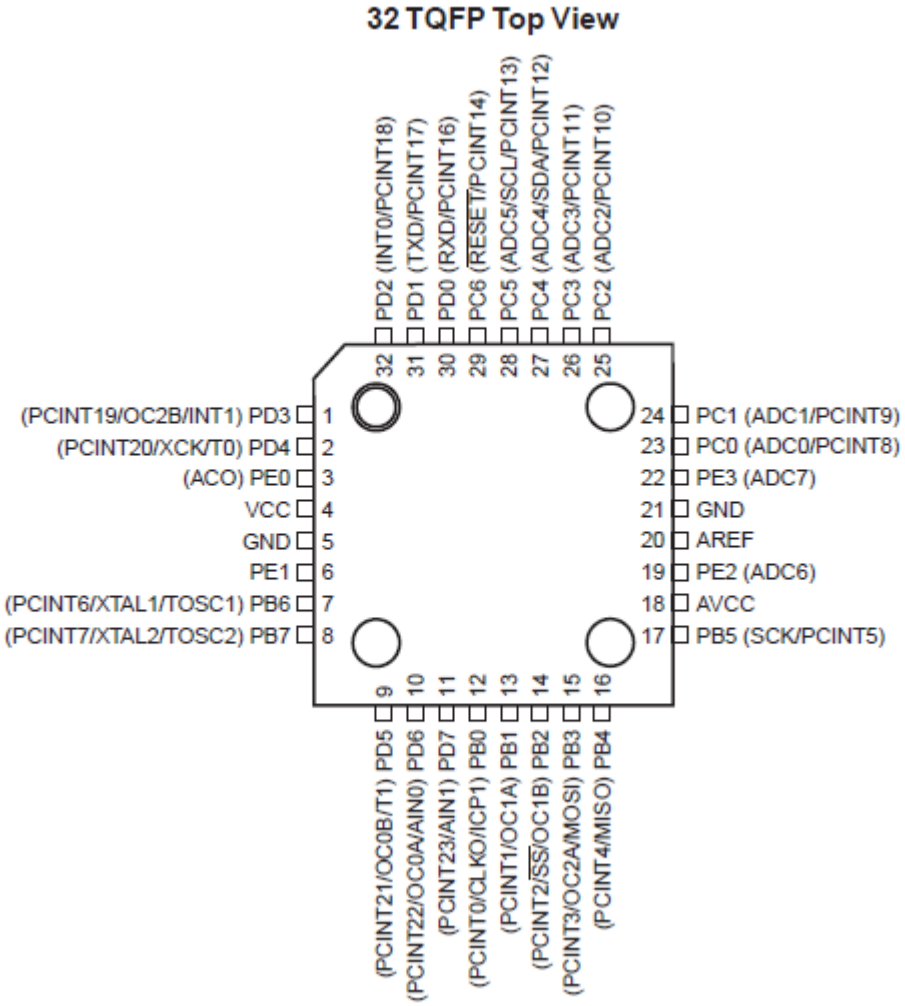
5.5.36 ATMEGA168P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.37 ATMEGA168PB

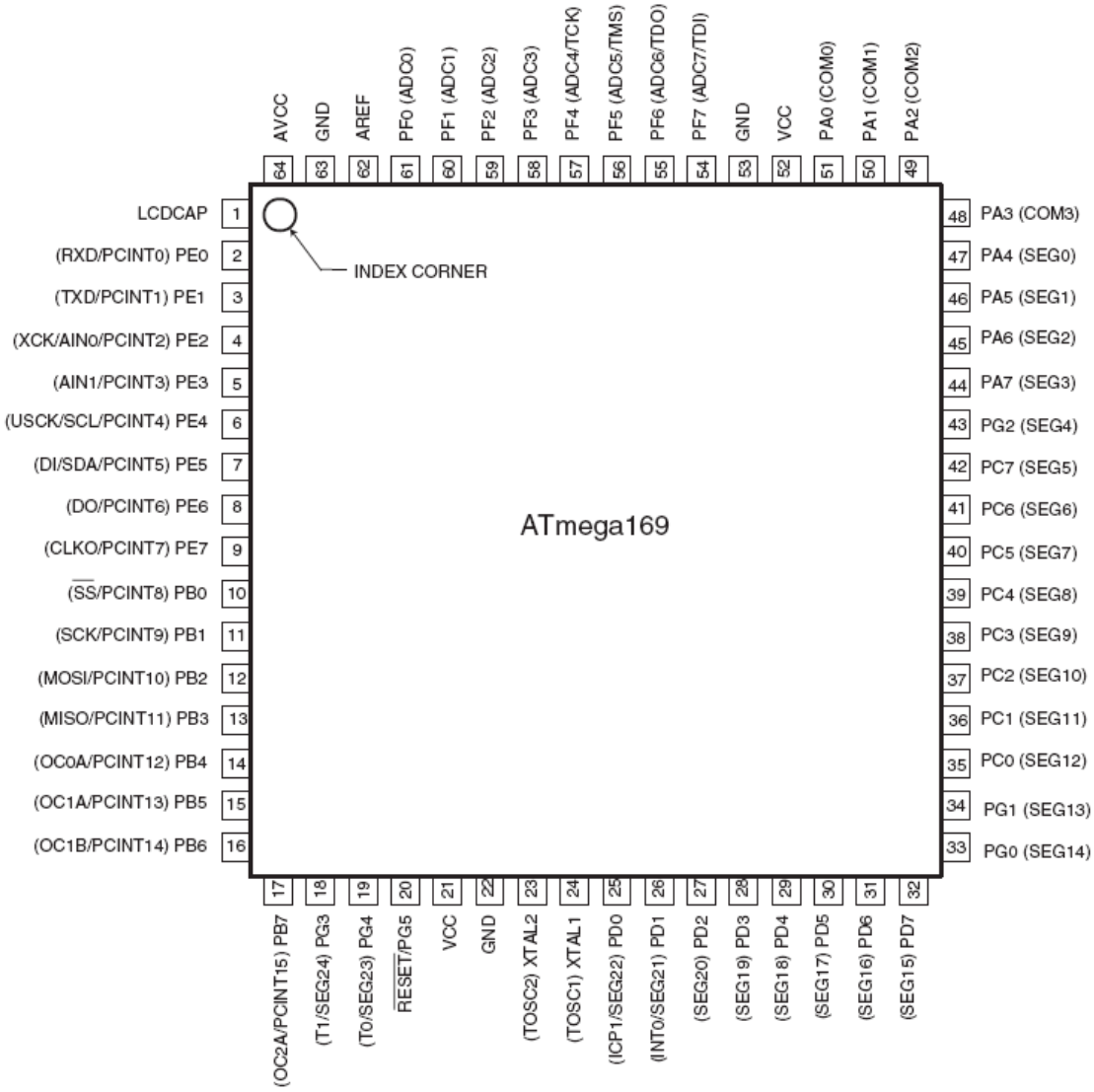
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



NO DIP version available

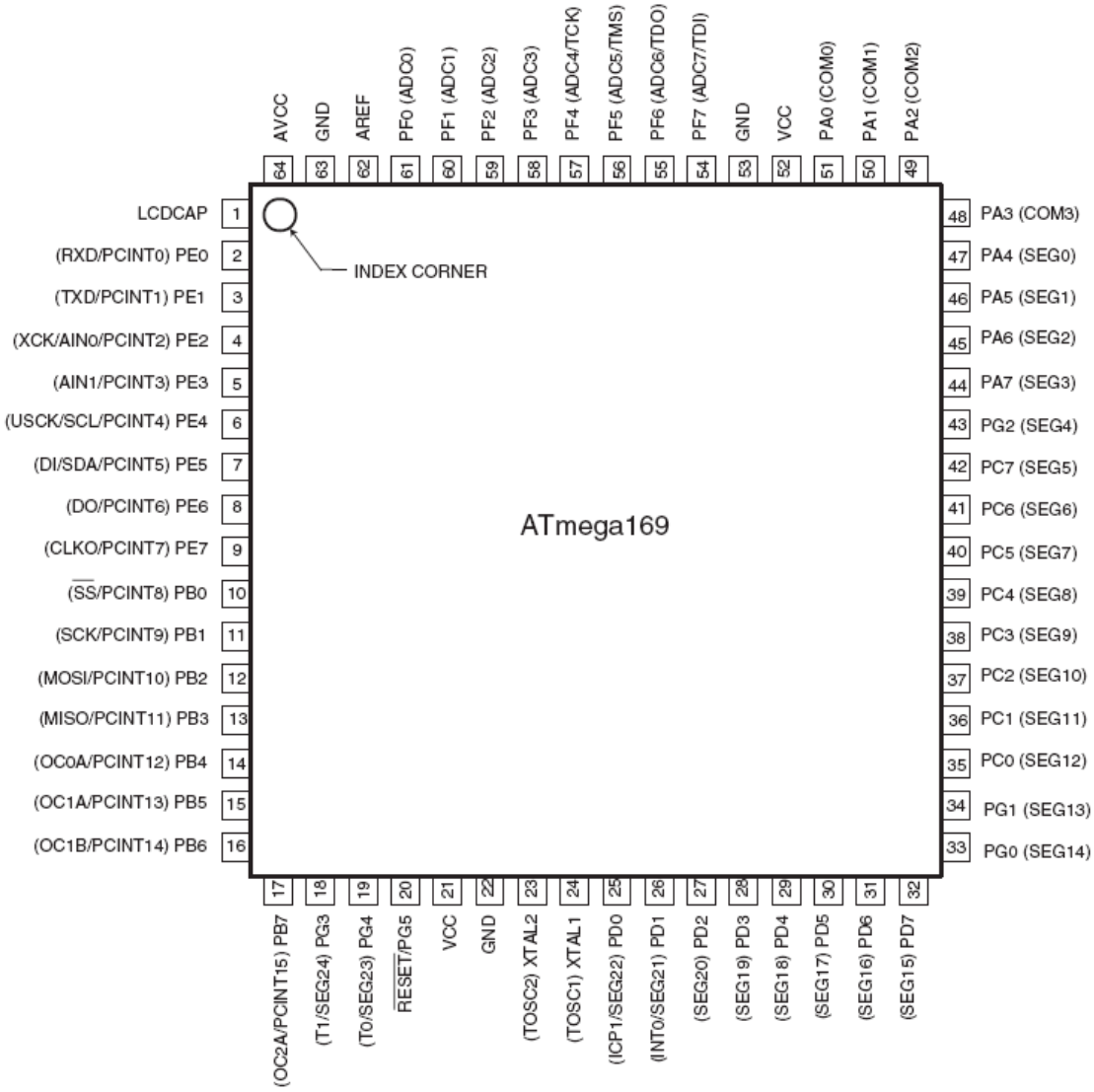
5.5.38 ATMEGA169

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



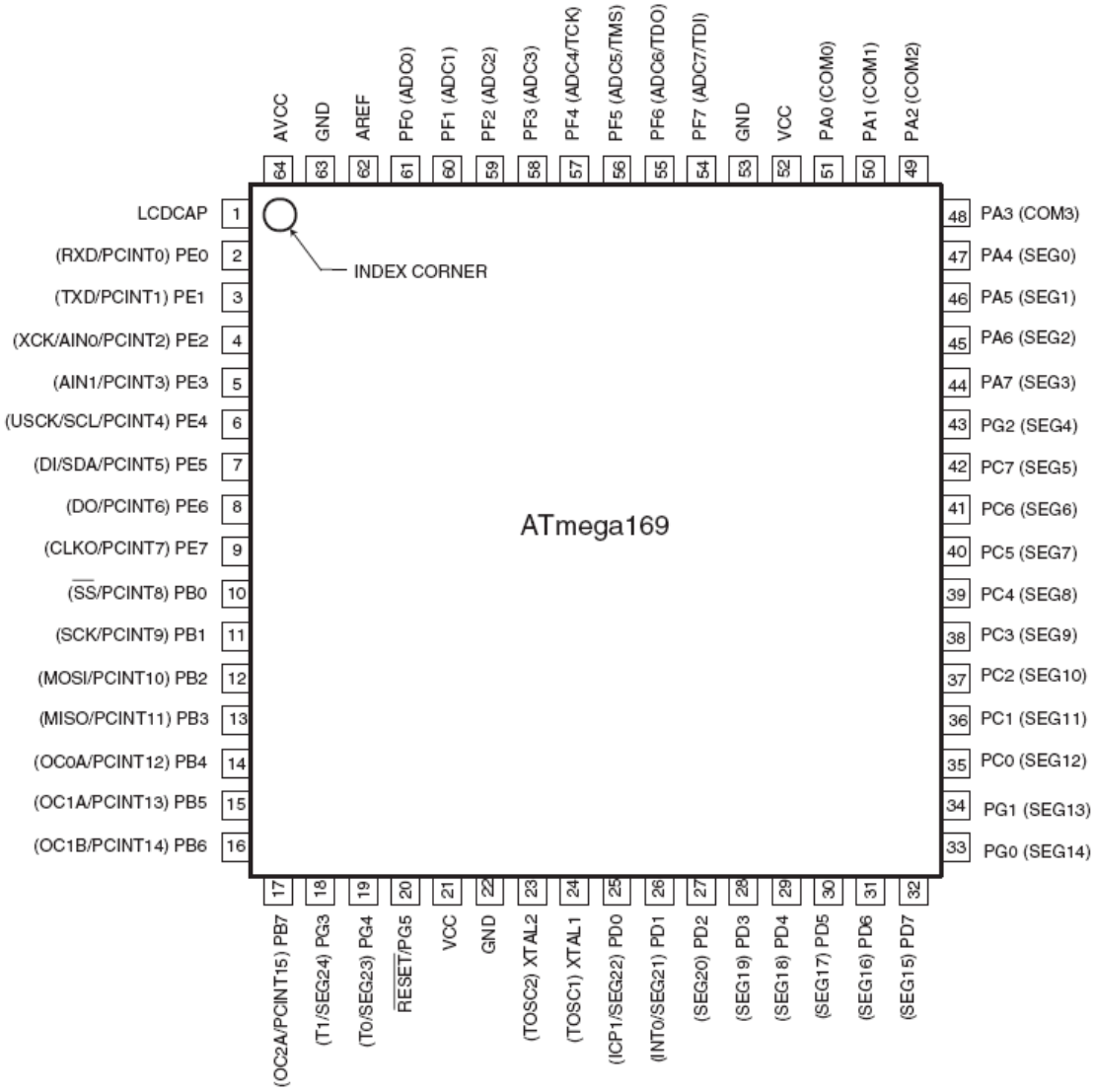
5.5.39 ATMEGA169P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



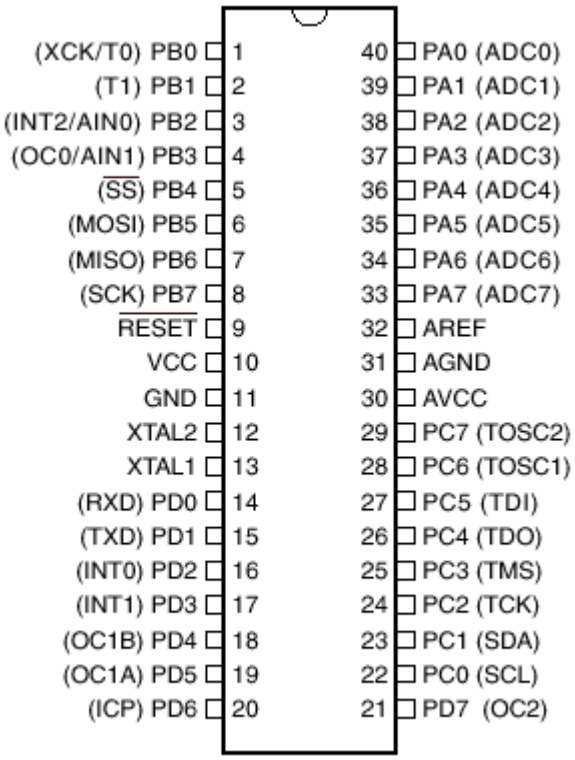
5.5.40 ATMEGA169PA

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.41 ATMEGA323

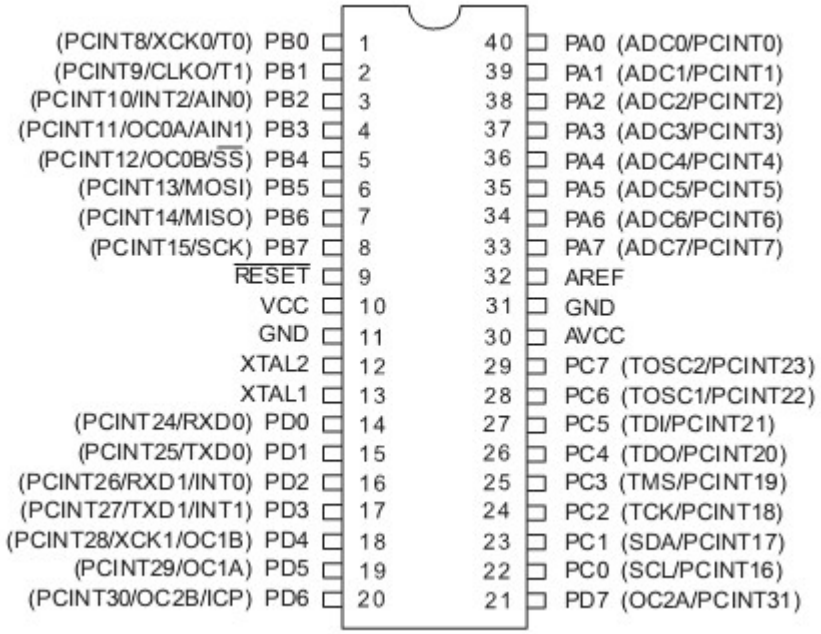
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The JTAG interface is enabled by default. This means that portC.2-portC.5 pins can not be used. Program the JTAG fuse bit to disable the JTAG interface.

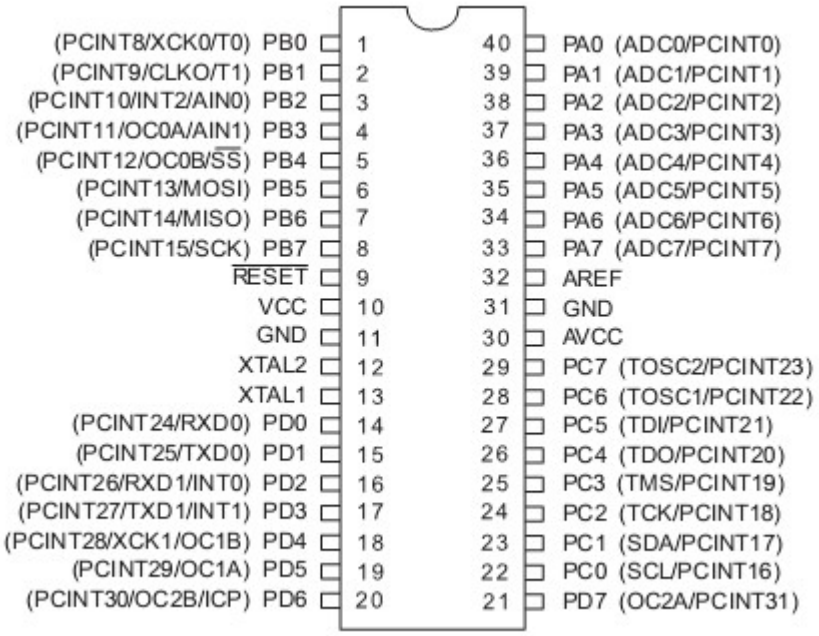
5.5.42 ATMEGA324A

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



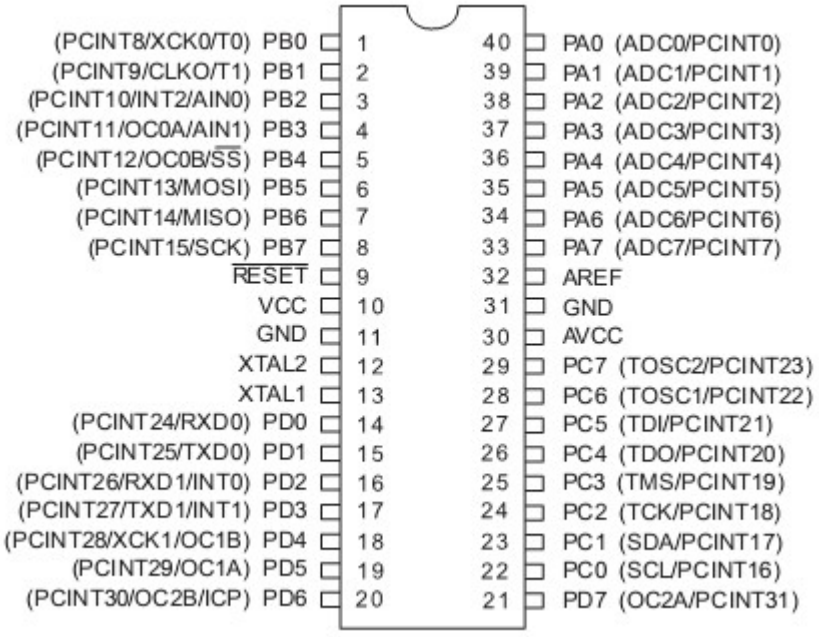
5.5.43 ATMEGA324P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



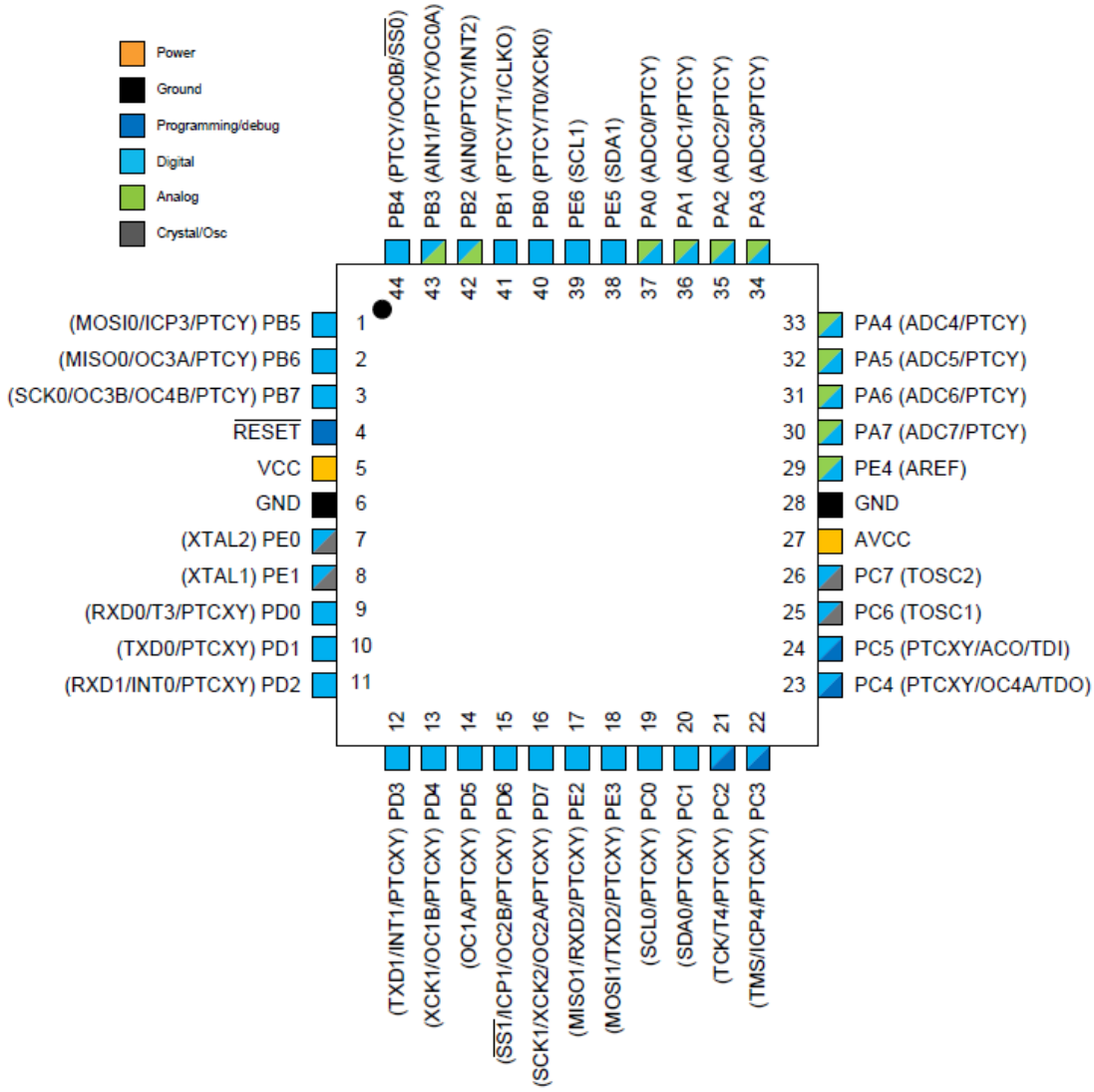
5.5.44 ATMEGA324PA

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.45 ATMEGA324PB

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



.
There is a bug in the chip : When you configure the second UART, the timer1 channel B will not work.
This info came from microchip :

Yes, your observation is correct. It is a known device bug. We already report this bug to our concern team.

This is due to the fact that timer 1 channel B is shared with XCK1 pin of UART1

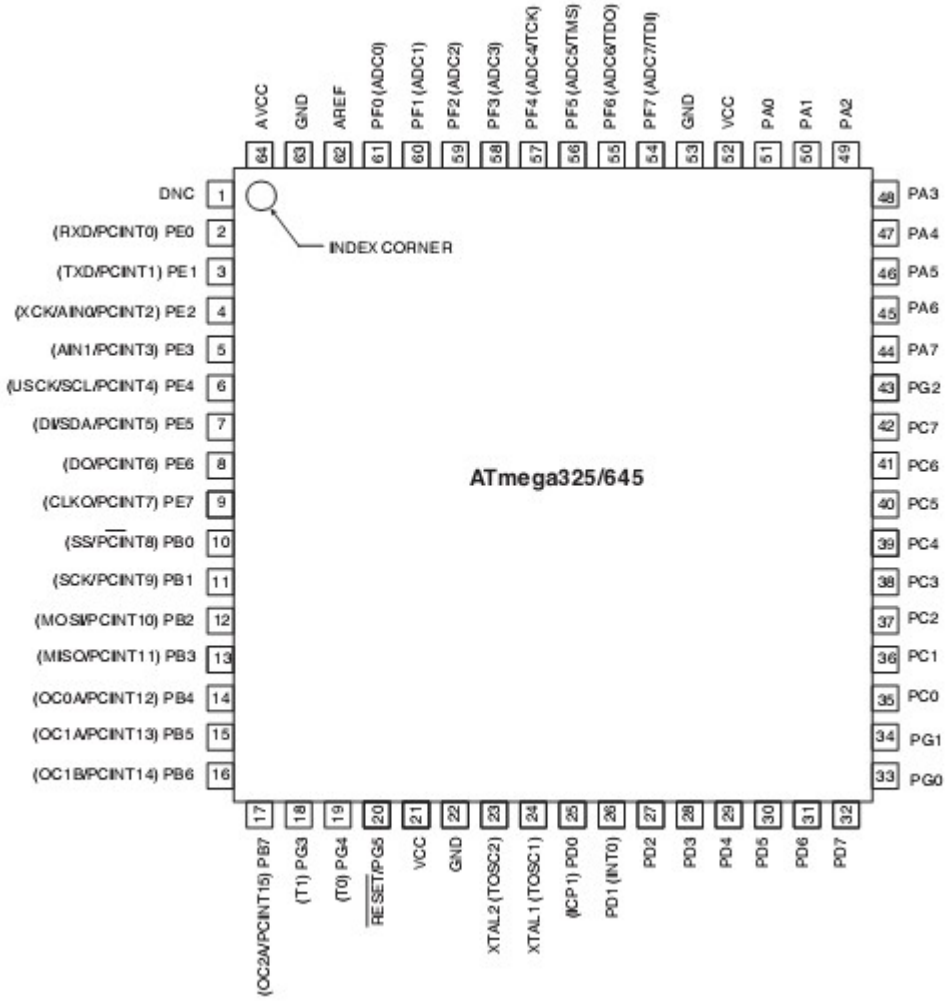
Usually this functionality should take priority over timer 1 channel B only when UART is configured in Synchronous mode but after discussing with our internal team confirmed that timer 1 channel B is disconnected based on UART1 activation

No matter even if the UART is configured in Asynchronous mode(in which case there is no use of XCK1) timer 1 channel B still gets disconnected.

This issue also presents in UART2 XCK2/OC2A.

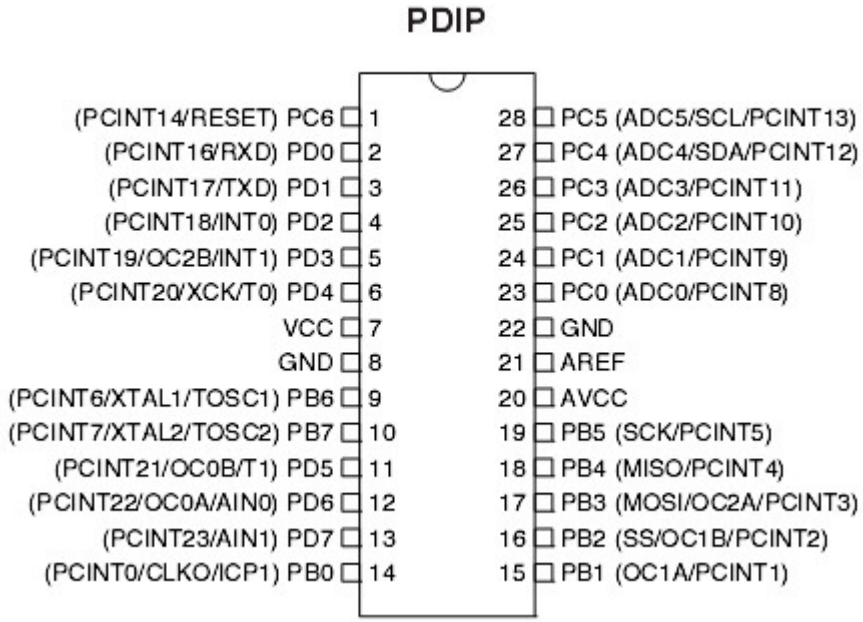
5.5.46 ATMEGA325

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



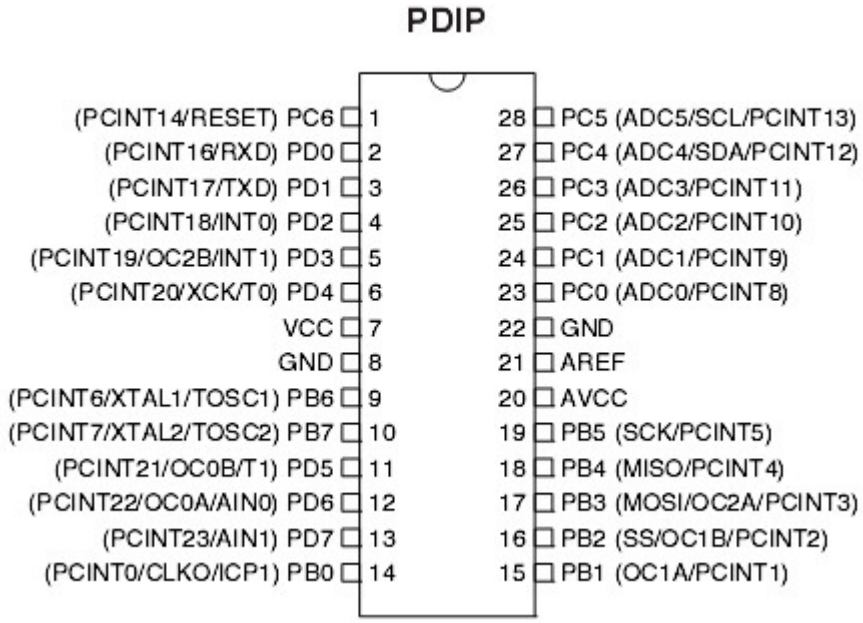
5.5.47 ATMEGA328

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



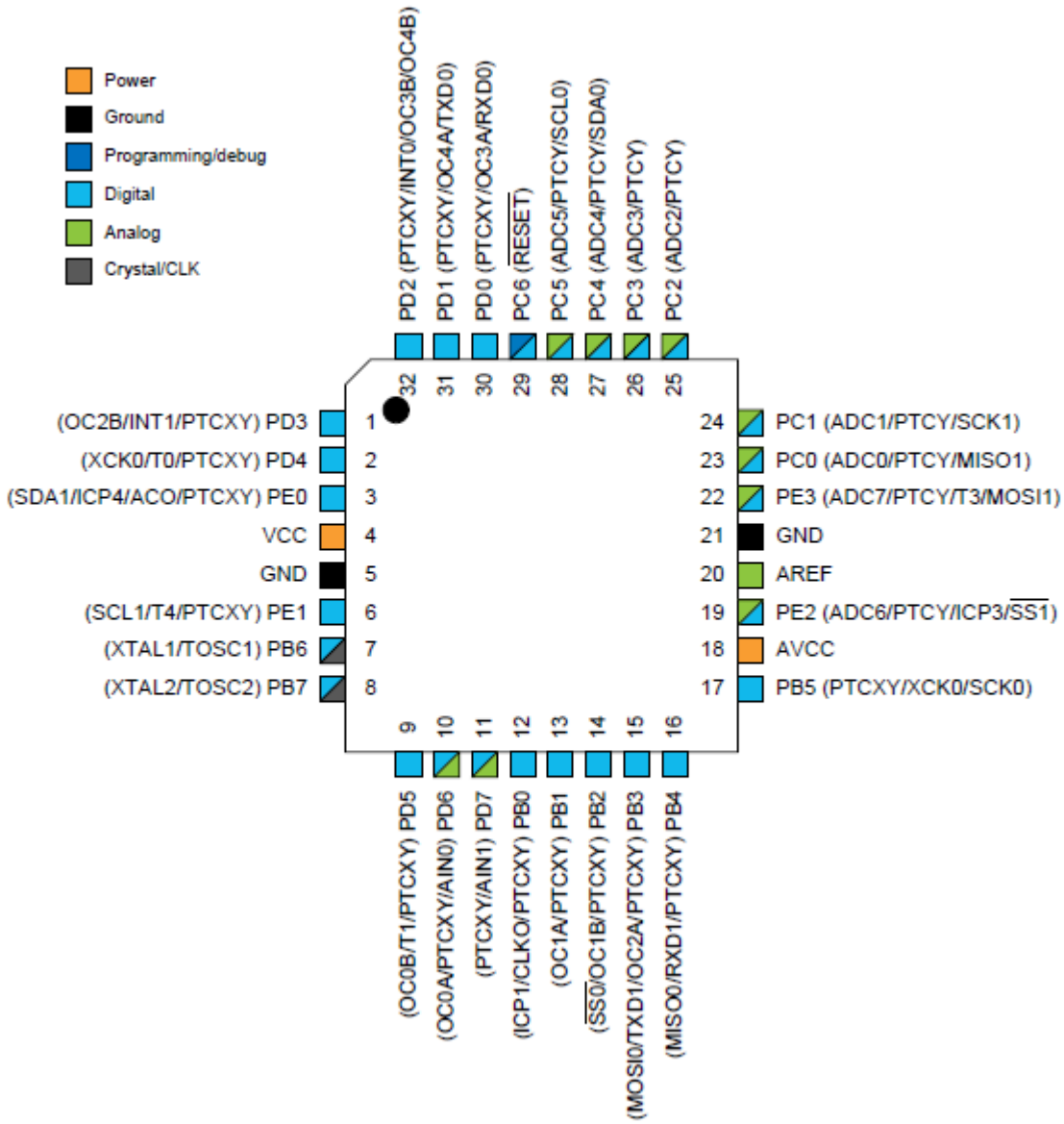
5.5.48 ATMEGA328P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.5.49 ATMEGA328PB

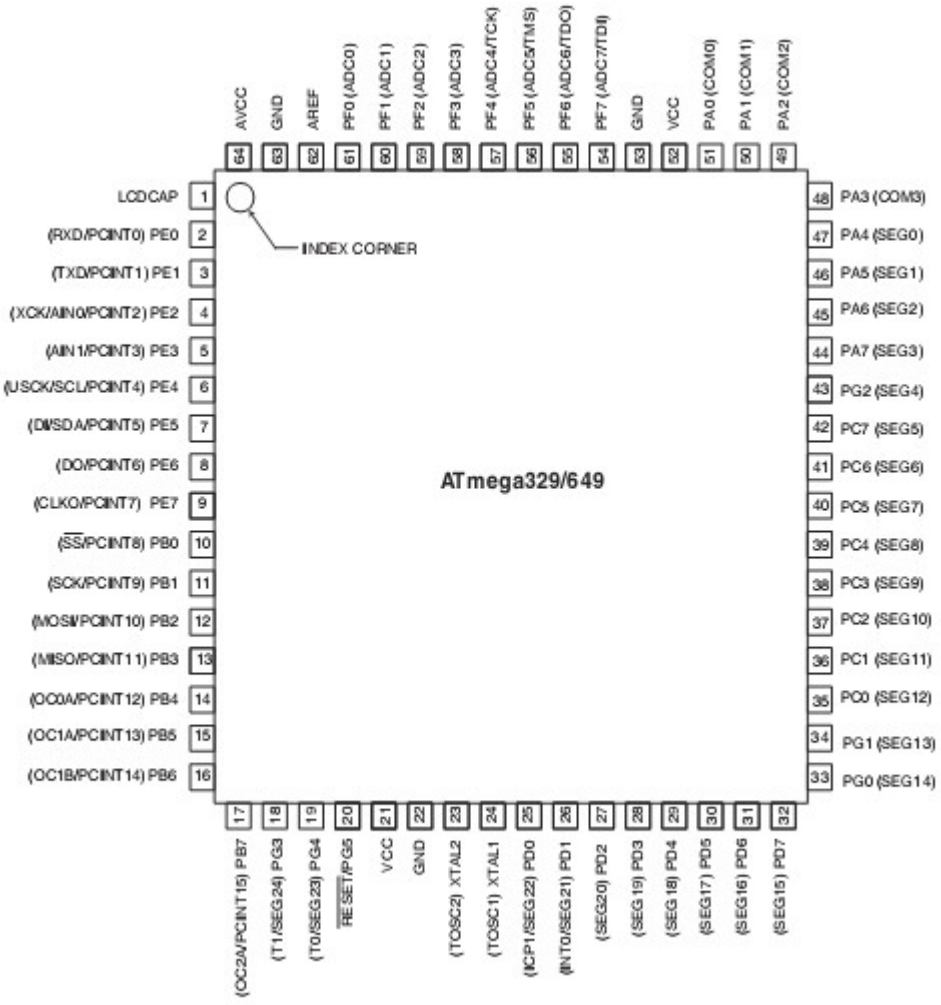
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



Notice that this processor is compatible with M328 but that extra pins have been added at location of VCC/GND.

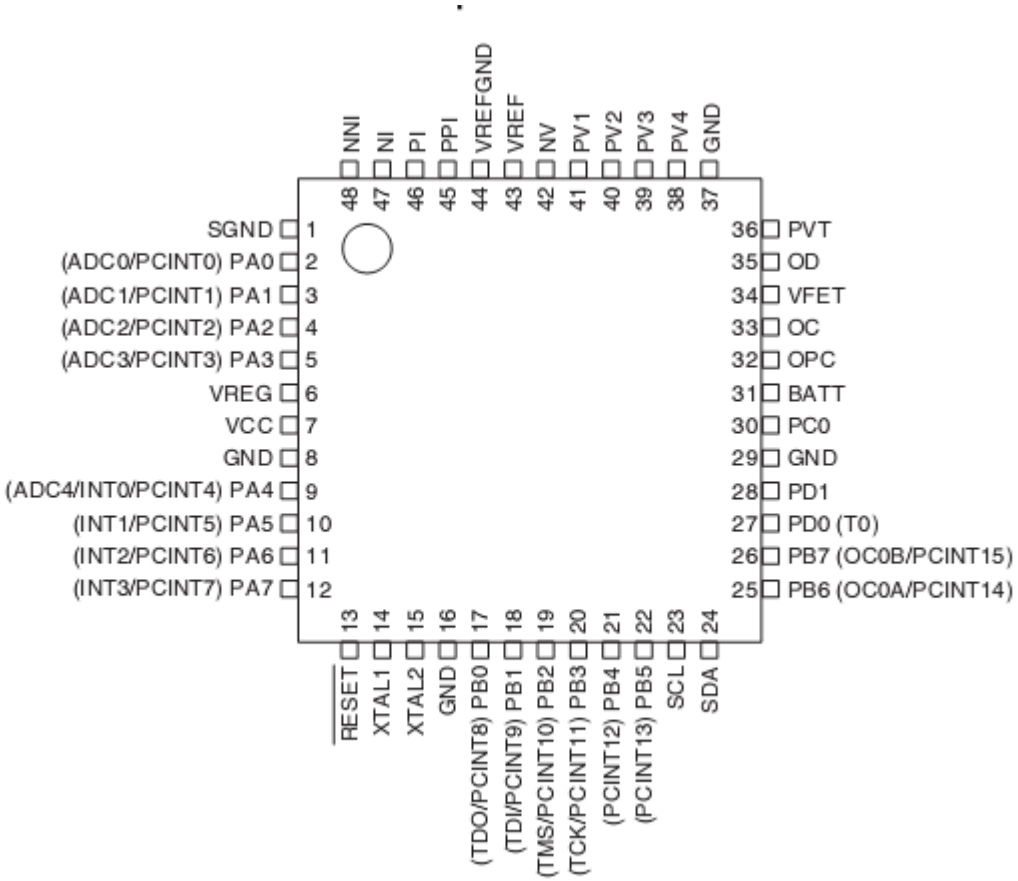
5.5.50 ATMEGA329

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



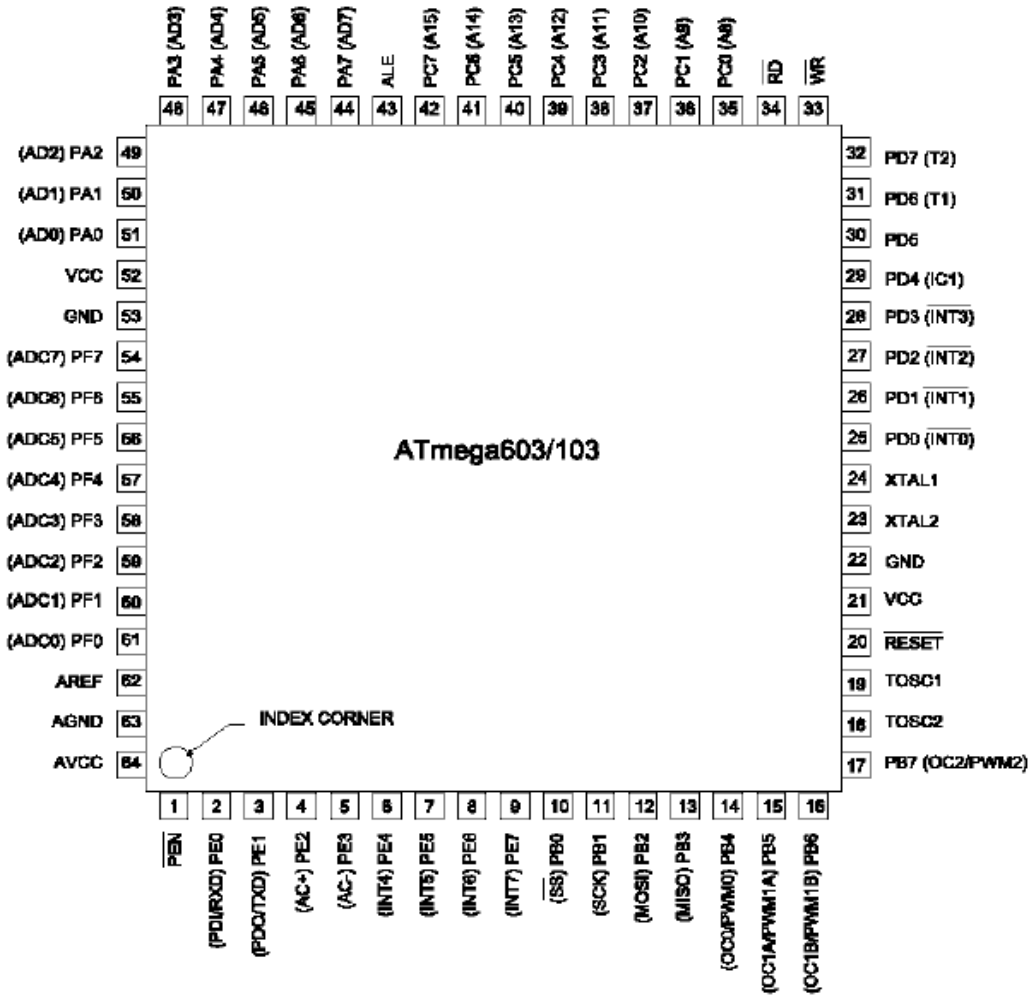
5.5.51 ATMEGA406

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
The image is from a preliminary data sheet. It is not clear yet if SCL and SDA have pin names too.
This chip can only programmed parallel and with JTAG. Normal (serial) ISP programming is not available.

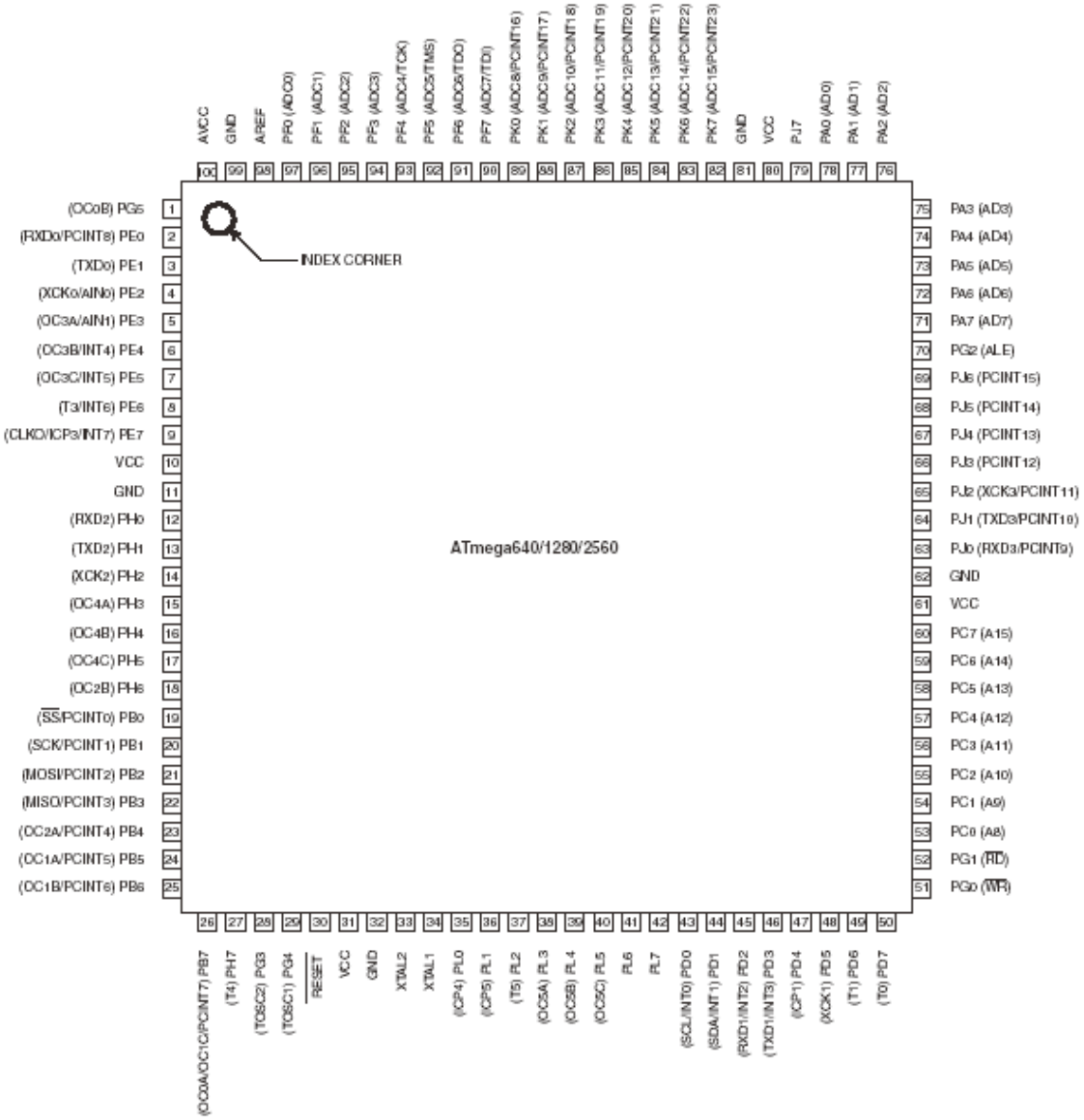


5.5.52 ATMEGA603

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
When you have a better image available, please send it to support@mcselec.com

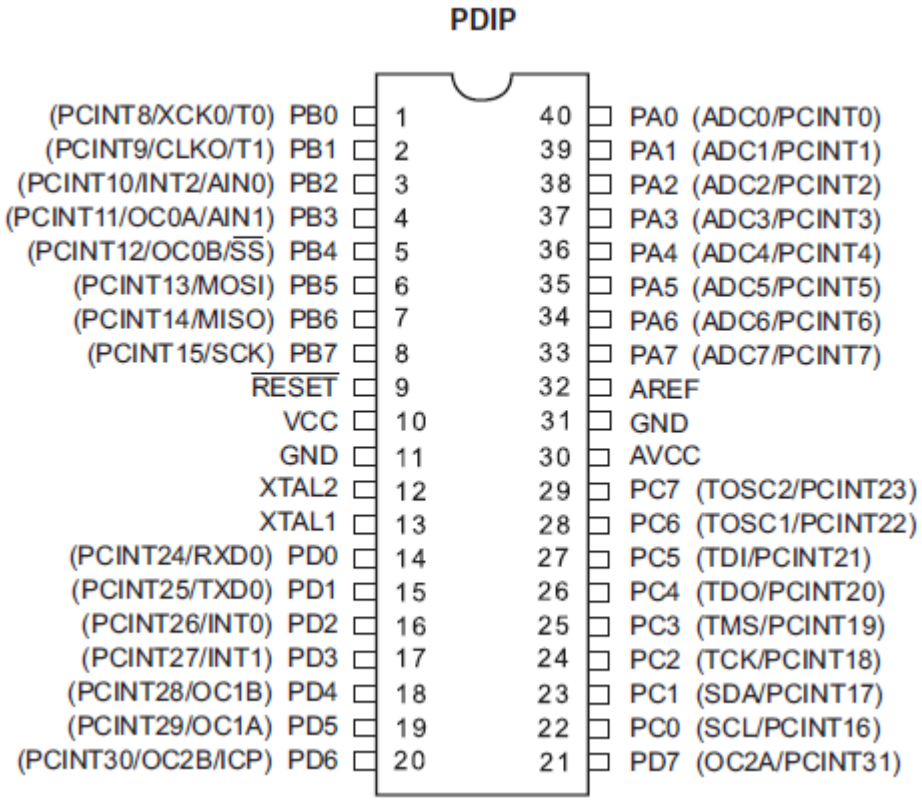


5.5.53 ATMEGA640



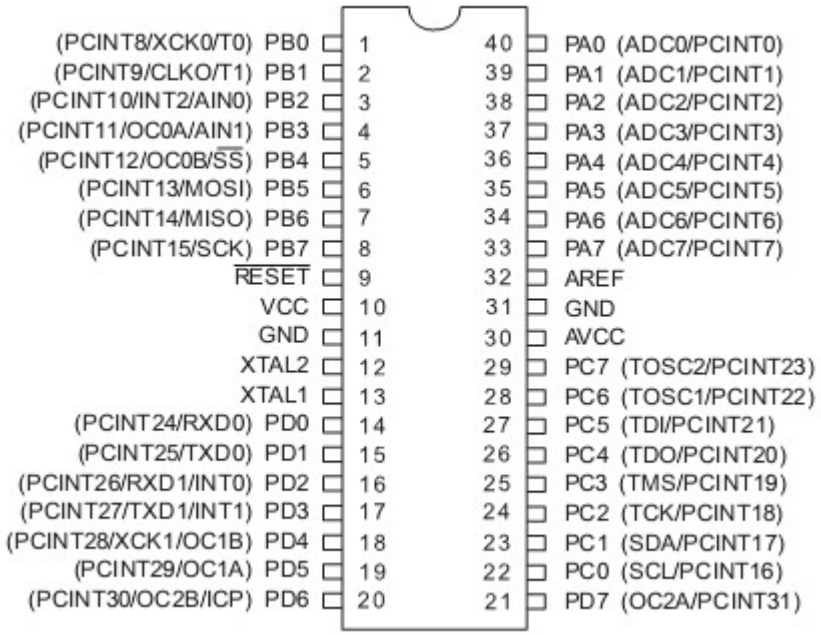
.

5.5.54 ATMEGA644



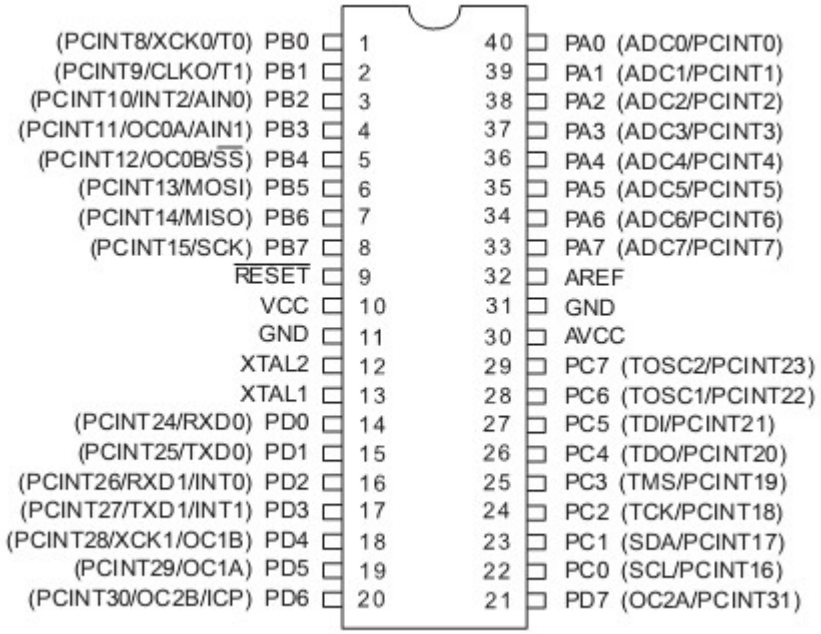
5.5.55 ATMEGA644P

Notice that there are Mega644 and Mega644P chips.
P stand for PICO power. You should use the P-version for new designs.
These Pico version usual add some functionality such as a second UART.



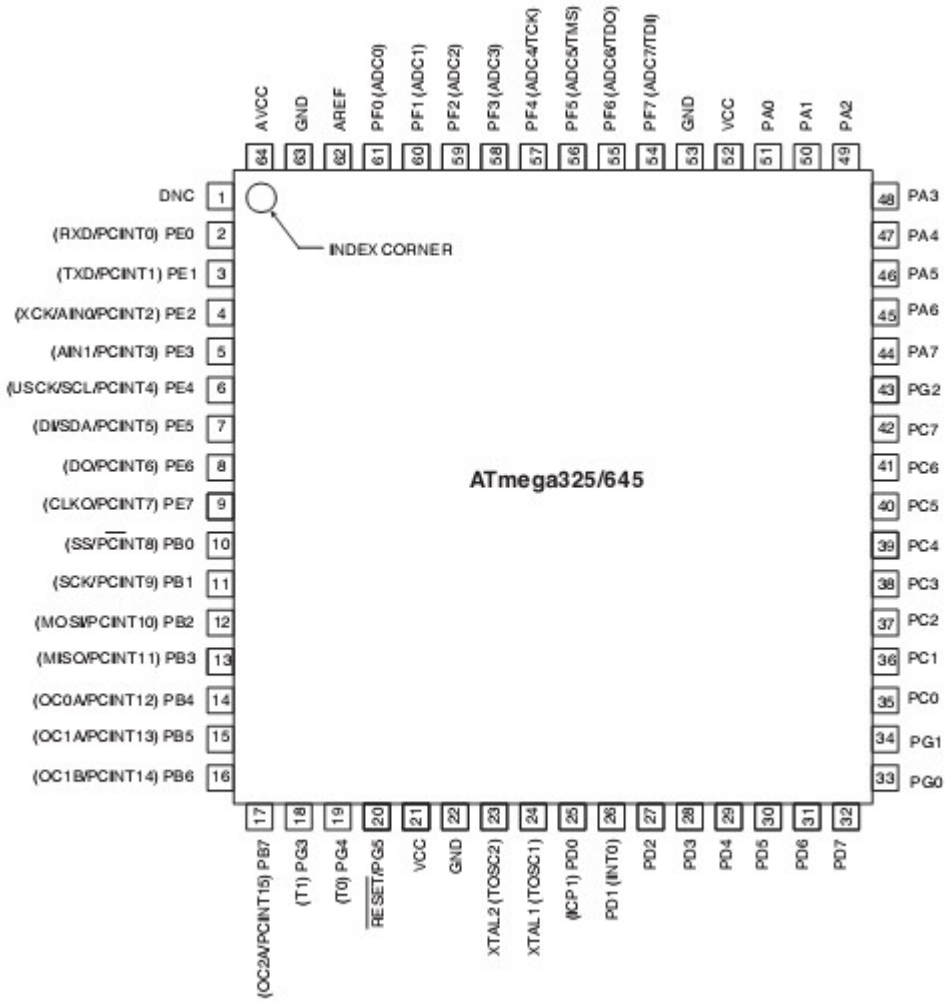
5.5.56 ATMEGA644PA

Notice that there are Mega644 and Mega644P chips.
P stand for PICO power. You should use the P-version for new designs.
These Pico version usual add some functionality such as a second UART.



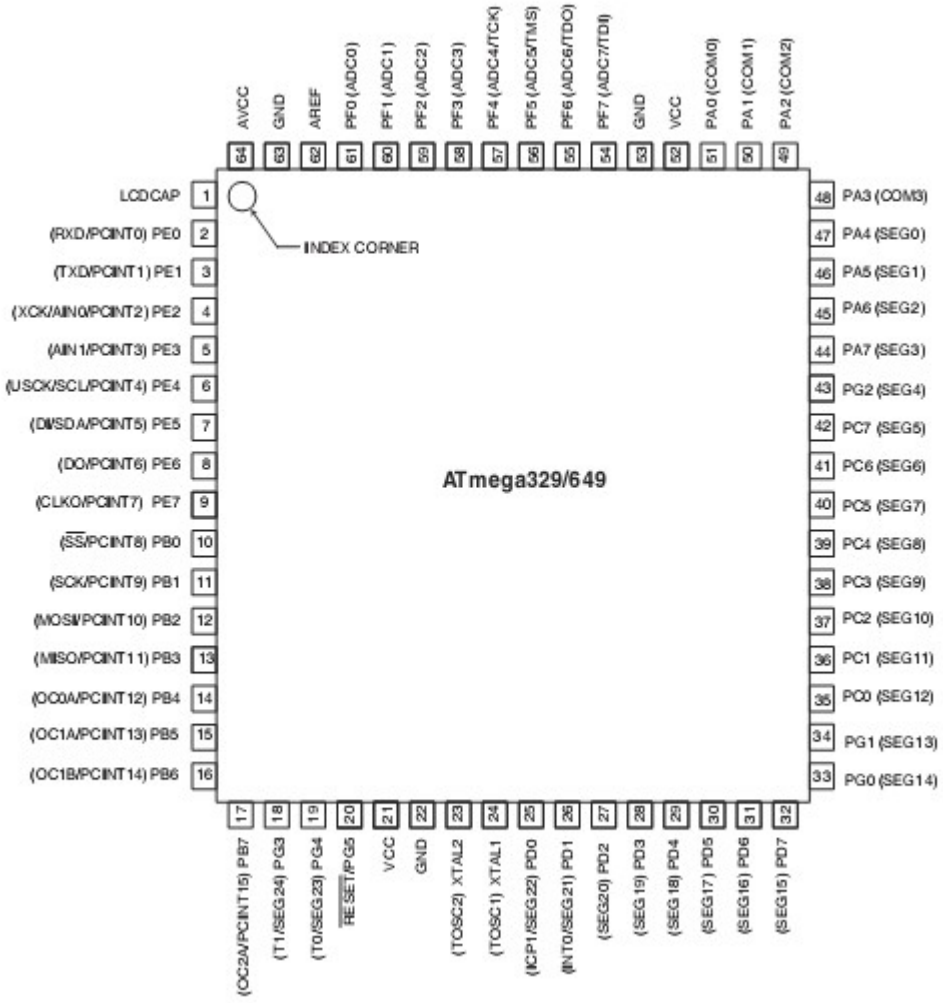
5.5.57 ATMEGA645

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



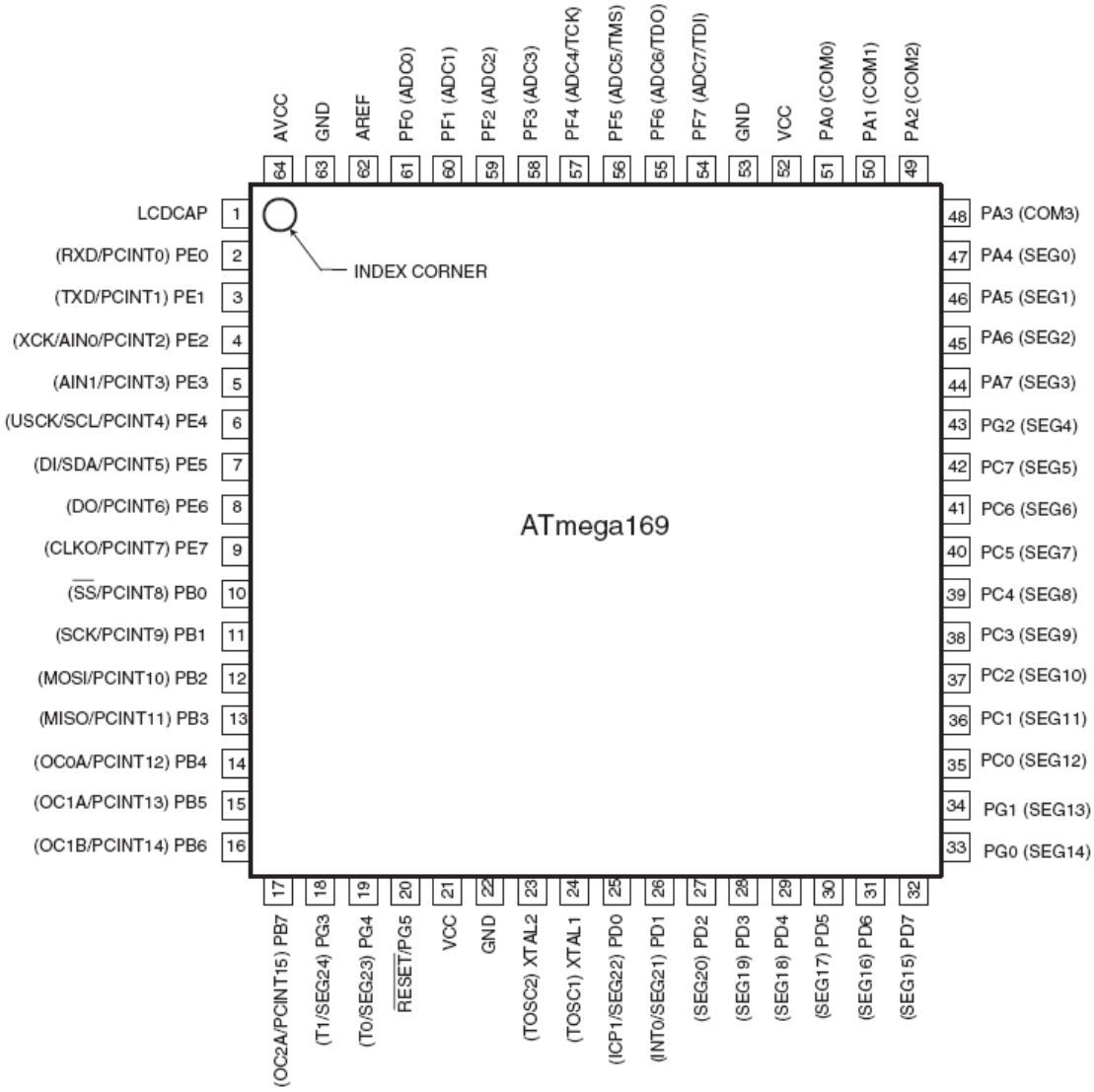
5.5.58 ATMEGA649

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

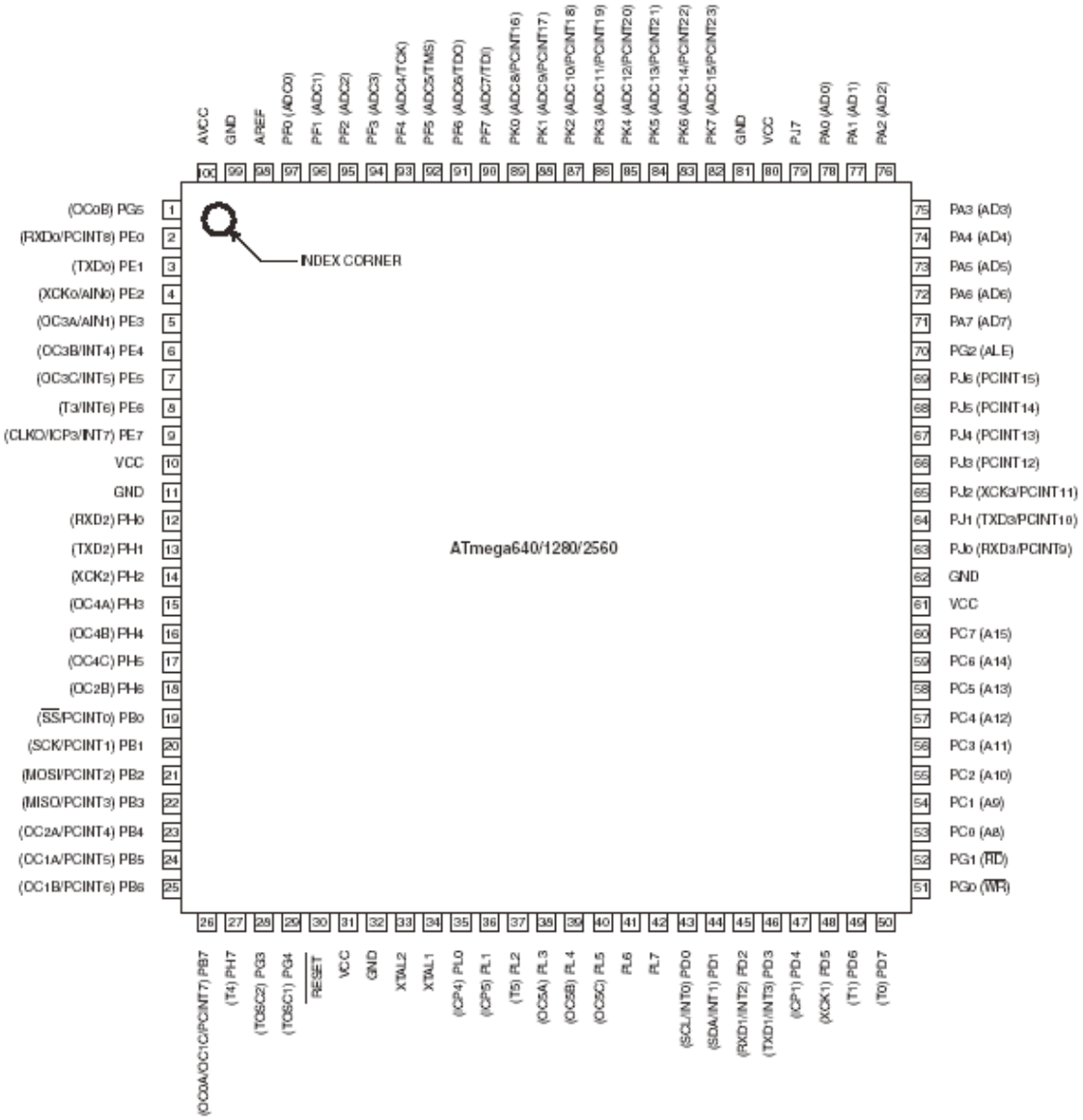


5.5.59 ATMEGA649PA

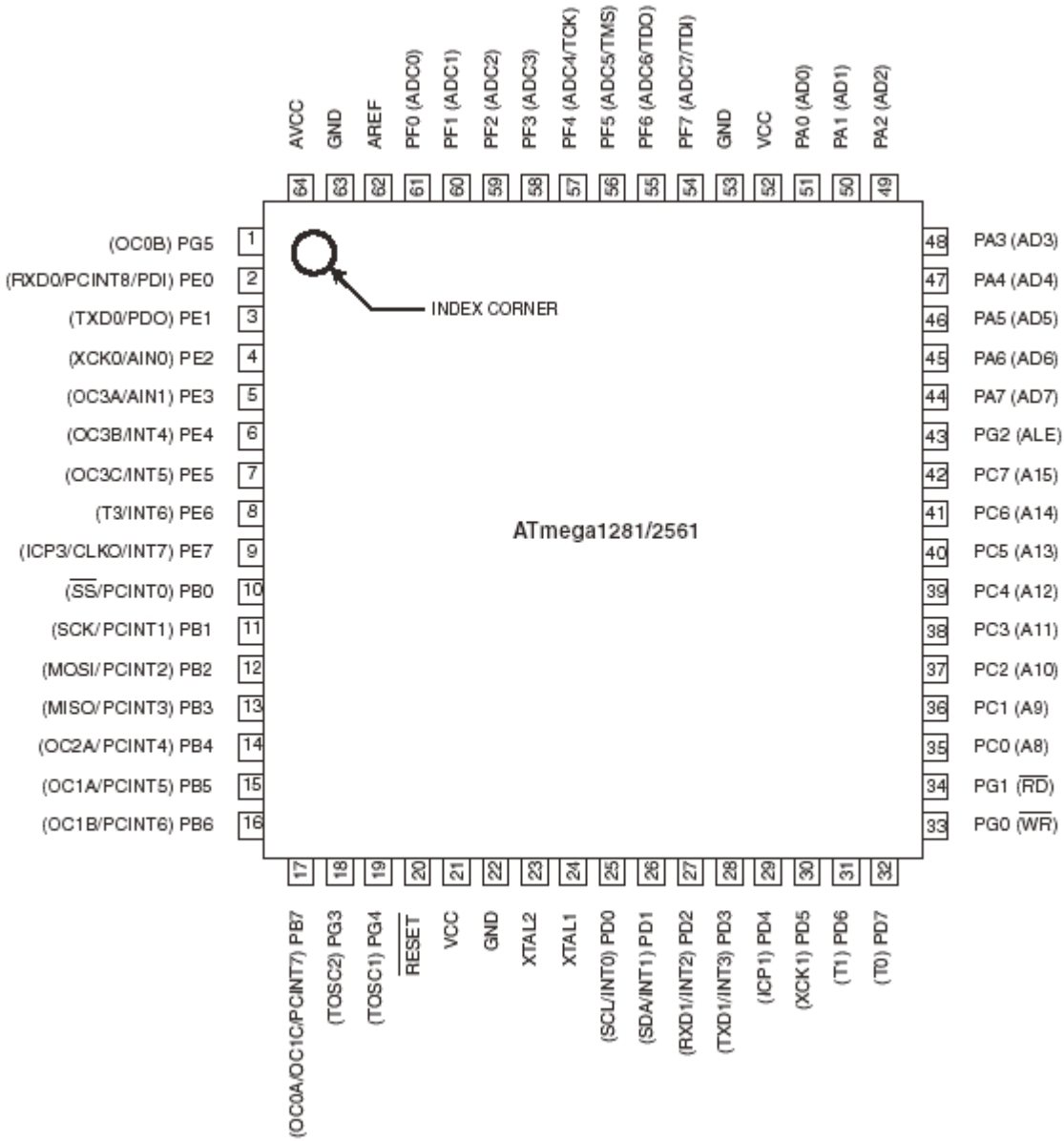
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



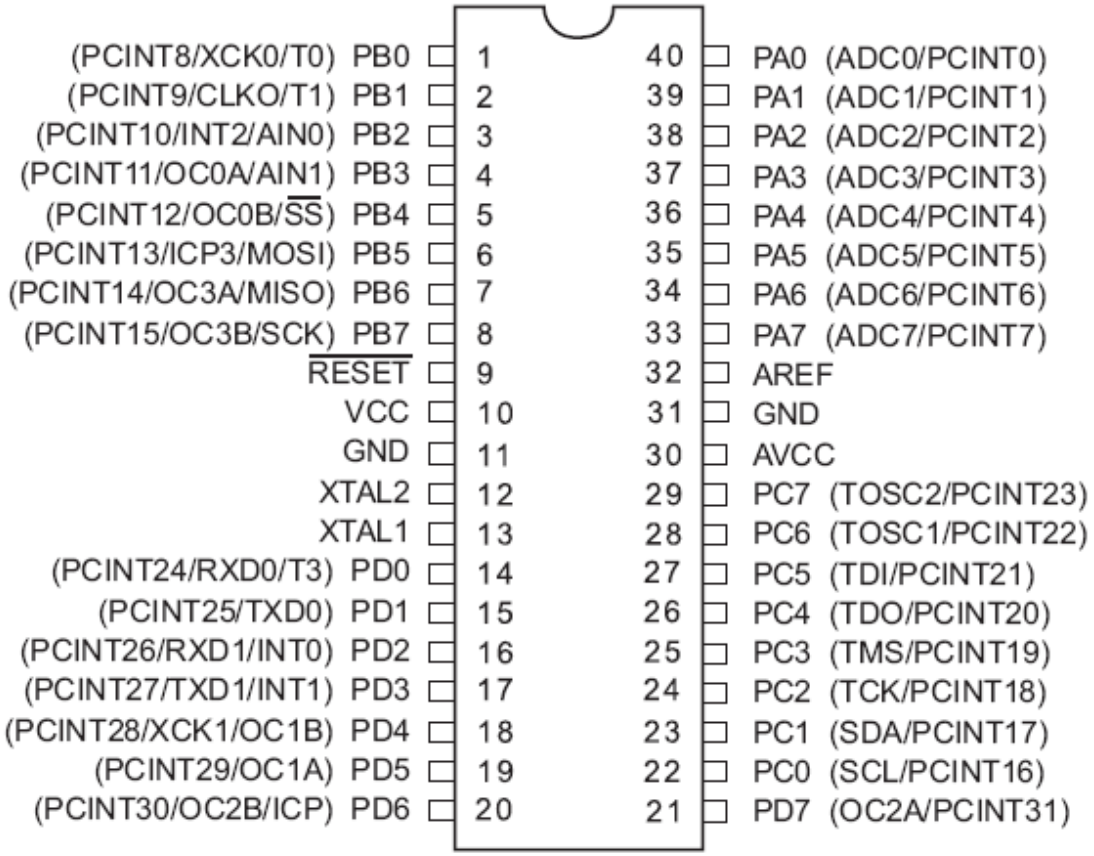
5.5.60 ATMEGA1280



5.5.61 ATMEGA1281



5.5.62 ATMEGA1284

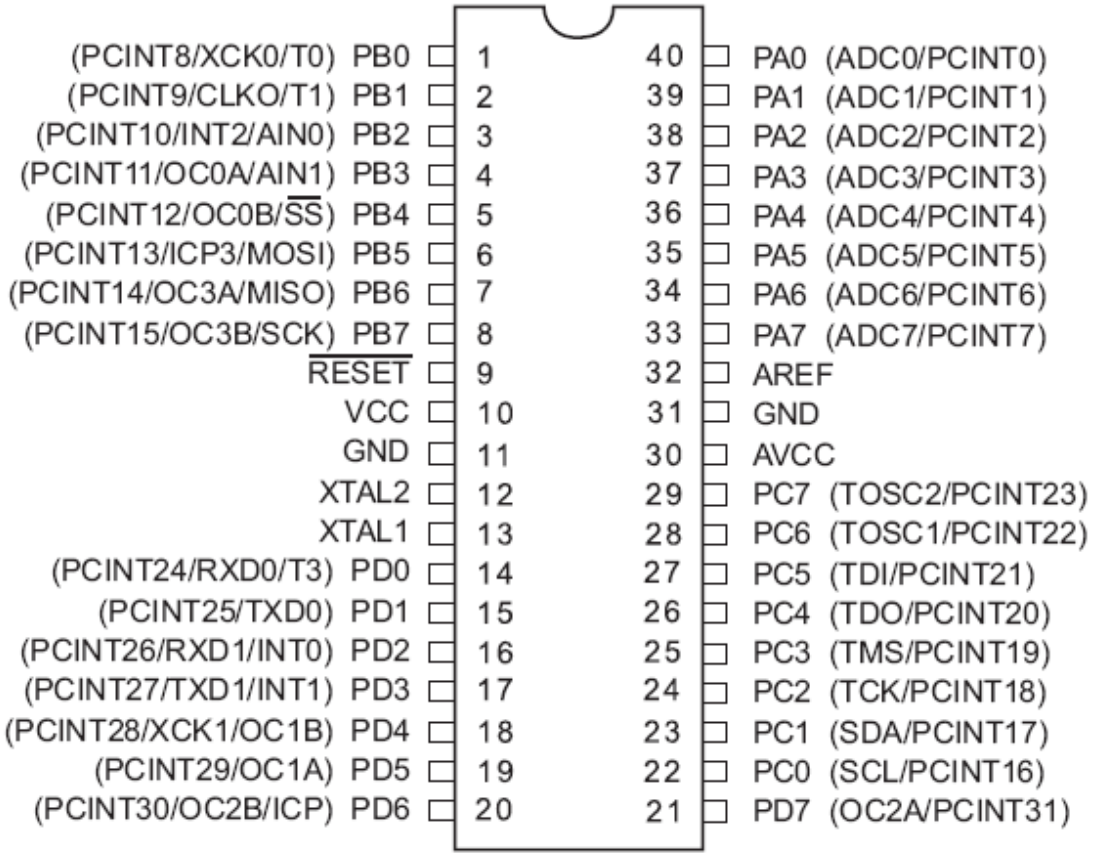


The M1284 seems to have an internal problem where large amounts of serial data can choke the processor.
A capacitor of 100pF on the RX pin to ground can solve this problem.
More info :
http://www.mcselec.com/index2.php?option=com_forum&Itemid=59&page=viewtopic&p=60860#60860

PORTC is connected to AVCC and not VCC. This can cause the brown out detection to trigger.

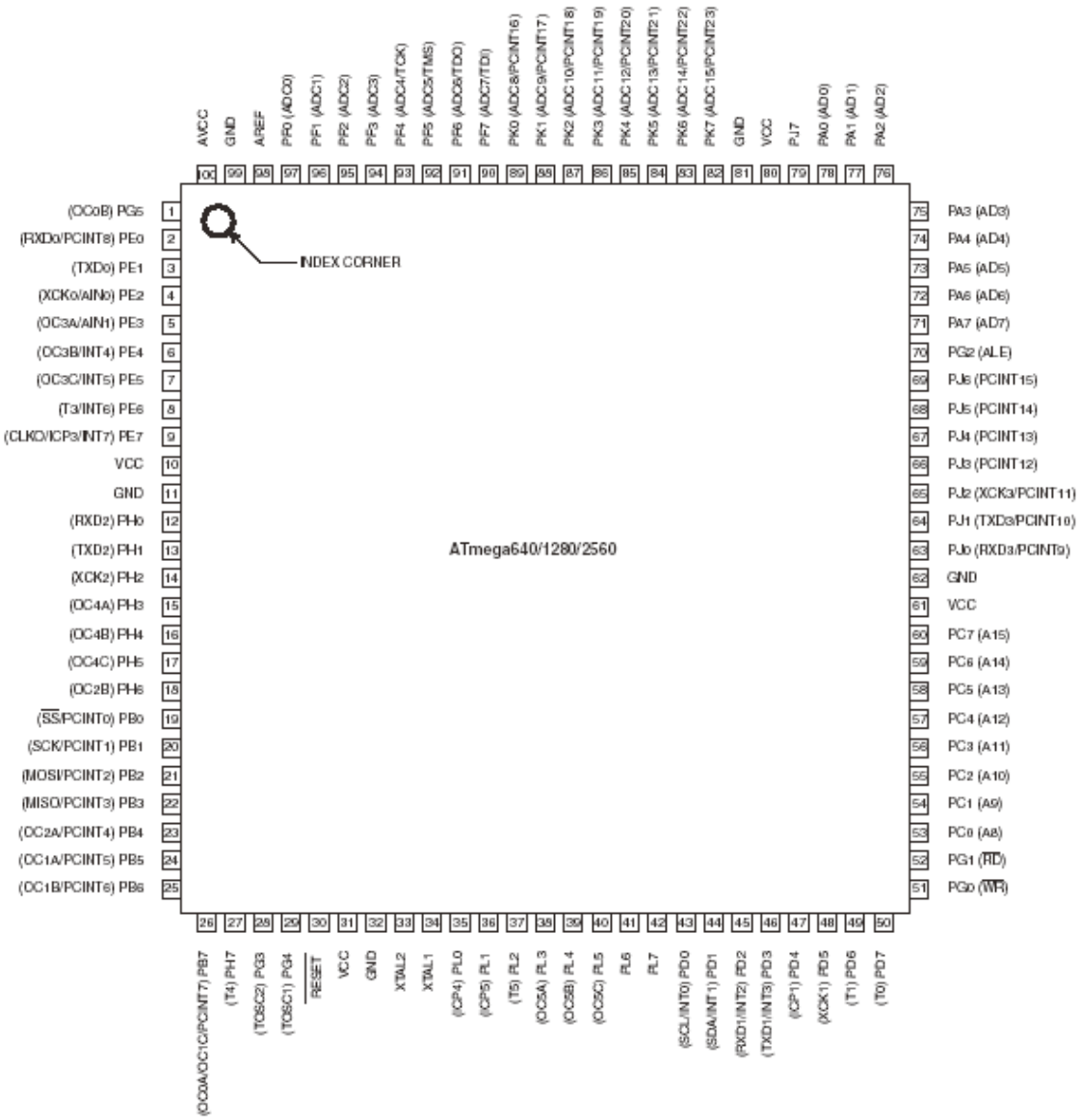
5.5.63 ATMEGA1284P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

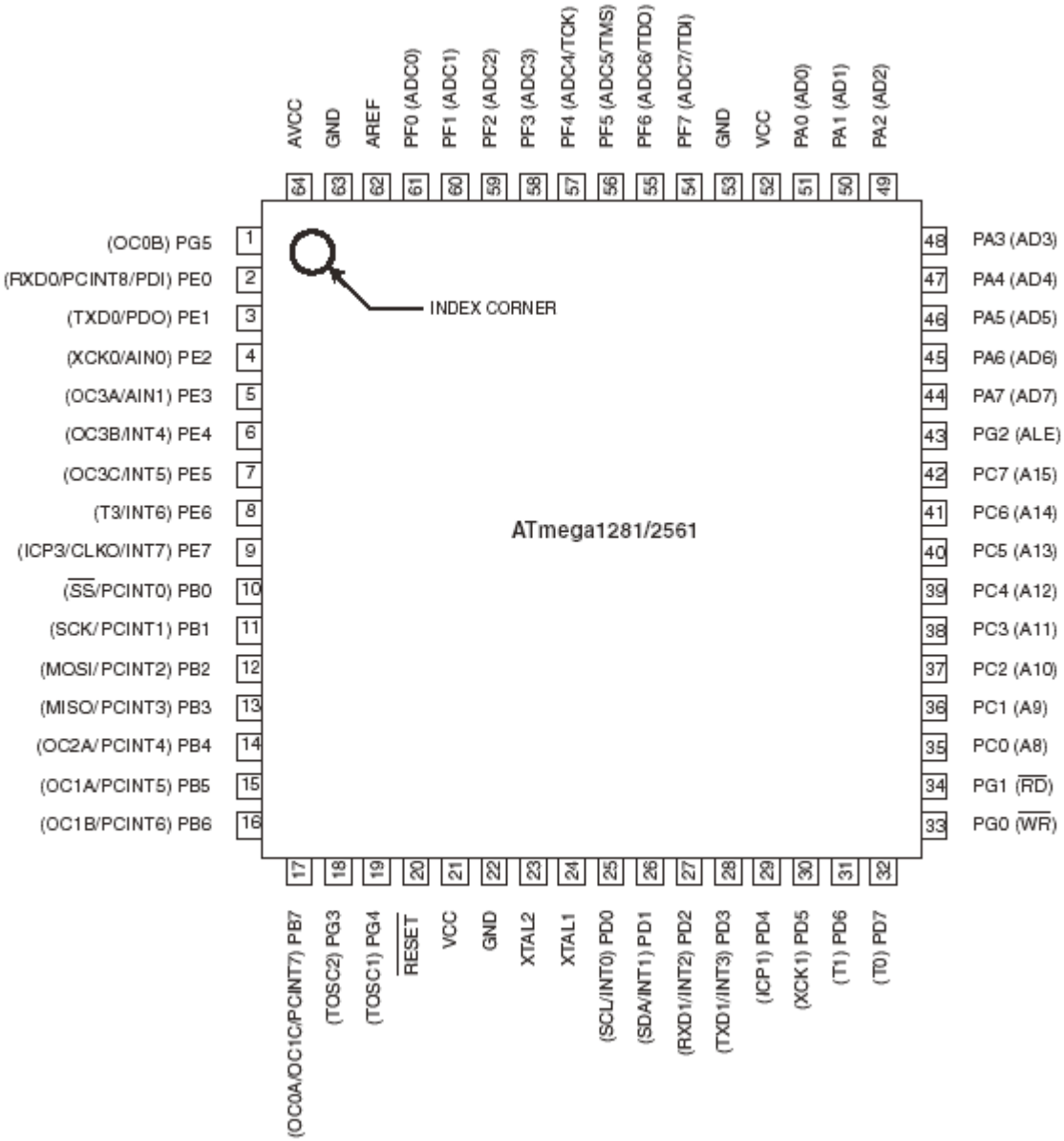


The M1284 seems to have an internal problem where large amounts of serial data can choke the processor.
A capacitor of 100pF on the RX pin to ground can solve this problem.
More info :
http://www.mcselec.com/index2.php?option=com_forum&Itemid=59&page=viewtopic&p=60860#60860

5.5.64 ATMEGA2560

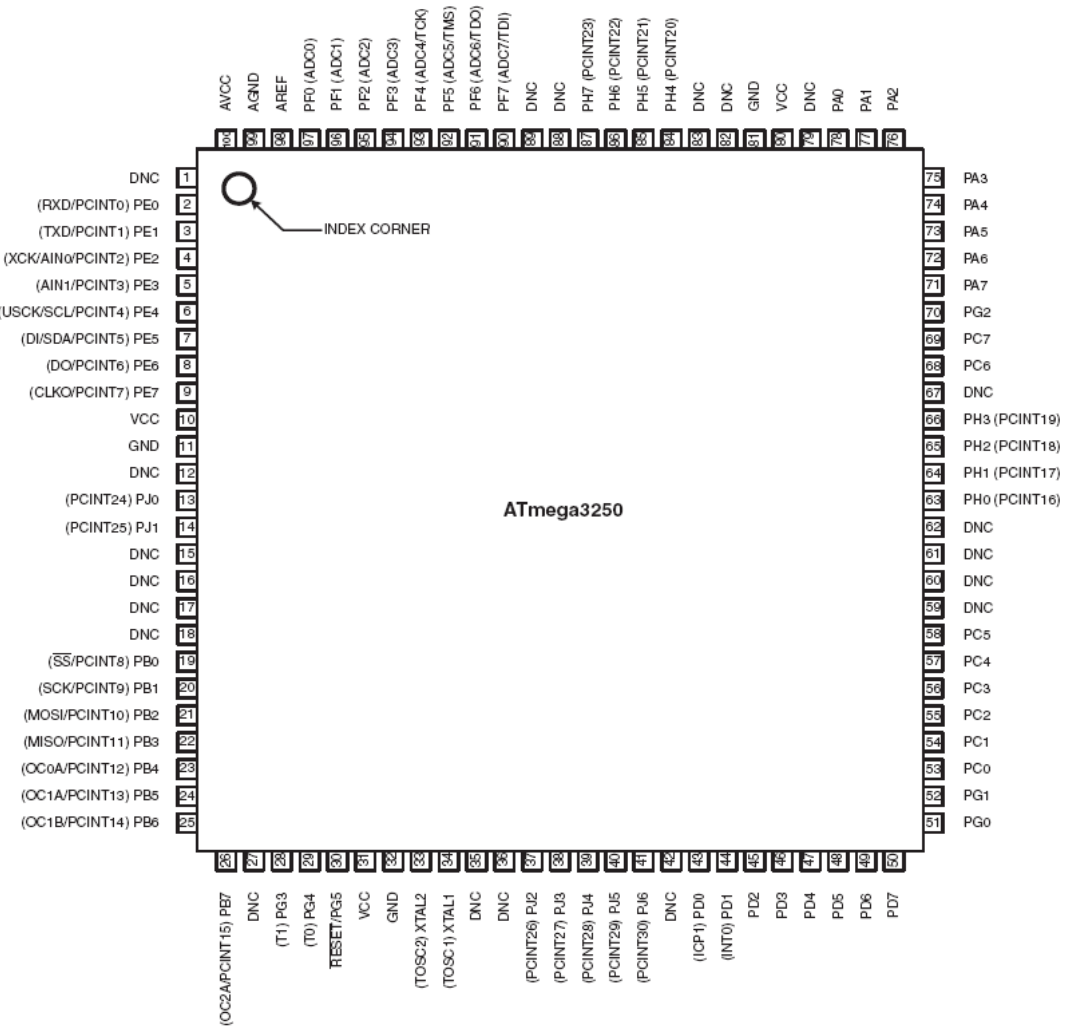


5.5.65 ATMEGA2561



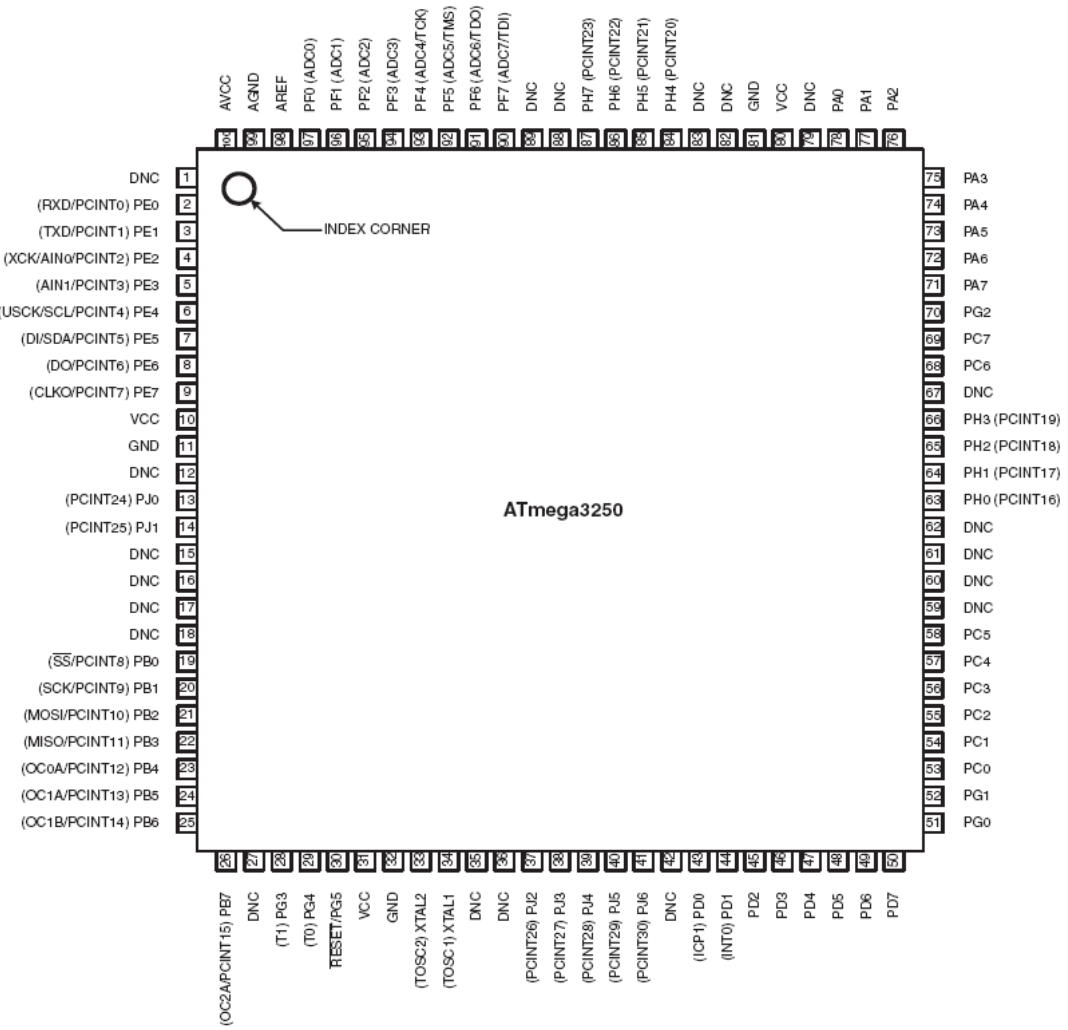
5.5.66 ATMEGA3250P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

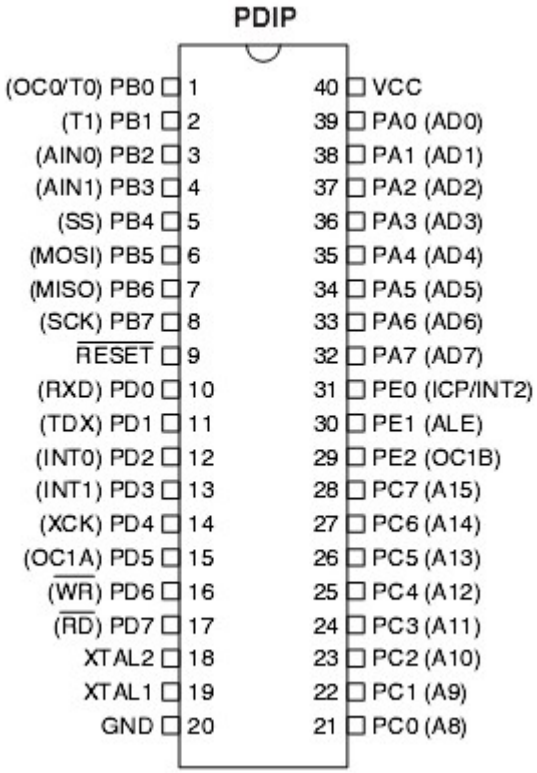


5.5.67 ATMEGA6450P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

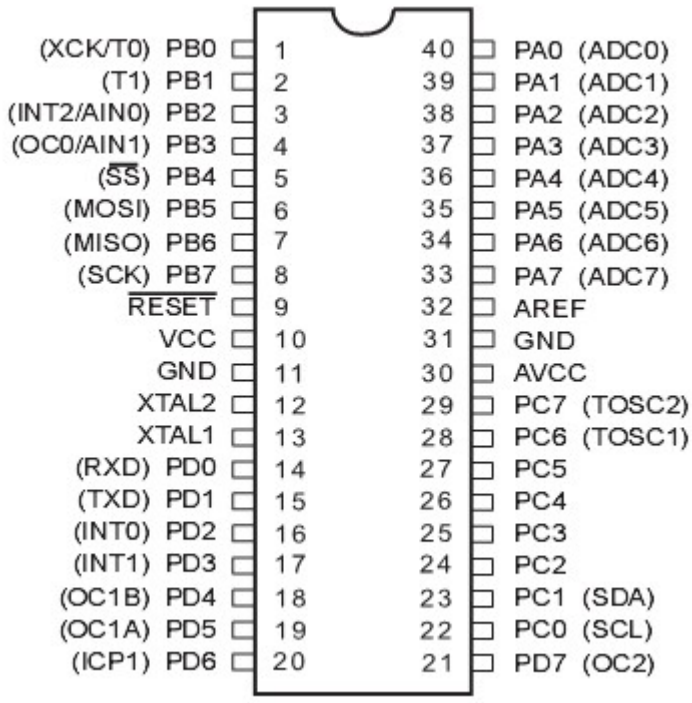


5.5.68 ATMEGA8515



5.5.69 ATMEGA8535

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.6 ATXMEGA

5.6.1 ATXMEGA

The ATXMEGA is a great new chip. It has a lot of hardware on board and a huge amount of hardware registers.
Some changes in the architecture are however breaking compatibility with normal AVR (ATMEGA/ATTINY) processors.

Dont miss the FAQ - ATXMEGA section below !

The ATXMEGA bring a huge amount of interfaces like UART, I2C, SPI, Counter/Timer, 12-Bit Analog Input/Output and also new features like DMA 937 (Direct Memory Access), the Event System 953 or AES hardware encryption/decryption.

Regarding more infos on ATXMEGA ANALOG DIGITAL CONVERTER (ADC) see here:
CONFIG ADCX 867

All ATXMEGA have their registers at the same address. Some chips might not have all registers because the hardware is not inside the chip, but all DAT* files are similar. And all hardware has a fixed offset. This allows to use dynamic code. For example Bascom-AVR can now use a variable for the UART and the code is only needed once because all hardware has a fixed offset.

* DAT files are the register files. The register files are stored in the BASCOM-AVR application directory and they all have the DAT extension. The register file holds information about the chip such as the internal registers and interrupt addresses. The register file info is derived from ATMEL definition files.

The ATXMEGA work with 3.3V so please do not connect something which output 5V to the XMEGA Pin's. Use a Level Shifter for that.
The maximum rating for a ATXMEGA Pin is 3.6 V.
When using the internal 32MHz oscillator you need at least 2.7V Vcc. The internal 32MHz oscillator is stable enough for a lot of applications. The maximum CPU Clock Frequency is 12MHz when using the XMEGA with just 1.6V Vcc.

Read Application Note AVR1012: XMEGA A Schematic Checklist for special considerations in your hardware design.

You can find Bascom samples for ATXMEGA in Bascom-AVR folder:
- General samples ……..\BASCOM-AVR\SAMPLES\XMEGA
- Bootloader samples in ……\BASCOM-AVR\SAMPLES\BOOT
- For chips like xm128a1 in …….\BASCOM-AVR\SAMPLES\CHIPS

Manuals for ATXMEGA: There are 2 manuals available from ATMEL for every ATXMEGA Chip

1. One Family Manual like for example for a ATXMEGA128**A**1 it is Atmel AVR XMEGA **A** Manual
2. Another Manual for the single chips like for example for an ATXMEGA128A1 it is the ATxmega64A1/128A1/192A1/256A1/384A1 Manual. In this Manual you find for example the Alternate Pin Functions. So you can find which Pin on Port C is the SDA and SCL Pin when you want to use the I2C/TWI Interface of this Port.

Beside the Manuals for the ATXMEGA chips there are a lot of Appliaction Notes available on the ATMEL website which explain for example the ATXMEGA Event

System or Direct Memory Access (DMA) and so forth.

What you need to get started with ATXMEGA and BASCOM-AVR

1. The latest Bascom-AVR FULL Version (The Demo Version of Bascom-AVR do not support ATXMEGA).
2. An evaluation board like the Atmel AVR XMEGA® Xplained evaluation kit or any other ATXMEGA evaluation board with PDI (Program and Debug Interface) header.
3. A Programmer like AVRISP MKII or any other PDI or JTAG programmer which support ATXMEGA.
4. Latest AVR-Studio 4.X or 5.X only for setting fuse bits and to flash Bootloader to ATXMEGA.
5. Programming the ATXMEGA can be done direct from BASCOM-IDE. See also LIBUSB 216 for further information.

The most important parts of an Bascom-AVR Program for XMEGA are:

1. The Register file for the chip, crystal init and Stacks

```
$regfile = "xm128a1def.dat"
$crystal = 32000000                    '32MHz
$hwstack = 64
$swstack = 40

$framesize = 64
```

2. Enable and configure the oscillator of your choice:

```
Config Osc = Enabled , 32mhzosc = Enabled              ' enable 2 MHz and 32 MHz internal
oscillators
```

3. Select the oscillator source for the system clock and prescaler (this must match with $crystal = XXXXXX). The following configure the internal 32MHz oscillator as system clock without prescaler so the system clock is 32MHz which match with $crystal = 32000000

```
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
```

4. If you intend to use interrupts you need to configure it before you use the Enable Interrupt command. Bascom-AVR will automatically enable the medium level interrupt when Enable Interrupt is in the code but not the low and high level interrupts.

```
Config Priority = Static , Vector = Application , Lo = Enabled , Med = Enabled , Hi =
enabled
```

5. Also when you want to use EEPROM you need to configure it before you can use it:

```
Config Eeprom = Mapped                 'Setup memory mode for EEPROM in XMEGA
```

6. After this you can add Enable Interrupts and your code in the Main Loop.

```
Do
 'Insert your code
Loop
```

7. End

```
End                                                    'end  program
```

A first simple program with ATXMEGA which toggle an output every second:

```
$regfile = "xm64a3def.dat"
$crystal =  32000000
$hwstack = 40
$swstack = 16
$framesize = 32

Config Osc =  Enabled ,  32mhzosc =  Enabled
Config  Sysclock = 32mhz ,    Prescalea = 1 ,    Prescalebc = 1_1
Config Portb = Output

Do
  toggle Portb.0
  Waitms 1000
loop

end 'end    program
```

FAQ - ATXMEGA

Q: How to set clock source and clock frequency with ATXMEGA ?
A: See CONFIG OSC 1008 and CONFIG SYSCLOCK 1081

Q: Do I need to set the XMEGA fuses to get started ?
A: **No**, to get started there is no need to set the fuses because opposed to megaAVR or ATTINY, where fuses are used to set the clock source and frequency you can set the clock source and frequency in the program by Bascom-AVR.
You also do not need an external clock source. You can use the internal 32MHz or 2MHz RC oscillator as clock source.

Q: Which Interrupt Level (Low, Med, High) is used when I do not specify the priority ?
A: MED is used when the priority is not specified.

Q: Is AVR-DOS supported for XMEGA ?
A: Yes, see AVR-DOS File System 1794

Q: What else is supported (Status: Bascom-AVR 2.0.7.6) where users of Bascom-AVR forum asked for ?
A: Following list:
- 1-WIRE, EEPROM, XRAM, Event System, Config Servos, AVR-DOS SDHC driver, DCF77, RTC32,
- INP/OUT support Xmega huge memory, xmega LCD simulation, dmxslave, RS-485, pulsein, pulseout, DMA
- Bascom Simulator, Event System, getrc5, CONFIG POWER_REDUCTION, buffered serial output (com1-com4)
- virtual portmap config, config tcXX for xmega timers, xmega comparator, $forcesofti2c will force the xmega to use software i2c
- AES encyption/decryption, xmega TWI, SPI, UART

Q: Where do I find which pin of ATXMEGA is SDA and SCL ?
A: There are 2 manuals available from ATMEL
1. One Family Manual like for example for a ATXMEGA128A1 it is Atmel AVR XMEGA A Manual
2. Another Manual for the single chips like for example for an ATXMEGA128A1 it is the

ATxmega64A1/128A1/192A1/256A1/384A1 Manual. **In this Manual you find for example the**
Alternate Pin Functions. So you can find which Pin on Port C is the SDA and SCL Pin when you want to
use the I2C/TWI Interface of this Port.

Q: How to program/flash an ATXMEGA ?
A: There is no ISP programming support. Only JTAG and PDI is supported. Of course the MCS Bootloader can be used but you need to program the chip first with for example an AVRISP MKII programmer. After this programming the
ATXMEGA can be done direct from BASCOM-IDE
Important is also that the AVRISP MKII programmer need 3.3V supply voltage from the Target.

Q: Is there a special boot loader source code I can use for ATXMEGA ?
A: There are example boot loader in following Bascom-AVR folder (C:\.....\BASCOM-AVR\SAMPLES\BOOT)
like ATXMEGA32A4 or ATXEMGA128A1. For other ATXMEGA Chips the source code can be easily adapted.

Q: What is the Program and Debug Interface (PDI) ?
A: The Program and Debug Interface (PDI) is an Atmel proprietary interface for external programming and on-chip
debugging of a device.

"The XMEGA doesn't have the SPI based In-System Programming (ISP) interface for
external programming, which has been used for megaAVR. Nor does it have the debugWIRE interface. These have been replaced by a two wire "Programming and Debugging Interface" (PDI)." [from Atmel App Note AVR1005]

Q: How to read/write from/to ATXMEGA Register ?
A: If you want or need to write or read ATXMEGA Registers direct you just need to find the name by using the
ATXMEGA DAT file.
For example if you want to read the ATXMEGA Revision there is the register **Mcu_revid** in the DAT file.
The DAT files can be found in the BASCOM-AVR folder.
Take care with protected registers. Before you can write to this registers you need to release it. An example for this is the Software Reset.

Q: How to initiate a software Reset of an ATXMEGA ?
A: Before you can write the Software Reset Bit you need to release the write protection for this bit and register.

```
'enable change of protected Registers for following 4 CPU Instruction Cycles
CPU_CCP = &HD8
'Initiate Software Reset by setting Bit0 of RST_CTRL Register
Rst_ctrl.0 = 1                    'When this bit is set a software reset occur
```

Q: How are External Interrupts (Port Interrupt) used with ATXMEGA ?
A: Each XMEGA port (like PortA or PortF) with pin's from Pin0....Pin7 has 2 interrupts (INT0 and INT1). So there is INT0 and INT1 available for every port.
Port Interrupts must be enabled before they can be used like for example Int1 on PortA = **PORTA_INT1**

Steps to use PortE.0 as Port Interrupt where INT0 is used:

1. enable the INT0 Interrupt on Port E

```
On Porte_int0 Port_e_int0__isr
Enable Porte_int0 , Lo 'Enable this Interrupt as
Lo Level Interrupt
Enable Interrupts
```

2. Config Porte.0 as Input:

```
Config Pine.0 = Input 'Set PINE.0 as Input
```

3. Configure the reaction:

```
Config Xpin = Porte.0 , Outpull = Pullup , Sense = Falling 'enable Pull up and
reaction on falling edge
```

4. Set the interrupt mask (which pin will be activated to generate an INT0 in this case):

```
Porte_int0mask = &B0000_0001 'include PIN0 in INT0 Mask
```

1 = Pin is activated for interrupt
0 = Pin is deactivated for interrupt

You could also set more pin's as activated (set to 1) but then you need to check in the interrupt service routine which pin was the root cause for generating the interrupt.

5. The Interrupt Service Routine:

```
'Port E INT0 Interrupt Service Routine
Port_e_int0__isr:

  'do something....

Return
```

Additional info for Port Interrupts:
For asynchronous sensing, only port pin 2 on each port has full asynchronous sense support. This means that for edge
detection, pin 2 will detect and latch any edge and it will always trigger an interrupt request. The other port pins have
limited asynchronous sense support [ATXMEGA A Manual].

See also below for an example for Port Interrupt (External Interrupt)

Q: For what do I need Fuse Bits (Fuses) with ATXMEGA ?
A: *"The Fuses are used to set important system function and can only be written from an external*
programming interface. The application software can read the fuses. The fuses are used to configure
reset sources such as Brown-out Detector and Watchdog, Start-up configuration, JTAG
enable and JTAG user ID, Bootloader…..An unprogrammed fuse or lock bit will have the value one, while a
programmed flash or lock bit will have the value zero.." [ATXEMGA A Manual]

Q: How to write to 16-Bit (Word) Register of ATXMEGA ?
A: You do not care about the 16-Bit (Word) Register because the compiler will handle this for you automatic.
For this you can find the [WIO] (Word IO) Section in the DAT file. You can only write direct to 16-Bit Register over the defined register in the [WIO] section of the DAT file

If there is a need to manual write/read to/from 16-Bit register you always need to write/read the LOW Byte (LSB)
and then the HIGH Byte (MSB).

Q: Is there also a linear memory architecture as with ATMEGA or ATTINY AVR's ?
A: The power of the AVR is/was the linear memory architecture. In the ATXMEGA this has been changed : the registers are placed into a separate address space. This makes code like this fail:

```
Clr r31
Ldi r30,10   ; point to register R10
Ld r24,z+    ; load value from R10 and inc pointer
```

Code like LDS r16, 0 will not load the content of register R0 either with Xmega
If your ASM code contains such code you need to rewrite it.

Q: Is the Bascom-AVR Demo version supporting ATXMEGA ?
A: ATXMEGA is not available in PDIP. This means that the ATXMEGA is not really suited for hobby projects.
As a result, the DEMO version does not support the ATXMEGA.

Q: For what are Virtual Port Registers good for ?
A: “Virtual port registers allow for port registers in the extended I/O memory space to be mapped virtually
in the I/O memory space. When mapping a port, writing to the virtual port register will be
the same as writing to the real port register. This enables use of I/O memory specific instructions
for bit-manipulation, and the I/O memory specific instructions IN and OUT on port register that
normally resides in the extended I/O memory space. There are four virtual ports, so up to four
ports can be mapped virtually at the same time. The mapped registers are IN, OUT, DIR and
INTFLAGS.” [from ATXMEGA A Manual]

Q: On which port of ATXMEGA can I find the COM1.....COM8 ?
A: See table below:

```
COM1 --> Usartc0
COM2 --> Usartc1
COM3 --> Usartd0
COM4 --> Usartd1
COM5 --> Usarte0
COM6 --> Usarte1
COM7 --> Usartf0
COM8 --> Usartf1
```

Q: Is Serialin and Serialout supported for UART Interfaces above COM4
A: Yes since version 2077 all 8 UARTS support buffered serial input and output.

For ATXMEGA the first 4 UARTS can use for example serialin:
SERIALIN : first UART/UART0 --> COM1
SERIALIN1 : second UART/UART1 --> COM2
SERIALIN2 : third UART/UART2 --> COM3

SERIALIN3 : fourth UART/UART3 --> COM4
SERIALIN4 : fourth UART/UART4 --> COM5
SERIALIN5 : fourth UART/UART5 --> COM6
SERIALIN6 : fourth UART/UART6 --> COM7
SERIALIN7 : fourth UART/UART7 --> COM8

For example with an ATXMEGA128A1 you get 8 UARTS:
Every of the 8 USART's has for example a Receive Interrupt which you can use to analyze incoming data:

ATXMEGA128A1 Receive Interrupts:
COM1 --> Usartc0_rxc
COM2 --> Usartc1_rxc
COM3 --> Usartd0_rxc
COM4 --> Usartd1_rxc
COM5 --> Usarte0_rxc
COM6 --> Usarte1_rxc
COM7 --> Usartf0_rxc
COM8 --> Usartf1_rxc

In the interrupt routine you need to use the inkey(#X) function because inkey(#X) is reading the data register and
therefore reset the interrupt flag. Without reading the data register or resetting the interrupt flag manual the
interrupt will fire again and again.

Example the interrupt routine:

```
Rxc_isr:
    Rs232 = Inkey( #1)
    'do something with the data
Return
```

Q: How to get the reason for a reset of ATXMEGA (like power on, watchdog or software rest)
A: There is a special register for that **RST_STATUS** you can read and analyze.
You can also read R0 register using GetReg(R0) function : myvar=GetReg(r0). You need to do this early in your code.

Q: How to auto calibrate the internal 2MHz and 32MHz Oscillators during runtime ?
A: The automatic runtime calibration of internal oscillators is activated by enabling the DFLL (Digital Frequency-locked Loops) and autocalibration.

```
'The internal 32.768 KHz Oscillator is used for calibration
 Osc_dfllctrl = &B00000000
 'Enable DFLL and autocalibration
 Set Dfllrc32m_ctrl. 0
```

Additional hint from ATMEL for some chip revisions:
"....**Both DFLLs and both oscillators has to be enabled for one to work**
In order to use the automatic runtime calibration for the 2 MHz or the 32 MHz internal oscillators,
the DFLL for both oscillators and both oscillators has to be enabled for one to work.
Problem fix/Workarund
Enabled both DFLLs and oscillators when using automatic runtime calibration for one of the internal oscillators....."

Q: How many Flash and EEPROM Write/Erase Cycles can be done with ATXMEGA ?
A: One write cycle consists of erasing a sector, followed by programming the same sector. You can find the maximum numbers for an ATXMEGA128A1 here:

XMEGA Flash and EEPROM Write/Erase Cycles:

For ATxmega128A1 devices

Flash:
25°C - 10K Write/Erase cycles
85°C - 10KWrite/Erase cycles

EEPROM:
25°C 80K- Write/Erase cycles
85°C 30K- Write/Erase cycles

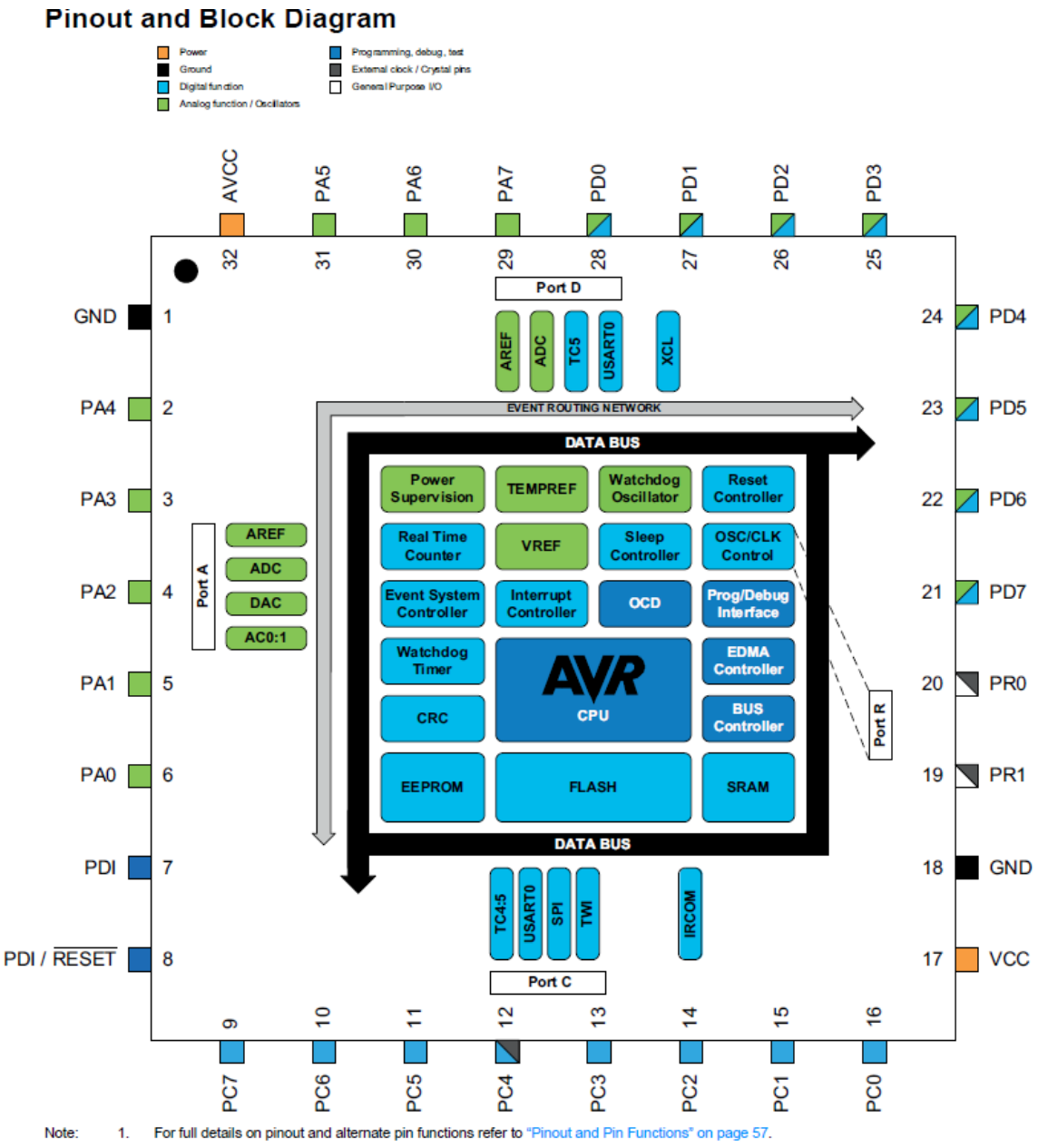
Example for XMEGA Port Interrupt (External Interrupt) from user hzz:

```
'-----------------------------------------------------------------------------------------------------------------
'         Configuring external interrupts with XMEGA
'      Tested OK with BASCOM 2.0.7.5 and an XMEGA128A3 board by Chip45
'       14-aug-2012

'-----------------------------------------------------------------------------------------------------------------
'_____________________________________________________________________________
$regfile = "xm128a3def.dat"
$hwstack = 256
$swstack = 128
$framesize = 128
'_____________________________________________________________________________
'      For 16MHz crystal
$crystal = 32000000
Config Osc = Disabled , Extosc = Enabled , Range = 12mhz_16mhz , Startup = Xtal_1kclk ,
32khzosc = Enabled
'  Set PLL OSC conditions:
Osc_pllctrl = &B1100_0010                                   '  reference external
oscillator, set the PLL' multiplication factor to 2 (bits 0 - 4)
Set Osc_ctrl.4                                              '  Enable PLL Oscillator
Bitwait Osc_status.4 , Set                                  '  wait until the pll clock
reference source is stable
Clk_ctrl = &B0000_0100                                      '  switch system clock to pll
Config Sysclock = Pll , Prescalea = 1 , Prescalebc = 1_1
'_____________________________________________________________________________
'    Setup:
Led1 Alias Portd.0 : Config Portd.0 = Output : Led1 = 1

'  Each XMEGA port has two interrupt vectors: INT0 and INT1.
'  For example, the XMEGA A3 has 7 ports: A, B, C, D, E, F, each with 8 pins ranging from
pin0 to pin7, and port R, with just two pins, normally used by an external XTAL.
'  Therefore up to 12 pins can be used as external interrupts with an XMEGA A3 (or up to
14 if an external XTAL is not used and PORTR.0 and PORTR.1 are free)
'  For each port, any two pins can be defined as an external interrupt source as follows:
'                The following example configures the following ports as external interrupt
sources:
'                  PORTB.0   using interrupt vector INT0 of PORTB
'                  PORTB.3   using interrupt vector INT1 of PORTB
'                     (no more port B pins can be defined as external interruopt sources )
'                  PORTA.3   using interrupt vector INT0 of PORTA

'     1) Set the Interrupt Service Routines Labels for the interrupt vectors used and
'                enable interrupts setting the intrerrupt level:
          On    Portb_int0    B0_b0_isr : Enable    Portb_int0 , Hi
          On    Portb_int1    B1_b3_isr : Enable    Portb_int1 , Lo
          On    Porta_int0    A0_a3_isr : Enable    Porta_int0 , Lo
'             I choose the label names to indicate the interruopt vector used and the pin that
will be assigned
'                next. For instance B1_b3_isr: uses INT vector 1 of port B assigned to pin b3
```

```
' 2) Config pins as inputs and define what should cause the interrupt: low level, hi
level, or transitions: rising, falling or both
          Config Portb.0 = Input : Config Xpin = Portb.0 , Sense = Rising
          Config Portb.3 = Input : Config Xpin = Portb.3 , Sense = Falling
          Config Porta.3 = Input : Config Xpin = Porta.3 , Sense = Both
'             Three switches are connected these pins, PORTB.0, PORTB.3 and PORTA.3, for testing

' 3) Assign pins to interrupt vectors:
        Portb_int0mask = &B0000_0001                    ' Assign pin b0 to Portb_int0
        Portb_int1mask = &B0000_1000                    ' Assign pin b3 to Portb_int1
        Porta_int0mask = &B0000_1000                    ' Assign pin a3 to Porta_int0

'         Notice that more than one pin can be assigned to the same vector. For instance, we
could have
'           written:
'       PROTB_INT0MASK = &B0000_1001                    ' Assign pin b0 and b3 to
Portb_int0
'         In this case, both "b0" and "b3" pins will result in executing the same ISR (If
possible, the ISR could
'           distinguish which pin has produced the interrupt and execute a different code.
This is a way of
'           having more than two external interrupt sources per port). Another way will be
assigning other
'           pins to event channels.

' 4) Write the Interrupt service routines (locate then after the do loop where they will
not be
'           executed except when they are called)
'           B0_b0_isr:
'               ' Do whatever at each rising edge of pin b0
'           Return
'
'           B1_b3_isr:
'               ' Do whatever at each falling edge of pin b3
'           Return
'
'           A0_a3_isr:
'               ' Do whatever at each rising and falling edges of pin a3
'           Return

'5) Don't forget to enable interrupts and config priorities
        Enable Interrupts
        Config Priority = Static , Vector = Application , Lo = Enabled , Med = Enabled ,
Hi = Enabled
'-------------------------------------------------------------------------------
Do
    ' No need to do anything here
Loop
'-------------------------------------------------------------------------------

B0_b0_isr:                                              ' The switch connected to
PORTB.0 will toggle the LED each time it goes HI
    Toggle Led1
Return

B1_b3_isr:                                              ' The switch connected to
PORTB.3 will toggle the LED each time it goes LO
    Toggle Led1
Return

A0_a3_isr:                                              ' The switch connected to
PORTA.3 will toggle the LED each time it goes LO or HI
    Toggle Led1
Return
```

5.6.2 ATXMEGA8E5

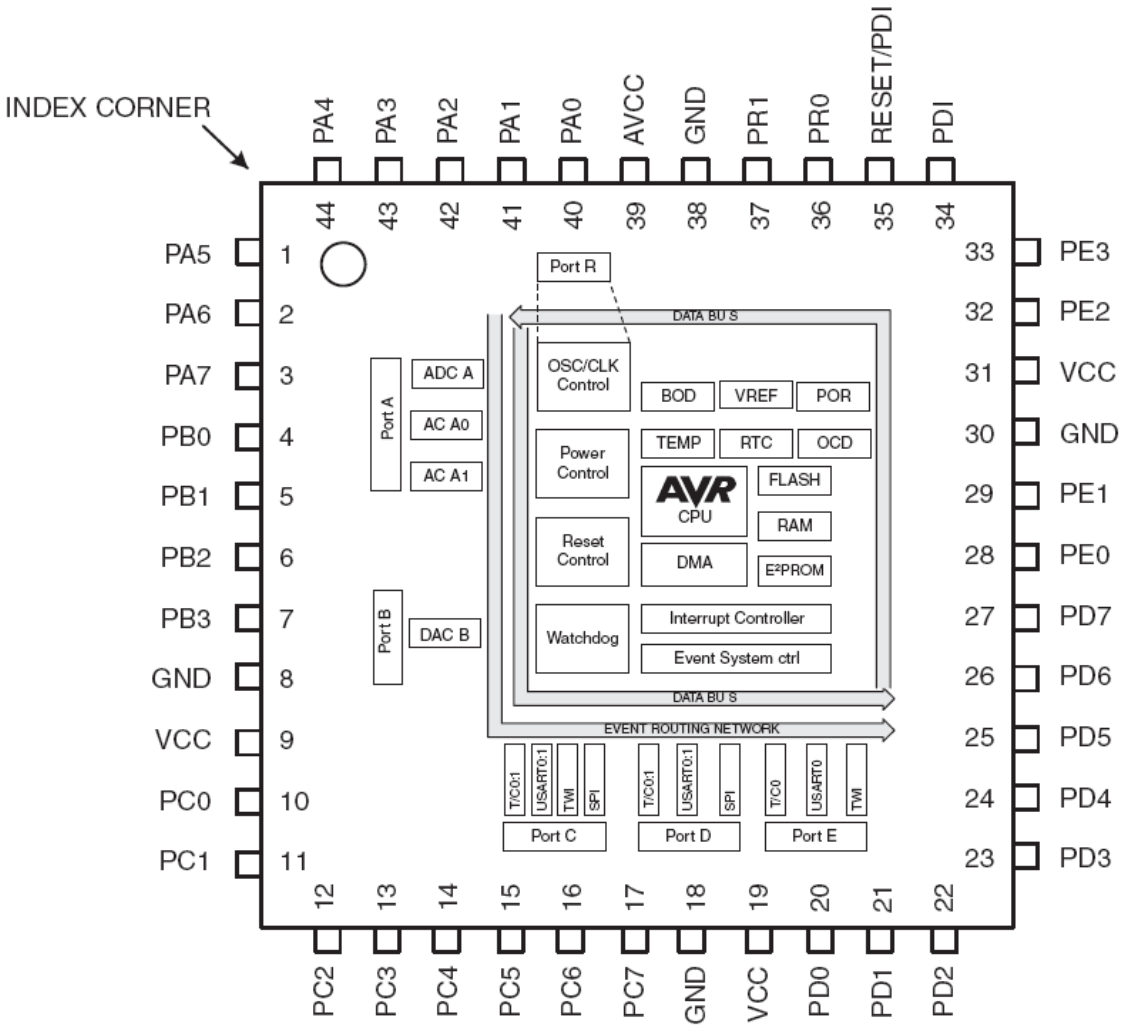
- The XMEGA E series requires that you reset the interrupt yourself. For example : TCC4_INTflags.0=1 'clear OV flag
- The ERASE_APP NVM command (&H20) erases the complete flash, thus the boot space included. Use &H25 instead to erase and write a page.
- There is a fixed map for the virtual ports :
  - VPORT0 - Virtual port A
  - VPORT1 - Virtual port C
  - VPORT2 - Virtual port D
  - VPORT3 - Virtual port R
- CONFIG XPIN slewrate is for the whole port, not for an individual pin



Note: 1. For full details on pinout and alternate pin functions refer to "Pinout and Pin Functions" on page 57.

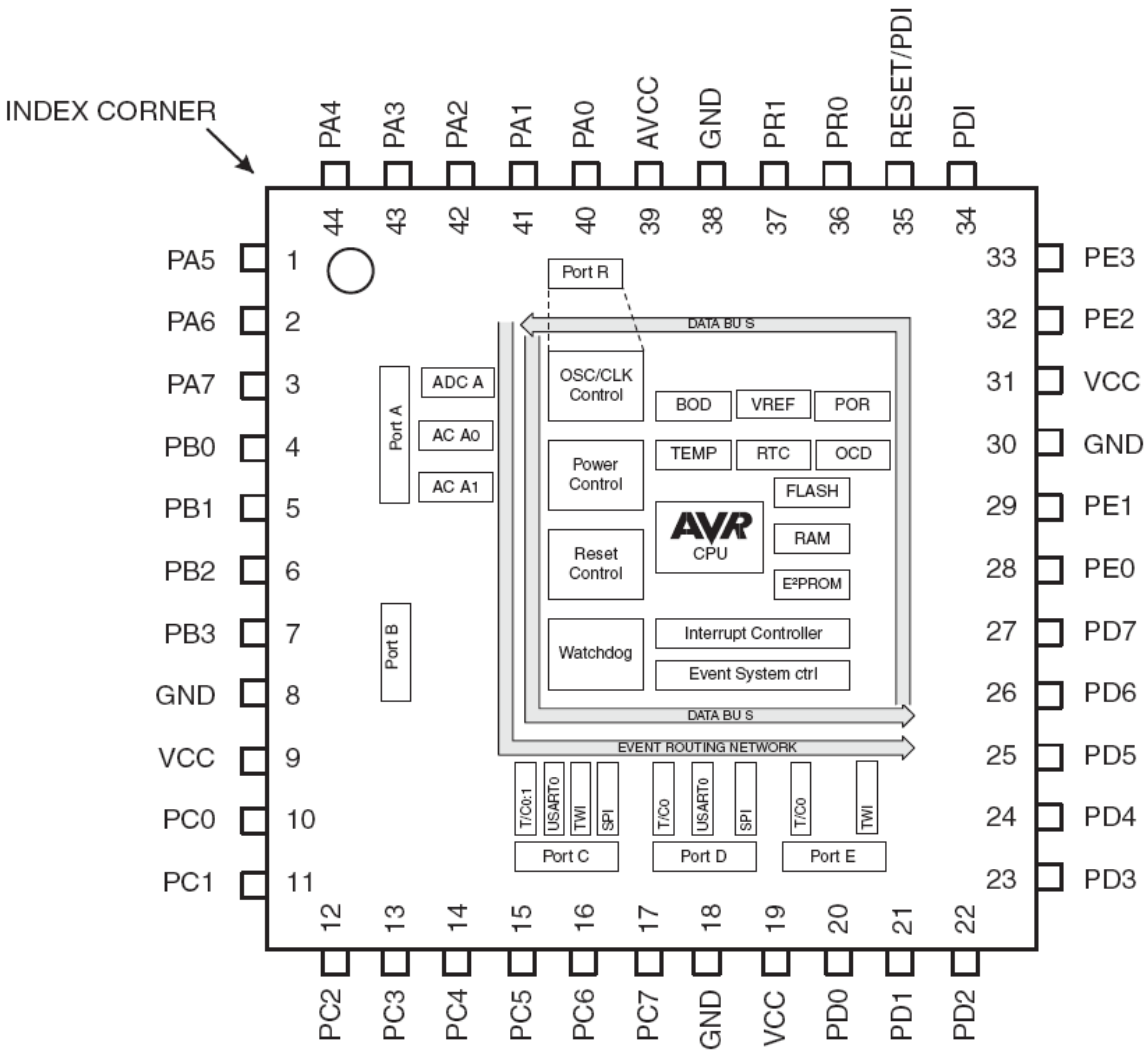
5.6.3 ATXMEGA16A4

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xmega.



5.6.4 ATXMEGA16D4

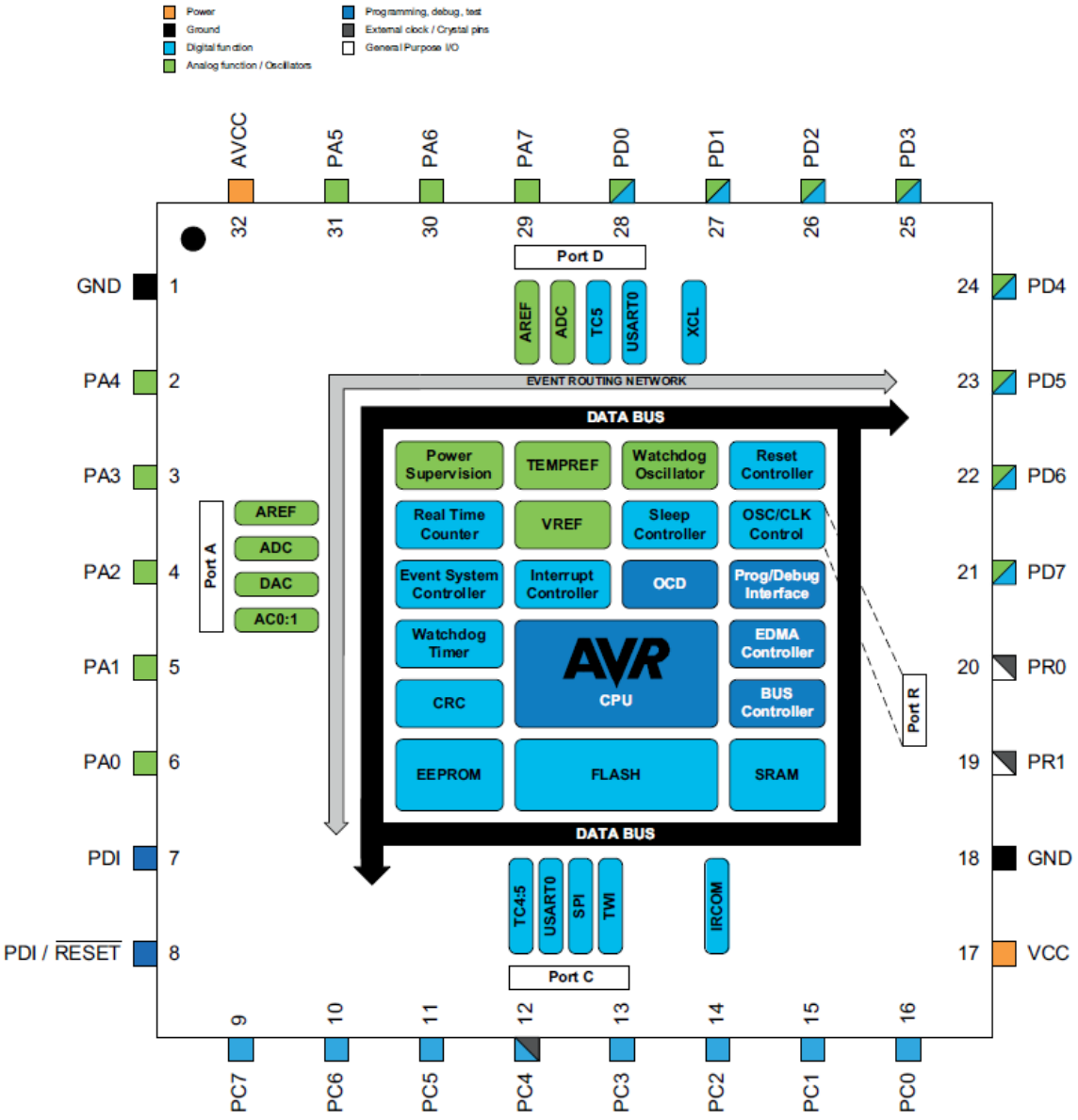
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xmega.



5.6.5 ATXMEGA16E5

- The XMEGA E series requires that you reset the interrupt yourself. For example : TCC4_INTflags.0=1 'clear OV flag
- The ERASE_APP NVM command (&H20) erases the complete flash, thus the boot space included. Use &H25 instead to erase and write a page.
- There is a fixed map for the virtual ports :
  - VPORT0 - Virtual port A
  - VPORT1 - Virtual port C
  - VPORT2 - Virtual port D
  - VPORT3 - Virtual port R
- CONFIG XPIN slewrate is for the whole port, not for an individual pin

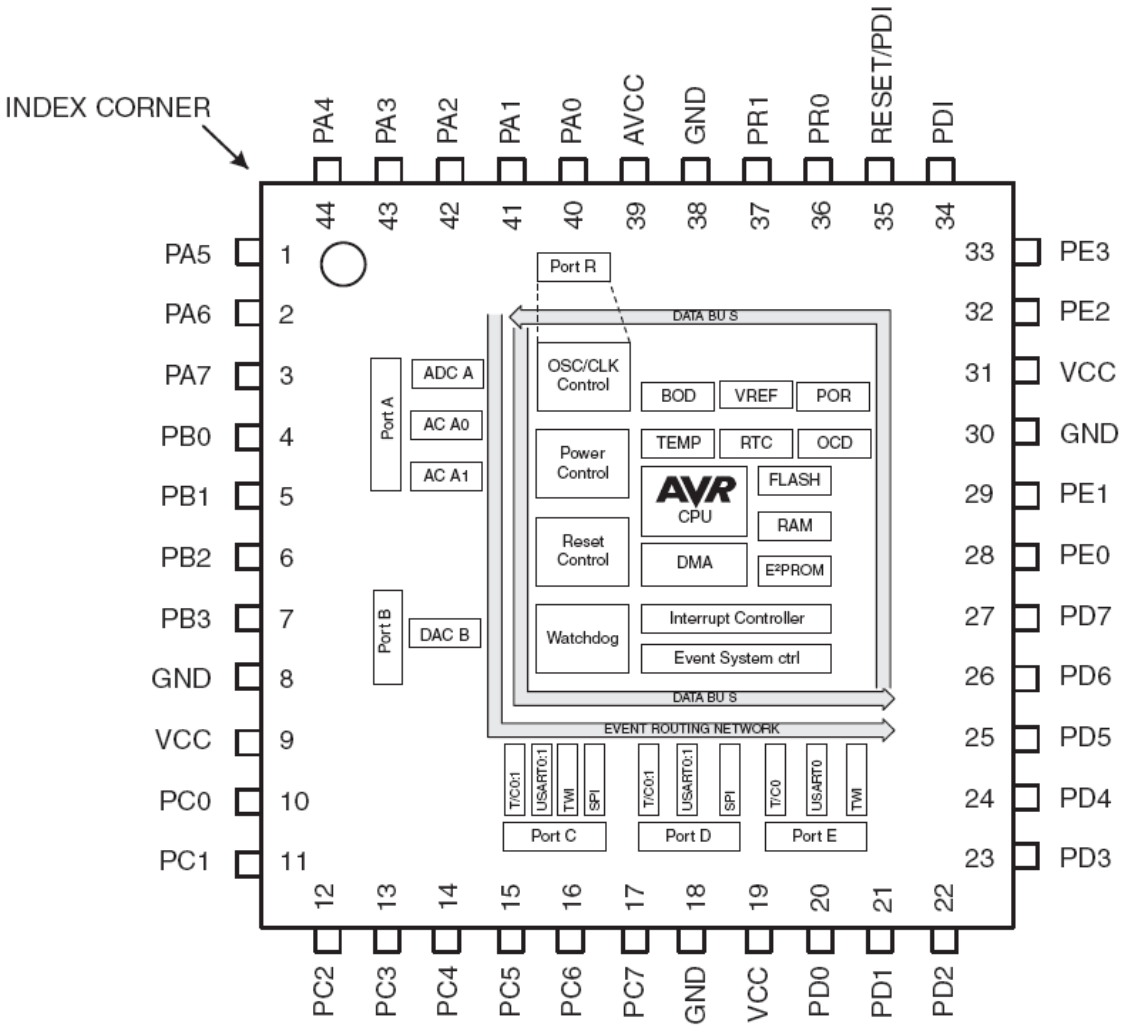
Pinout and Block Diagram



Note: 1. For full details on pinout and alternate pin functions refer to "Pinout and Pin Functions" on page 57.

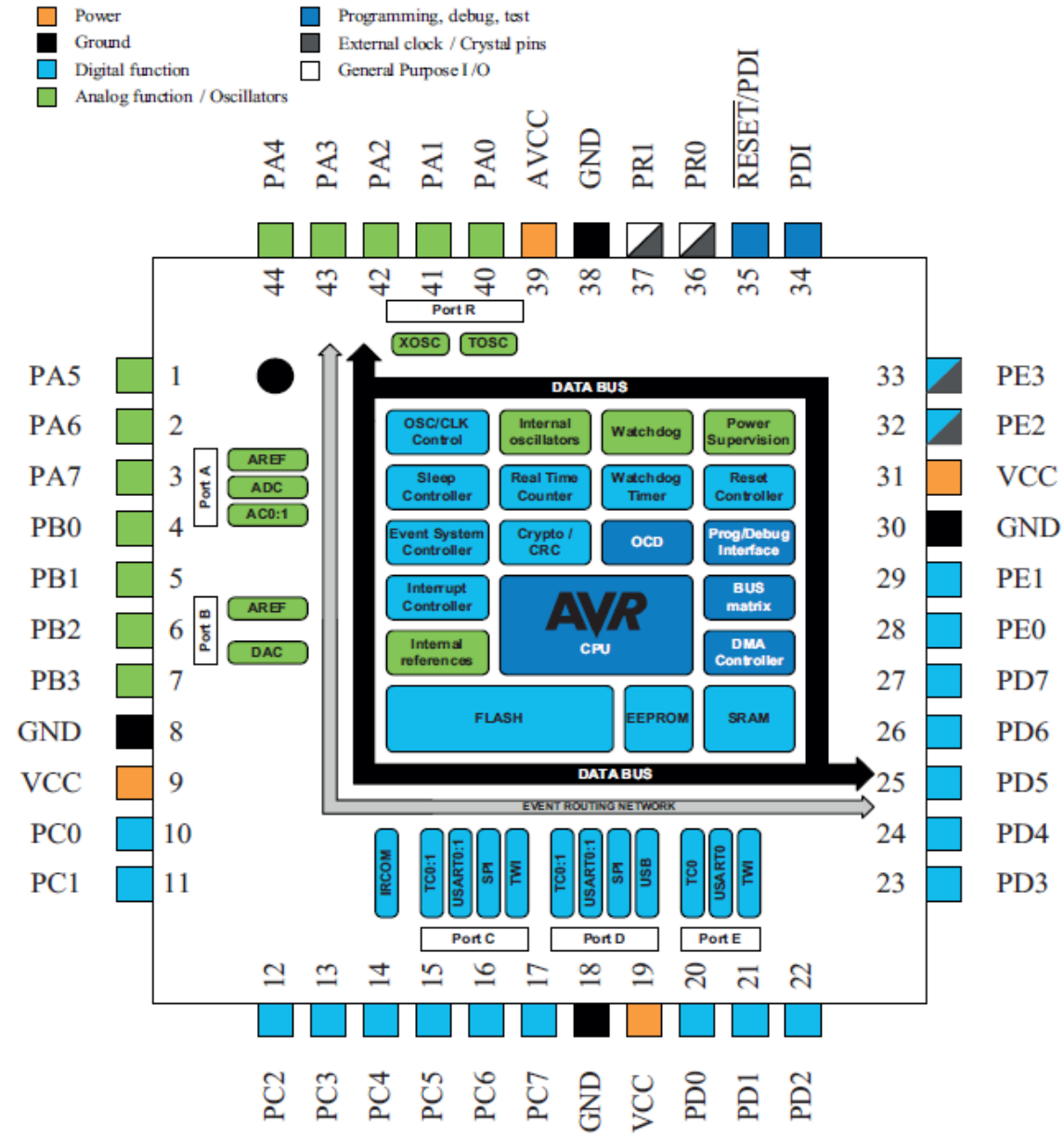
5.6.6 ATXMEGA32A4

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xmega.



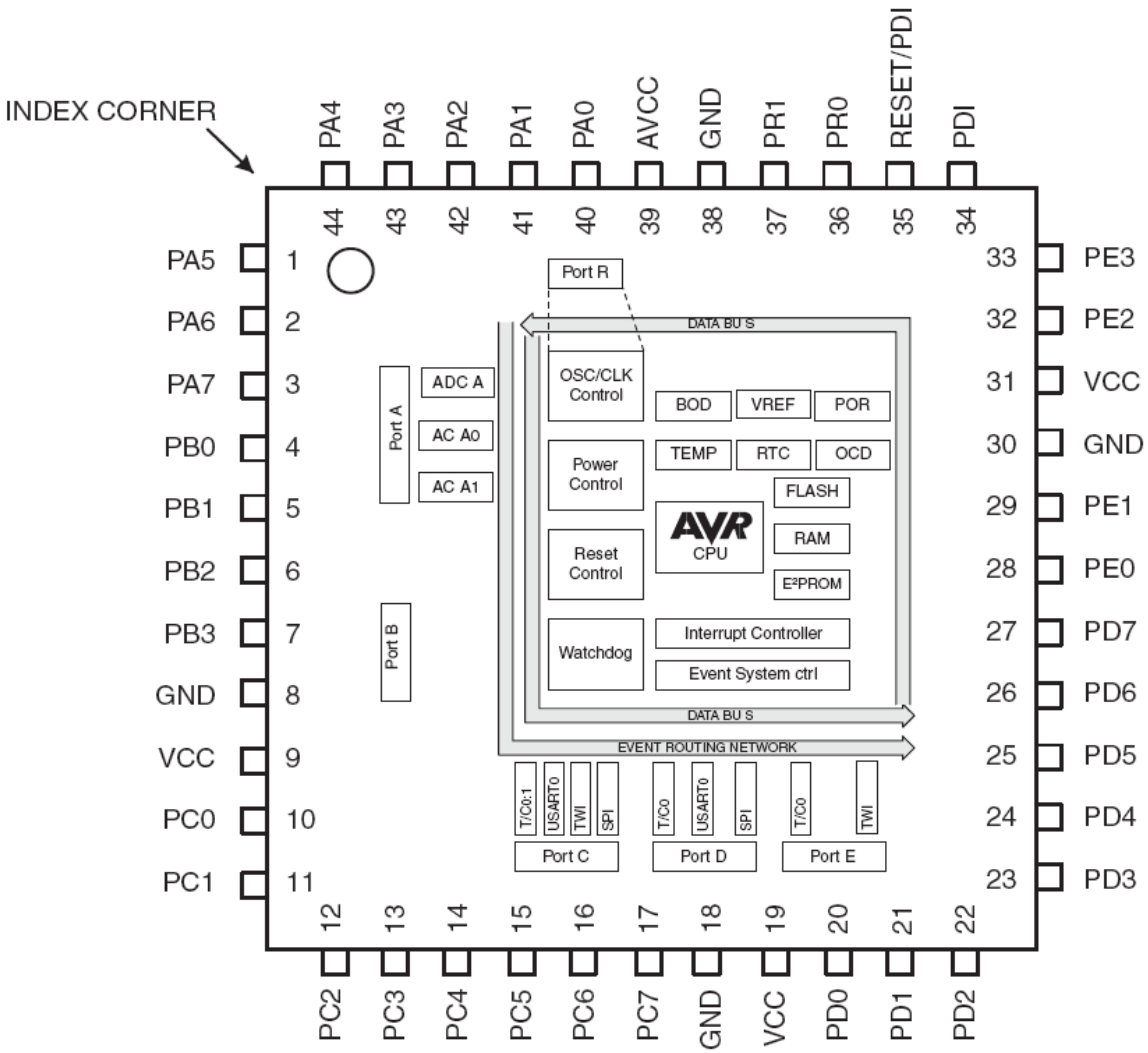
5.6.7 ATXMEGA32A4U

Figure 2-1. Block Diagram and QFN/TQFP pinout



5.6.8 ATXMEGA32D4

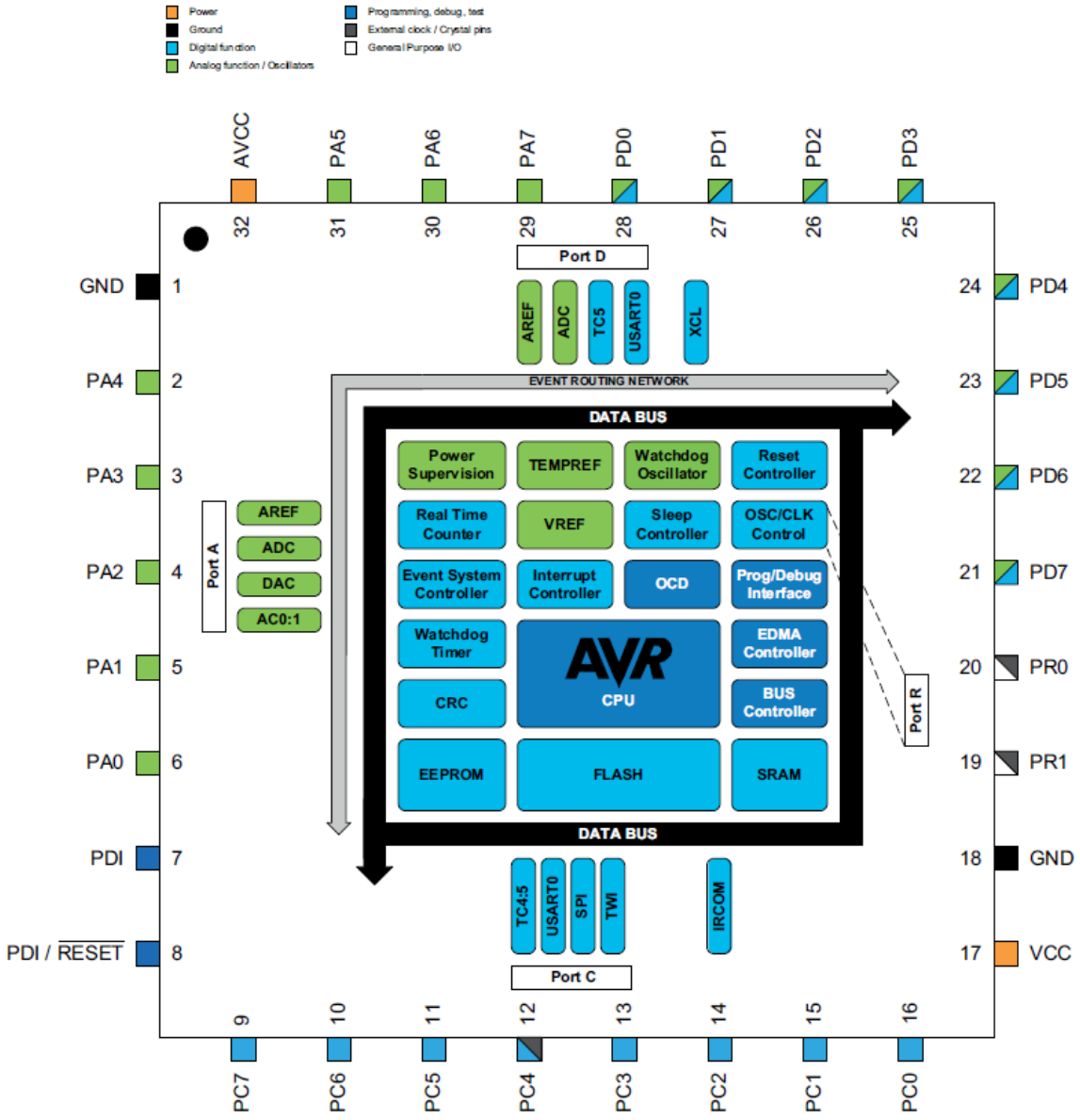
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xmega.



5.6.9 ATXMEGA32E5

- The XMEGA E series requires that you reset the interrupt yourself. For example : TCC4_INTflags.0=1 'clear OV flag
- The ERASE_APP NVM command (&H20) erases the complete flash, thus the boot space included. Use &H25 instead to erase and write a page.
- There is a fixed map for the virtual ports :
  - VPORT0 - Virtual port A
  - VPORT1 - Virtual port C
  - VPORT2 - Virtual port D
  - VPORT3 - Virtual port R
- CONFIG XPIN slewrate is for the whole port, not for an individual pin

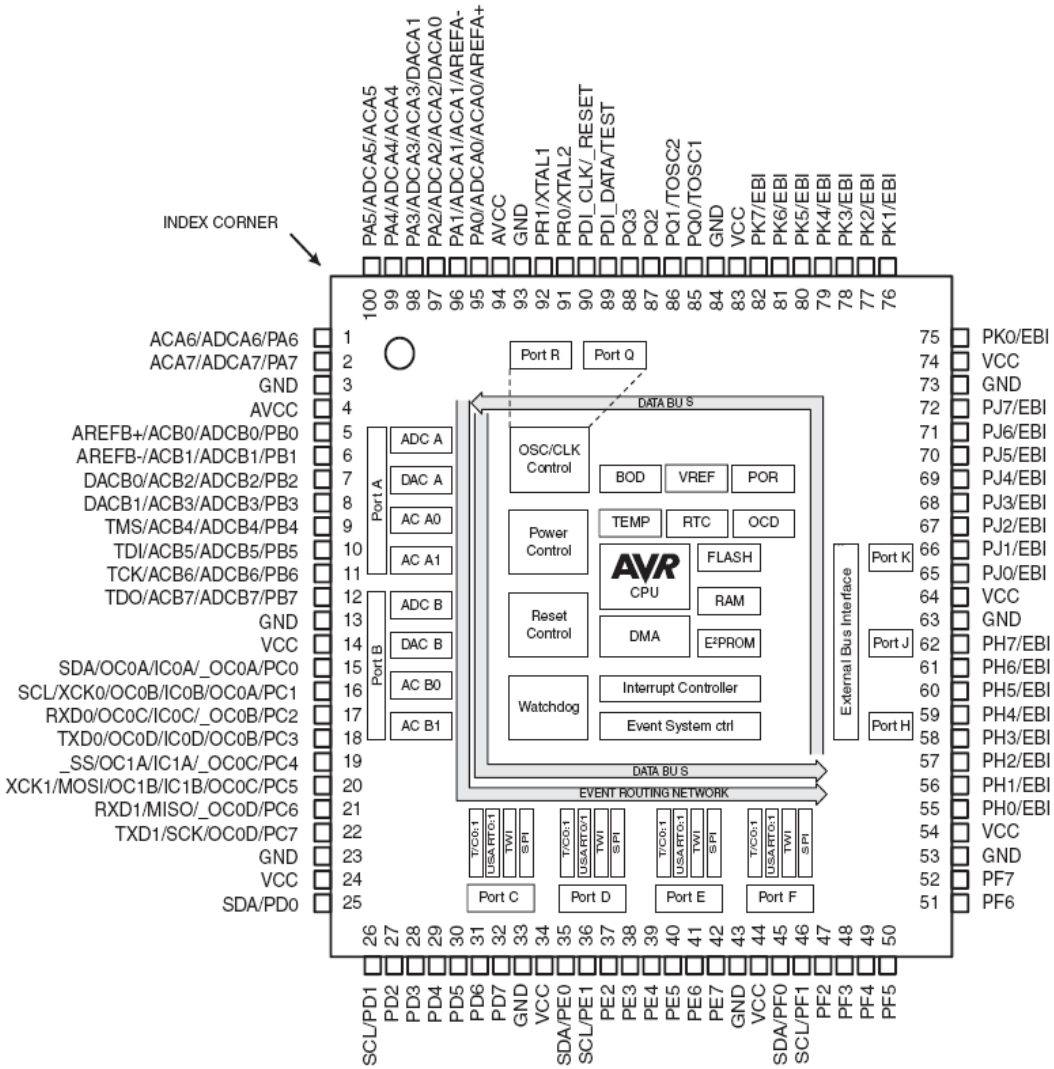
Pinout and Block Diagram



Note: 1. For full details on pinout and alternate pin functions refer to "Pinout and Pin Functions" on page 57.

5.6.10 ATXMEGA64A1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



Here a note about the spike detector :

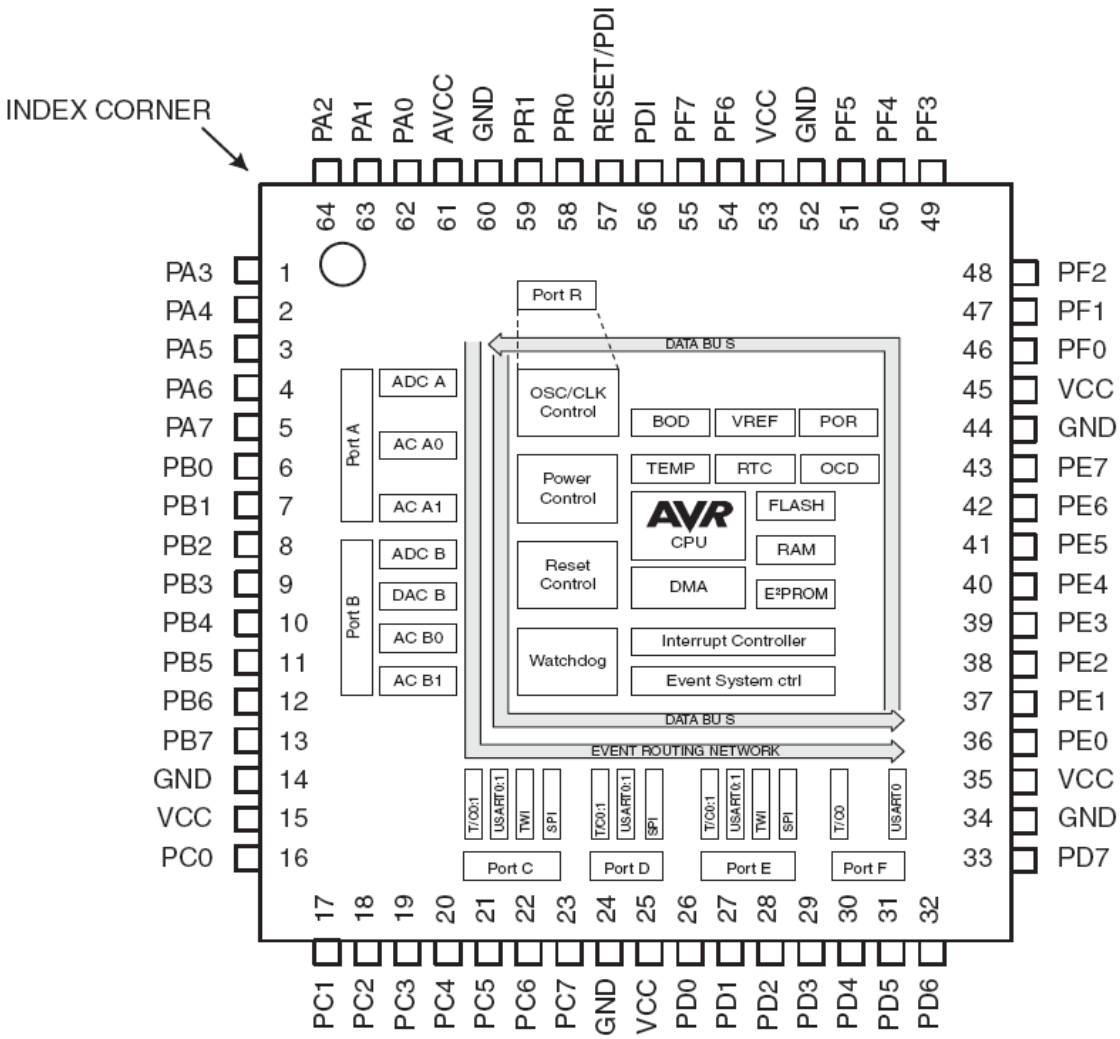
>The calibration byte in the production signature row show 0xFF and 0x00
>for the ADC Calibration byte. Are these really the calibration values ?
>And I'm not able to set the HIGH Byte of the calibration register.
>Errata of Rev H don't show something from calibration bytes.

Reply from Atmel :

The voltage spike detector has been removed from the latest revision of the XMEGA A manual.
This is because we have, unfortunately, not been able to validate the spike detector fully. The module is disabled in currently available parts to avoid unforeseen problems for any customers.

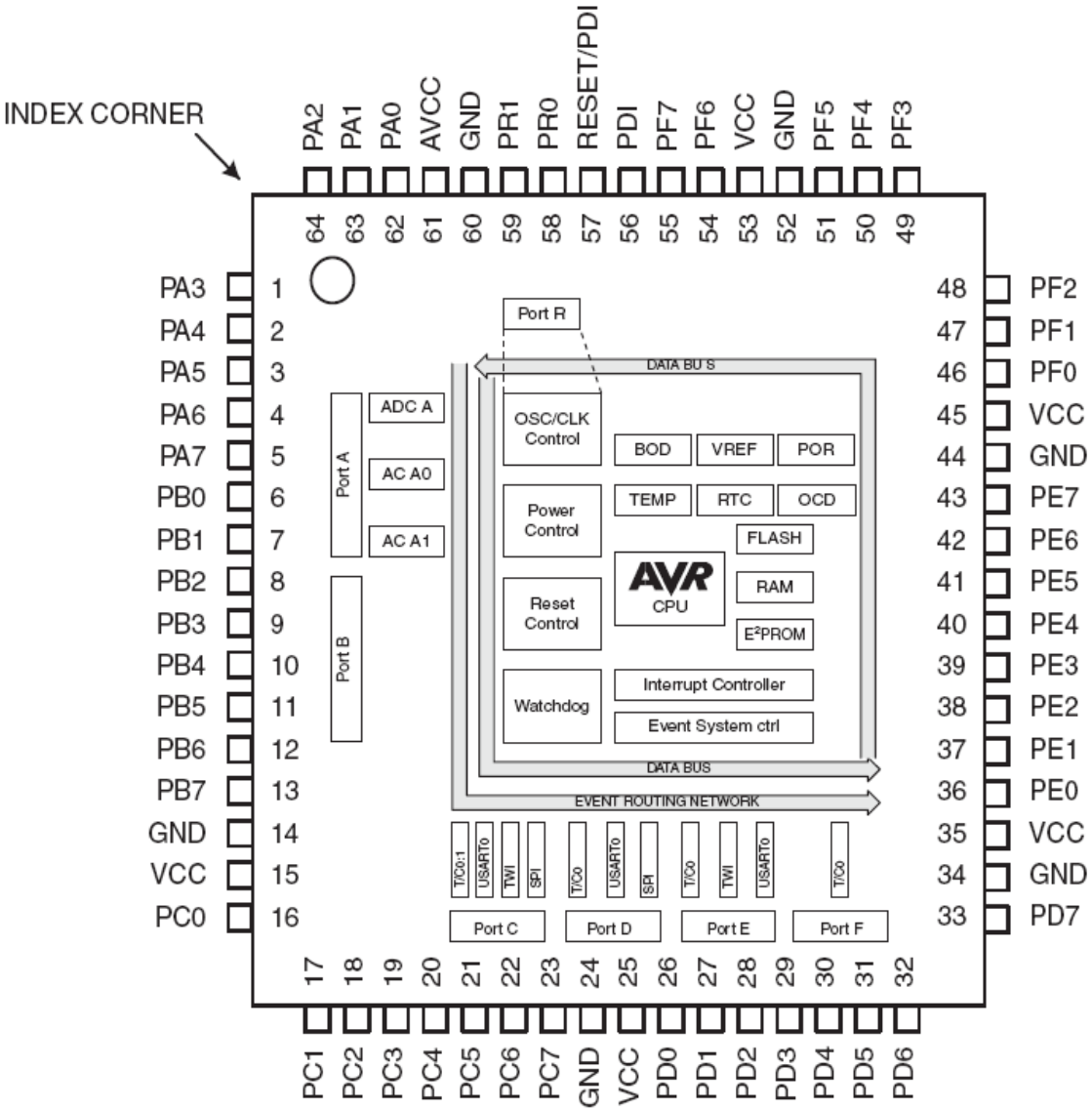
5.6.11 ATXMEGA64A3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



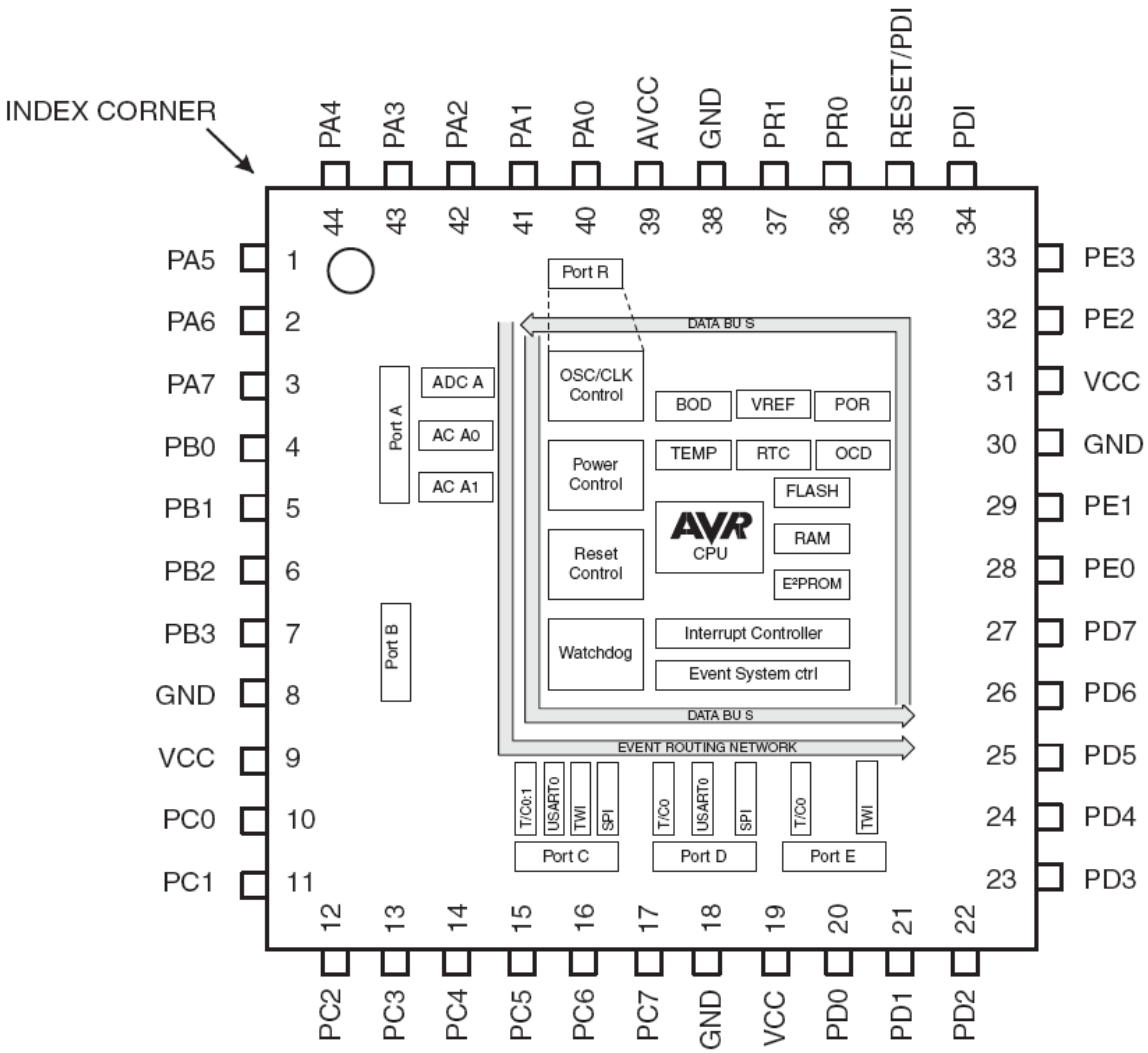
5.6.12 ATXMEGA64D3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



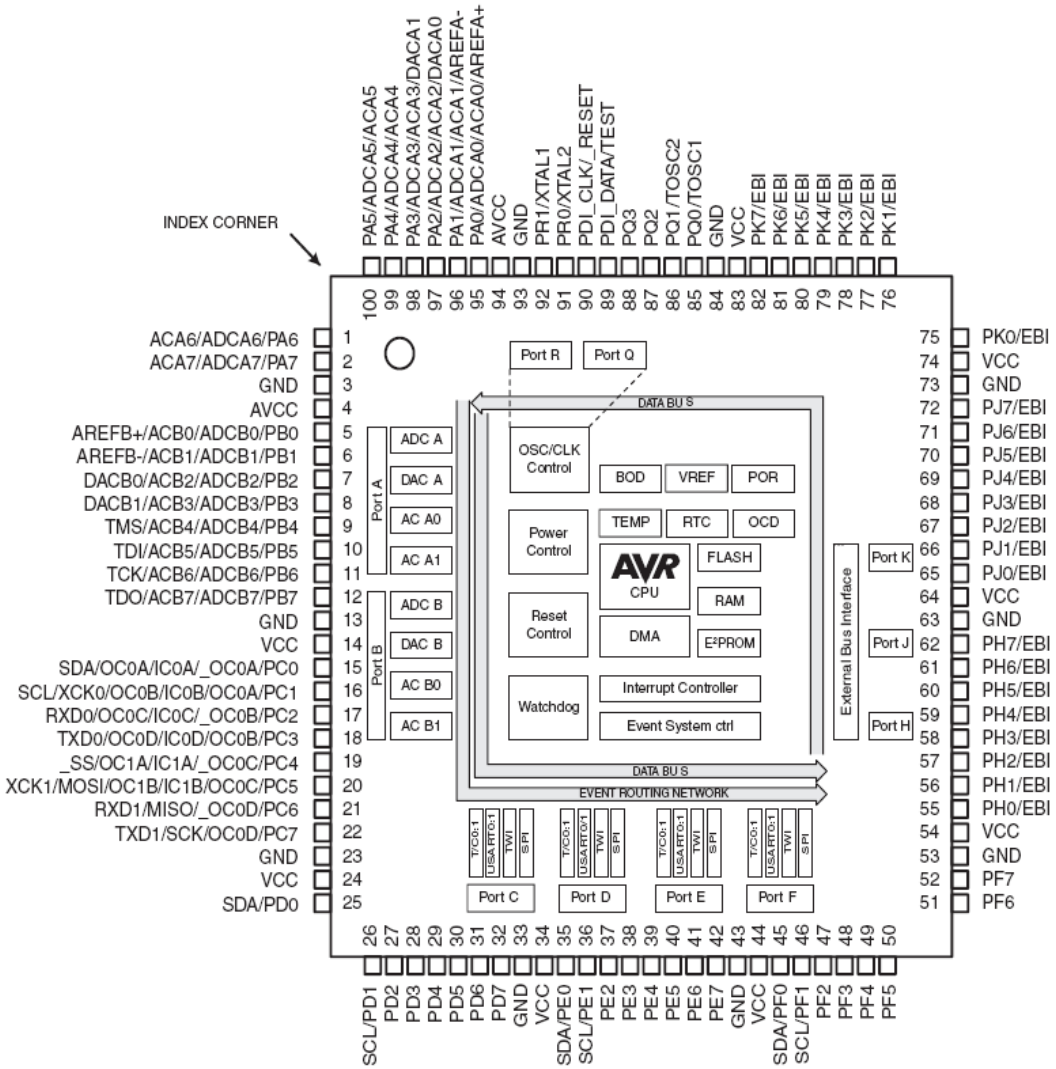
5.6.13 ATXMEGA64D4

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xmega.



5.6.14 ATXMEGA128A1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



Question: The DVDSON FUSE BIT the ATxmega A MANUAL says that for characterization data on VDROP and tSD consult the device data sheet.
(Device: ATXMEGA128A1 RevH). But I can't find this Information in the datasheet ?

Answer: The voltage spike detector has been removed from the latest revision of the XMEGA A manual.
This is because we have, unfortunately, not been able to validate the spike detector fully.

Question: The calibration byte in the production signature row show 0xFF and 0x00 for the ADC Calibration byte. Are these really the calibration values ?
Errata of Rev H don't show something from calibration bytes. (Device: ATXMEGA128A1 RevH)

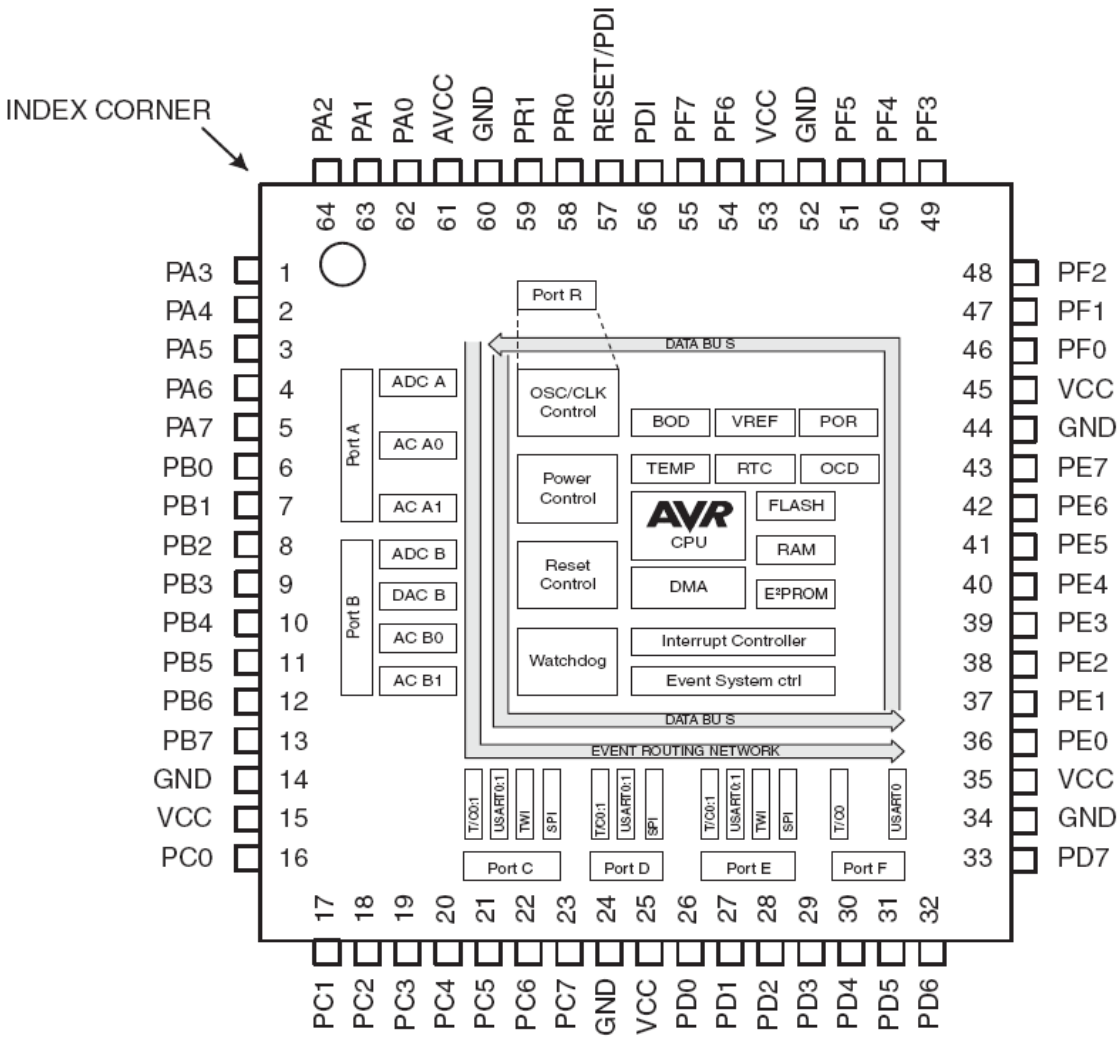
Answer: Yes this is a known issue with ATXMEGA128A1 RevH. We will be fixing up this
issue in the later version of the device.

You should write the code for loading the calibration registers in the
firmware so that when we fix it in the later version you do not have to fix

the code. Also loading it now will not cause any problem in the ADC operation.

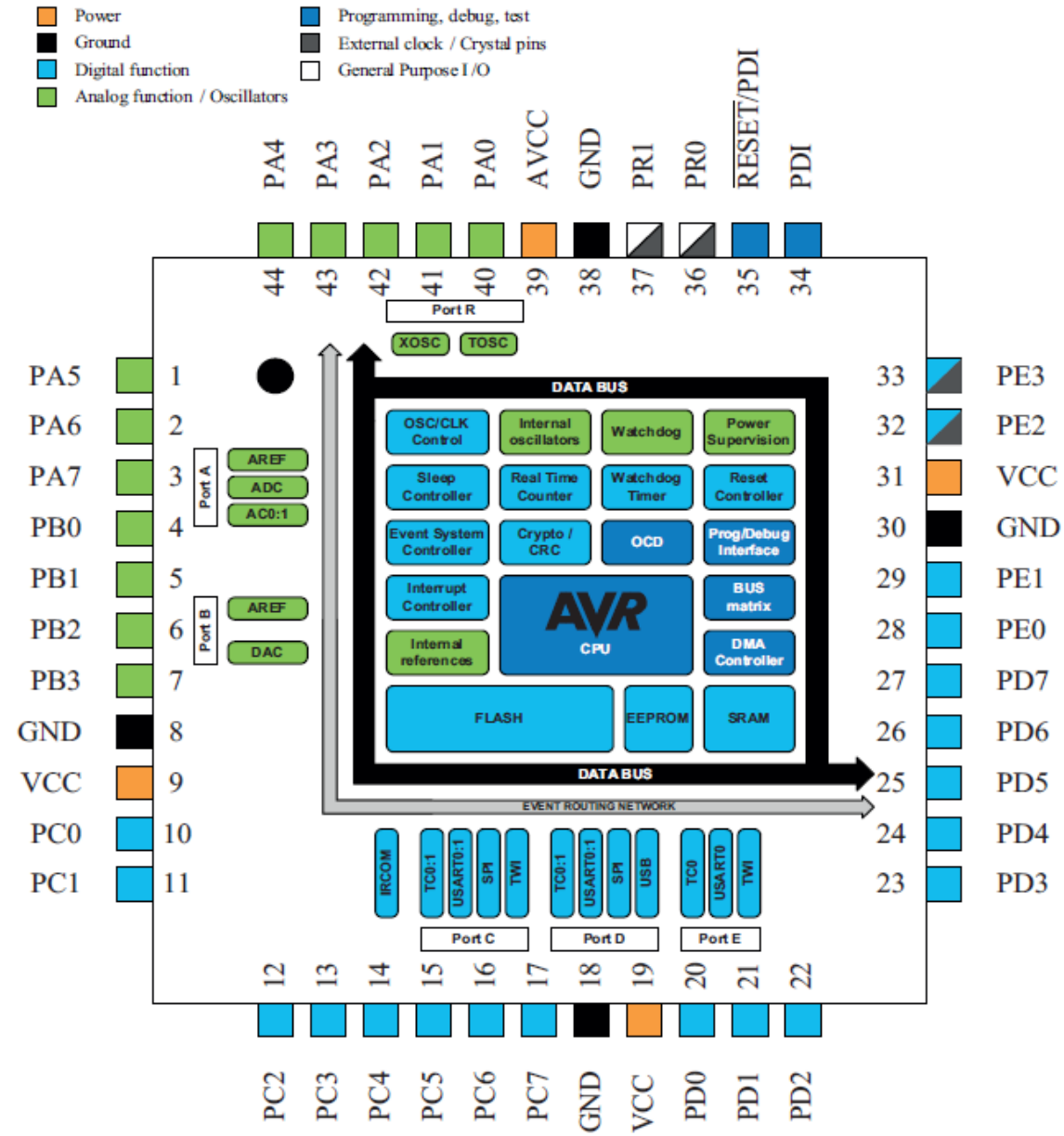
5.6.15 ATXMEGA128A3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.6.16 ATXMEGA128A4U

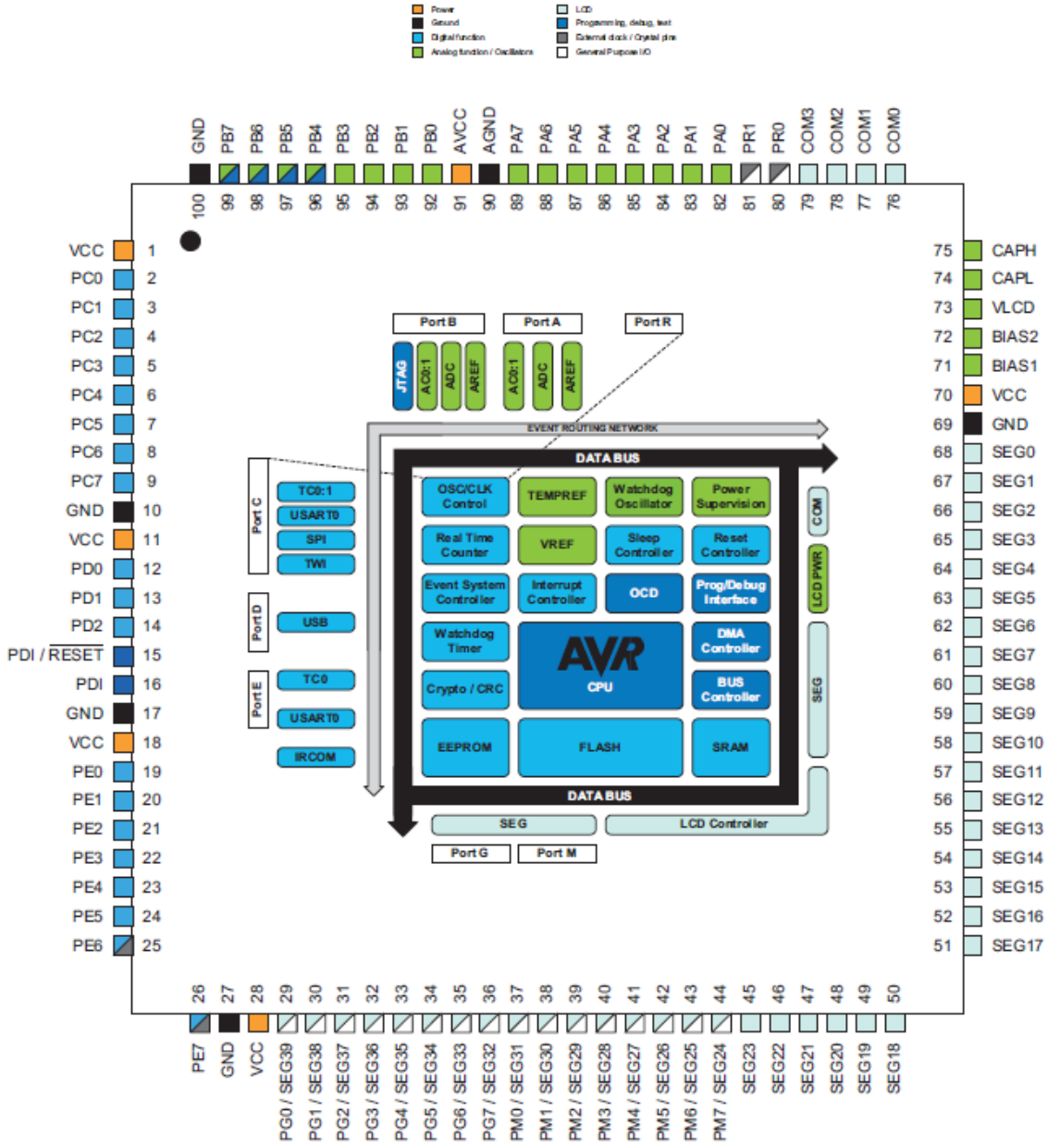
Figure 2-1. Block Diagram and QFN/TQFP pinout



5.6.17 ATXMEGA128B1

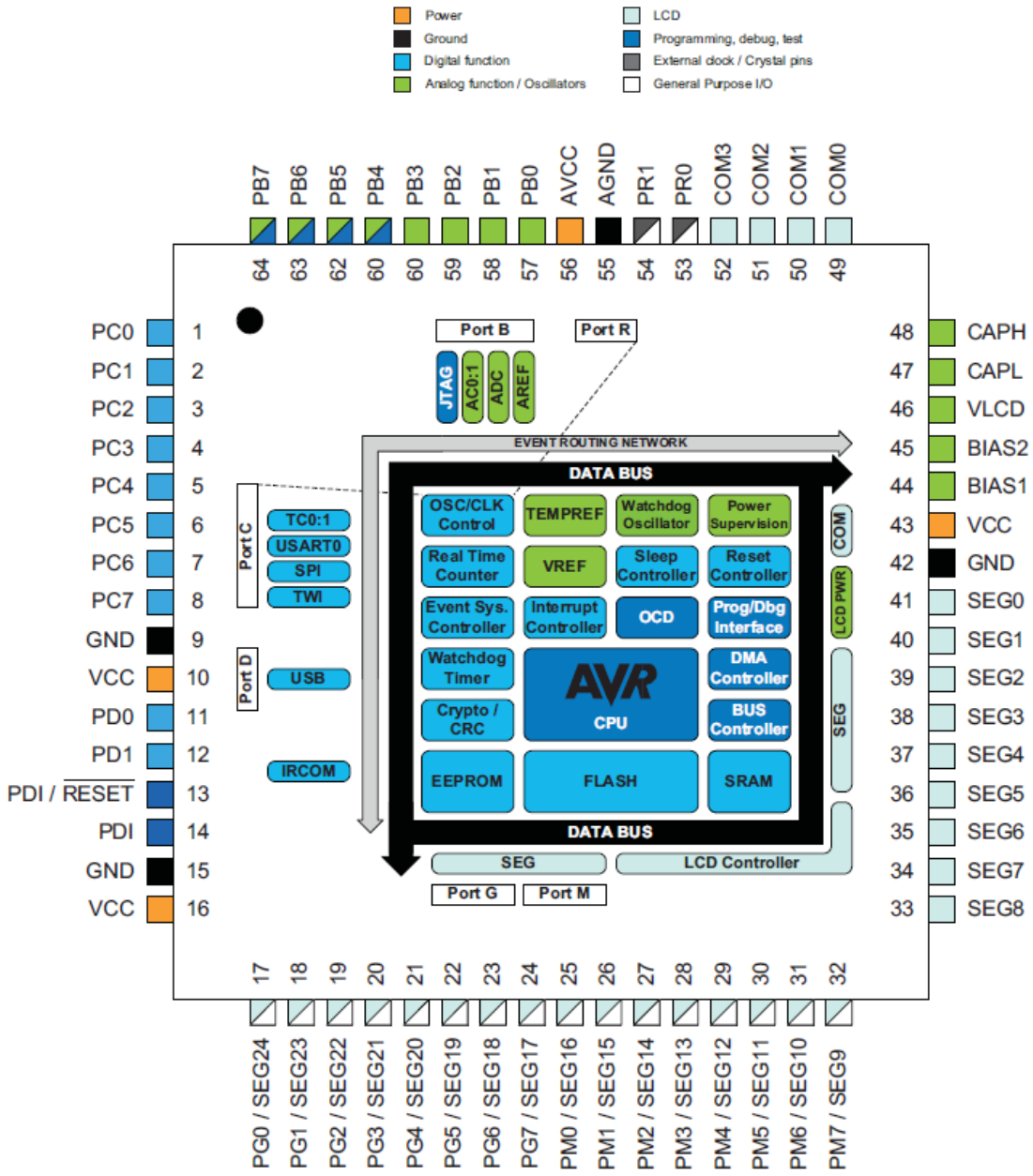
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 2-1. Block Diagram and Pinout



5.6.18 ATXMEGA128B3

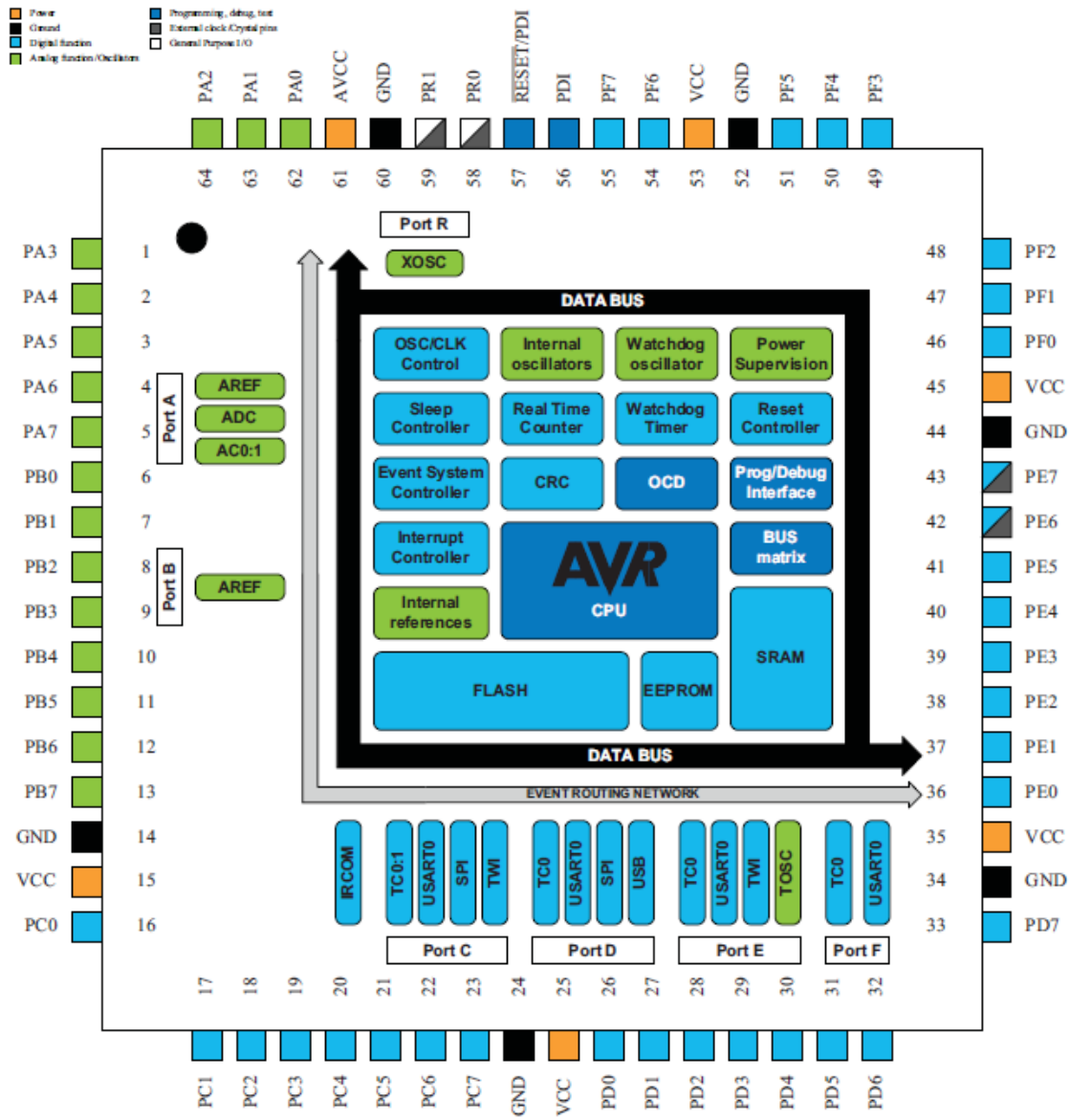
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.6.19 ATXMEGA128C3

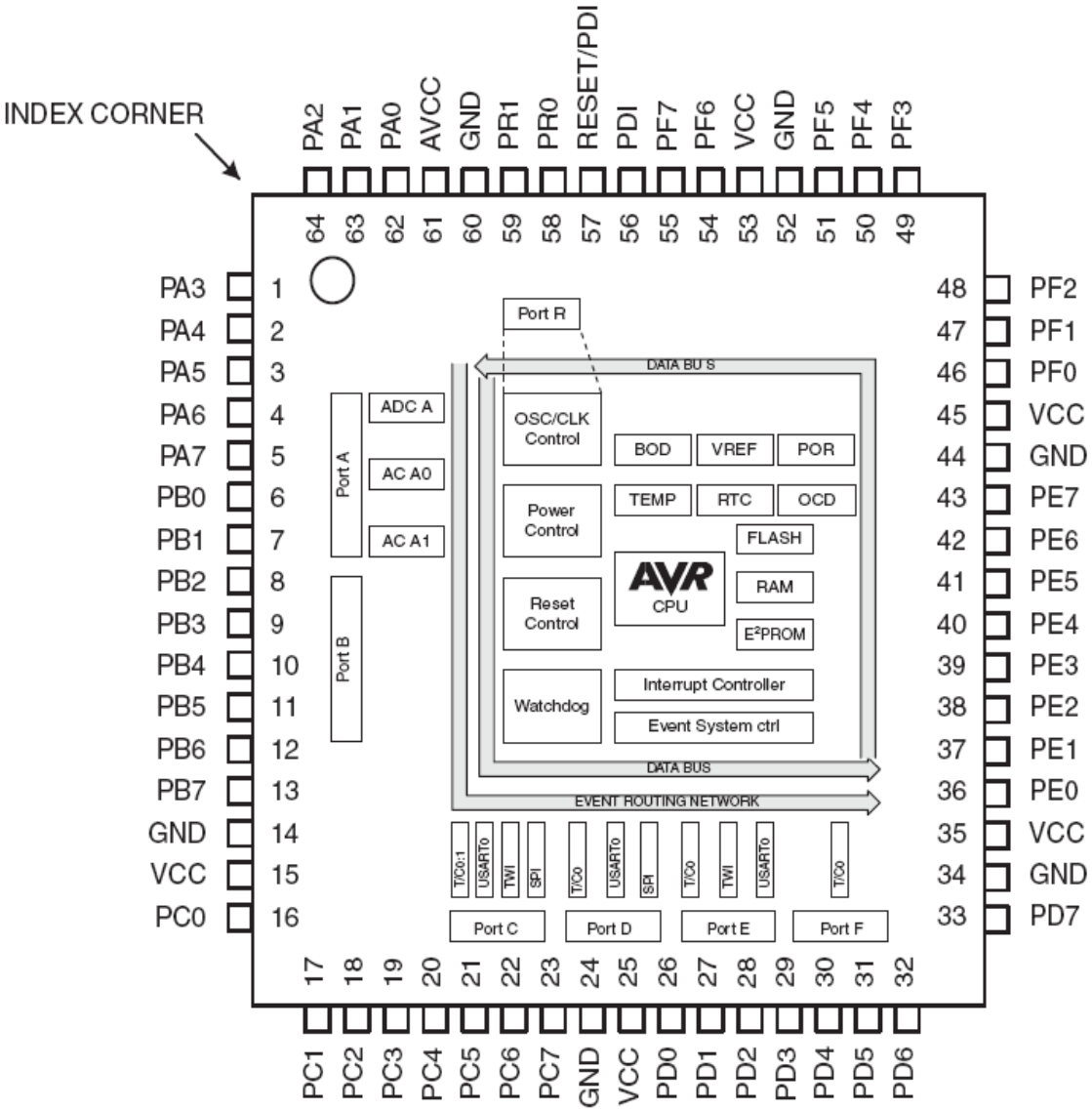
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 2-1. Block Diagram and Pinout



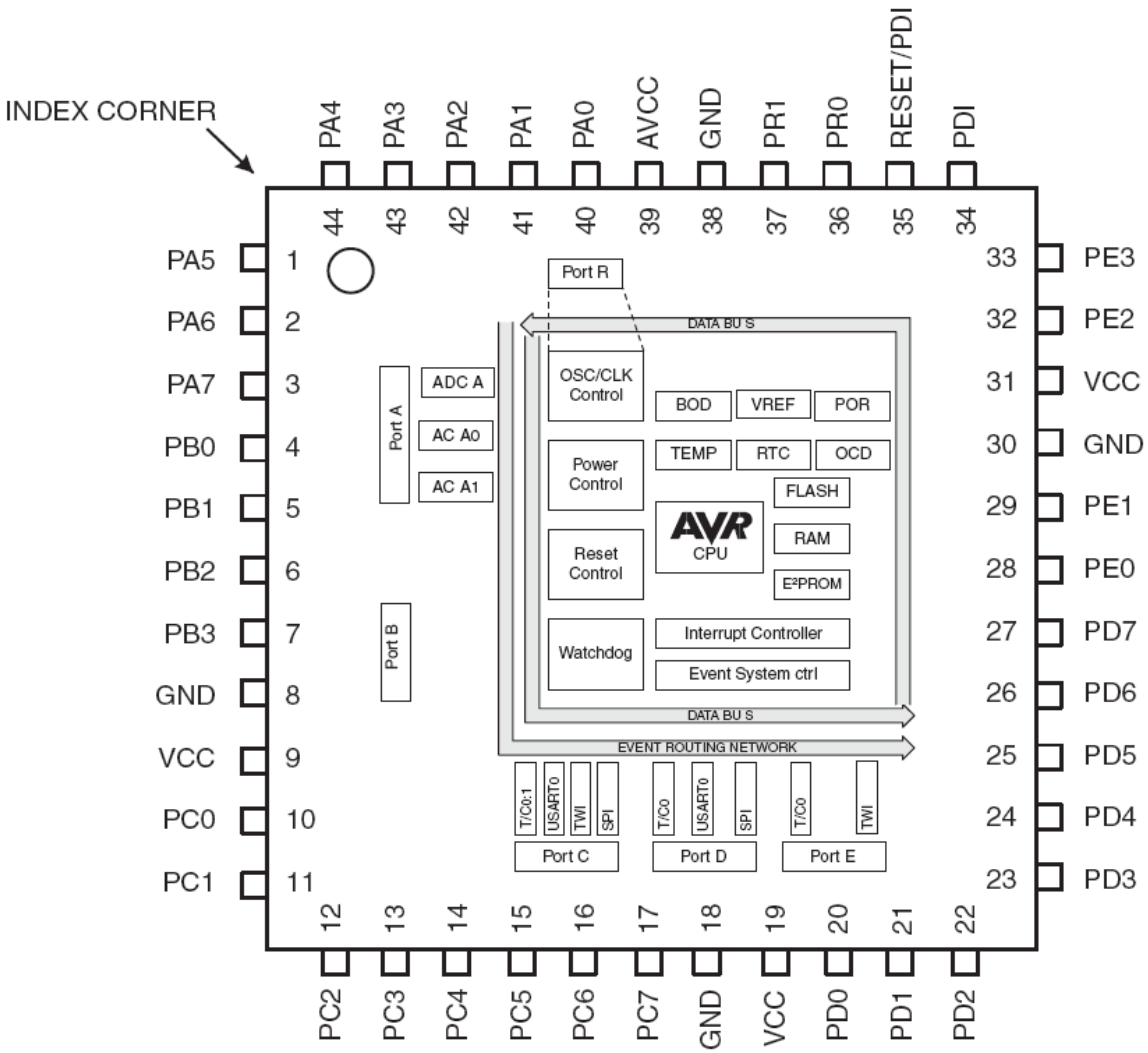
5.6.20 ATXMEGA128D3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



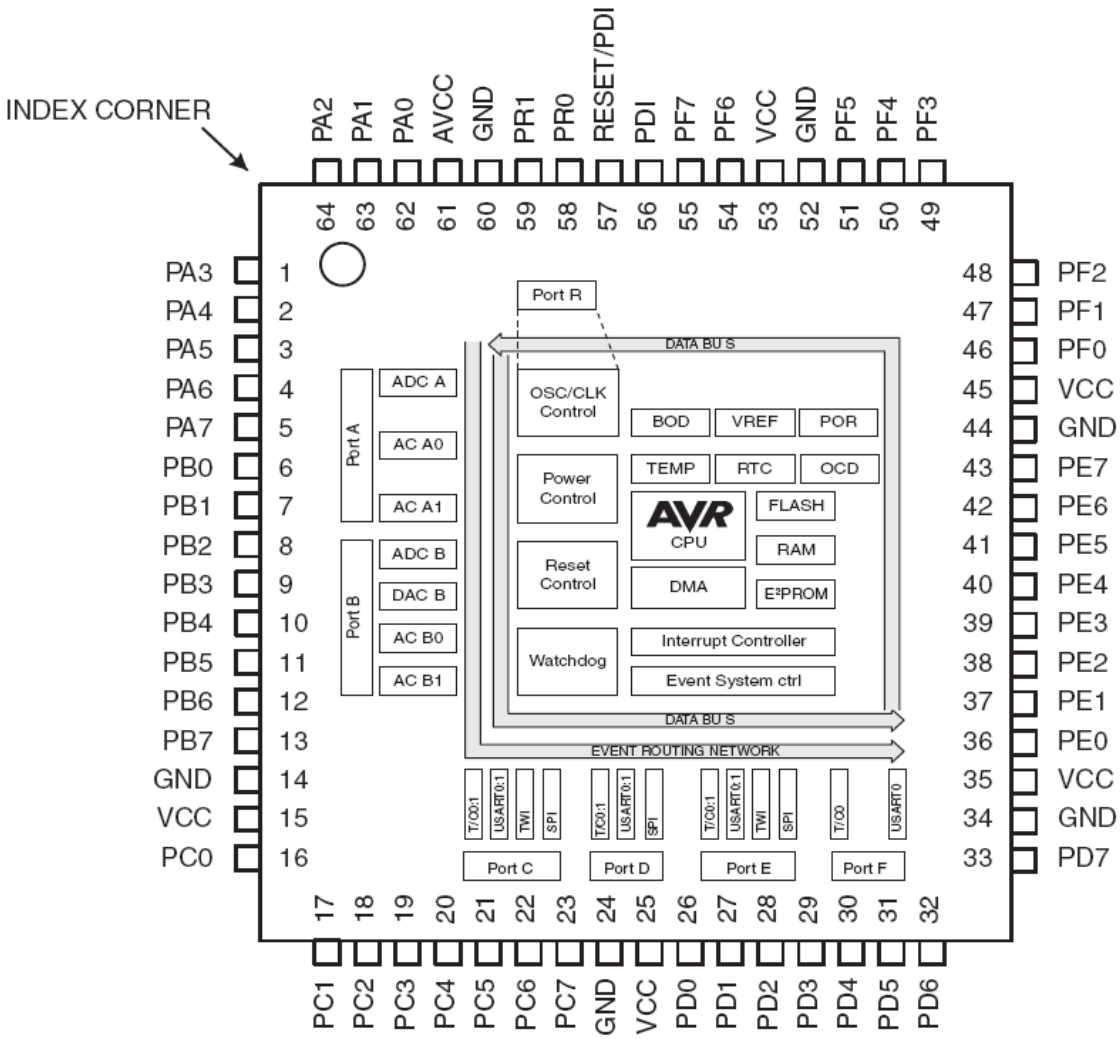
5.6.21 ATXMEGA128D4

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xmega.



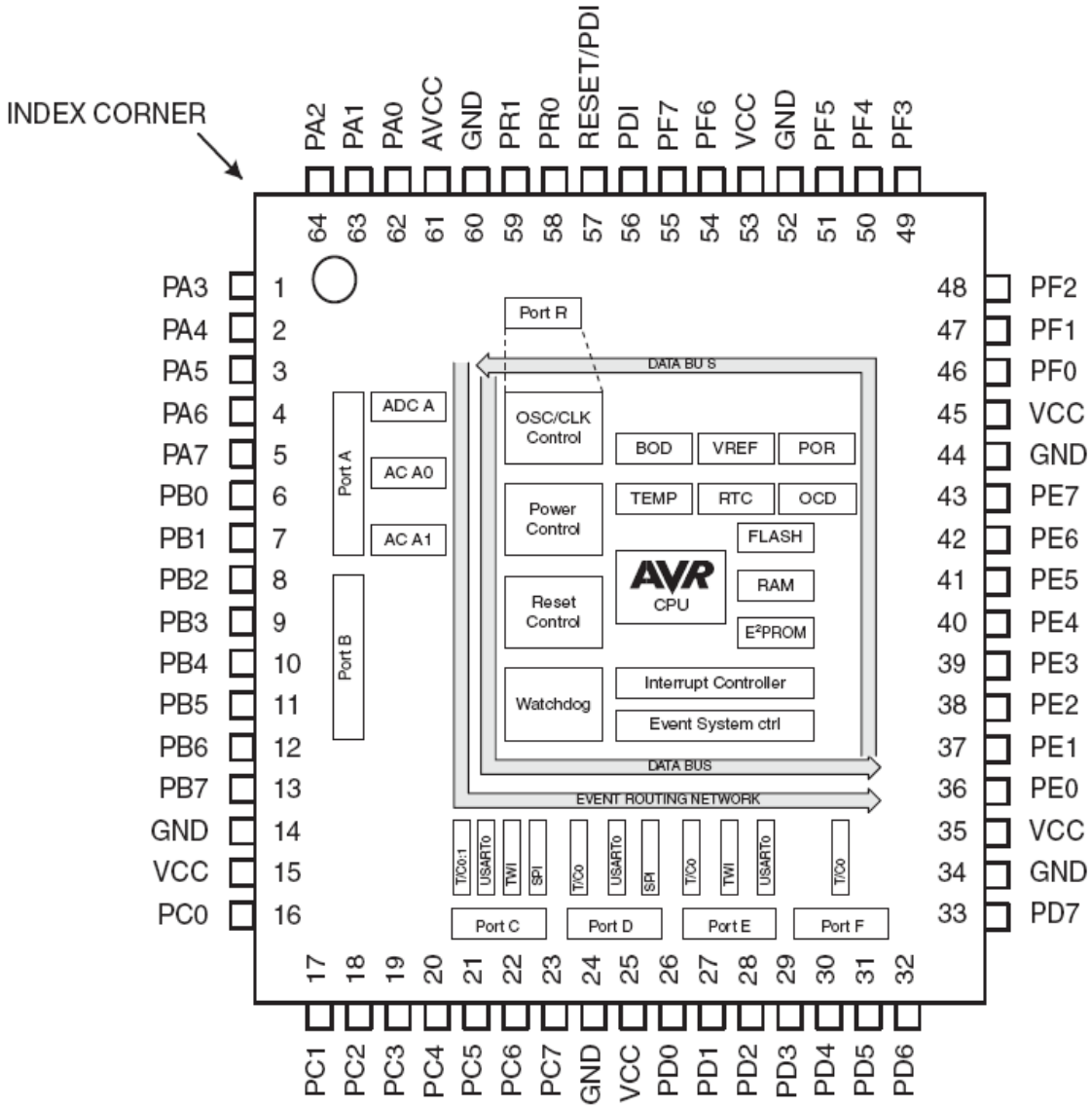
5.6.22 ATXMEGA192A3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



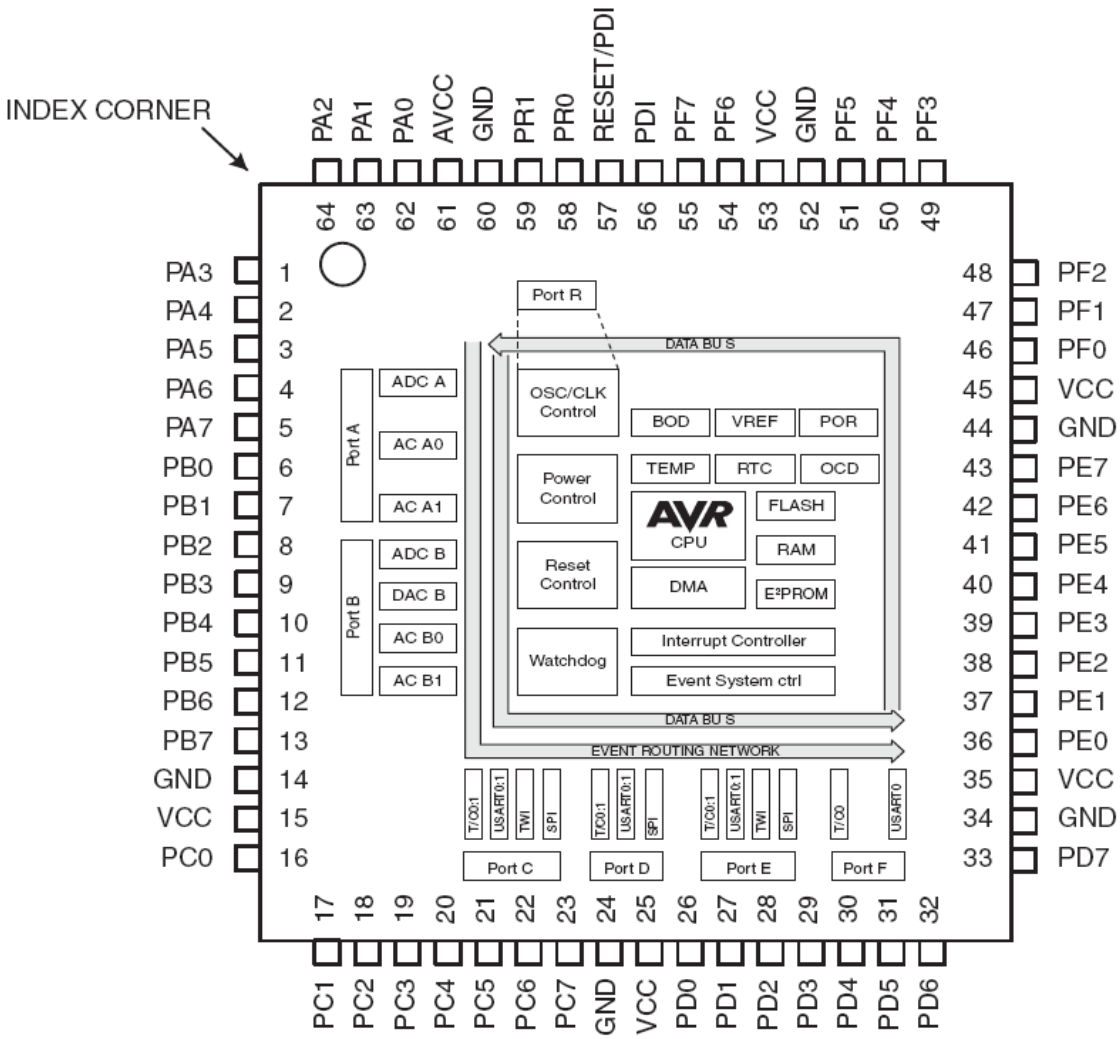
5.6.23 ATXMEGA192D3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



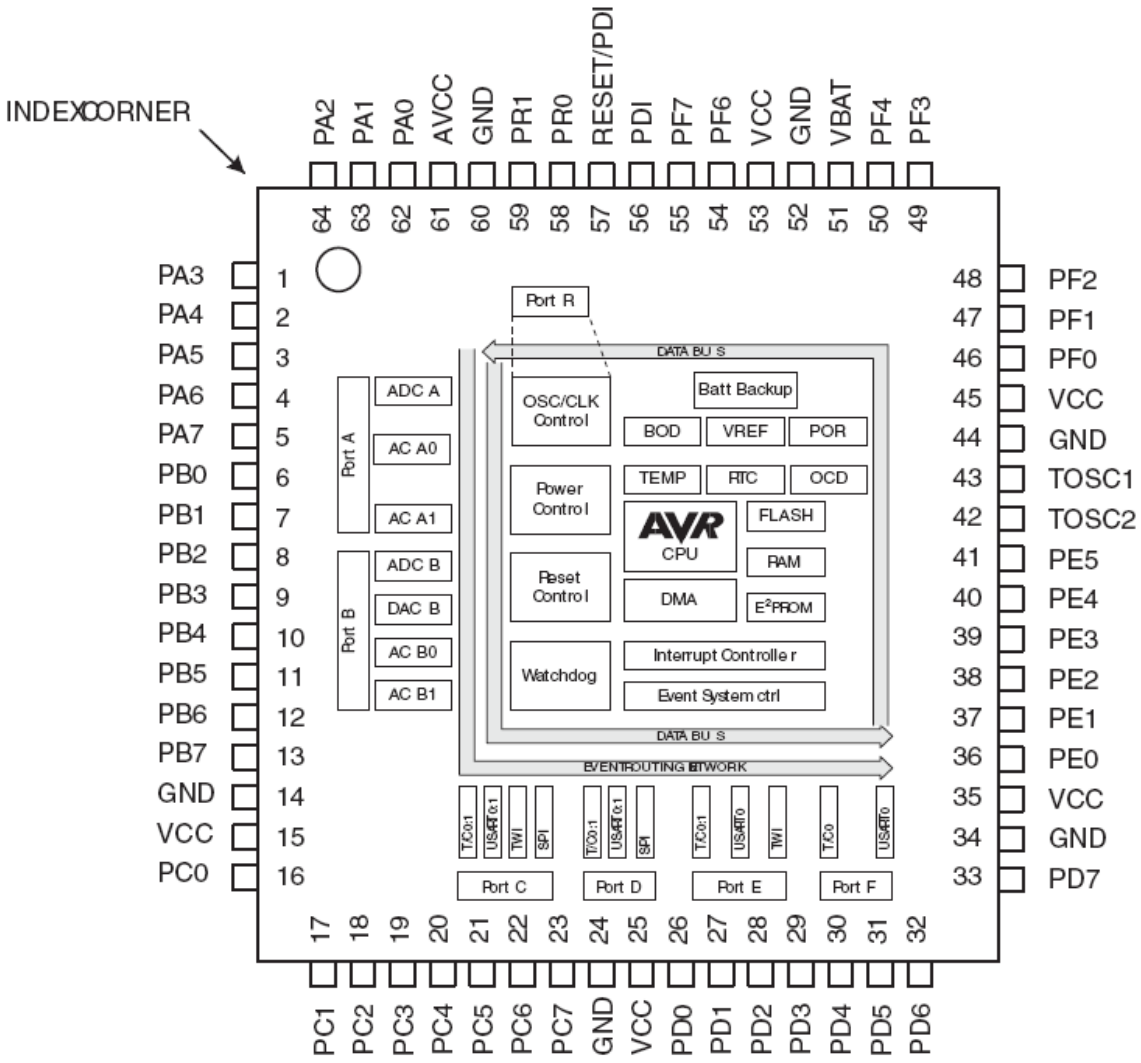
5.6.24 ATXMEGA256A3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

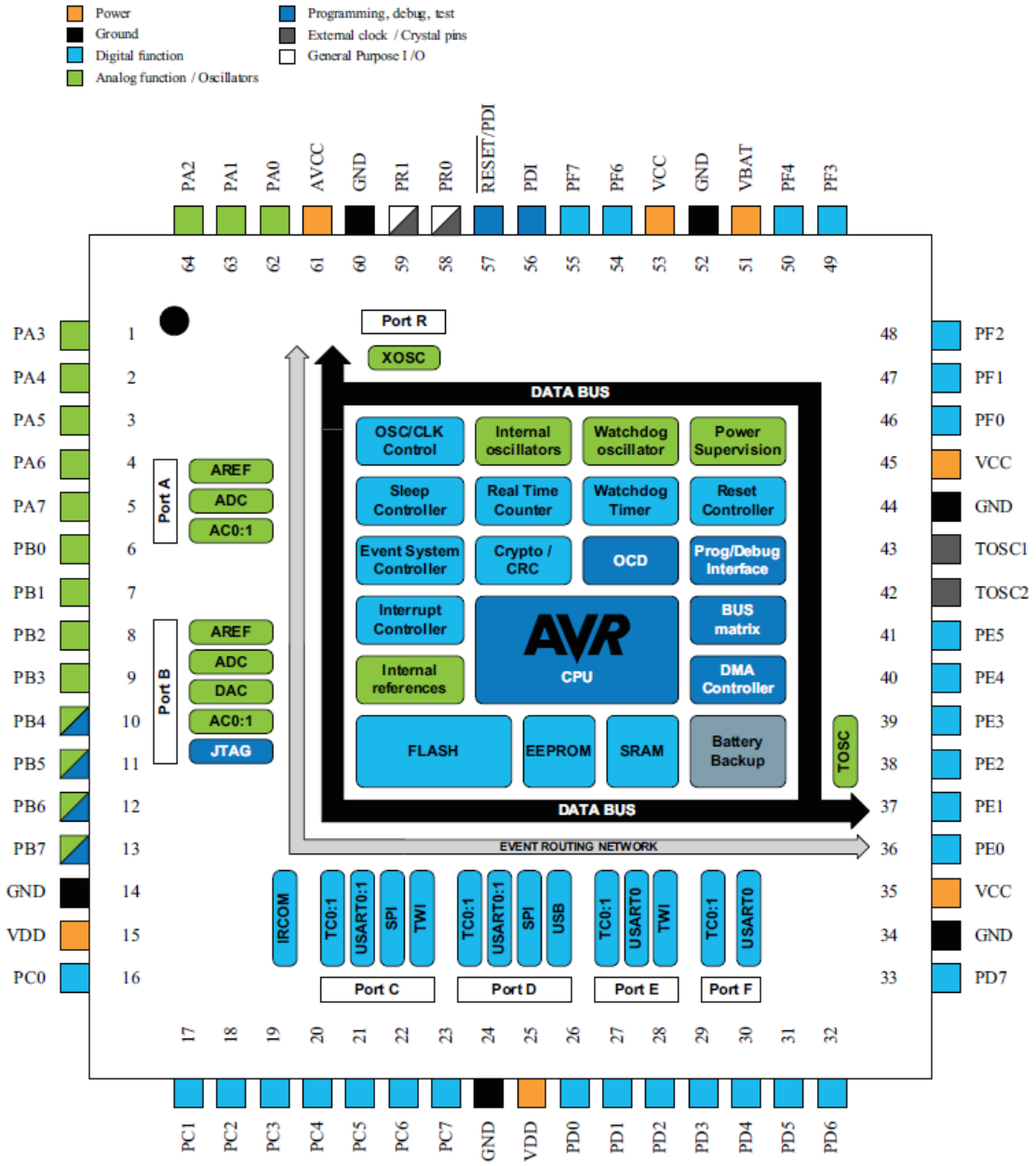


5.6.25 ATXMEGA256A3B

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

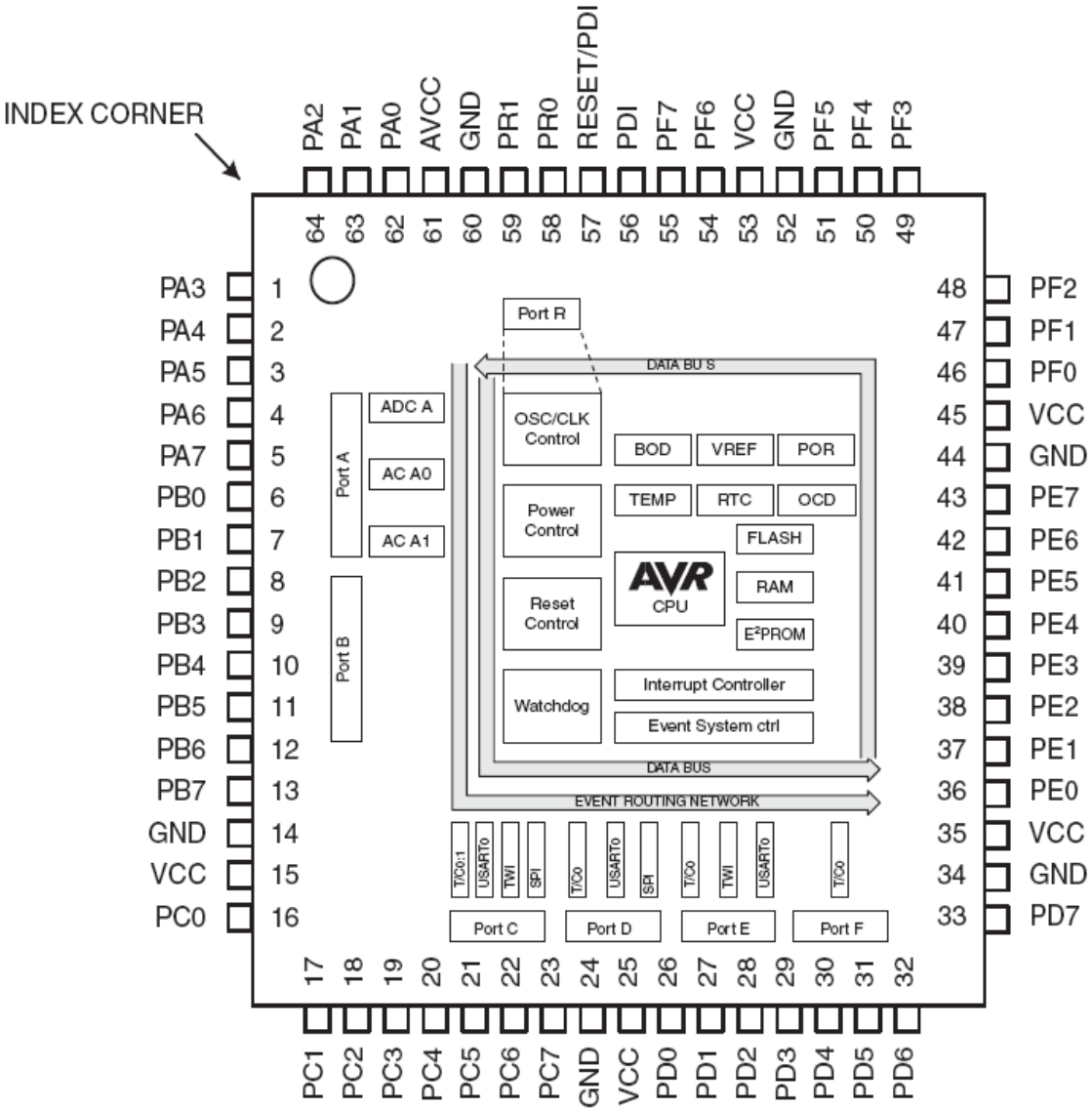


5.6.26 ATXMEGA256A3BU



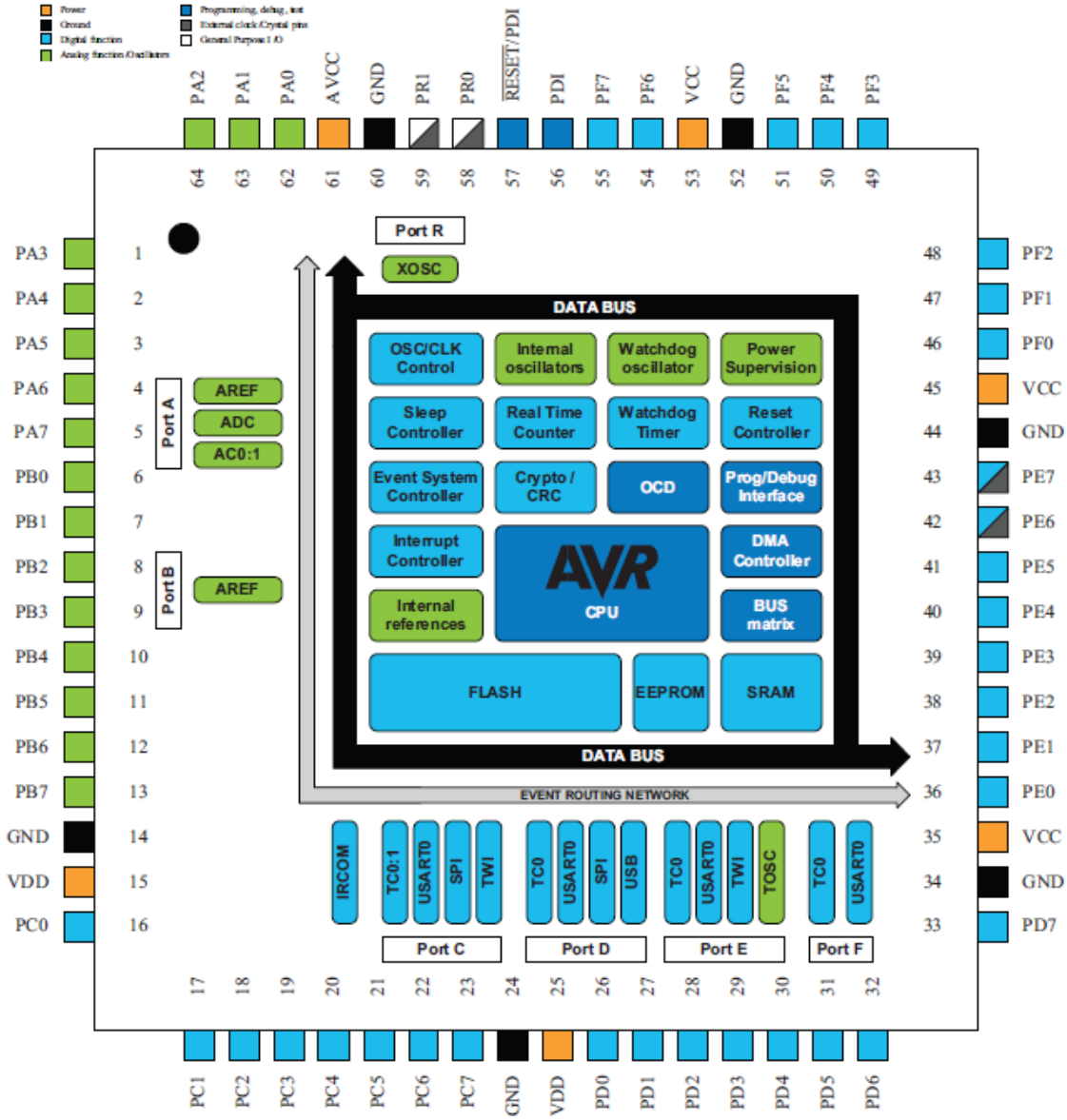
5.6.27 ATXMEGA256D3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



5.6.28 ATXMEGA384C3

Figure 2-1. Block diagram and pinout.



5.7 XTINY

5.7.1 XTINY

The names XTINY, MEGAX, AVRX and UPDI are used in this manual. We do mean the same thing ; the microchip processor with UPDI interface. Since the XTINY was the first platform implemented, you find that XTINY is used a lot.
But it will apply to MEGAX and AVRX as well which were implemented later unless specified otherwise.

At MCS we refer to the new range of TINY processors as the XTINY. This because they look like Xmega processors but smaller.
The XTINY processor is a great new processor. It uses the AVR instruction set. But it has a lot of the hardware found in the Xmega.
But do not make a mistake : these processors are different in many ways. They are the next generation of AVR processors.

They use little power.

When XMega was supported we had to make some decisions about this support. The goal is to keep the code BASCOM compatible. And for Xtiny we were faced with the same dilemmas.
So again there will be new CONFIG statements and because the hardware is different, there will be other changes as well.

You should read the data sheet of the processor like ATTINY816. We used the attiny 816/817 for testing.
The processors are not available anymore in DIP. But you can use SOIC with a converter board. Like Xmega, this makes the processor less usable for hobbyists. At least for those that do not make their own PCB's.

Like the Xmega the internal registers can not be addressed by a pointer. So we had to change code using register pointers.

The DAT files contain a constant that identifies the platform. This constant is named _XTINY. For normal AVR and Xmega is is absent.

Platform	Value
XTINY	1
MEGAX	2
AVRX DA, DB, DD	3
AVRX EA	4

Since the NVM controller differs among platforms, this constant is used in the xtiny.lib for writing the EEPROM.

UPDI
A big difference is that you once again need a new programmer since the programming interface is using UPDI.
The good news however is that you can use a simple serial port and a resistor to do the programming.
BASCOM-AVR supports the UPDI 202 protocol with a dedicated programmer.
You will notice when you read the fuses that there are a lot of fuses and other settings.

OSC
Like the Xmega, the Xtiny has extensive internal and external oscillator support. This means that you need to chose the system clock using the CONFIG SYSCLOCK statement.
This is essentially the only difference in code you need to make compared to the normal AVR.
The options and values differ from the Xmega.

UART
In order to use the UART, you need to use CONFIG COMx. There is no default configuration when you use $baud. We would recommend to use CONFIG COM anyway. It can set all relevant settings.
Also notice that the UART has separate registers for reading and writing. That is something new, not seen earlier. For this purpose we mapped registers UDRW and UDRR to the USART0_RXDATA / TXDATA registers.

ERAM

The EEPROM is more complex since it only works with pages. But the good news is that the EEPROM can be read like normal variables.

PORTS

The ports have more options compared to normal AVR but less compared to Xmega. While Xmega can do a setting for an entire port, this feature is missing in Xtiny. The virtual port option is also different. In Xtiny there is just a register in the lower IO space mapped to the three most important port registers : DDR, PORT and PIN. These are named VPORTA_DIR, _OUT and _IN.
In the first XTINY support we added DDR0, PORT0, PIN0 to make them compatible with Xmega. We also added VDDRA, VPORTA, and VPINA. All these registers are located in lower IO memory. In 2084 we removed these aliases.

IMPORTANT CHANGE in 2084

It turned out that using the same scheme as for the Xmega was not optimal. So for 2084 we changed some ALIAS.
We suggest you read the following information.
In the normal AVR there are PIN, PORT and DDR registers. They always have a low IO address so the instructions like CBI/SBI can be used.
This means that only 1 asm instruction is required to set/reset a bit of a port related register.
For example for the 2313def.dat we have PORTD:
PORTD =$12
DDRD =$11
PIND =$10

Since low IO space is limited and AVR processors with more ports were made, these ports got an extended IO address. Which means that the address is higher.
It also means that instructions like CBI/SBI will not work since they only work on low memory addresses.
For example from the m128RFA1.DAT
PORTL= $10b
DDRL= $10a
PINL= $109
To change a bit in such a register, the value must be loaded, altered and written back.

With Xmega the port related addresses got a high address. But also a virtual port was added. These are dynamic virtual ports. Which means that you can chose at run time to which port a virtual register is mapped to. These virtual port addresses are placed in the low IO address space. So CBI/SBI could be used.
For example for the xm128A4Udef.dat these are some virtual registers
VPORT0_DIR = 16
VPORT0_OUT = 17
VPORT0_IN = 18

These registers can be made to map to PORTA, PORTB, etc. And you can change this at run time. So there is no fixed relation between the virtual and the actual port. Writing to a virtual port will write to the actual register. Thus when VPORT0 is mapped to PORTA, both writing to PORTA and VPORT0 will do the same thing.
Since there is no fixed relation, an ALIAS to PORTA points to PORTA_OUT. This because there is no PORT/DDRA/PINA register. So as a user you can use the port as you always did.

For the XTINY there is again a virtual register but it has a fixed relation.
For example for the attiny816 :

```
VPORTA_DIR=0                    ; 0000
VPORTA_OUT=1                    ; 0001
VPORTA_IN=2                     ; 0002
```

The relation is fixed. Each port has a virtual register.
So instead of pointing to PORTA_OUT for PORTA, we now point to the VPORTA_OUT register.
This will give better code.
But it also required compiler changes. The problem with the virtual address is that the order is different. It would have been great if the traditional order was used.
Because the ports are similar to xmega ports we point to PORTx_DIR.

Lets have a look at VPORTA in the DAT file :

```
VPORTA_DIR=0                ; 0000
DDRA=0                          ; 0000 alias
VPORTA_OUT=1                ; 0001
PORTA=1                         ; 0001 alias
VPORTA_IN=2                 ; 0002
PINA=2                          ; 0002 alias
```

The alias we use are DDRA, PORTA and PINA.

Since all ports are aliased all port related code should work without problems.

Pull Up

The pin behavior can be further configured using CONFIG XPIN 1158.
This is important since the pull up need to be configured. The pull up can no longer be activated by writing a 1 to the PORT register !

INTERRUPTS

The interrupt system is extended with an option to specify one high priority interrupt. So this is a nice extension.
You should also notice that a lot of interrupts need to be cleared manually. For example using a PIN interrupt requires to reset the INTFLAG register by writing a 1. While normal AVR pin interrupts let you guess which pin caused the interrupt, the Xtiny has registers that tell which pin caused the interrupt.

WATCHDOG

The watchdog works different since it has no on/off control. A timeout value of 0 will turn off the WD timer.
Any other time will start the WD timer.
START WATCHDOG will use the last value found from CONFIG WATCHDOG. When not found or available an error will be thrown by the compiler
STOP WATCHDOG will write a value of 0 to turn of the WD.

DAT files

When you look in the DAT file you will notice that almost all registers have new names. Also compared to Xmega. Of course BASCOM will use the right register when you use BASCOM code.

When using ASM you need to check the datasheet and the DAT file.
When you are familiar with Xmega the transition should be smooth.

ADC and DAC

The Xtiny chips are very complete. They also have AD converter and a DA converter on board.

TWI/I2C

The TWI is almost identical to the TWI in Xmega. This also means that when using TWI you need to create a byte variable named TWI_START in your user code.
The reason for this is that the TWI hardware sends a START and the slave address at once.
Traditionally this was separated by an I2CSTART and I2CWBYTE.
The TWI_START variable holds the state of I2CSTART. And when you write the address, it will use the proper instruction to send start and slave address.
When you want to use soft I2C, you need to use the $FORCESOFTI2C directive.
Do NOT include the "i2c_twi.lbx" library since this library is only for old AVR processors.

1WIRE

Since the ports have a different mapping, the 1wire code needed to be rewritten as well. The code is placed in the xtiny.lib

LIB

All Xtiny specific code you can find in Xtiny.lib
It contains overloaded versions of the code found in mcs.lib
Xtiny specific code in mcs.lib or other libs can be found by looking for the _XTINY constant.
This constant is set when you use a processor from the Xtiny range.
For normal Xtiny the value is set to 1. For megaX (like M4809) it is set to 2.

BOOT

The XTINY has a different memory model. The BOOT area is located at the start of memory. After this memory the normal memory is located. With a FUSE you can set how many pages of the BOOT area you would like to use for boot code. A value of 1 will reserve a space of 256 bytes(this depends on the page size). When your loader binary code is 1024 bytes you would set it to : 1024/256 = 4
After the normal application code there is also the optional application data. This area can also be set with a fuse.
When using a boot loader you need to take care of the fact that your normal application code must start after the boot code. This can be done by using $ROMSTART in your code.

TCAx

Timer TCA has 1 or more wave outputs. As a user you need to set the pin to the output mode.

XTINY/MEGAX/AVRX

The newest processors can configure each pin of a port. This is done using the CONFIG XPIN statement.
The sense parameter controls how the pin is used :

INT_DISABLED : no interrupt will occur but input buffer is enabled
BOTH : on a rising or falling edge an interrupt will occur
RISING : a rising edge will trigger an interrupt
FALLING : a falling edge will trigger an interrupt
INP_DISABLED : input and digital input buffer are disabled
LOW_LEVEL : interrupt will occur on a low level

So instead of using ENABLE you need to use CONFIG XPIN and select the proper trigger mode.
Instead of DISABLE you need to use CONFIG XPIN and select INT_DISABLED for the sense mode.

Add on

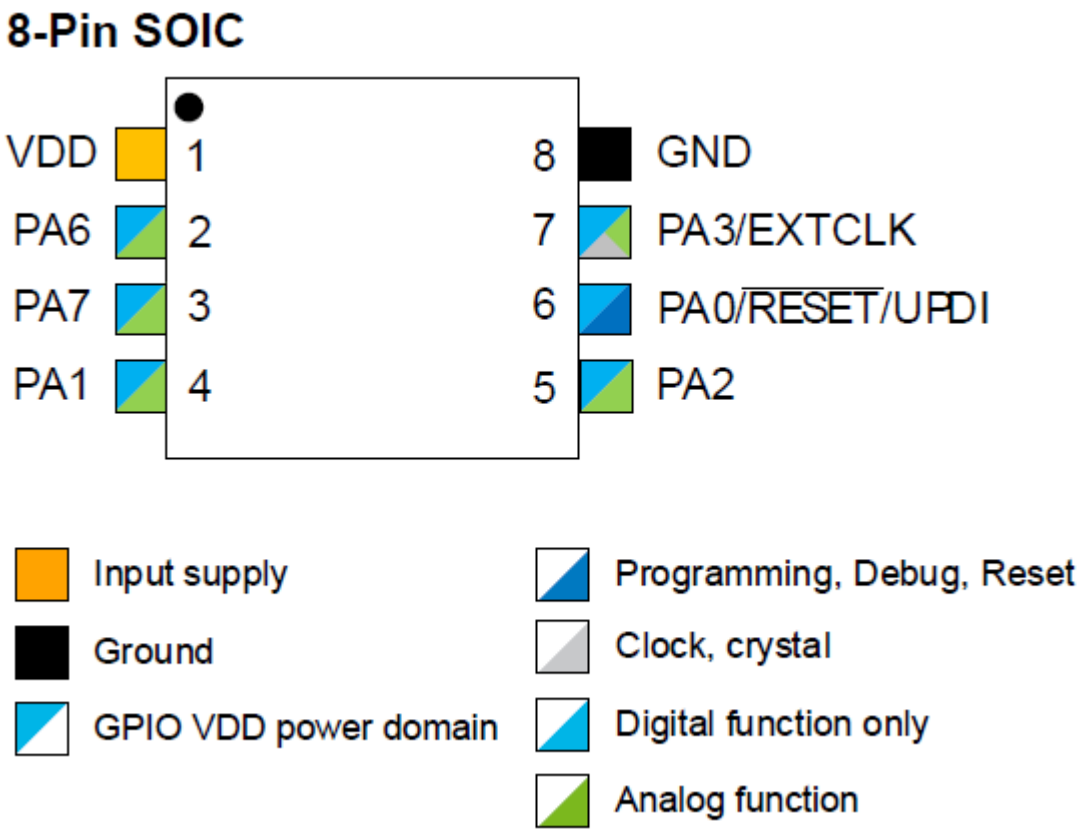
The addition of Xmega to BASCOM was a lot of work. This because only the instruction set was the same. We worked long on UPDI processor support and still this is a work in progress. So expects some updates specific for Xtiny/UPDI.
The Xtiny addition is not free of charge. It requires a commercial add on. It is available from the MCS shop.
The Xtiny add on will support all Xtiny processors. And by that we mean processors from the mega range like mega4808, DA/DB/DD range like avr128da28 etc.

See also

MEGAX 490, AVRX 503

5.7.2 ATTINY202

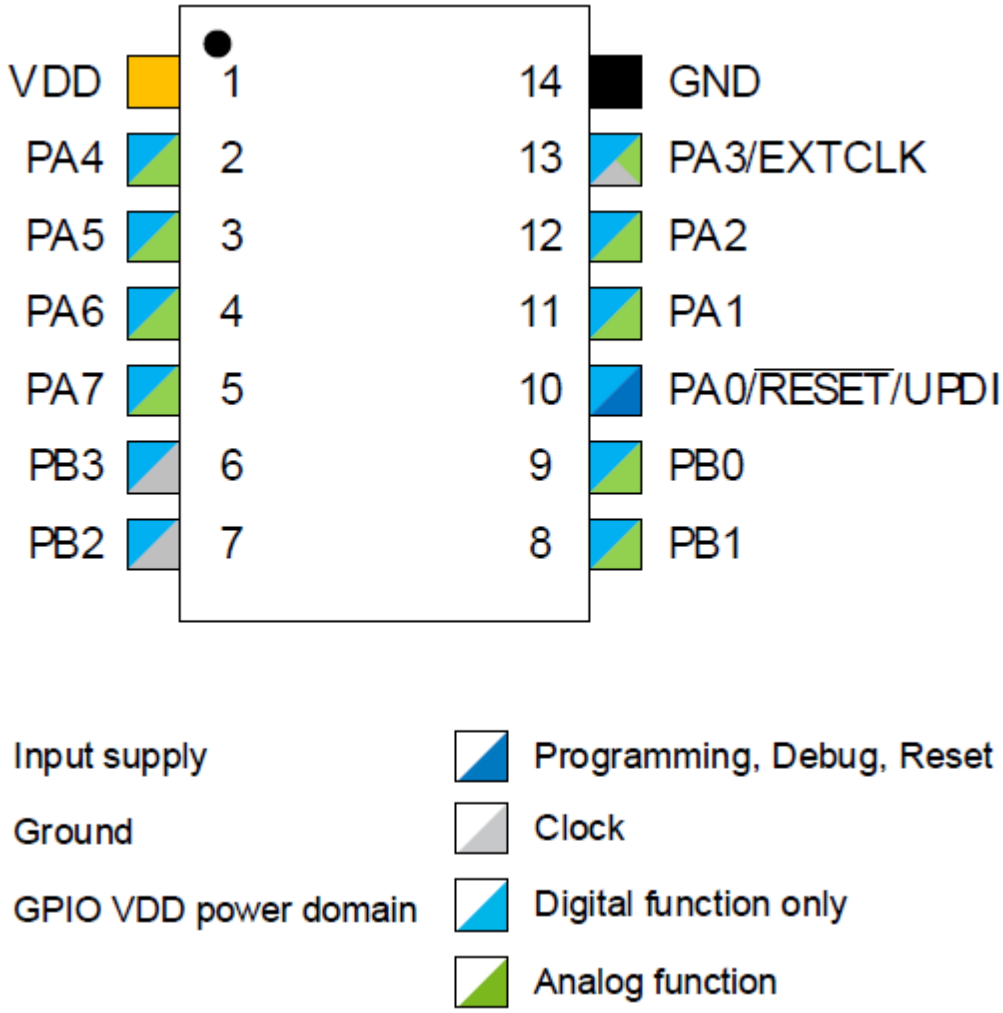
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.
The ATTINY202 and ATXTINY402 are the 2K/4K flash variants
- The manufacturer inc files have reference to VPORTB/VPORTC but these do not exist



5.7.3 ATTINY204

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

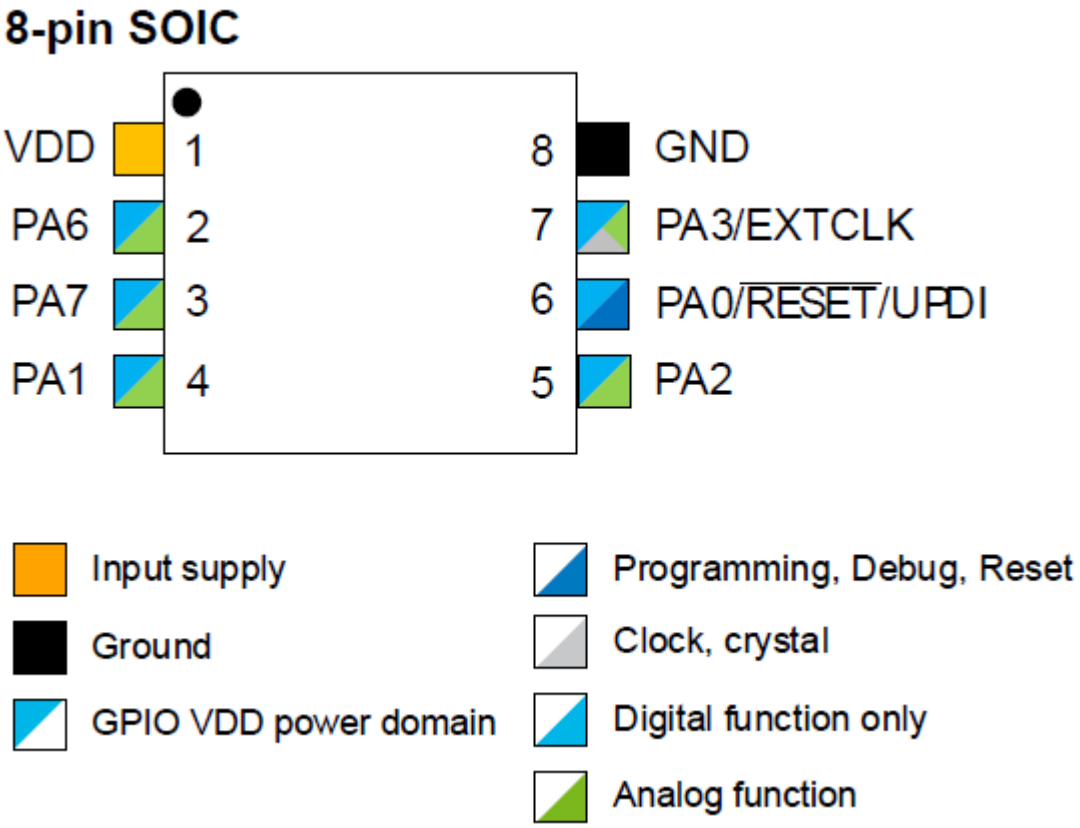
The ATTINY204, 404, 804 and 1604 are the 2K/4K/8K/16K flash variants



5.7.4 ATTINY212

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

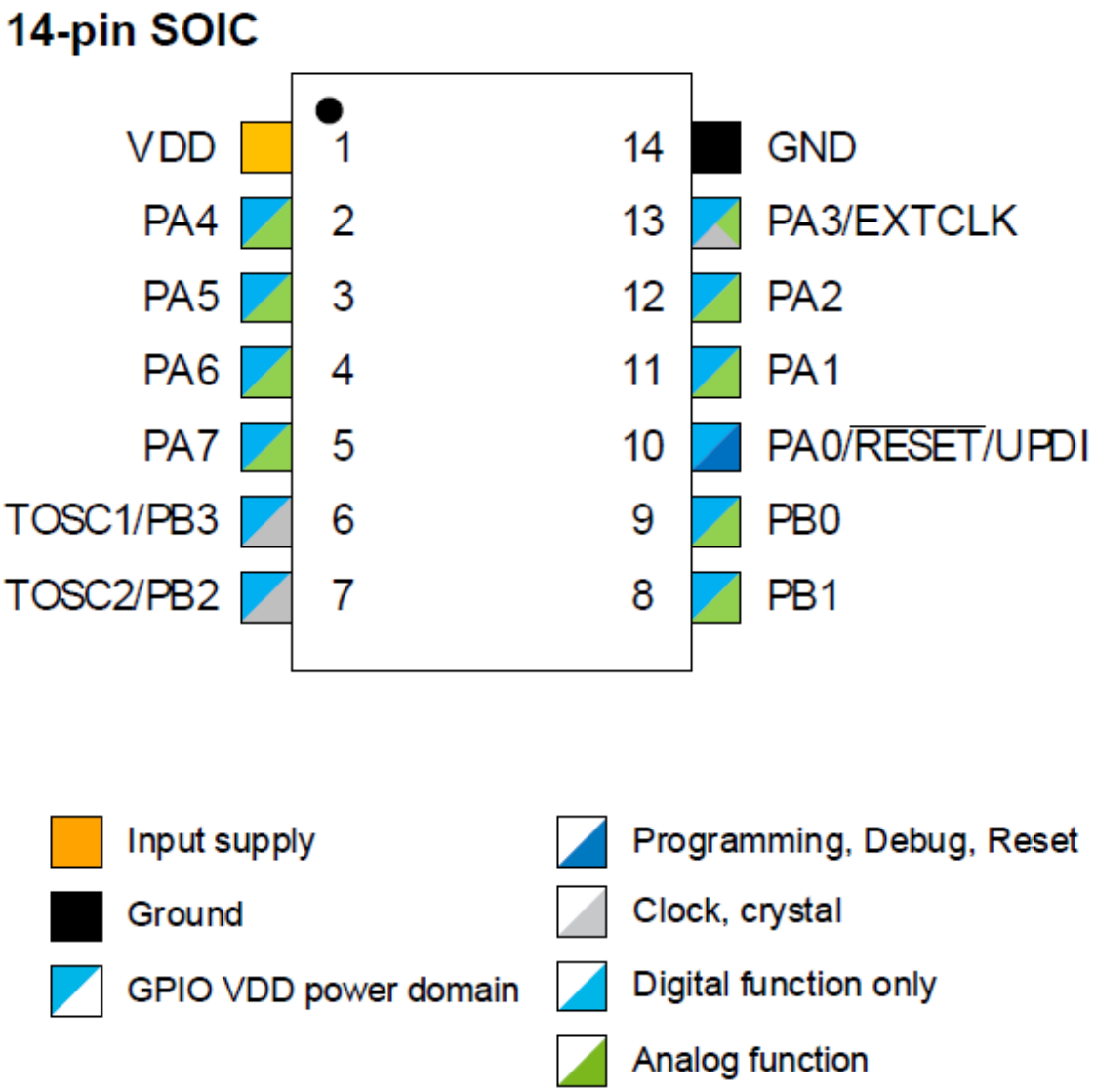
The ATTINY212 and 412 are the 2K/4K flash variants



5.7.5 ATTINY214

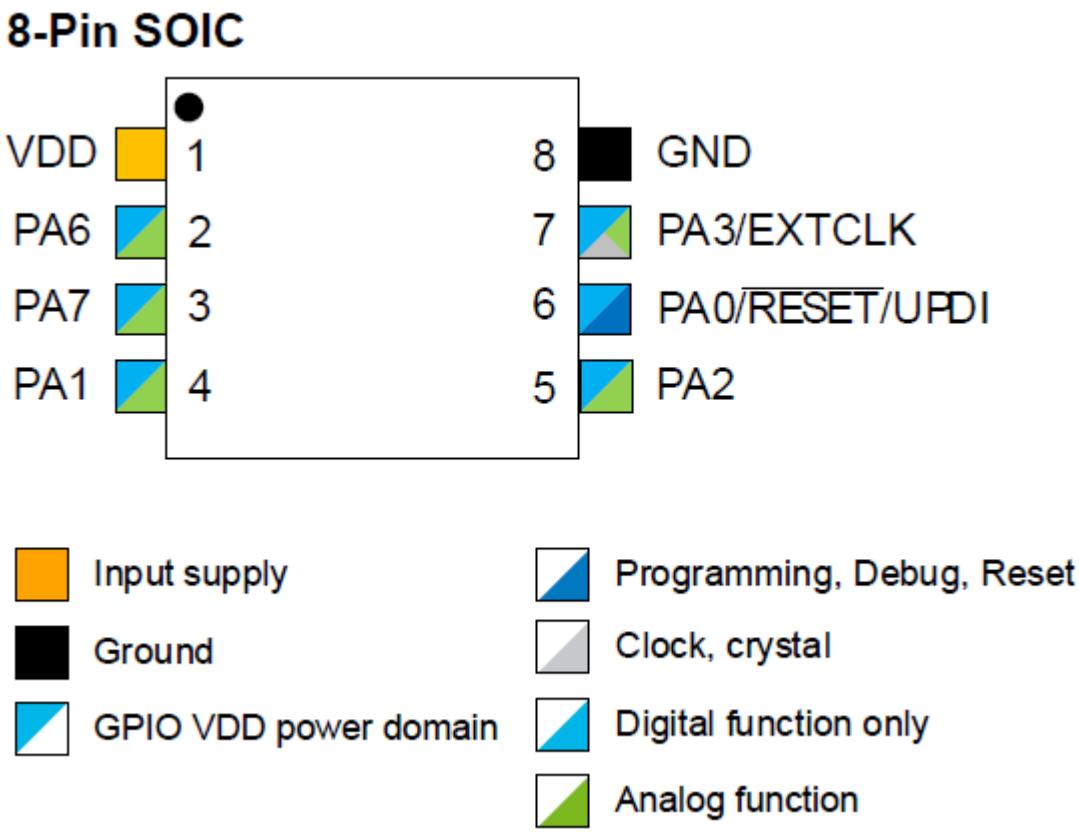
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY214/414 and ATXTINY814 are the 2K/4K/8K flash variants



5.7.6 ATTINY402

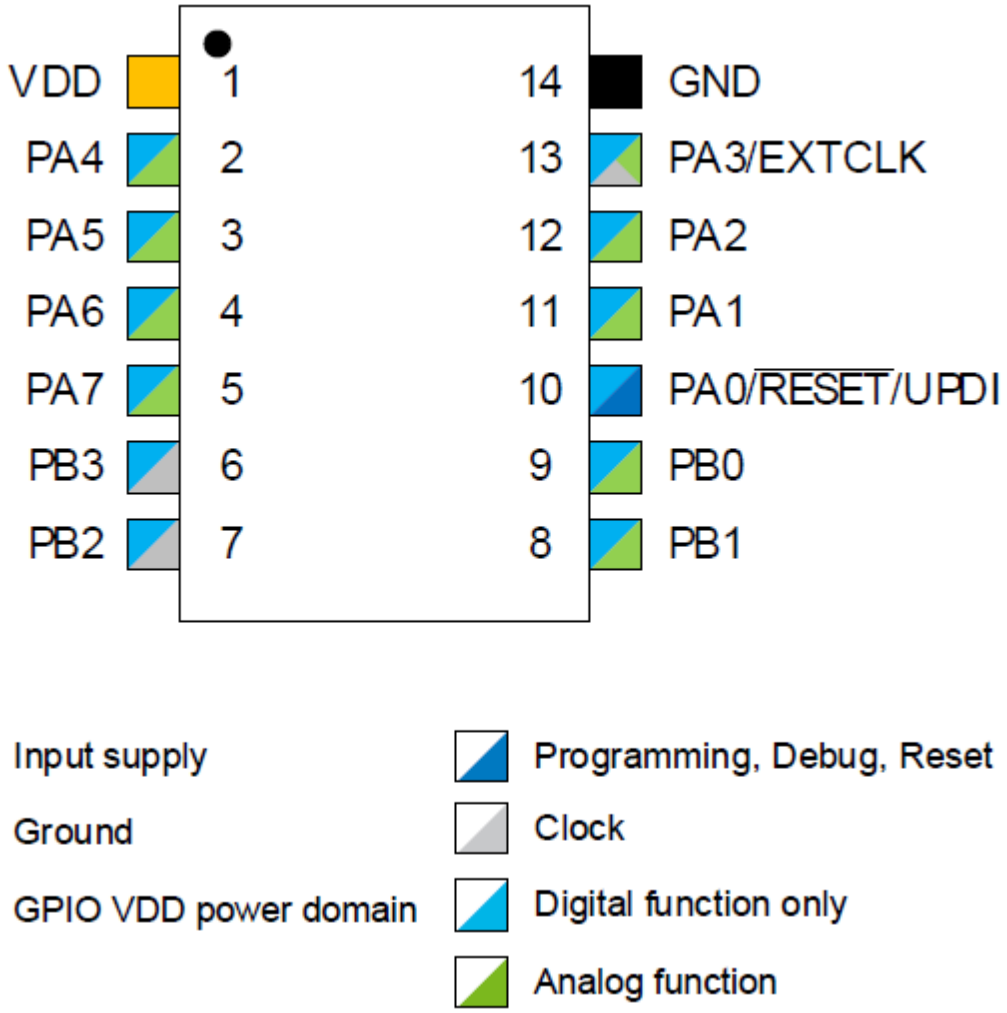
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.
The ATTINY202 and ATXTINY402 are the 2K/4K flash variants
- The manufacturer inc files have reference to VPORTB/VPORTC but these do not exist



5.7.7 ATTINY404

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY204, 404, 804 and 1604 are the 2K/4K/8K/16K flash variants

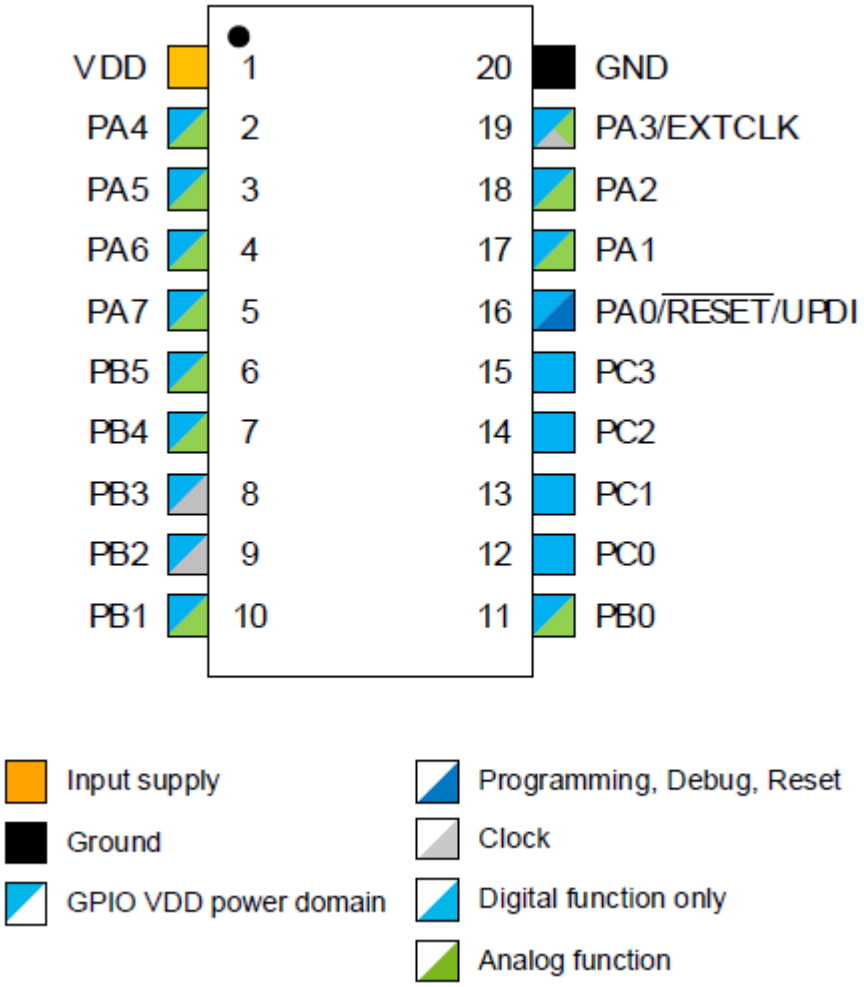


.

5.7.8 ATTINY406

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

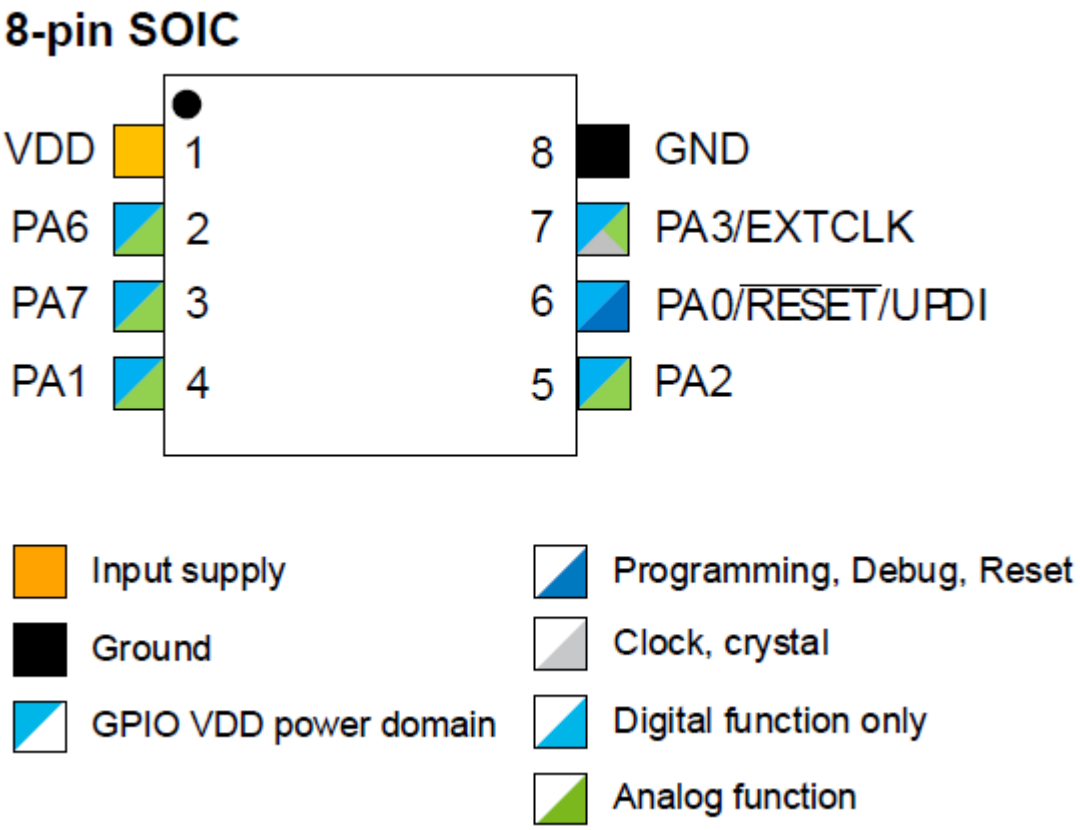
The ATTINY406, 804 and 1606 are the 4K/8K/16K flash variants



5.7.9 ATTINY412

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

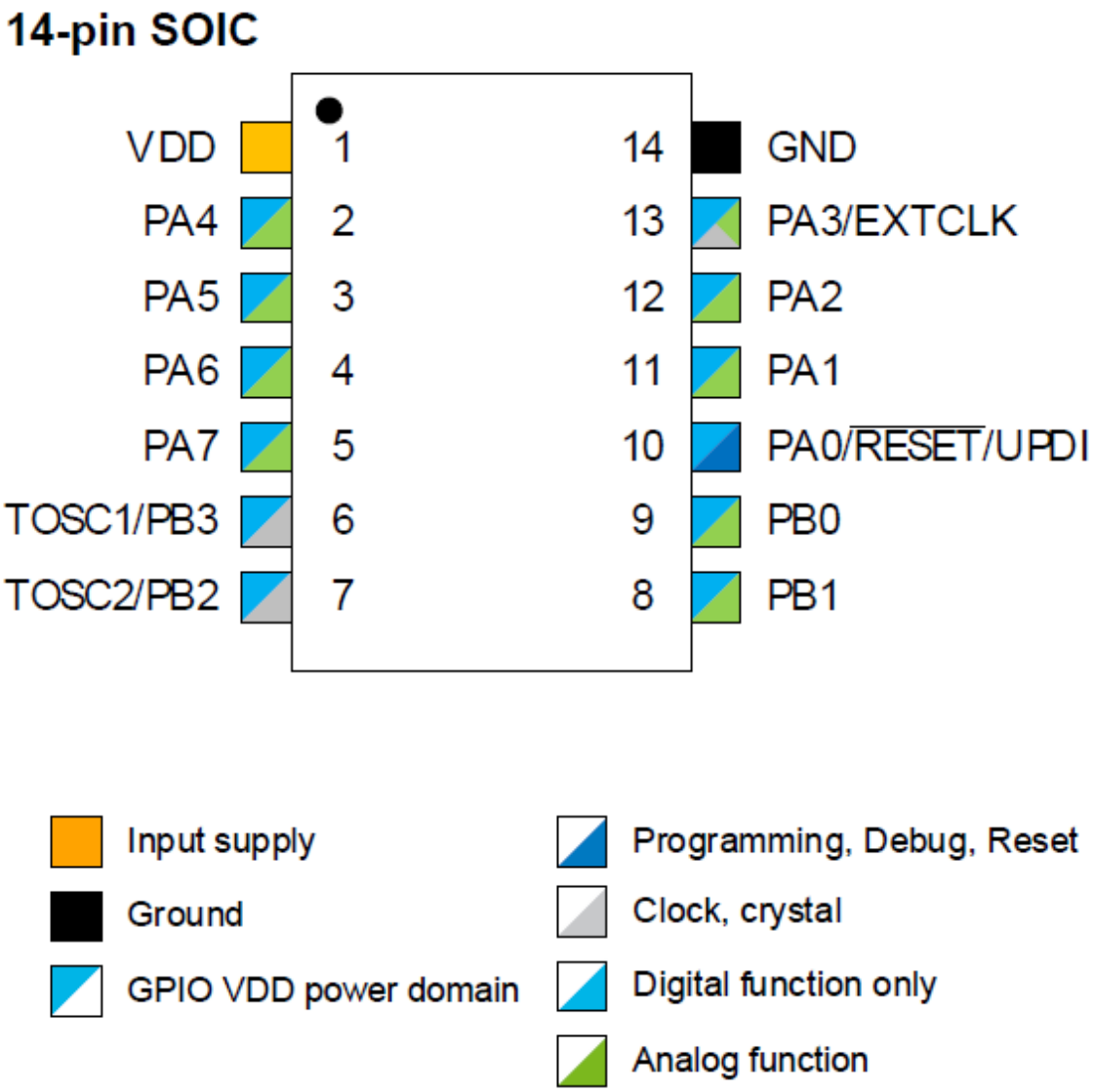
The ATTINY212 and 412 are the 2K/4K flash variants



5.7.10 ATTINY414

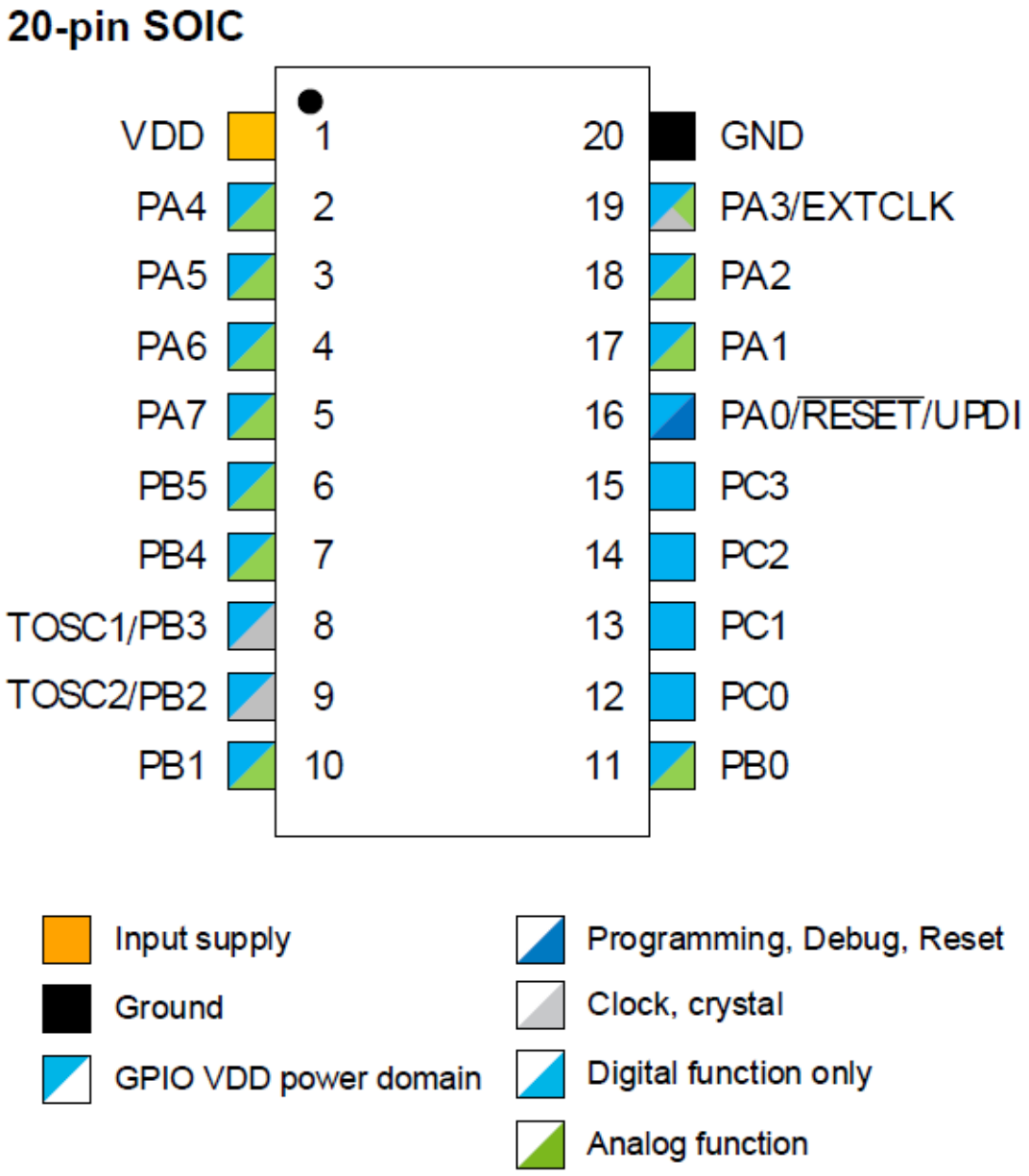
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY214/414 and ATXTINY814 are the 2K/4K/8K flash variants



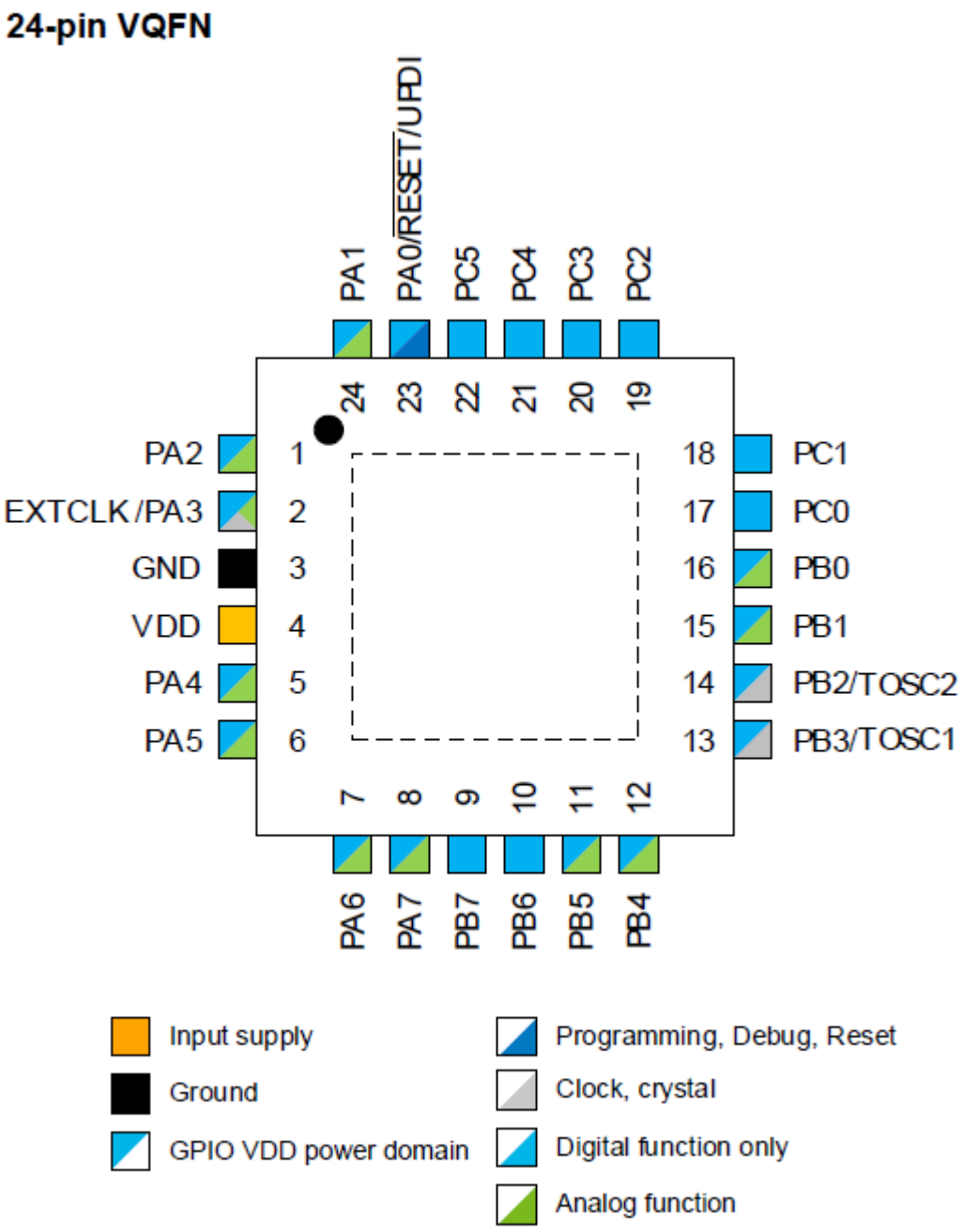
5.7.11 ATTINY416

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.
The ATTINY416 and 816 are the 4K/8K flash variants



5.7.12 ATTINY417

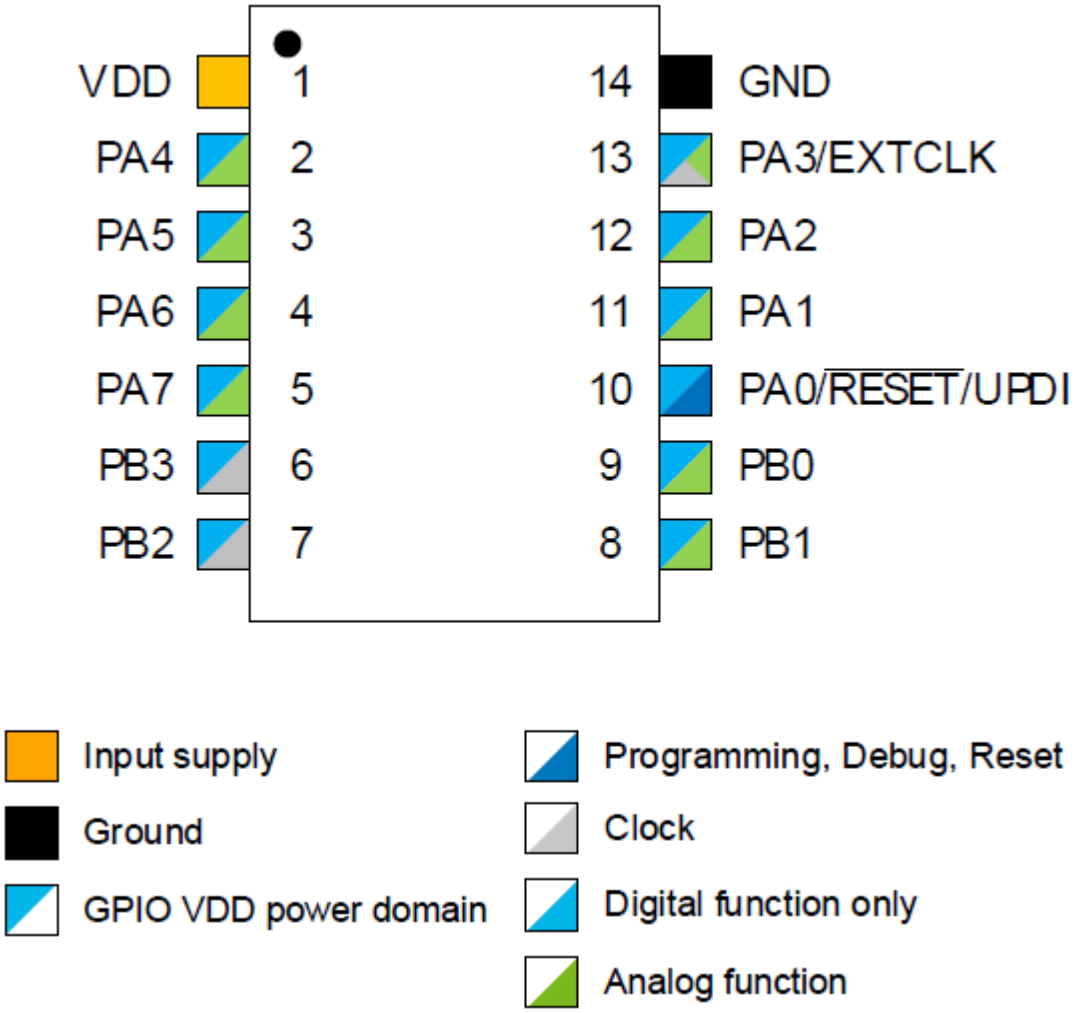
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.
The ATTINY417 and 817 are the 4K/8K flash variants in 24 pin casing



5.7.13 ATTINY804

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

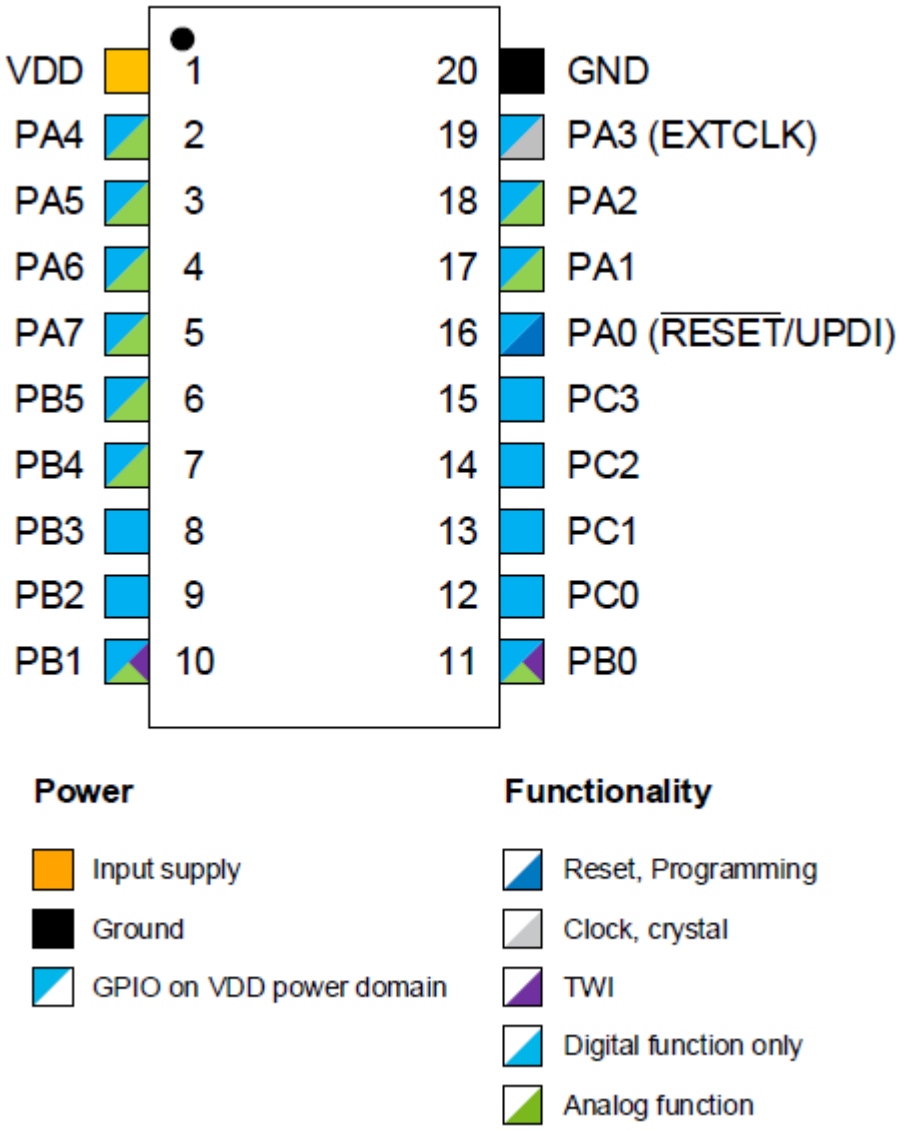
The ATTINY204, 404, 804 and 1604 are the 2K/4K/8K/16K flash variants



5.7.14 ATTINY806

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

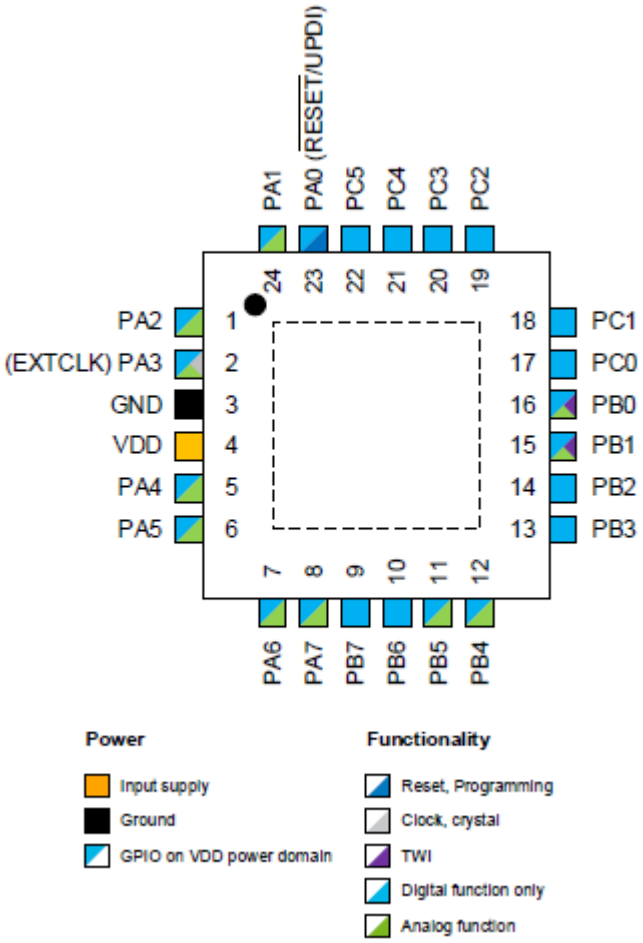
The ATTINY406, 804 and 1606 are the 4K/8K/16K flash variants



5.7.15 ATTINY807

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

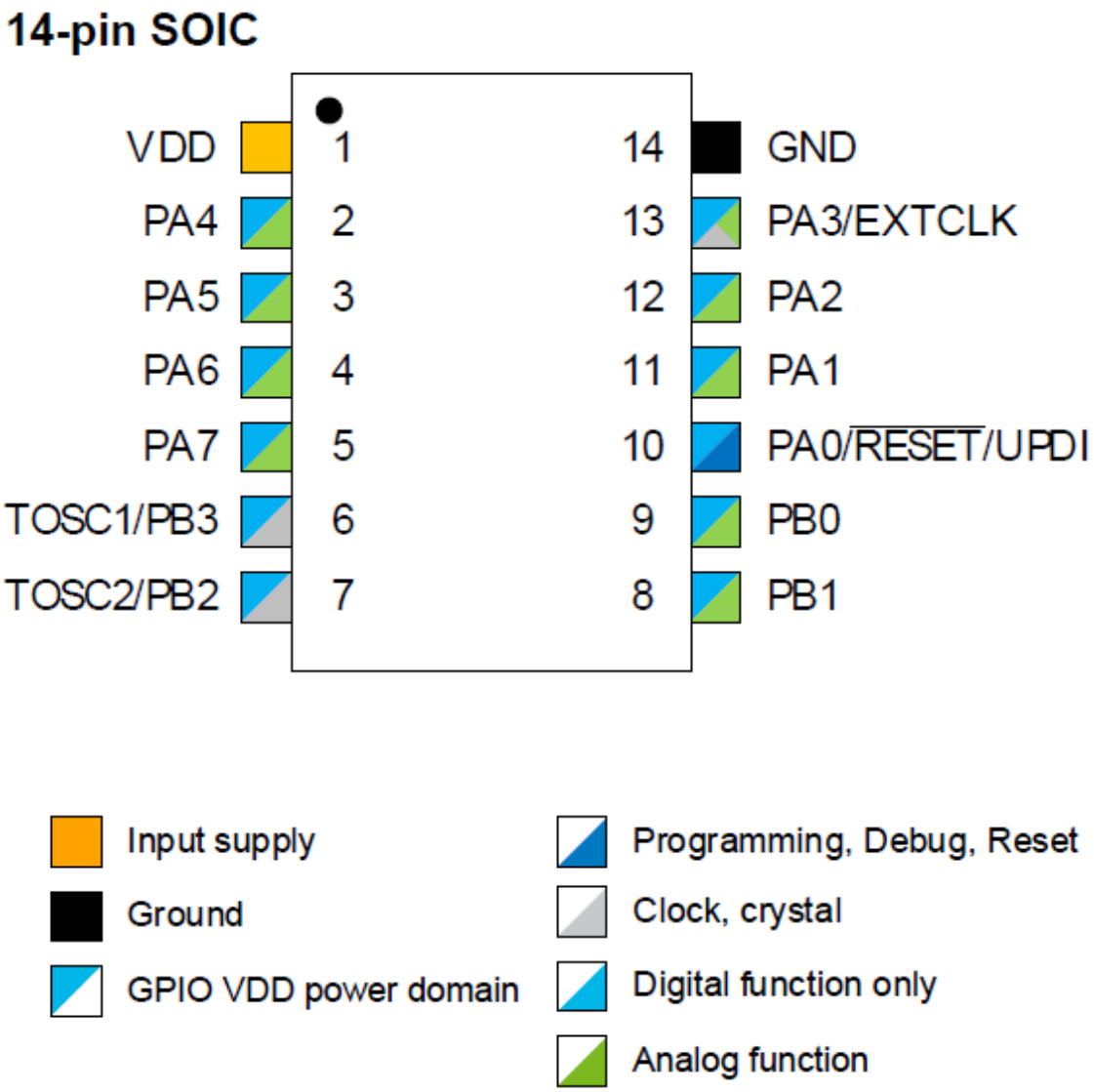
The ATTINY807 and 1607 are 8K/16K flash variants



5.7.16 ATTINY814

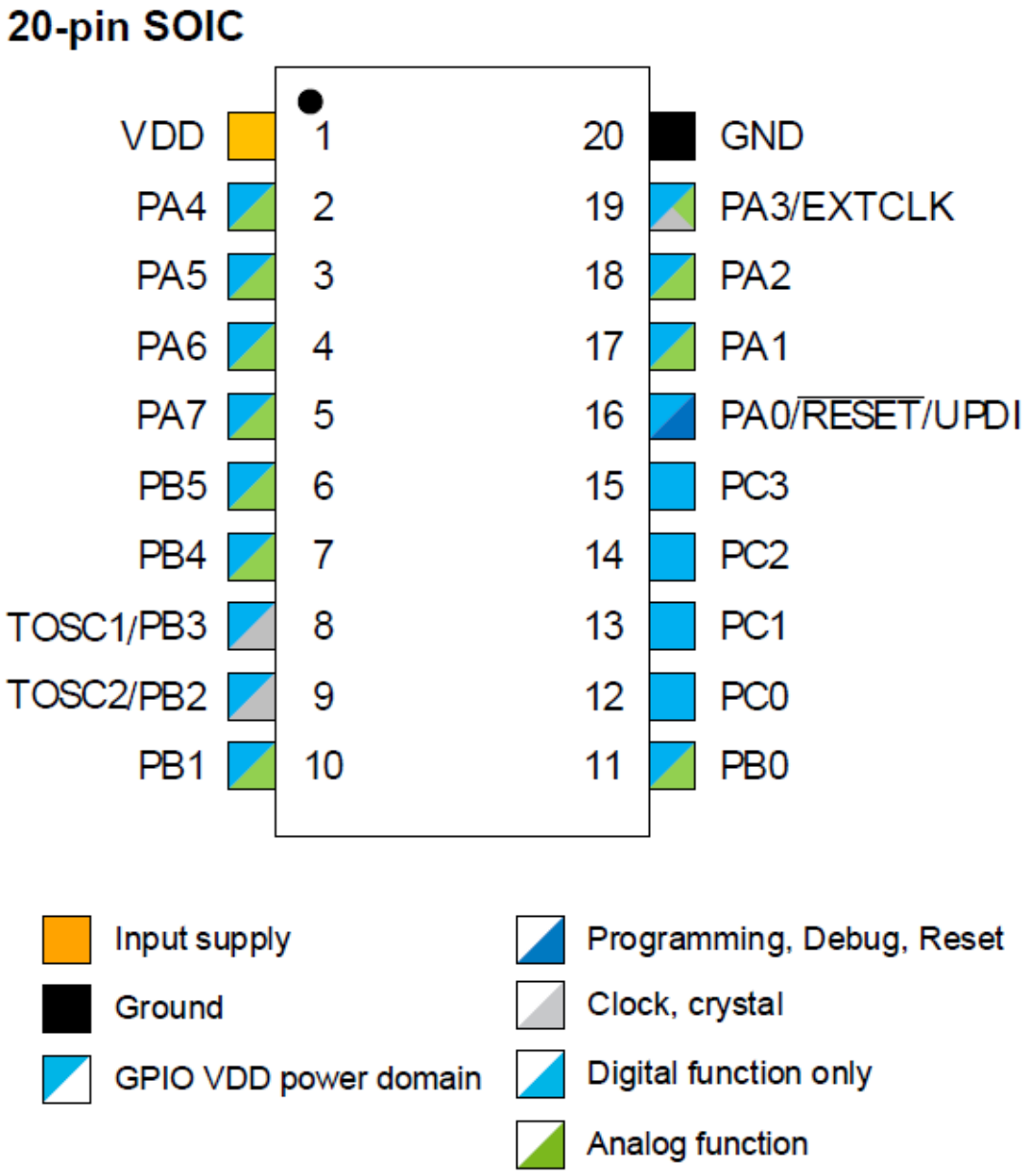
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY214/414 and ATXTINY814 are the 2K/4K/8K flash variants



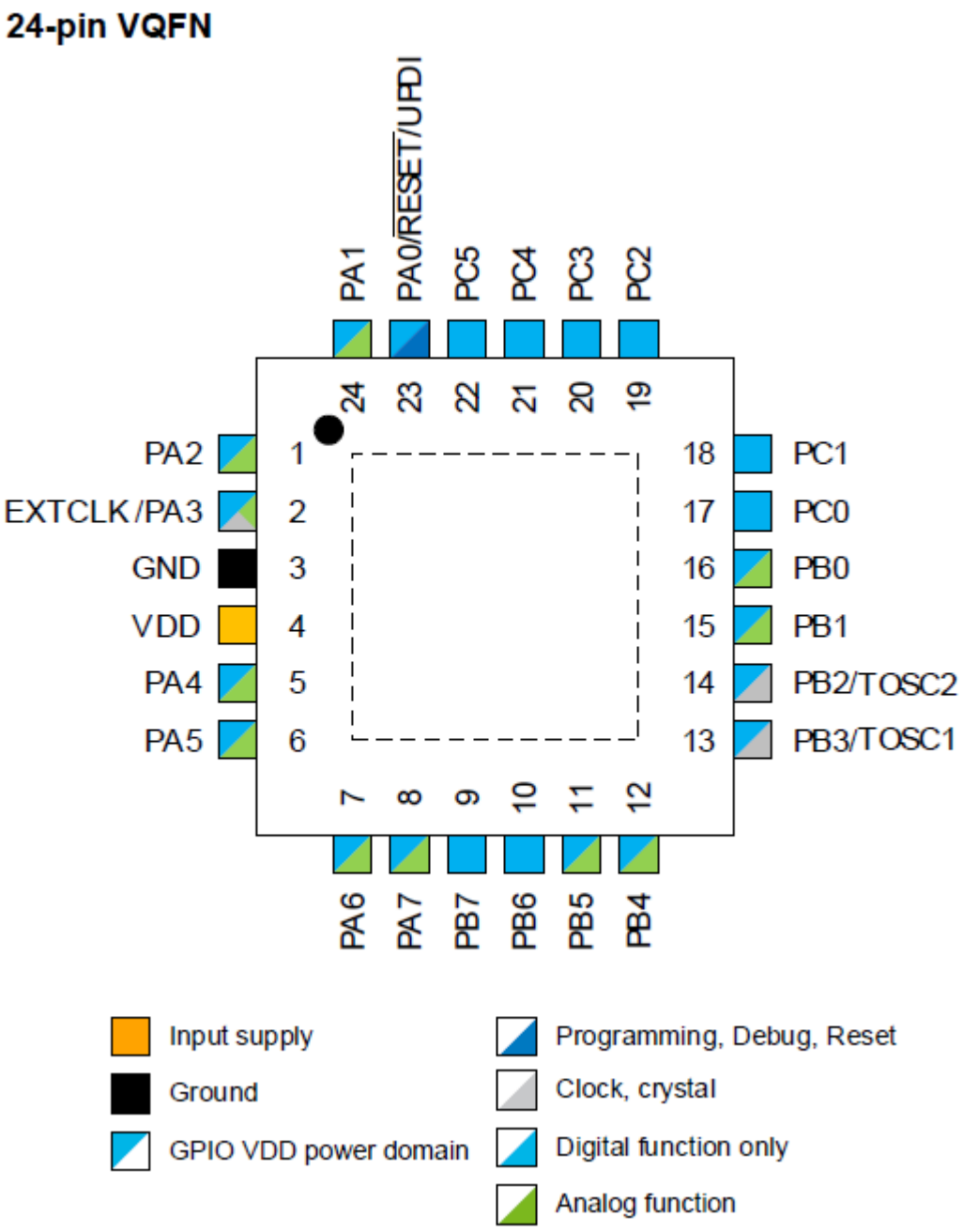
5.7.17 ATTINY816

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.
The ATTINY416 and 816 are the 4K/8K flash variants



5.7.18 ATTINY817

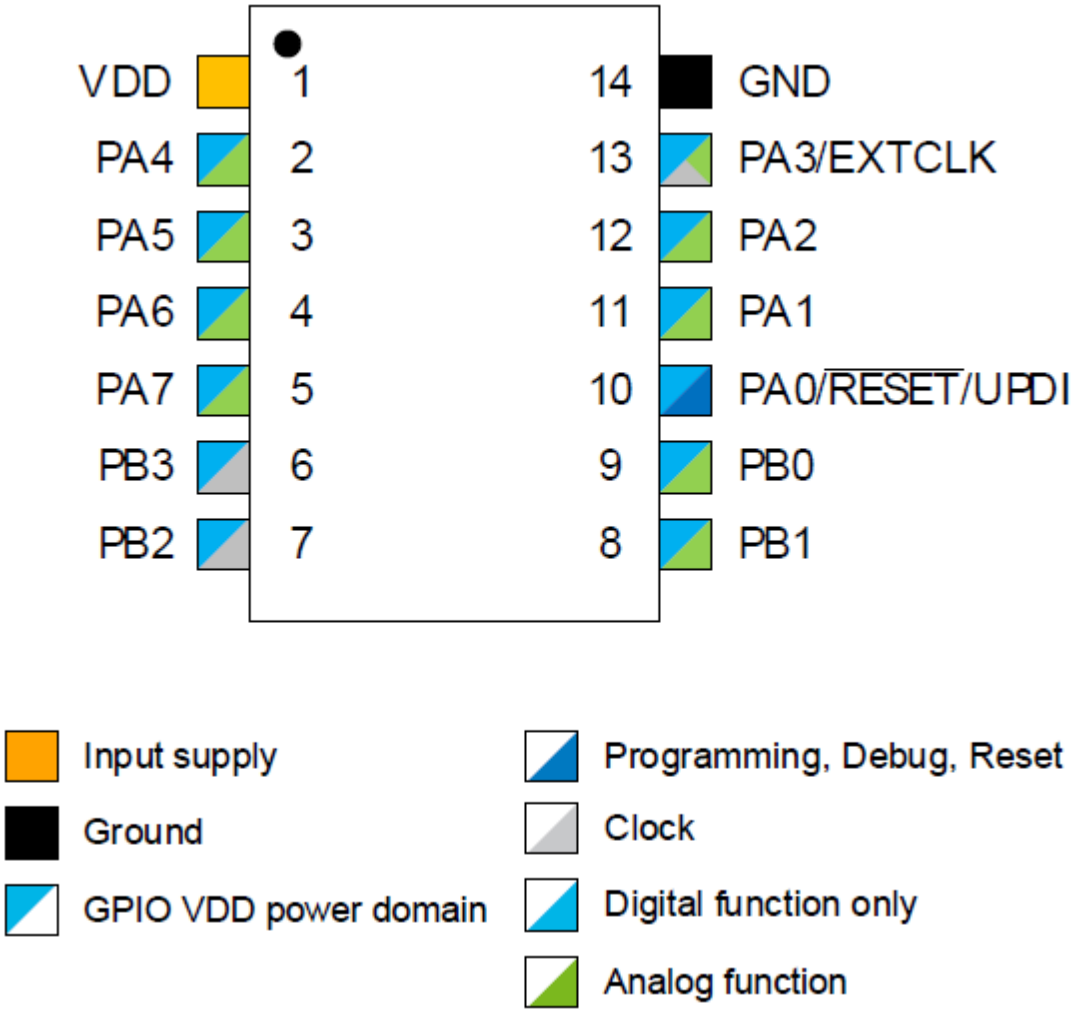
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.
The ATTINY417 and 817 are the 4K/8K flash variants in 24 pin casing



5.7.19 ATTINY1604

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

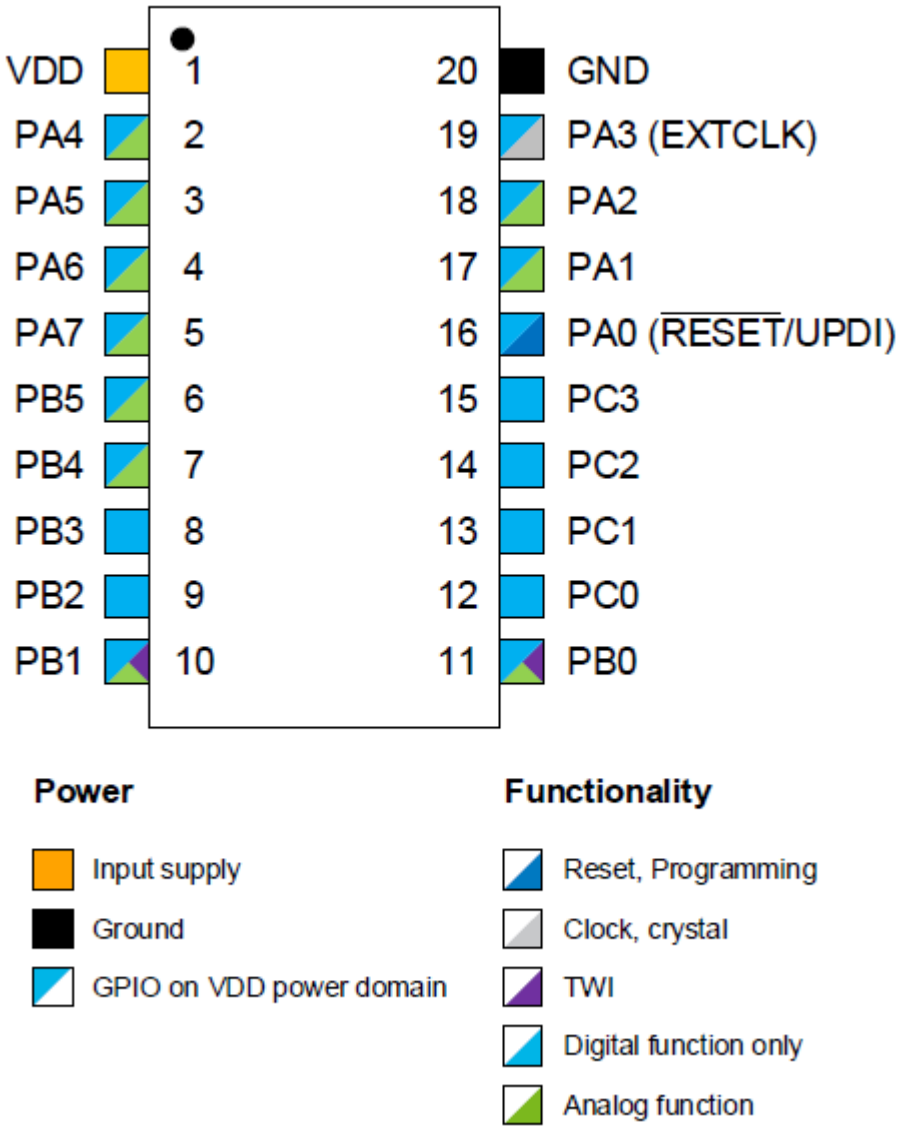
The ATTINY204, 404, 804 and 1604 are the 2K/4K/8K/16K flash variants



5.7.20 ATTINY1606

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

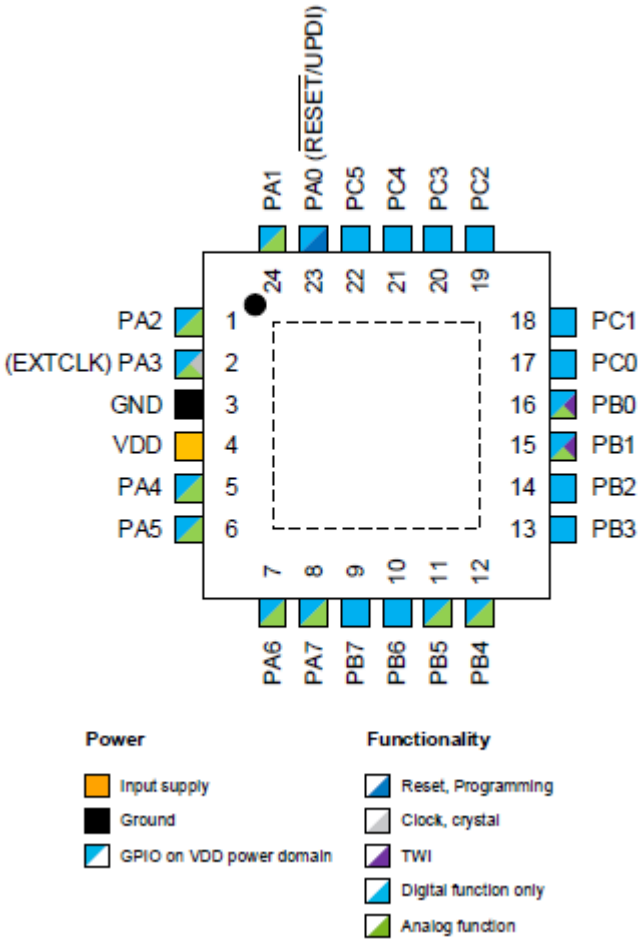
The ATTINY406, 804 and 1606 are the 4K/8K/16K flash variants



5.7.21 ATTINY1607

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

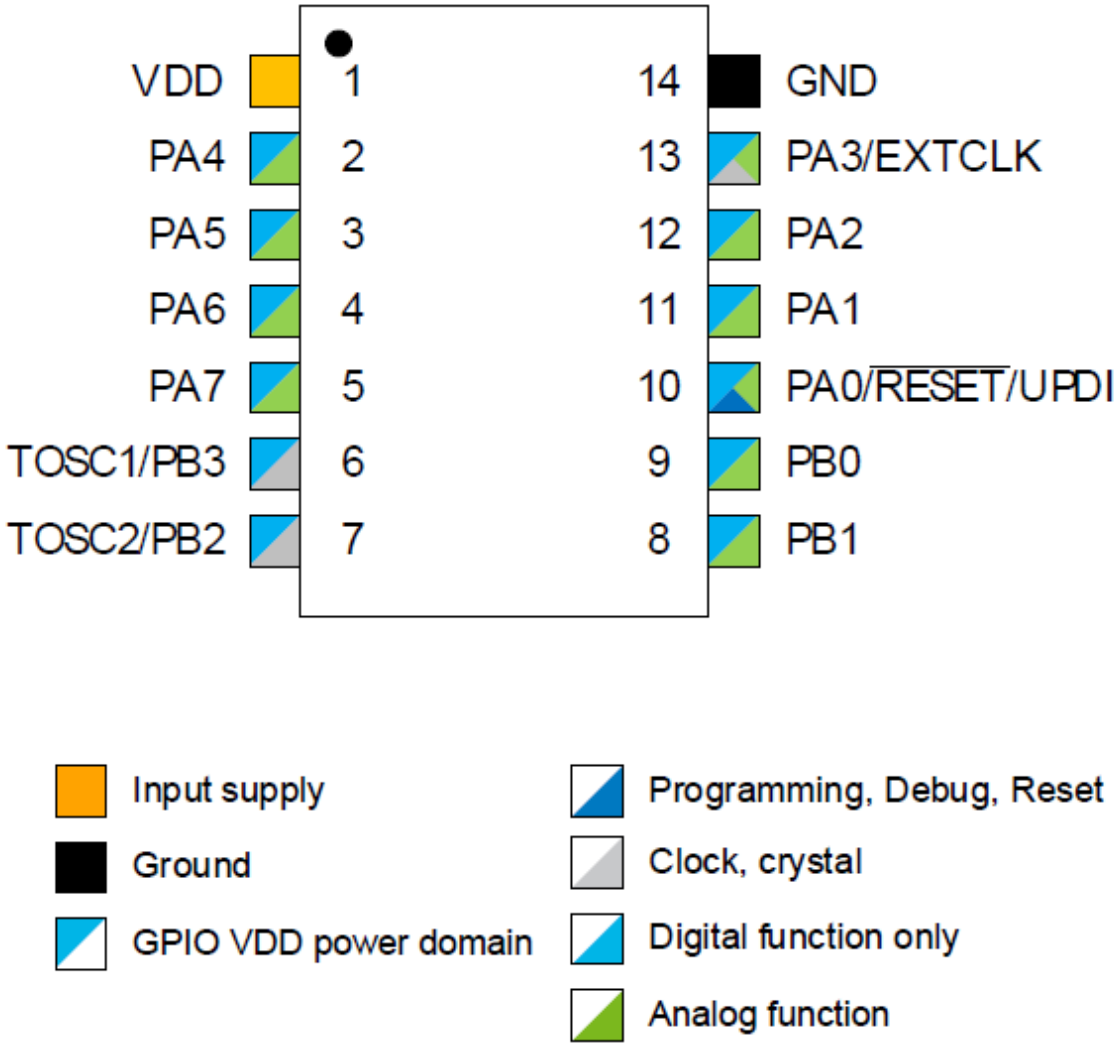
The ATTINY807 and 1607 are 8K/16K flash variants



5.7.22 ATTINY1614

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY1614/1616 and 1617 are 16K flash variants in 14/20/24 pin variants

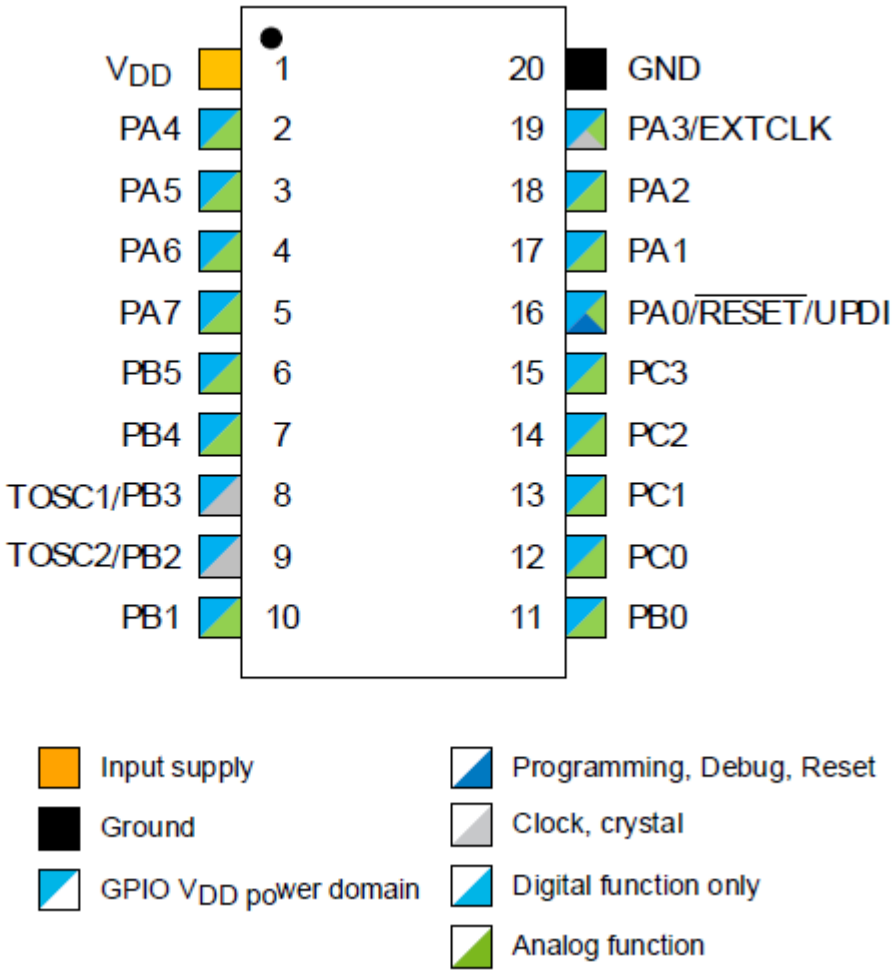


5.7.23 ATTINY1616

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY1614/1616 and 1617 are 16K flash variants in 14/20/24 pin variants

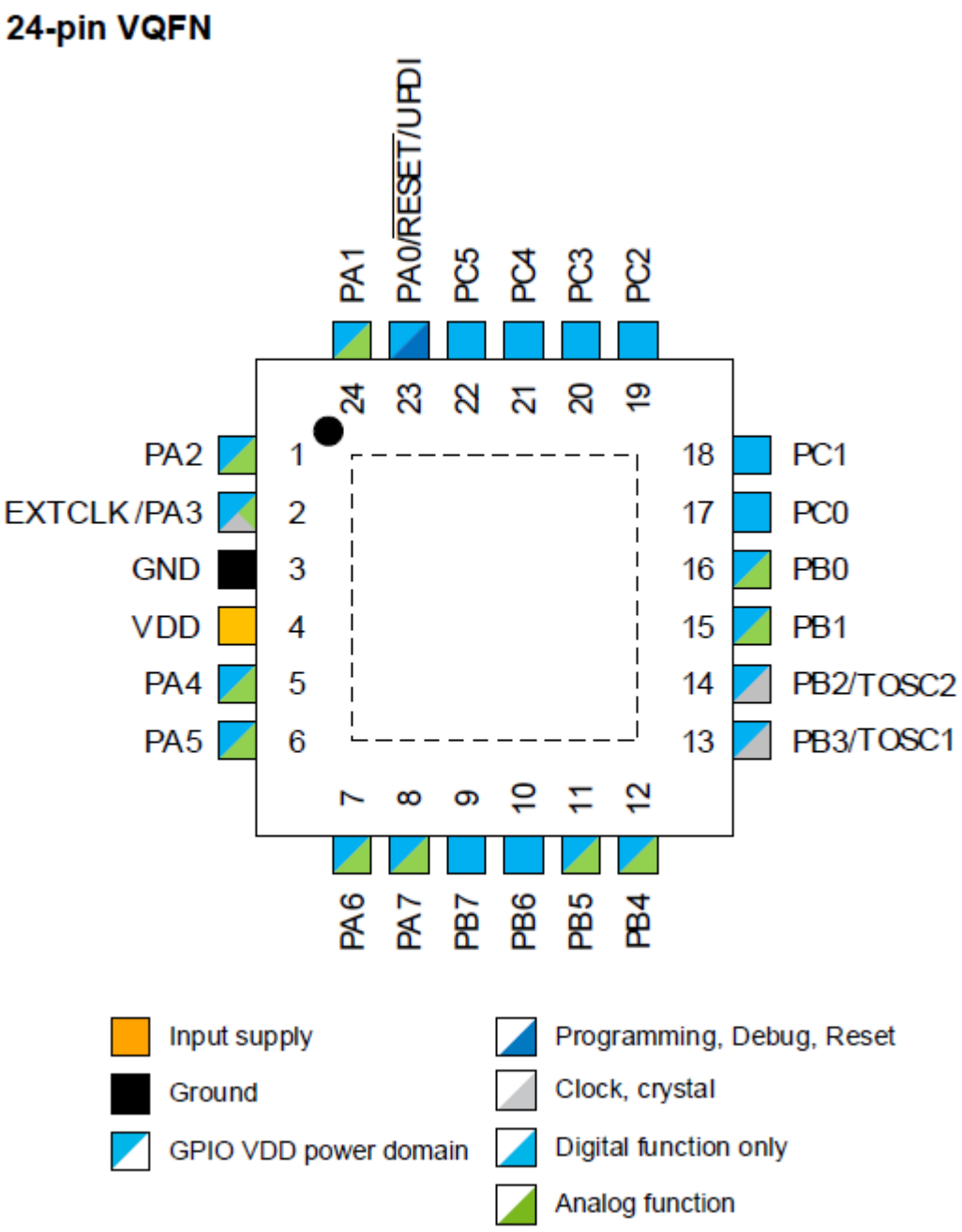
20-Pin SOIC



5.7.24 ATTINY1617

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

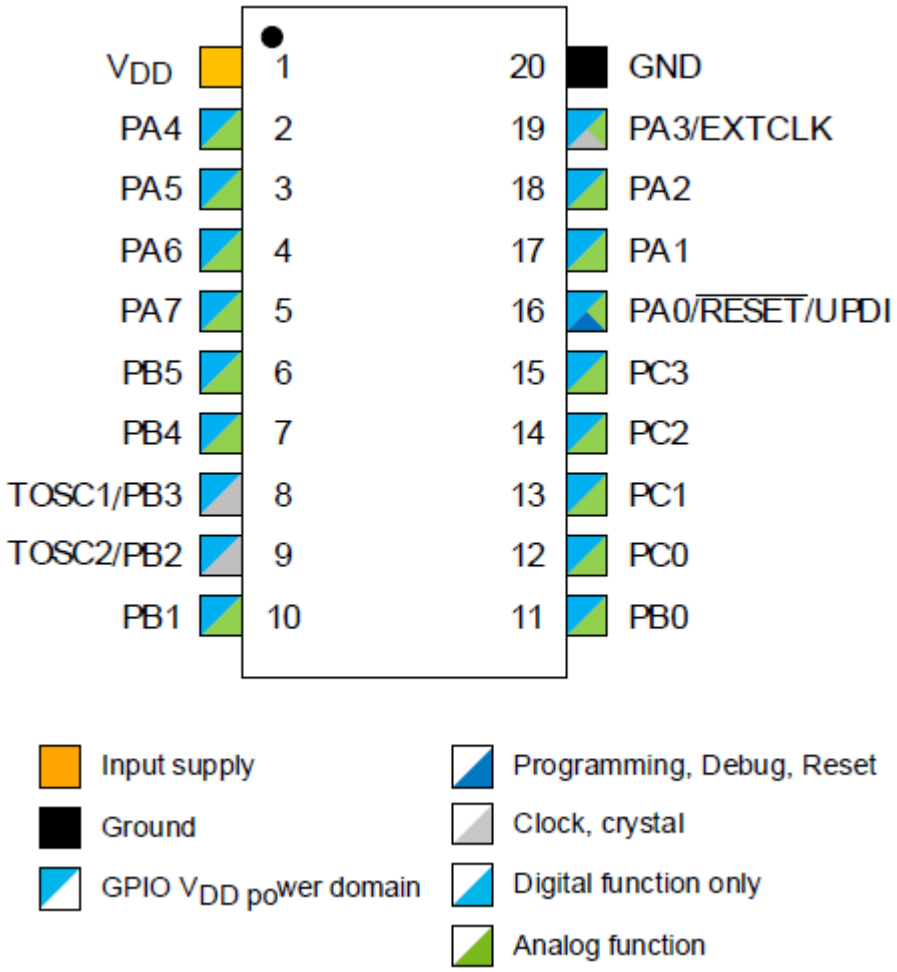
The ATTINY1614/1616 and 1617 are 16K flash variants in 14/20/24 pin variants



5.7.25 ATTINY3216

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY3216 and 3217 are the 32K flash variants in 20 pin and 24 pin casings

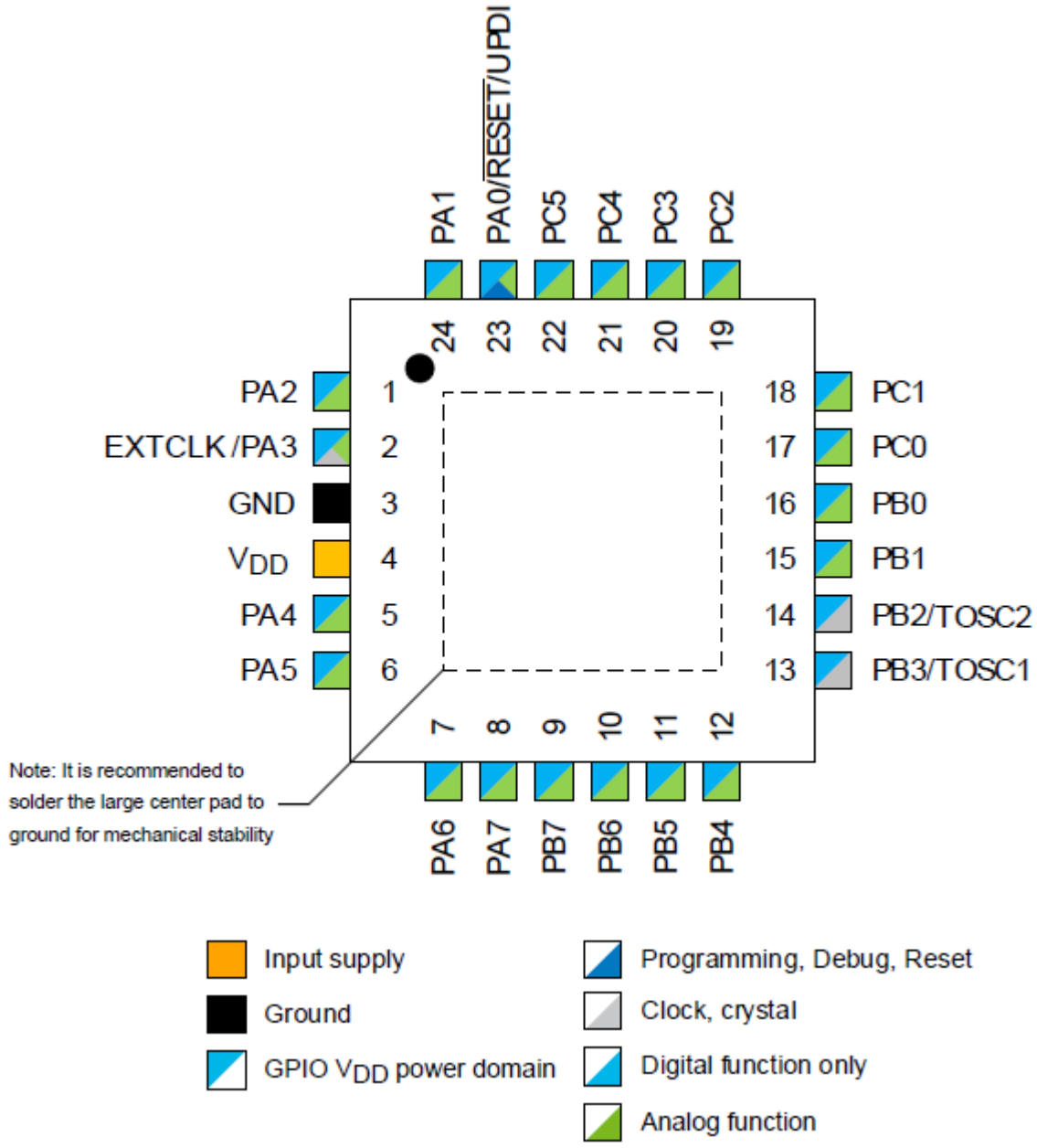


5.7.26 ATTINY3217

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460.

The ATTINY3216 and 3217 are the 32K flash variants in 20 pin and 24 pin casings

24-Pin VQFN



5.8 ATMEGAX

5.8.1 MEGAX

At MCS we refer to the new range of MEGA processors as the MEGAX. This because they look like Xmega processors but smaller.
Since the name XMEGA was taken by the actual XMEGA, and we use XTINY for the attinyX processors, we use MEGAX.
So why this distinction? And not use the atmel/microchip name? The answer is simple.
By using different names for the DAT file we want to make it clear that some hardware is different.
In fact these are all AVR chips but the core differs. As a user you do not need to care much. You can use the processors as usual.
The only important change is the programmer interface which is UPDI. Which is supported by BASCOM.

XMEGA and XTINY users will find many similar options. We based XTINY support on XMEGA. And MEGAX support on XTINY.
If you are unfamiliar with XTINY/MEGAX you best read the information about XTINY 460.
In fact we list all the differences under the XTINY topic. When not mentioned otherwise, the same applies for MEGAX.

The MEGAX is a bigger XTINY with more memory and hardware.
Some also come in DIP form.
We will only list the differences here with the XTINY.

 When you find info about the XTINY in the help, this info is also for the MEGAX unless there is a note about a difference. So when you read XTINY you can consider it equal to MEGAX.

Like the XTINY the MEGAX requires an add on. This is the same add on as the XTINY. So the XTINY Add On supports the MEGAX processors as well.
Use the update function to update the add on.

All tests have been performed using the MEGA4809-40 pins DIP.

- The biggest difference with XMEGA is that the MEGAX voltage range goes up to 5V.
- Biggest difference with XTINY is that there is more hardware like multiple USART's.

See also

AVRX 503 , XTINY 460

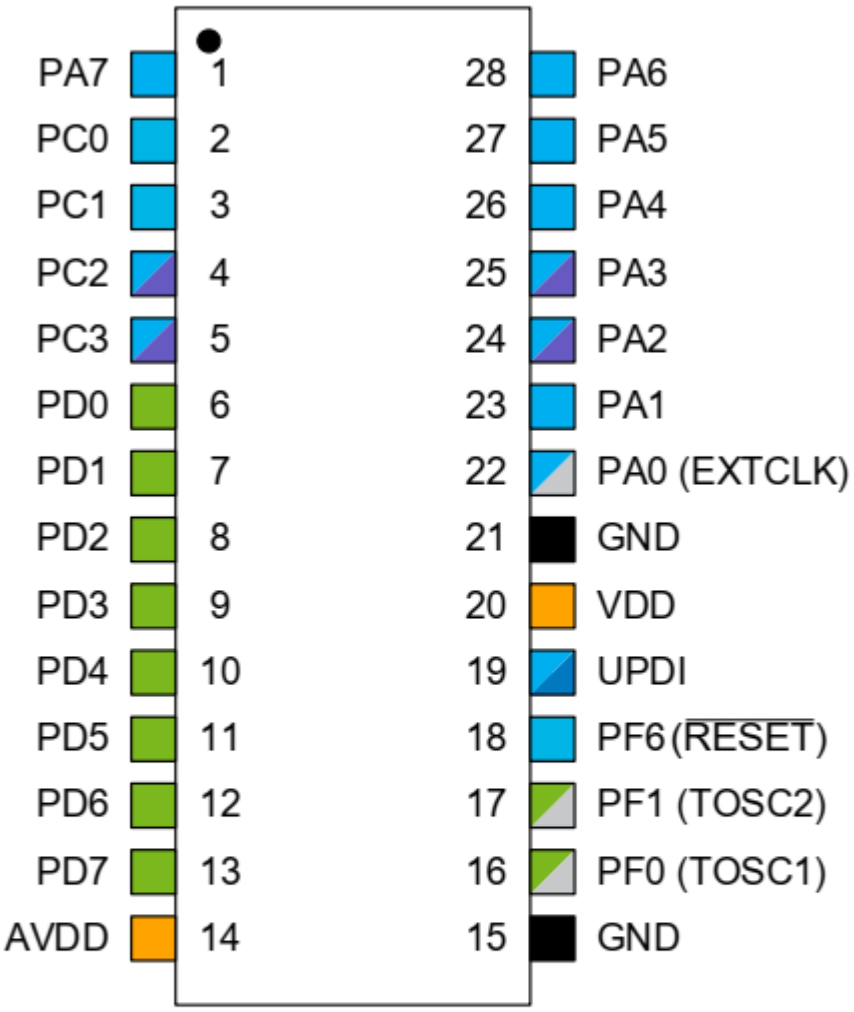
5.8.2 ATMEGA808

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 and MEGAX 490.

The ATMEGA808 comes in different casings. It has 8KB of flash and a maximum of 32 pins.
The ATMEGA809 has more pins : 40 and 48

Pinout

28-Pin SSOP



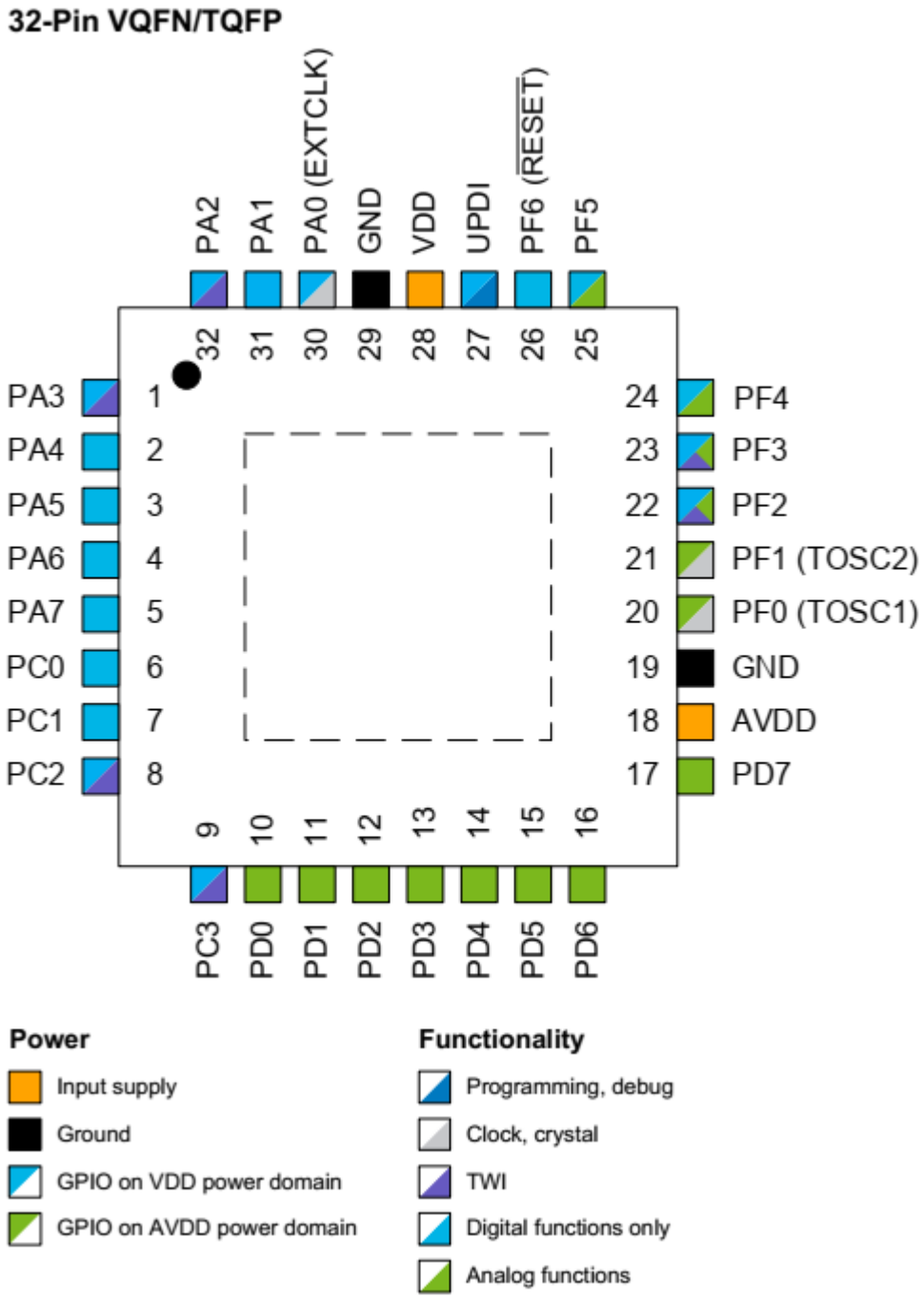
Power

- Input supply
- Ground
- GPIO on VDD power domain
- GPIO on AVDD power domain

Functionality

- Programming, debug
- Clock, crystal
- TWI
- Digital functions only
- Analog functions

VQFN/TQFP 32



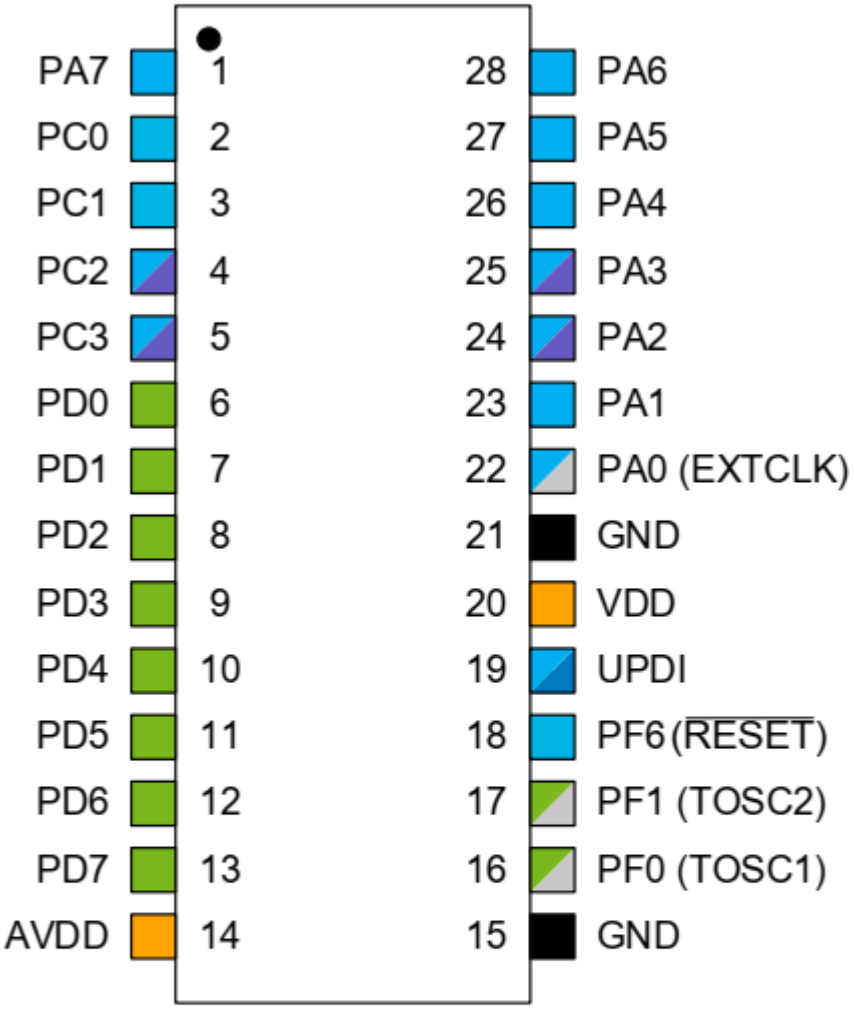
5.8.3 ATMEGA1608

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 and MEGAX 490.

The ATMEGA1680 comes in different casings. It has 16KB of flash and a maximum of 32 pins.
The ATMEGA1609 has more pins : 40 and 48

Pinout

28-Pin SSOP



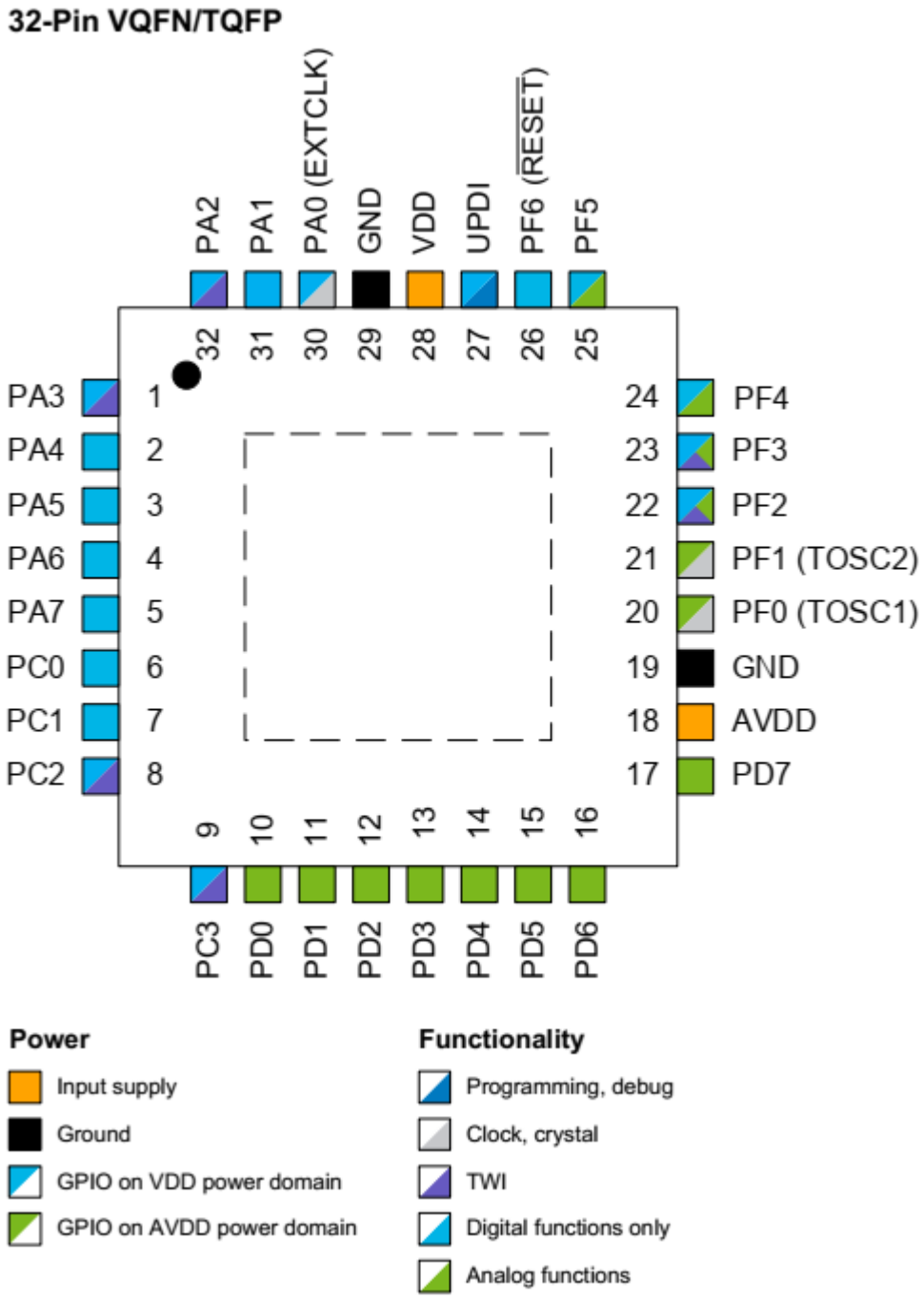
Power

- Input supply
- Ground
- GPIO on VDD power domain
- GPIO on AVDD power domain

Functionality

- Programming, debug
- Clock, crystal
- TWI
- Digital functions only
- Analog functions

VQFN/TQFP 32



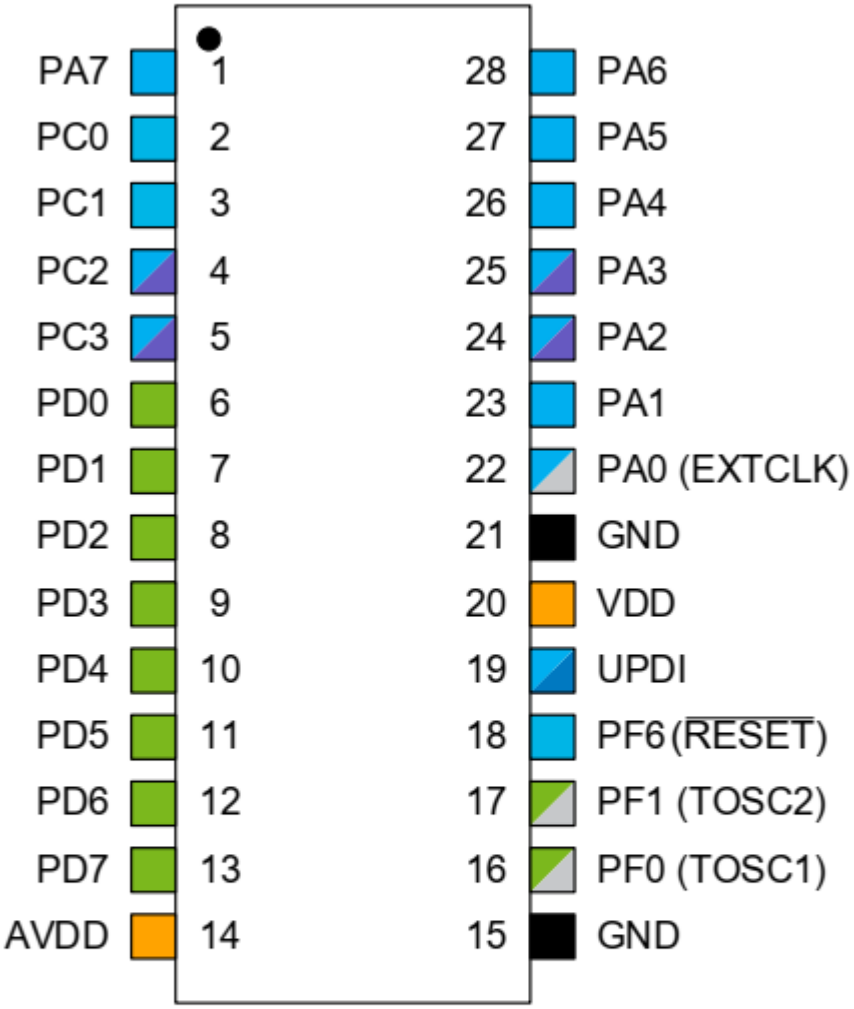
5.8.4 ATMEGA3208

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 and MEGAX 490.

The ATMEGA3280 comes in different casings. It has 32KB of flash and a maximum of 32 pins.
The ATMEGA3209 has more pins : 40 and 48

Pinout

28-Pin SSOP



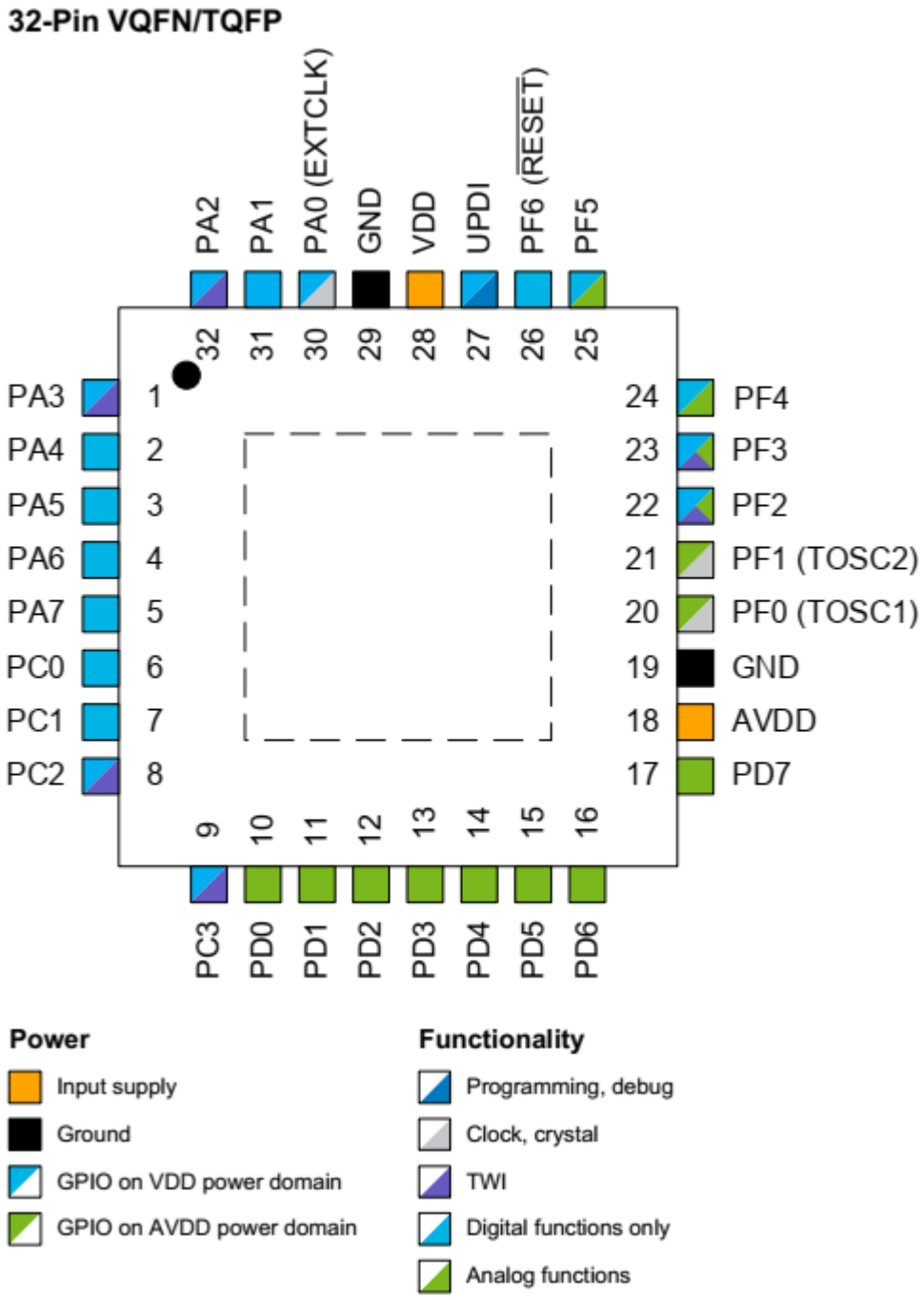
Power

- Input supply
- Ground
- GPIO on VDD power domain
- GPIO on AVDD power domain

Functionality

- Programming, debug
- Clock, crystal
- TWI
- Digital functions only
- Analog functions

VQFN/TQFP 32

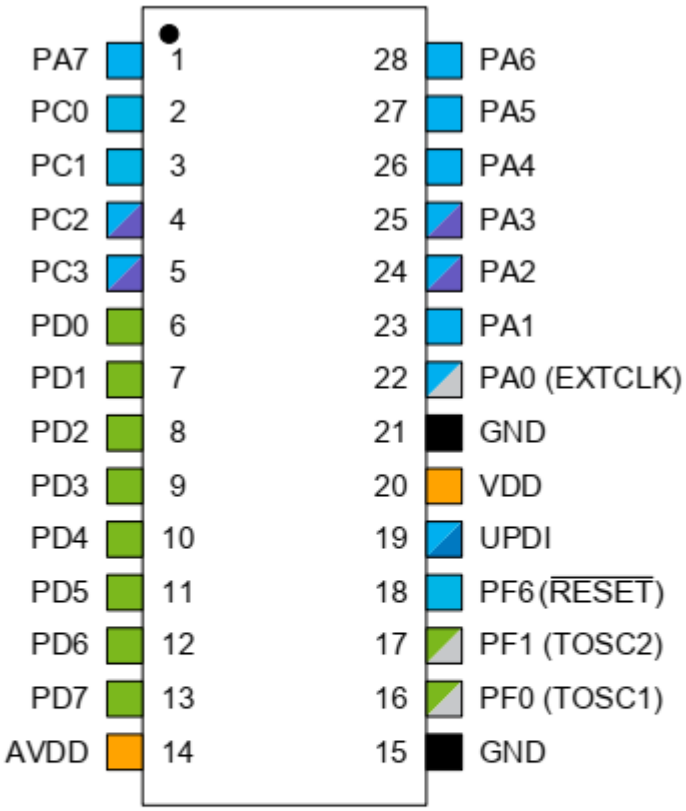


5.8.5 ATMEGA4808

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 and MEGAX 490.

The ATMEGA4808 comes in VQFN and TQFP. It has 48KB of flash and 32 pins.
The ATMEGA4808 also comes in SSOP. It has 48KB of flash and 28 pins.

28-Pin SSOP



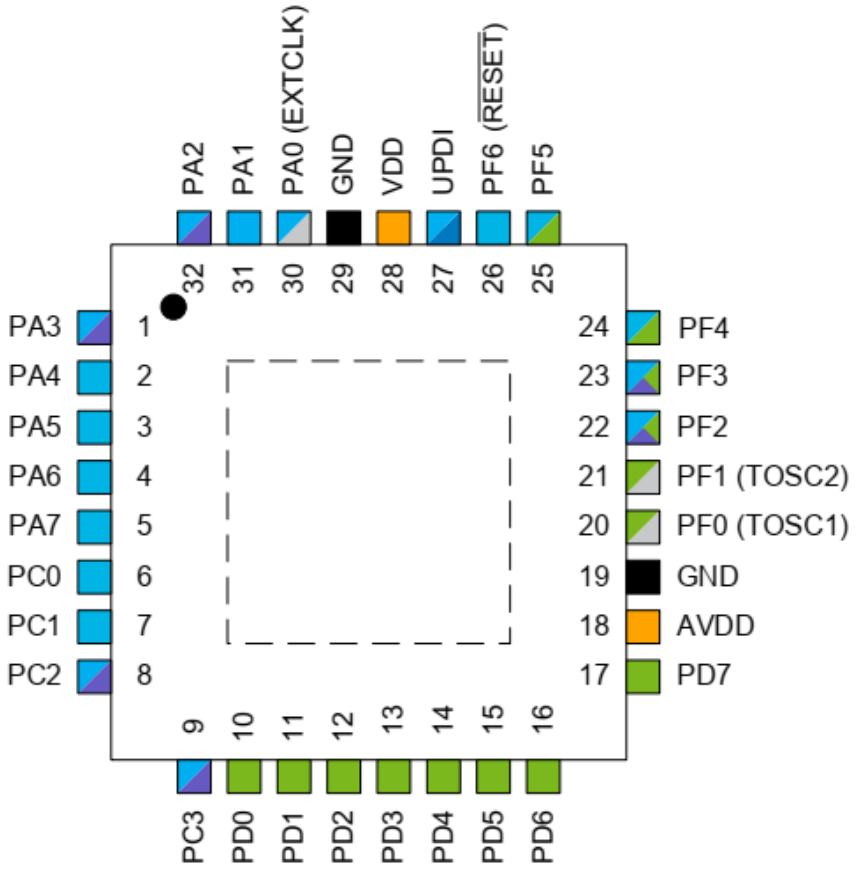
Power

- Input supply
- Ground
- GPIO on VDD power domain
- GPIO on AVDD power domain

Functionality

- Programming, debug
- Clock, crystal
- TWI
- Digital functions only
- Analog functions

32-Pin VQFN/TQFP



Power

- Input supply
- Ground
- GPIO on VDD power domain
- GPIO on AVDD power domain

Functionality

- Programming, debug
- Clock, crystal
- TWI
- Digital functions only
- Analog functions

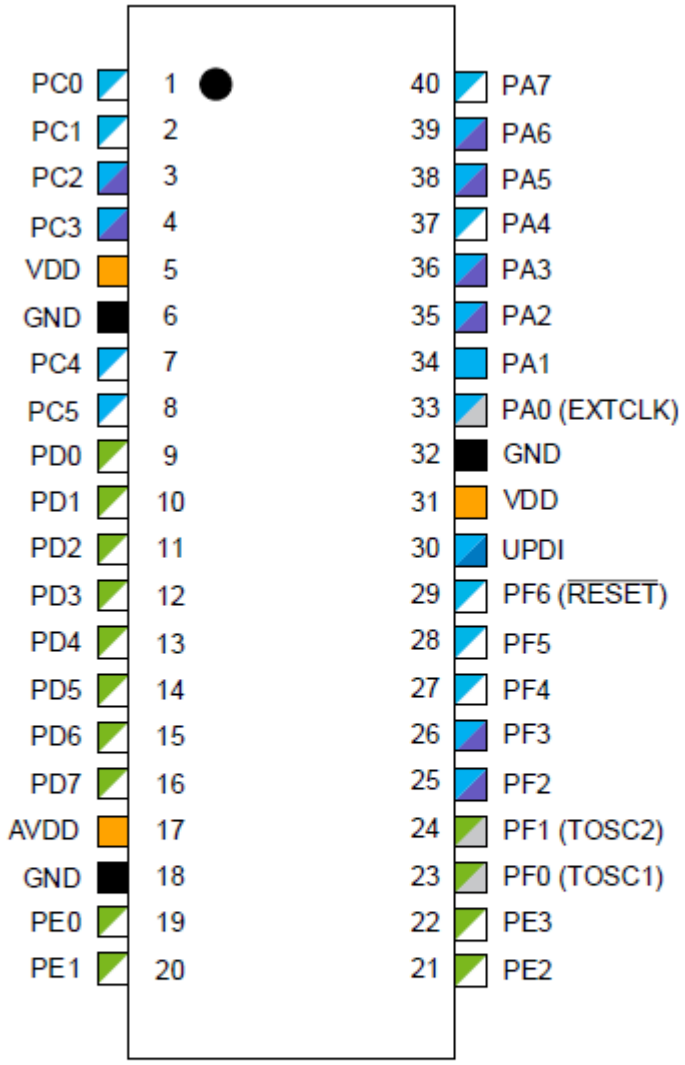
Note: The center pad underneath the QFN packages can be connected to PCB ground or left electrically unconnected. Solder or glue it to the board to ensure good mechanical stability. If the center pad is not attached, the package might loosen from the board.

5.8.6 ATMEGA4809

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 and MEGAX 490.

The ATMEGA4809 comes in different casings. It has 48KB of flash and a maximum of 48 pins.
There is a DIP version with 40 pins with the same die. PORTB has no pins however

40-Pin PDIP



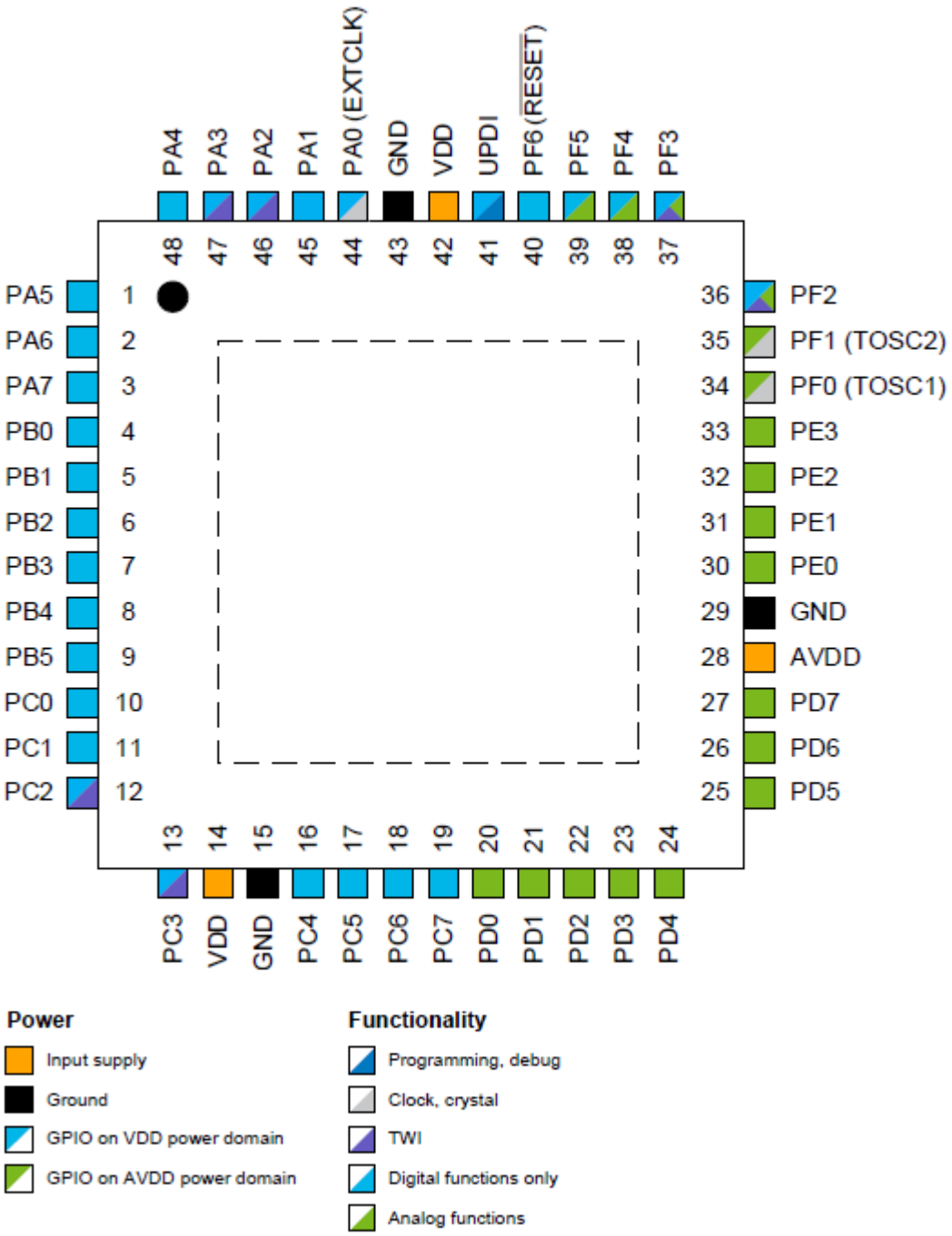
Power

- Input supply
- Ground
- GPIO on VDD power domain
- GPIO on AVDD power domain

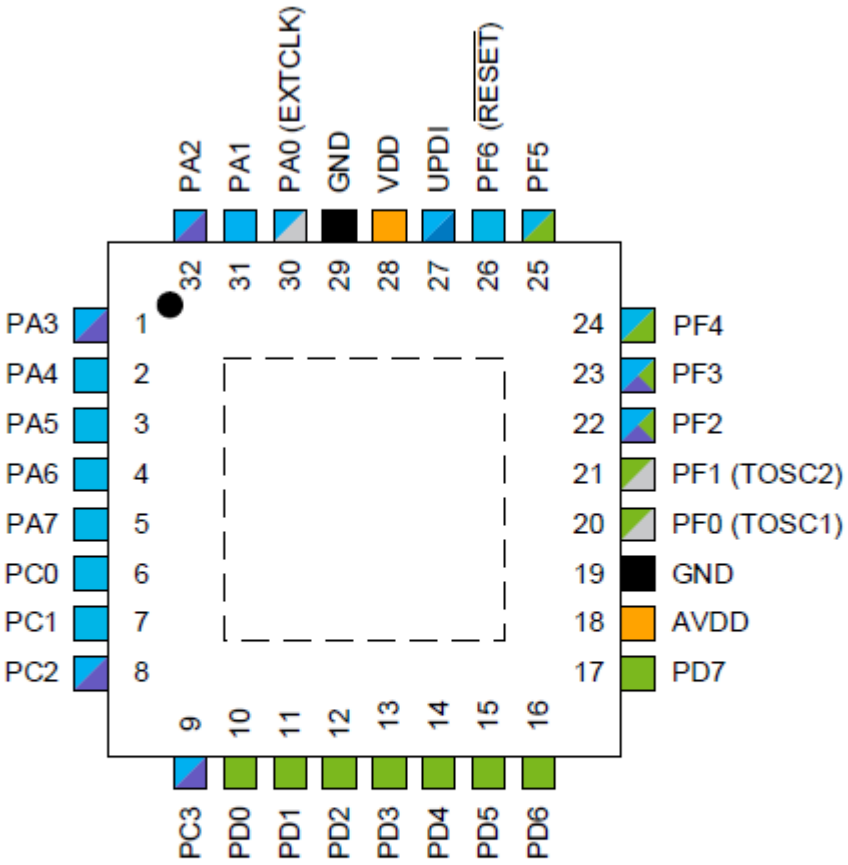
Functionality

- Programming, debug
- Clock, crystal
- TWI
- Digital functions only
- Analog functions

48-Pin VQFN/TQFP



32-Pin VQFN/TQFP

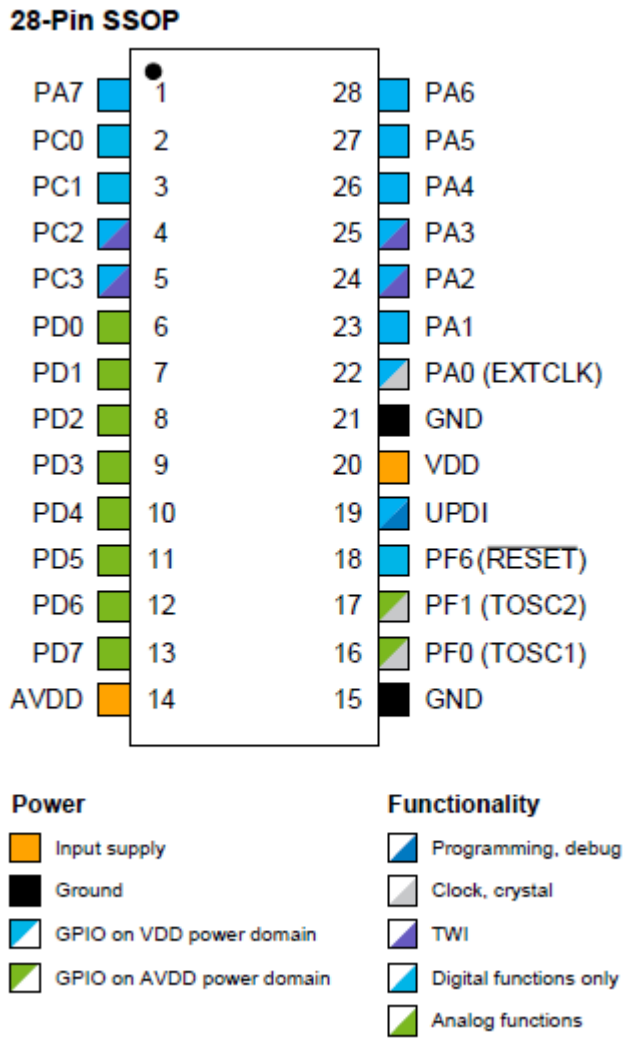


Power

- Input supply
- Ground
- GPIO on VDD power domain
- GPIO on AVDD power domain

Functionality

- Programming, debug
- Clock, crystal
- TWI
- Digital functions only
- Analog functions



5.9 AVRX

5.9.1 AVRX

This chapter describes the AVRDBxx processors. To make a distinction between all the different AVR cores , MCS names the DB series the AVRX. This will include all the AVRxxDByy processors and AVRxxDAyy processors.
The difference between the DB and DA series seems to be that DA is missing the MVIO option.
The AVRX processors can be seen as new Xmega processors. They are the big brother of the ATMEGAX (MEGA4809).
The advantage is that they are cheaper and also run on a voltage between 1v8 and 5V.

The AVRX also has the UPDI interface. It is 24 bit wide since the flash code exceeds the 64KB.

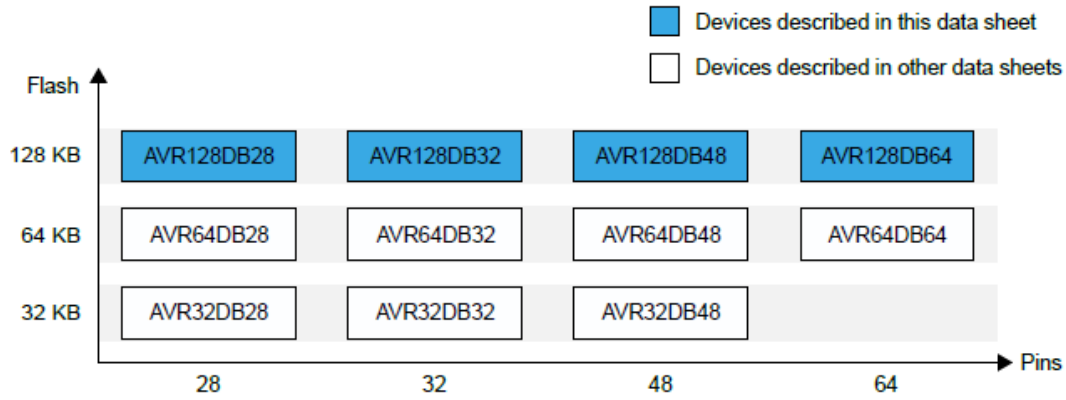
One change compared with other processors is that each pin version has its own ID. This means that the AVR128DB28 (28 pins) and the AVR128DB64 (64 pins) have different ID's.
Since there is one data sheet you need to take good care that the hardware exist in the chip you select. For example, the AVR128DB64 has 6 UARTS. But the

AVR128DB28 has 3 UARTS.

From the PDF

The AVR128DB28/32/48/64 microcontrollers of the AVR® DB family of microcontrollers are using the AVR® CPU with
hardware multiplier running at clock speeds up to 24 MHz. They come with 128 KB of Flash, 16 KB of SRAM, and
512 bytes of EEPROM. The microcontrollers are available in 28-, 32-, 48- and 64- pin packages. The AVR® DB
family uses the latest technologies from Microchip with a flexible and low-power architecture, including Event System,
accurate analog subsystems, and advanced digital peripherals.

Figure 1. AVR® DB Family Overview



XTINY/MEGAX

Please read the topics about XTINY 460 and MEGAX 490. Unless noted the information applies to AVRX as well.

MVIO

We at MCS think that the DB series is great. It is cheap, available in PDIP and has many features.
The DB series has MVIO (Multi Voltage I/O). For the DB series this means that pins or PORTC have a different voltage domain.
You can power the processor from 5V and the MVIO you can power at 3V3. This is great for mixed voltage designs.
Of course you can also connect both domains to the same voltage source.

DAC

There is one DAC with 10 bit resolution.
It can be enabled with the CONFIG statement : config DAC0=ENABLED|DISABLED, OUT_ENABLE=ENABLED|DISABLED, RUNMODE=ENABLED|DISABLED
The DAC has an output pin which can be enabled.

ADC

There is a 12 bit A/D converter. The input pins depend on the processor model.
The A/D converter has differential channels. Notice that not all input pins can be used for differential input.

ZCD

Also new is the Zero Cross Detector. Up to 3 detectors are available depending on the model.
Config Zcd0 1167 = Enabled|DISABLED , Runmode = Disabled|ENABLED, INVERT=ENABLED|DISABLED, OUT_ENABLE=ENABLED|DISABLED
We recommend to use an opto coupler with the detector when you interface this with other electronics/voltages.

~~The interrupt for the ZCD is special. In normal AVR there is a register to enable an interrupt, and some other register might be available to handle specific properties. In the AVRX this is all done in the interrupt register. There is only one vector.~~
~~The ENABLE/DISABLE statements have been extended with the options :~~
~~ENABLE ZCD0_BOTH~~
~~ENABLE ZCD0_FALLING~~
~~ENABLE ZCD0_RISING~~
~~And the same for DISABLE.~~

~~When you enable the ZCD the proper setting will be written to the interrupt register. When you disable the ZCD, no matter which setting you use, it will disable the interrupt.~~
~~So DISABLE ZCD0_RISING and DISABLE ZCD0_BOTH will have the same effect.~~

2087
In version 2087 we redesigned the interrupt handling. Interrupts that have multiple settings but only one address get a general name. For ZCD0 this becomes ZCD0_INT
The ENABLE/DISABLE is extended with a section in the DAT file named [ENADISABLE]
When you press ENABLE and CTRL+SPACE you get a list of interrupts you can enable/disable. For ZCD this will be ZDC0_NONE, ZCD0_RISING, ZDC0_FALLING and ZDC0_BOTH.
Only one of the modes can be active at the same time.
Multiple modes also apply to PIN interrupts. Each port can have 1 interrupt for example PORTD_INT. But each port pin can be enabled/disabled individually to generate an interrupt.

OPAMP
Also new is the OPAMP. There are up to 3 OPAMP's which can also be connected to each other. The CONFIG OPAMP 1005 has many options.

TIMERS
There are many timers. All with a different purpose. There are timers of type A(16 bit), B(16 bit) and D(12 bit).

VREGPWR
The Sleep controller also has a voltage regulator control register. CONFIG VREGPWR 1148 configures the power options.

All the remaining options are similar to the XTINY and MEGAX series.

The supported AVRX series are : DA, DB, DD and EA.

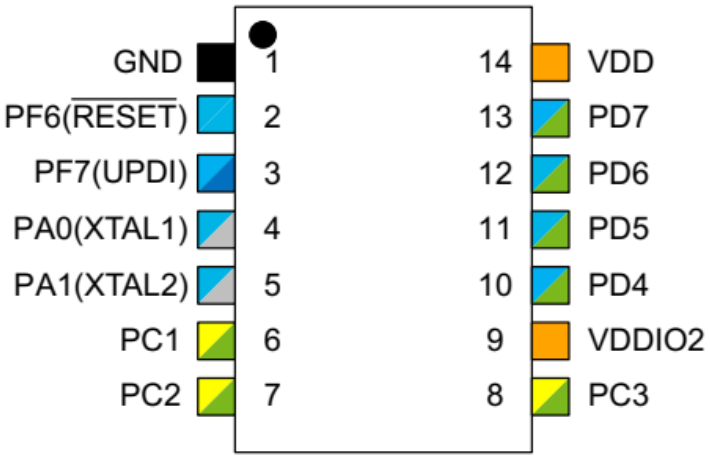
See also
MEGAX 490, XTINY 460

5.9.2 AVR16DD14

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR16DD14 comes in SOIC-14. It has 16KB of flash and 14 pins. It has 2KB SRAM and 256 bytes EEPROM.

14-Pin SOIC



Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

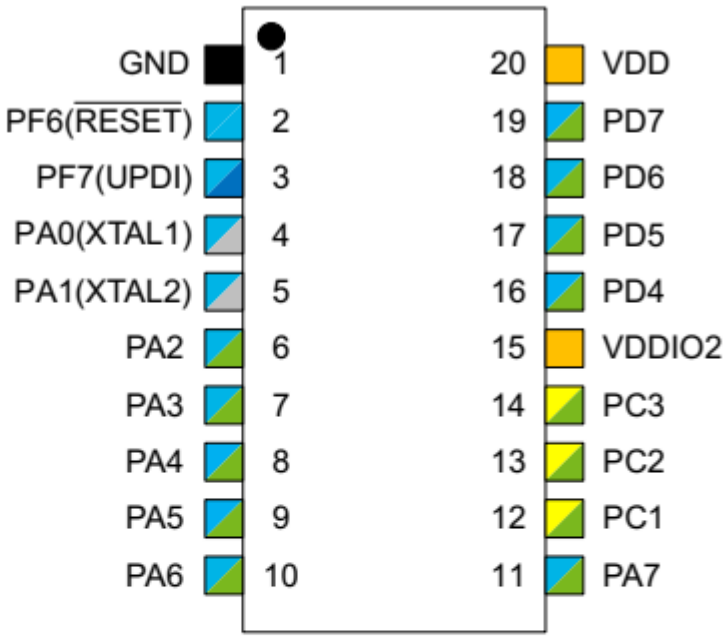
- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

5.9.3 AVR16DD20

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR16DD20 comes in SOIC-20 and VQFN20. It has 16KB of flash and 20 pins. It has 2KB SRAM and 256 bytes EEPROM.

20-Pin SOIC



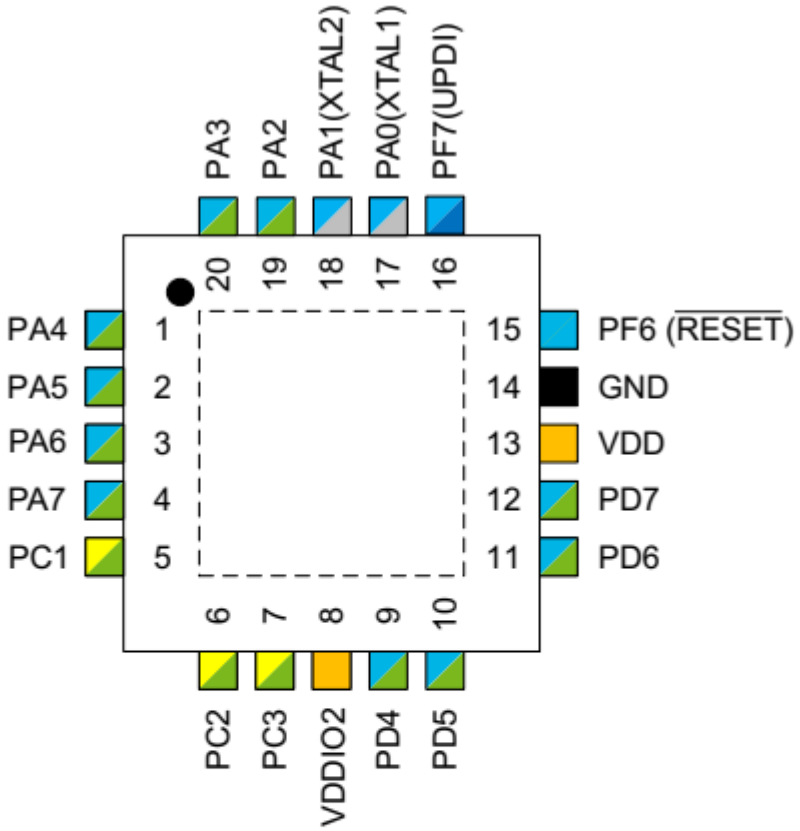
Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

20-Pin VQFN



Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

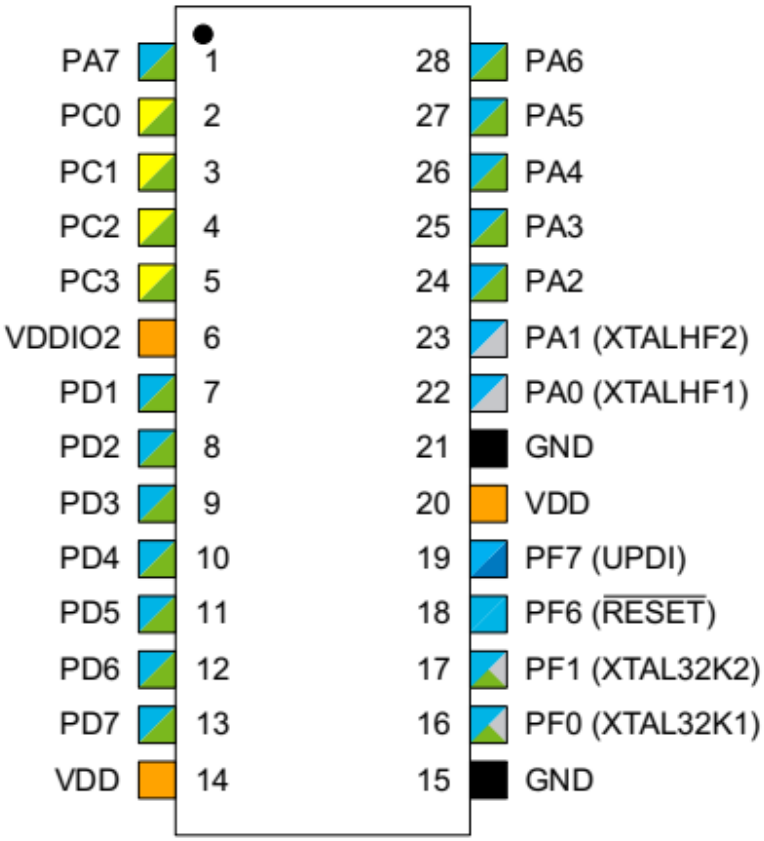
- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

5.9.4 AVR16DD28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR16DD28 comes as SPDIP, SSDOP, SOIC and VQFN. It has 16KB of flash, 2KB SRAM and 256 bytes EEPROM.

28-Pin SPDIP, SSOP and SOIC



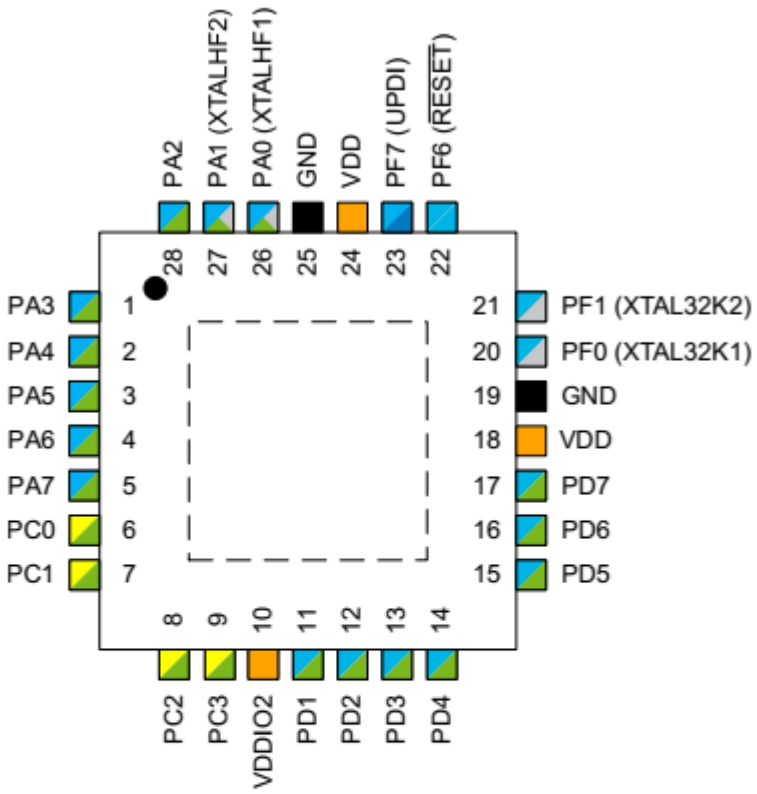
Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

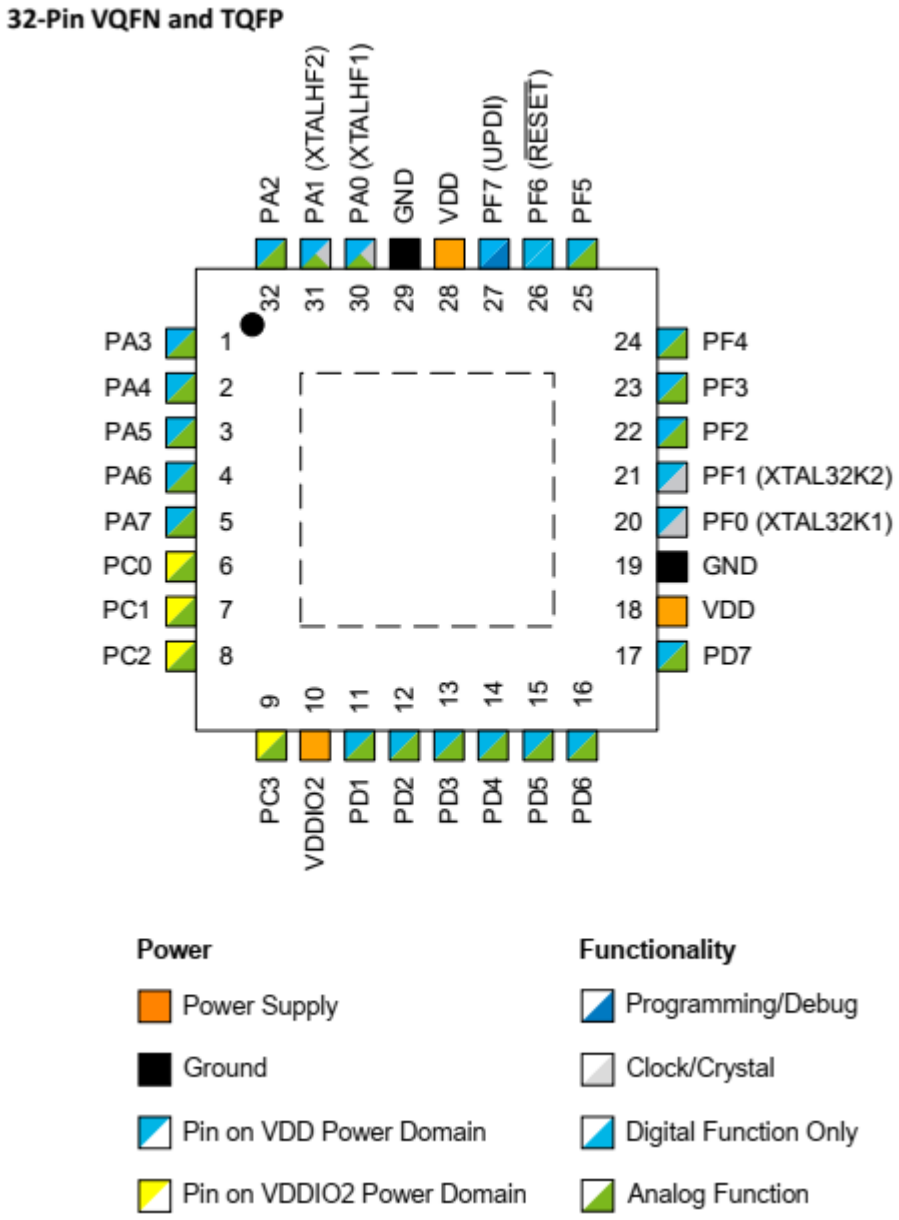
28-Pin VQFN



5.9.5 AVR16DD32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR16DD32 comes as VQFN and TQFP. It has 16KB of flash, 2KB SRAM and 256 bytes EEPROM.



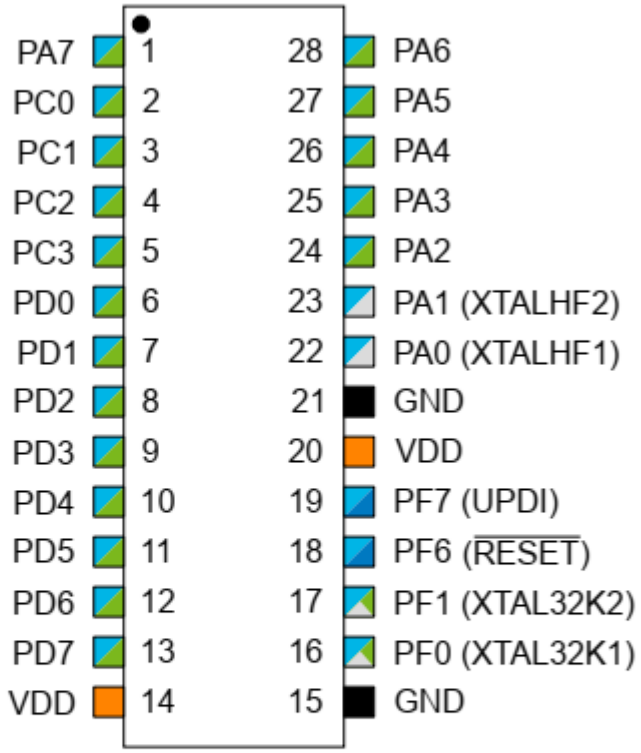
5.9.6 AVR16EA28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR16EA28 comes as SPDIP, SSOP and VQFN. It has 16KB of flash, 2KB SRAM and 512 bytes EEPROM.

Pinout

28-pin SPDIP and SSOP

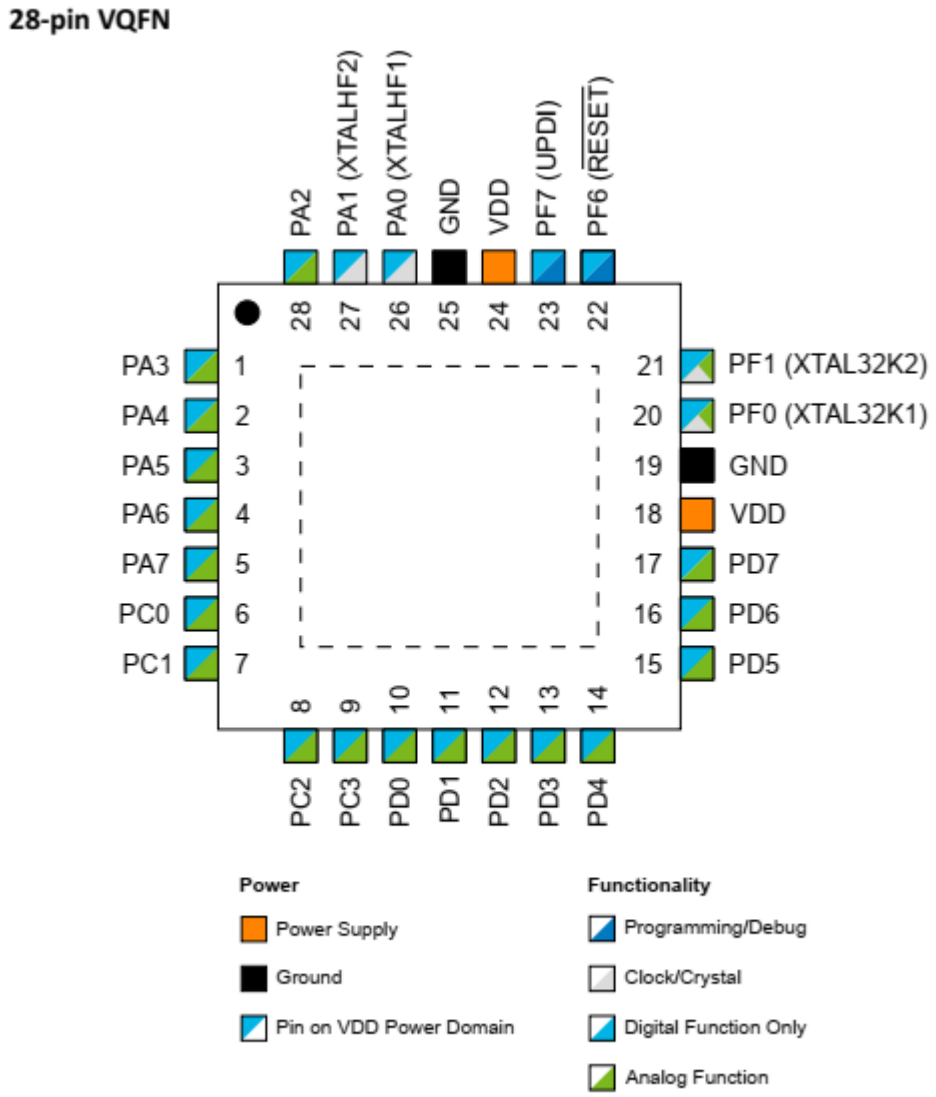


Power

- Power Supply
- Ground
- Pin on VDD Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

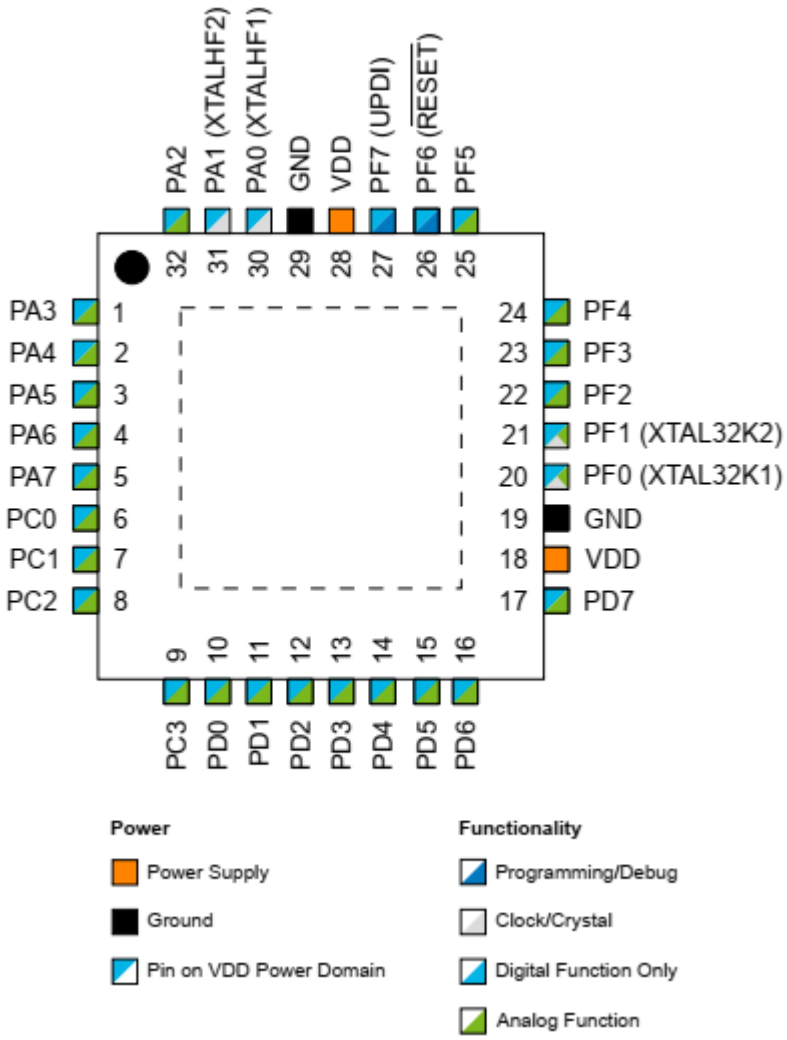


5.9.7 AVR16EA32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR16EA32 comes as VQFN and TQFP. It has 16KB of flash, 2KB SRAM and 512 bytes EEPROM.

32-pin VQFN and TQFP

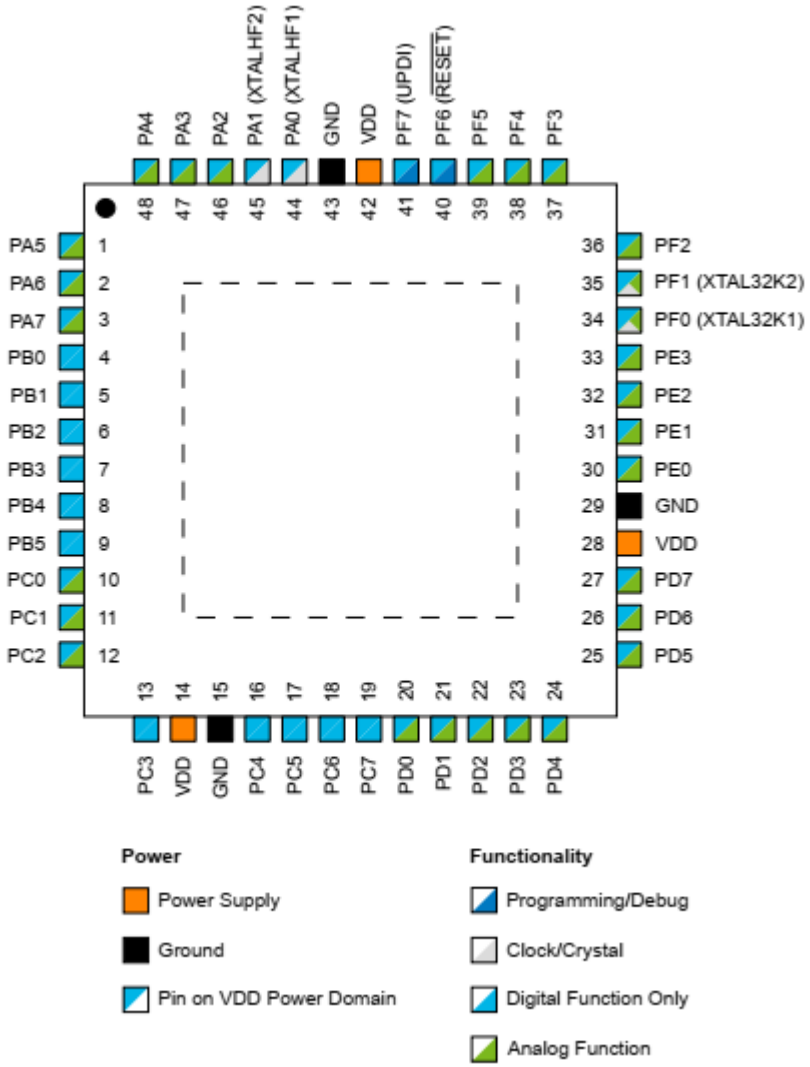


5.9.8 AVR16EA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR16EA48 comes as VQFN and TQFP. It has 16KB of flash, 2KB SRAM and 512 bytes EEPROM.

48-pin VQFN and TQFP



5.9.9 AVR32DA28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DA28 comes in SPDIP, SSOP and SOIC. It has 32KB of flash and a maximum of 48 pins. Internal SRAM size is 4KB with 512 bytes EEPROM

2.1 28-Pin SPDIP, SSOP and SOIC

Note: For the AVR® DA Family, the VDD and AVDD are internally connected (no separate power domains).

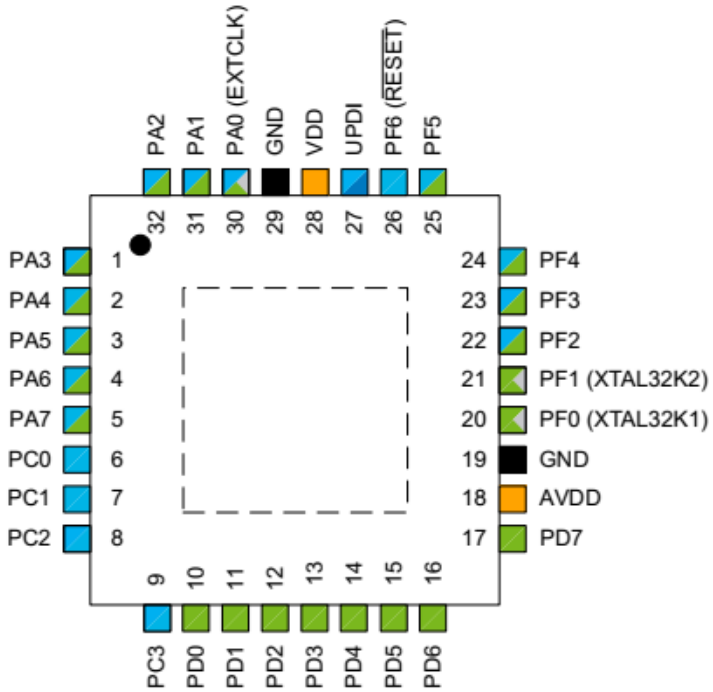
5.9.10 AVR32DA32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DA32 comes in VQFN and TQFP. It has 32KB of flash and a maximum of 48 pins. Internal SRAM size is 4KB with 512 bytes EEPROM

.

2.2 32-Pin VQFN and TQFP



Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on AVDD Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

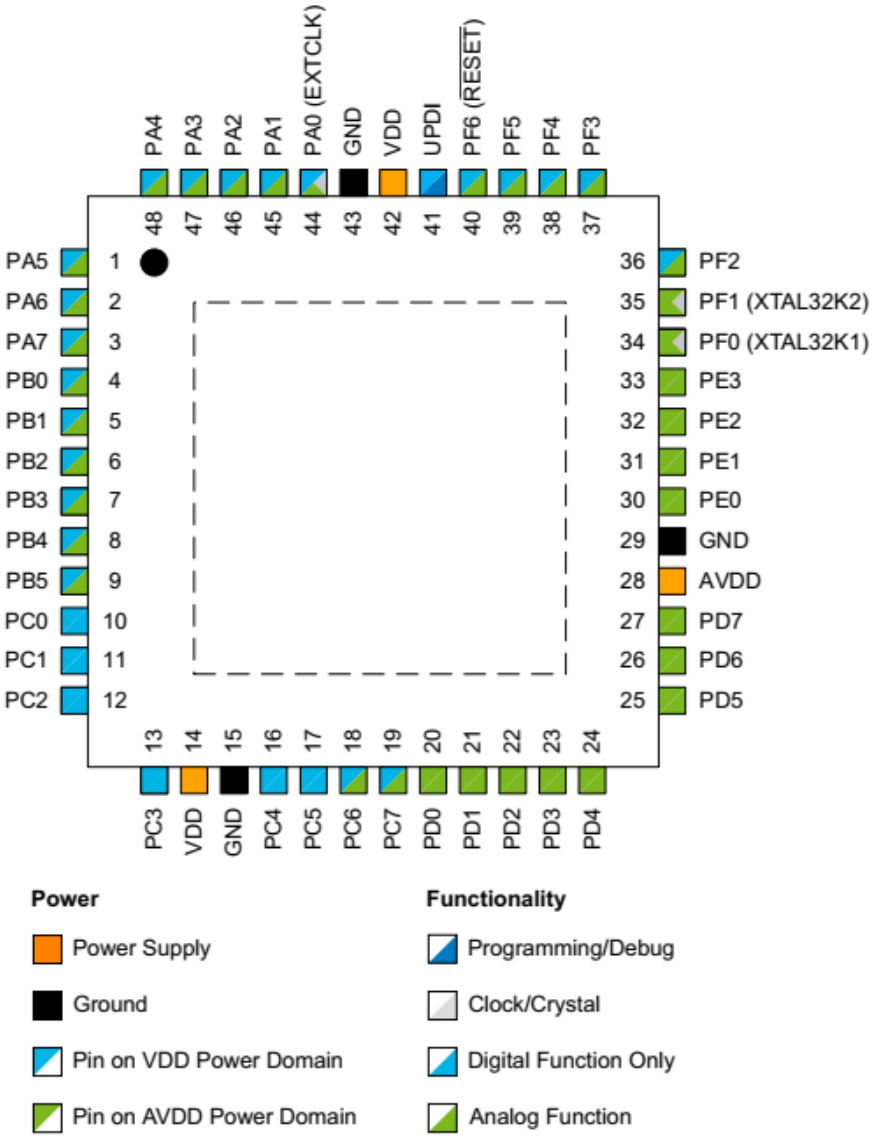
Note: For the AVR® DA Family, the VDD and AVDD are internally connected (no separate power domains).

5.9.11 AVR32DA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DA48 comes in TQFP and VQFN. It has 32KB of flash and a maximum of 48 pins. Internal SRAM size is 4KB with 512 bytes EEPROM

2.3 48-Pin VQFN and TQFP



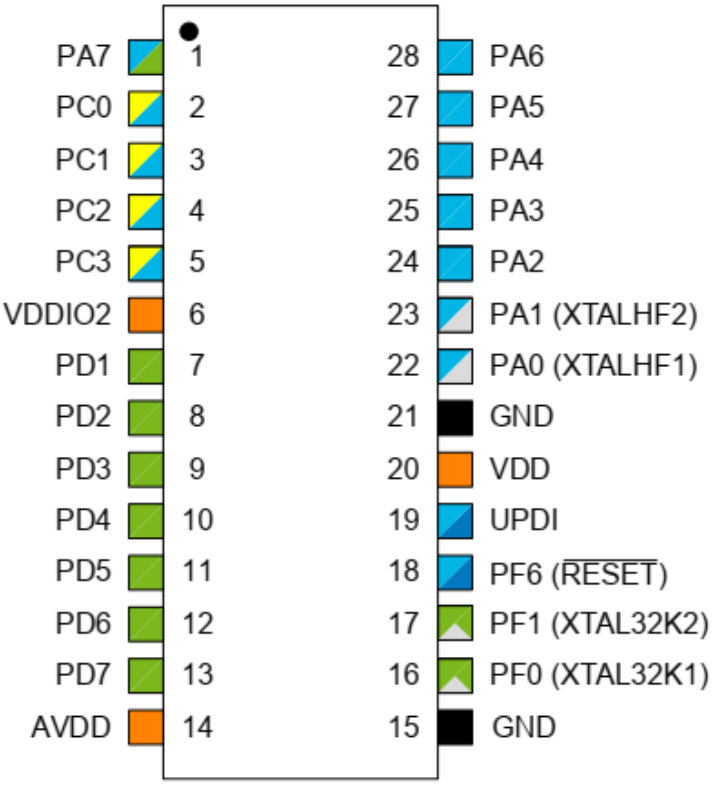
Note: For the AVR® DA Family, the VDD and AVDD are internally connected (no separate power domains).

5.9.12 AVR32DB28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DB28 comes in SSOP, SOIC and SPDIP. It has 32KB of flash and 28 pins. Internal SRAM size is 4KB with 512 bytes EEPROM

28-pin SSOP, SOIC and SPDIP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

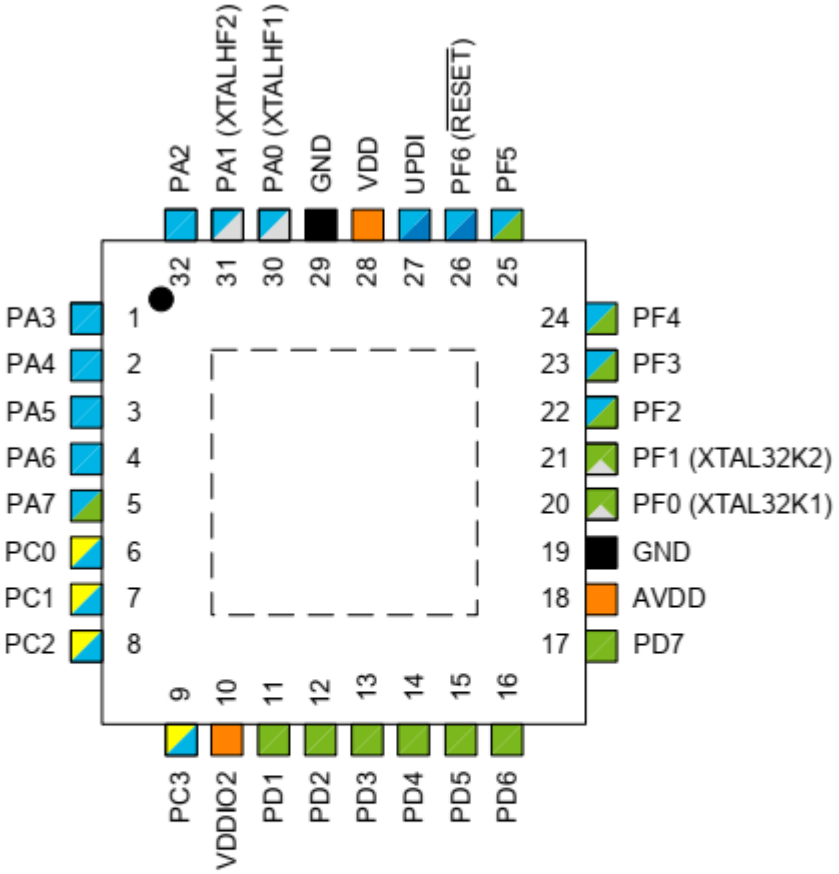
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

5.9.13 AVR32DB32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DB32 comes in VQFN and TQFP. It has 32KB of flash and 32 pins. Internal SRAM size is 4KB with 512 bytes EEPROM

32-pin VQFN and TQFP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

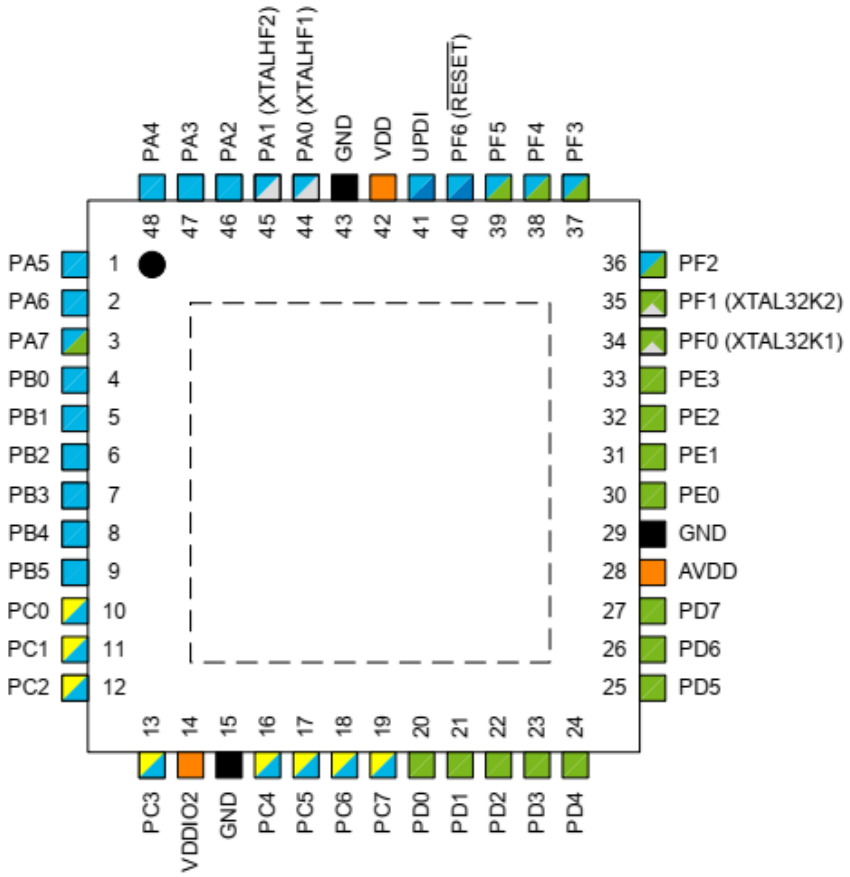
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

5.9.14 AVR32DB48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DB48 comes in TQFP and VQFN. It has 32KB of flash and 32 pins. Internal SRAM size is 4KB with 512 bytes EEPROM

48-pin VQFN and TQFP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

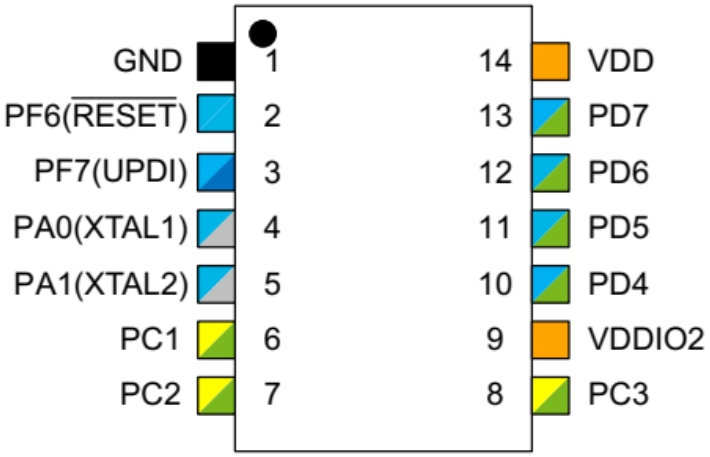
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

5.9.15 AVR32DD14

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DD14 comes in SOIC-14. It has 32KB of flash and 14 pins. It has 4KB SRAM and 256 bytes EEPROM.

14-Pin SOIC



Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

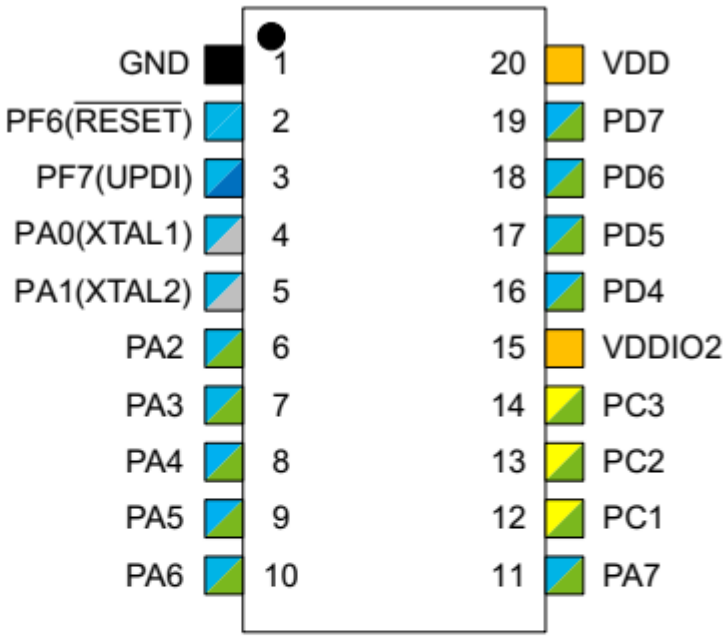
- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

5.9.16 AVR32DD20

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DD20 comes in SOIC and VQFN. It has 32KB of flash and 20 pins. It has 4KB SRAM and 256 bytes EEPROM.

20-Pin SOIC



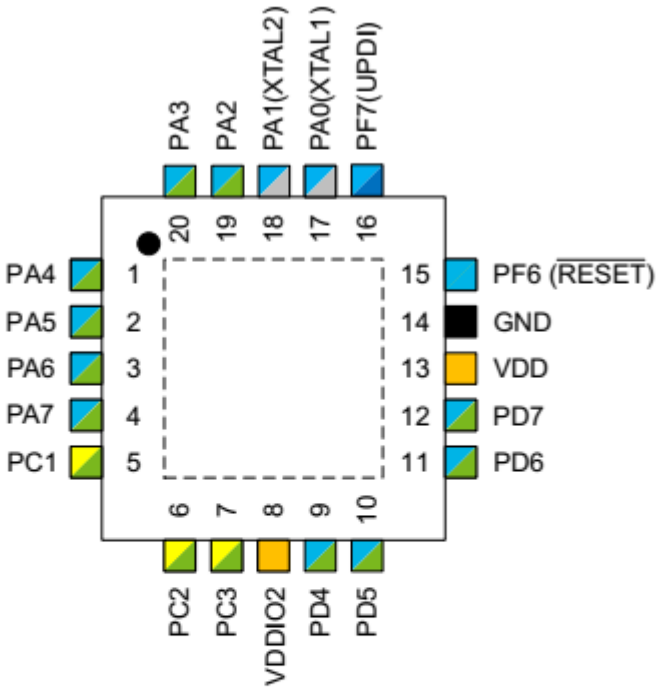
Power

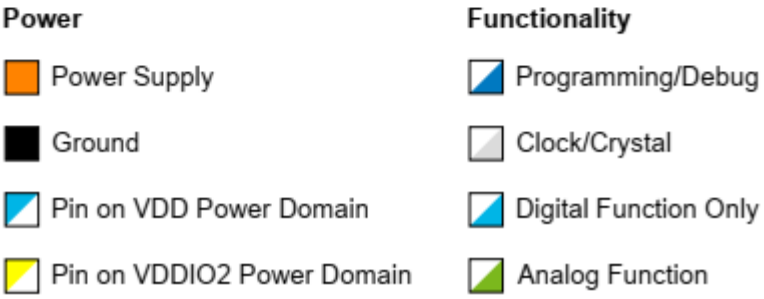
- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

20-Pin VQFN



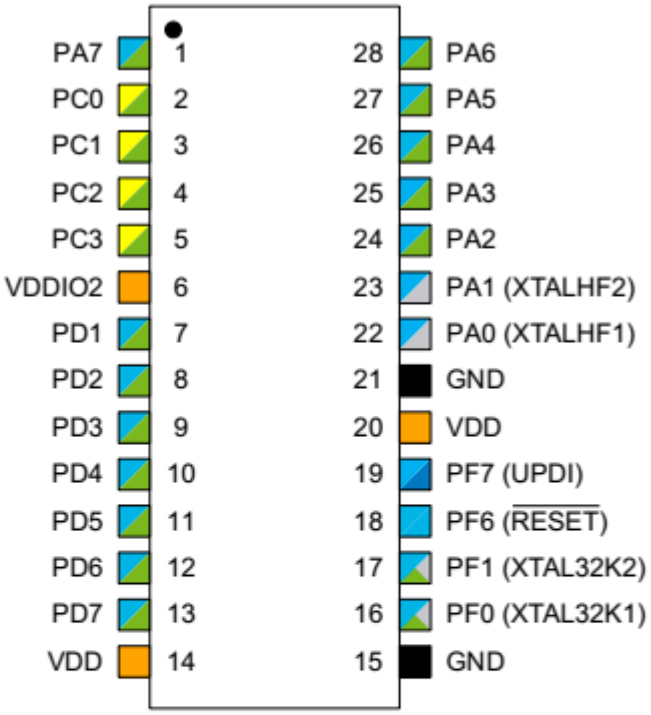


5.9.17 AVR32DD28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DD28 comes in SPDIP, SSOP, SOIC and VQFN. It has 32KB of flash and 28 pins. It has 4KB SRAM and 256 bytes EEPROM.

28-Pin SPDIP, SSOP and SOIC



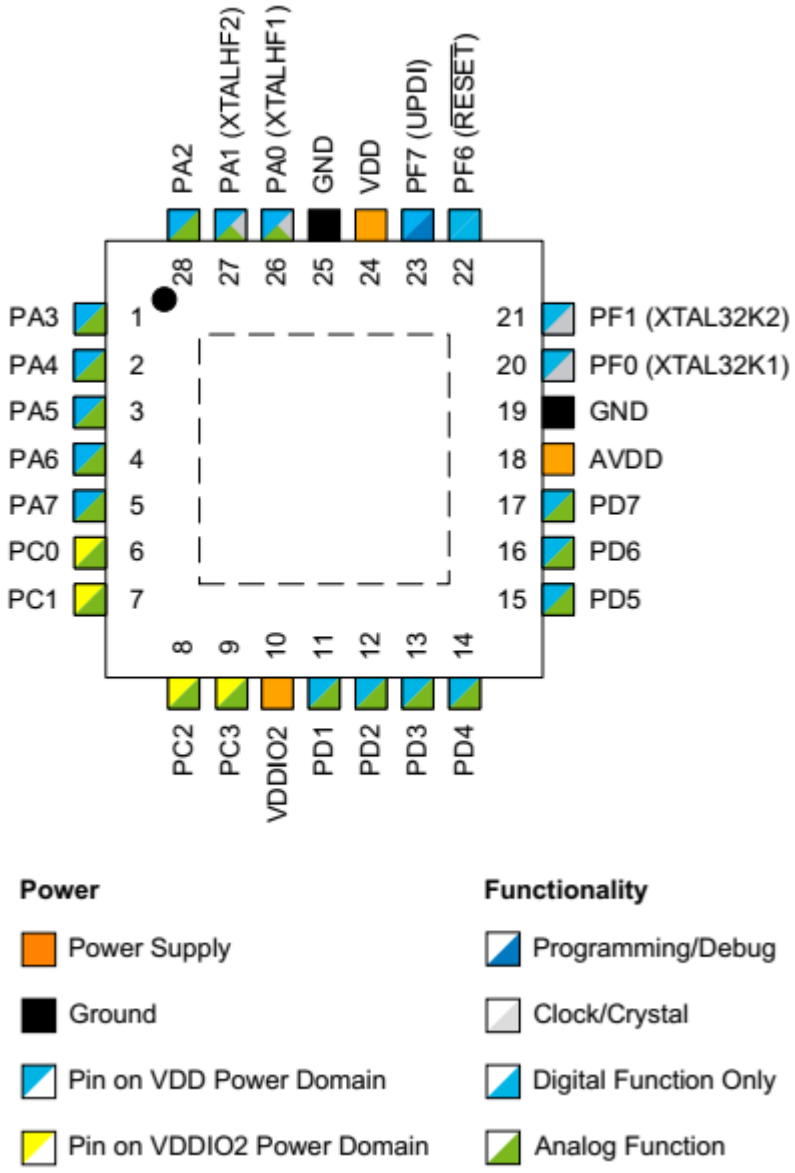
Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

28-Pin VQFN

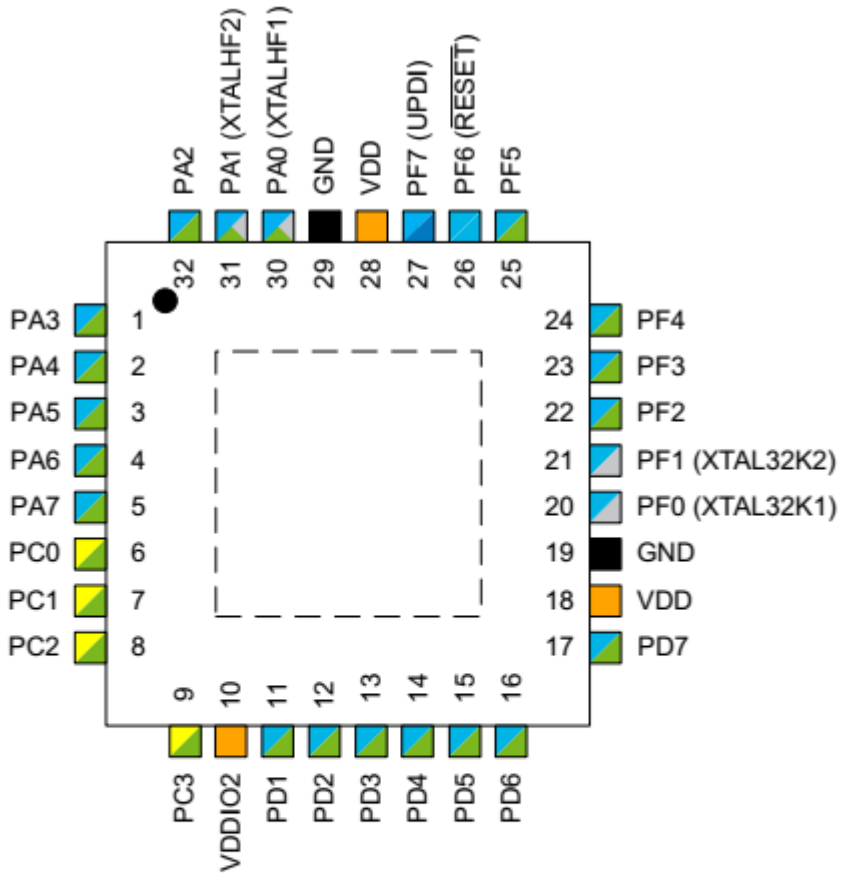


5.9.18 AVR32DD32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32DD32 comes in VQFN and TQFP. It has 32KB of flash and 32 pins. It has 4KB SRAM and 256 bytes EEPROM.

32-Pin VQFN and TQFP



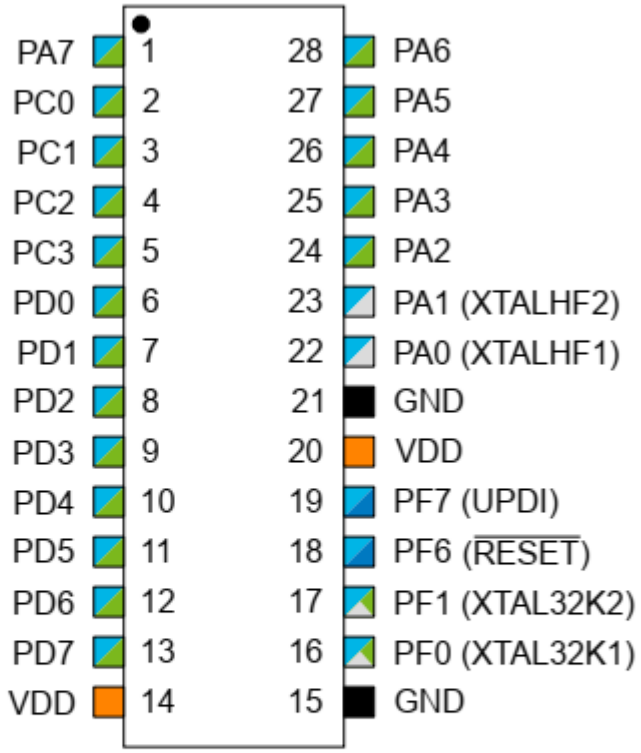
5.9.19 AVR32EA28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

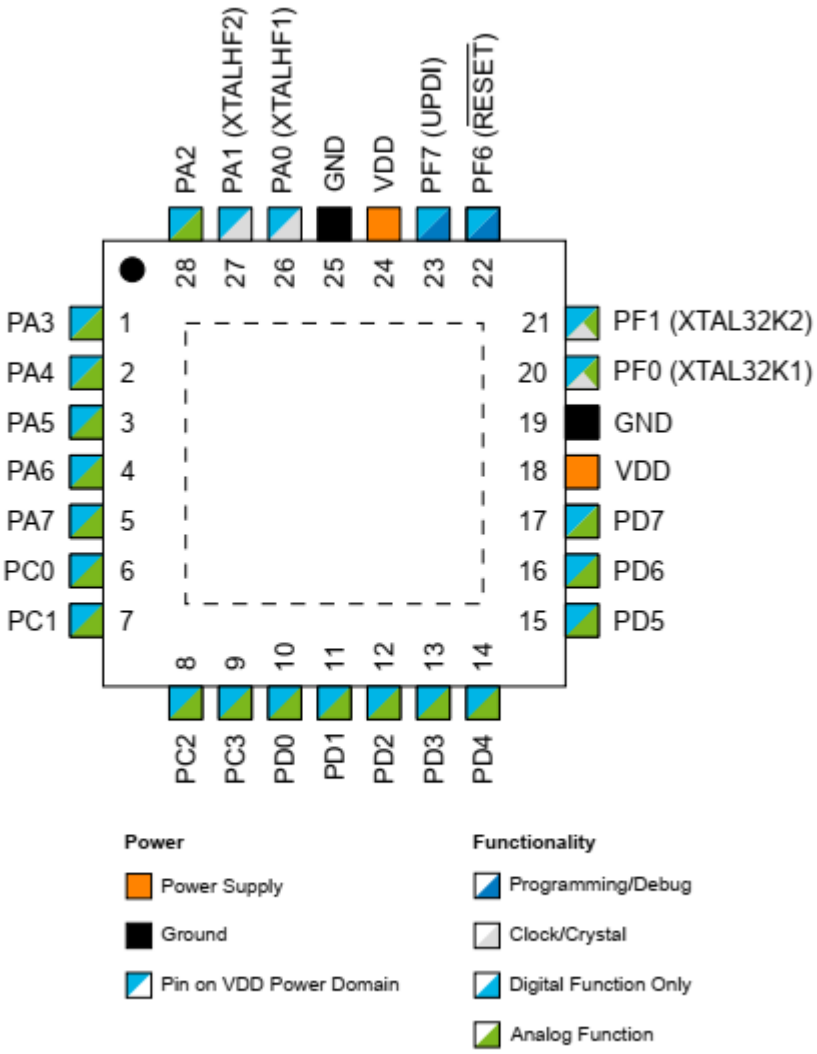
The AVR32EA28 comes in SPDIP, SSOP and VQFN. It has 32KB of flash and 28 pins. It has 4KB SRAM and 512 bytes EEPROM.

Pinout

28-pin SPDIP and SSOP



28-pin VQFN

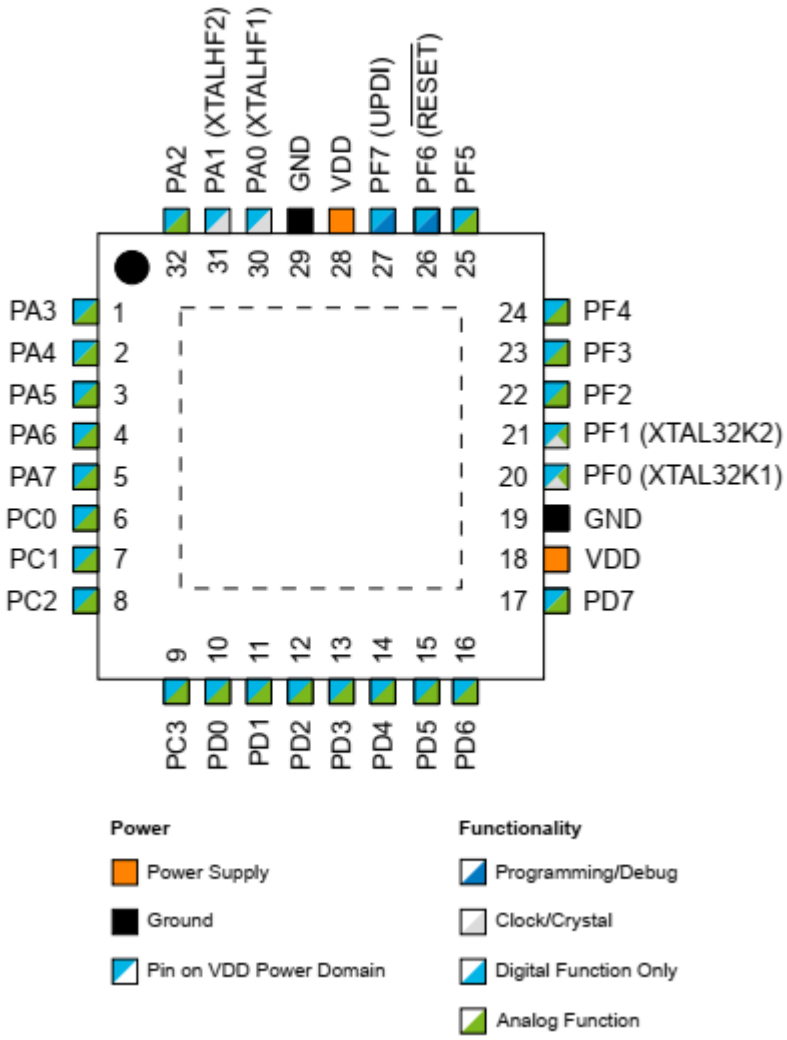


5.9.20 AVR32EA32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32EA32 comes in VQFN and TQFP. It has 32KB of flash and 32 pins. It has 4KB SRAM and 512 bytes EEPROM.

32-pin VQFN and TQFP

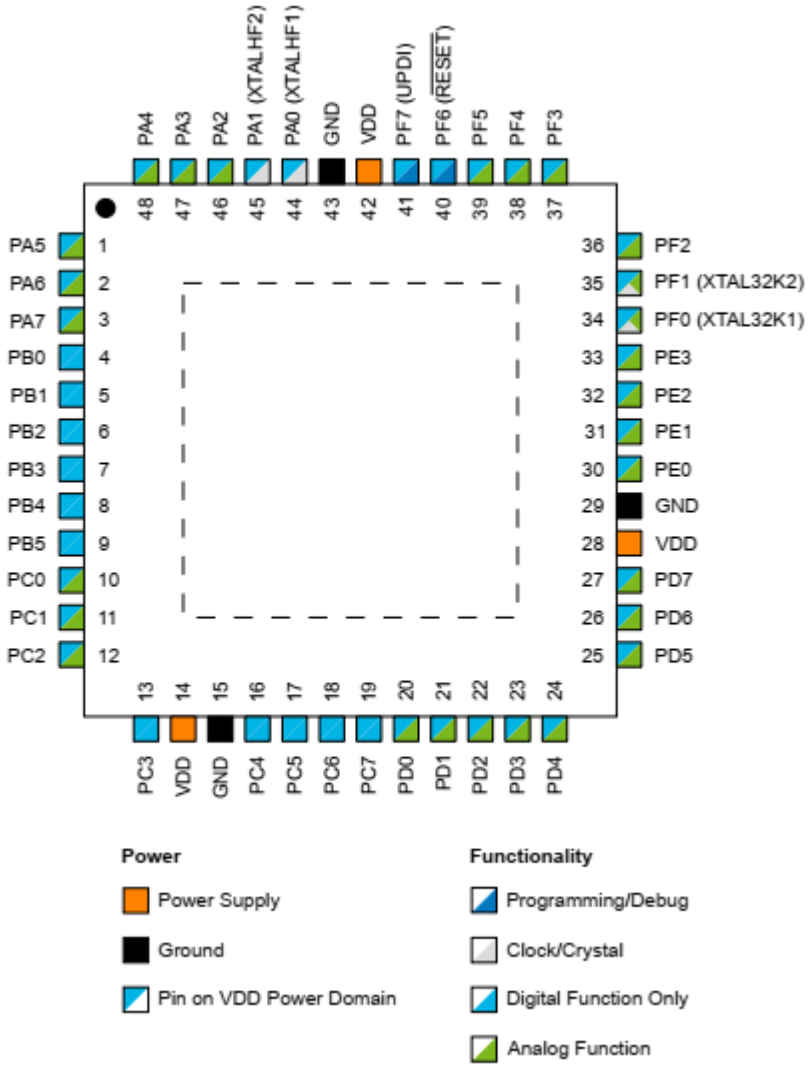


5.9.21 AVR32EA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR32EA48 comes as VQFN and TQFP. It has 32KB of flash, 4KB SRAM and 512 bytes EEPROM.

48-pin VQFN and TQFP

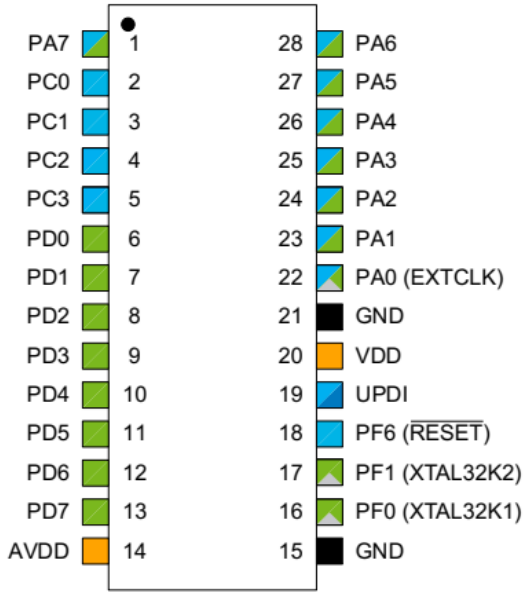


5.9.22 AVR64DA28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DA28 comes in 28 pin SPDIP, SSOP and SOIC. It has 64KB of flash and 28 pins. The SRAM size is 8KB and EEPROM 512 bytes

2.1 28-Pin SPDIP, SSOP and SOIC



Note: For the AVR® DA Family, the VDD and AVDD are internally connected (no separate power domains).

5.9.23 AVR64DA32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DA32 comes in 32 pin VQFN and TQFP. It has 64KB of flash and 32 pins. The SRAM size is 8KB and EEPROM 512 bytes

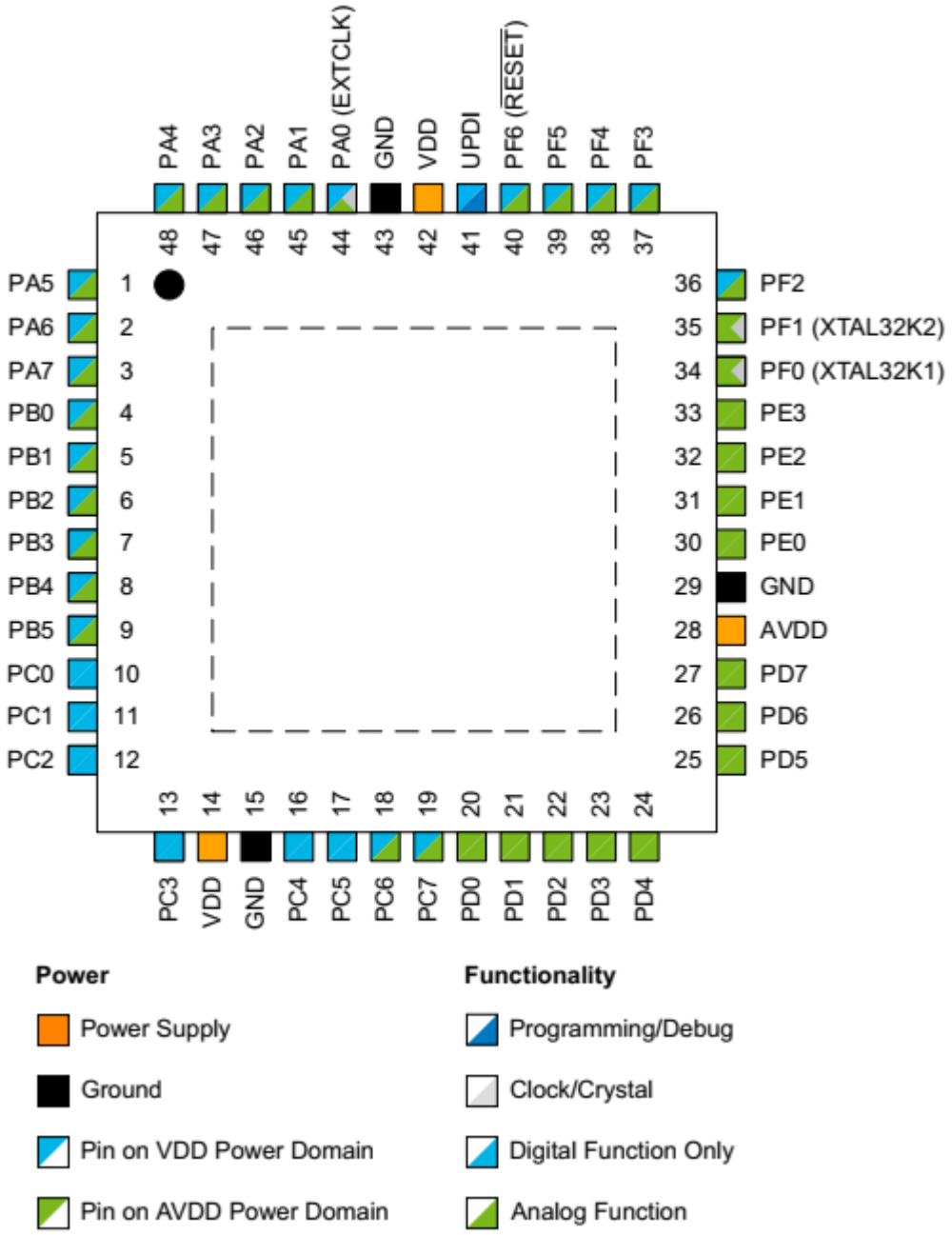
32-Pin VQFN and TQFP

5.9.24 AVR64DA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DA48 comes in 48 pins VQFN and TQFP. It has 64KB of flash and 48 pins. 8KB SRAM and 512 bytes EEPROM.
The difference with the DB series is that there is no MVIO (multi voltage IO).

48-Pin VQFN and TQFP

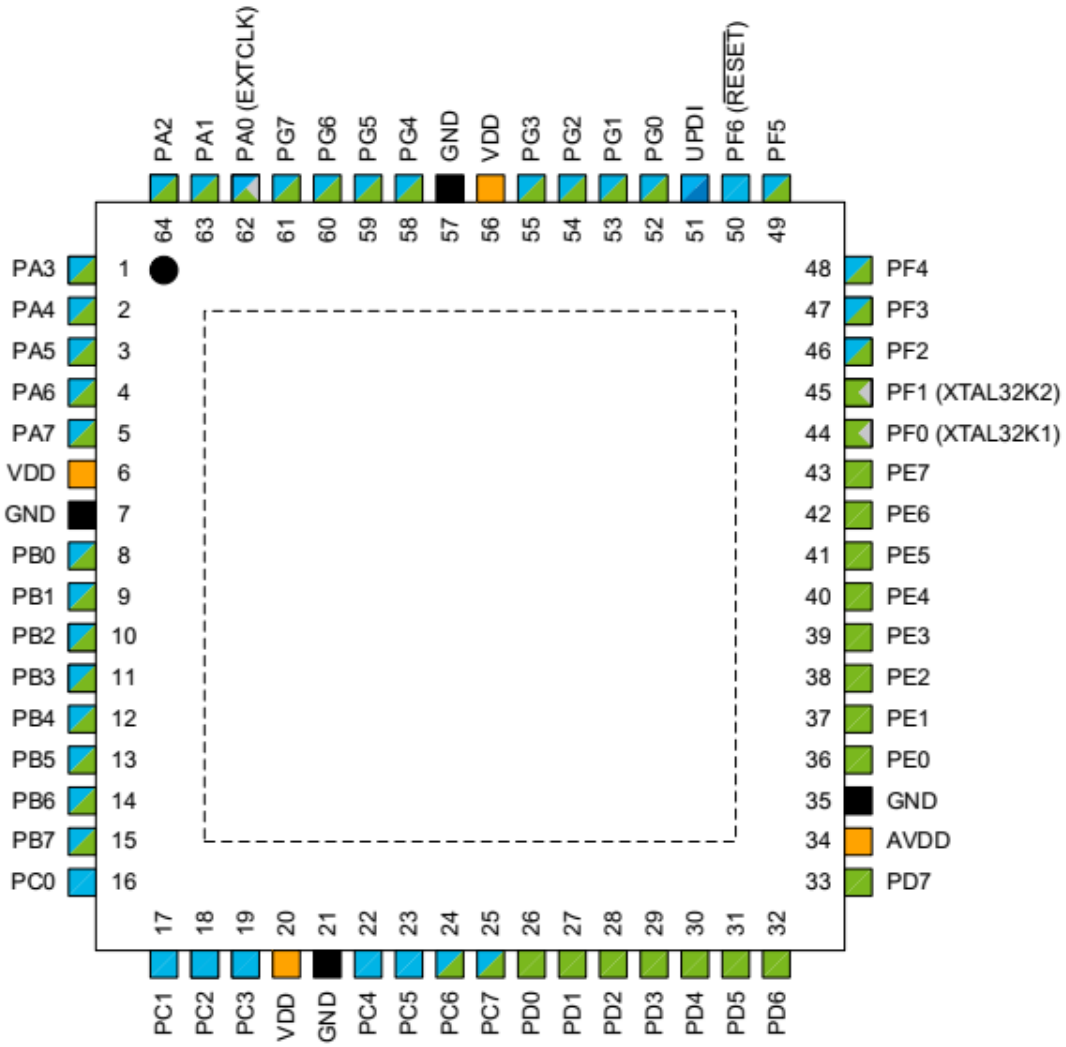


5.9.25 AVR64DA64

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DA64 comes in TQFP and VQFN. It has 64KB of flash and a maximum of 64 pins. It has 8KB SRAM and 512 bytes EEPROM.

64-Pin VQFN and TQFP

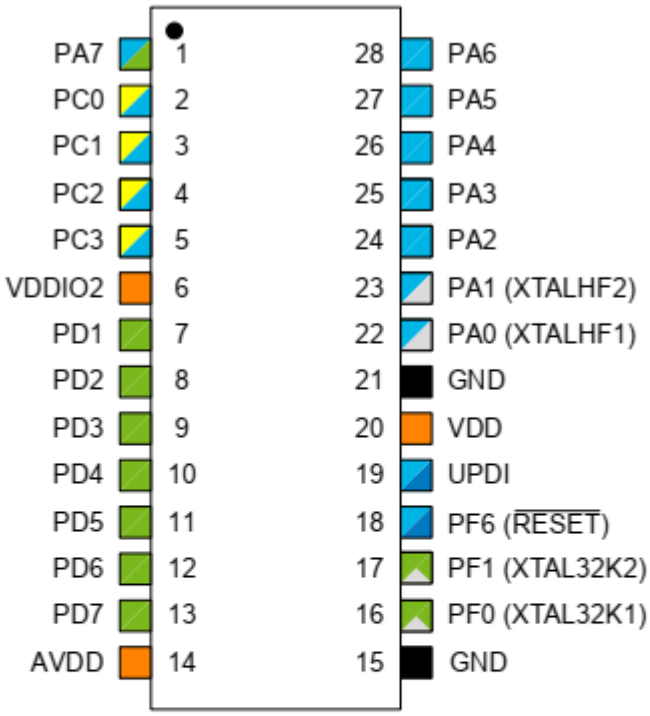


5.9.26 AVR64DB28

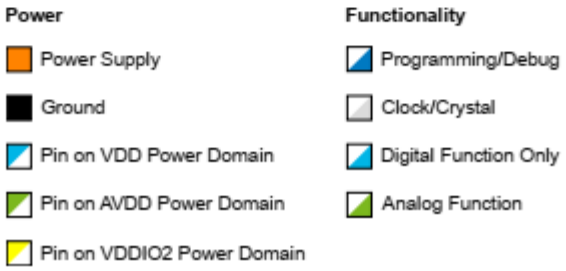
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DB28 comes in SSOP, SOIC and SPDIP. It has 64KB of flash, 8KB SRAM and 512 bytes EEPROM

28-pin SSOP, SOIC and SPDIP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

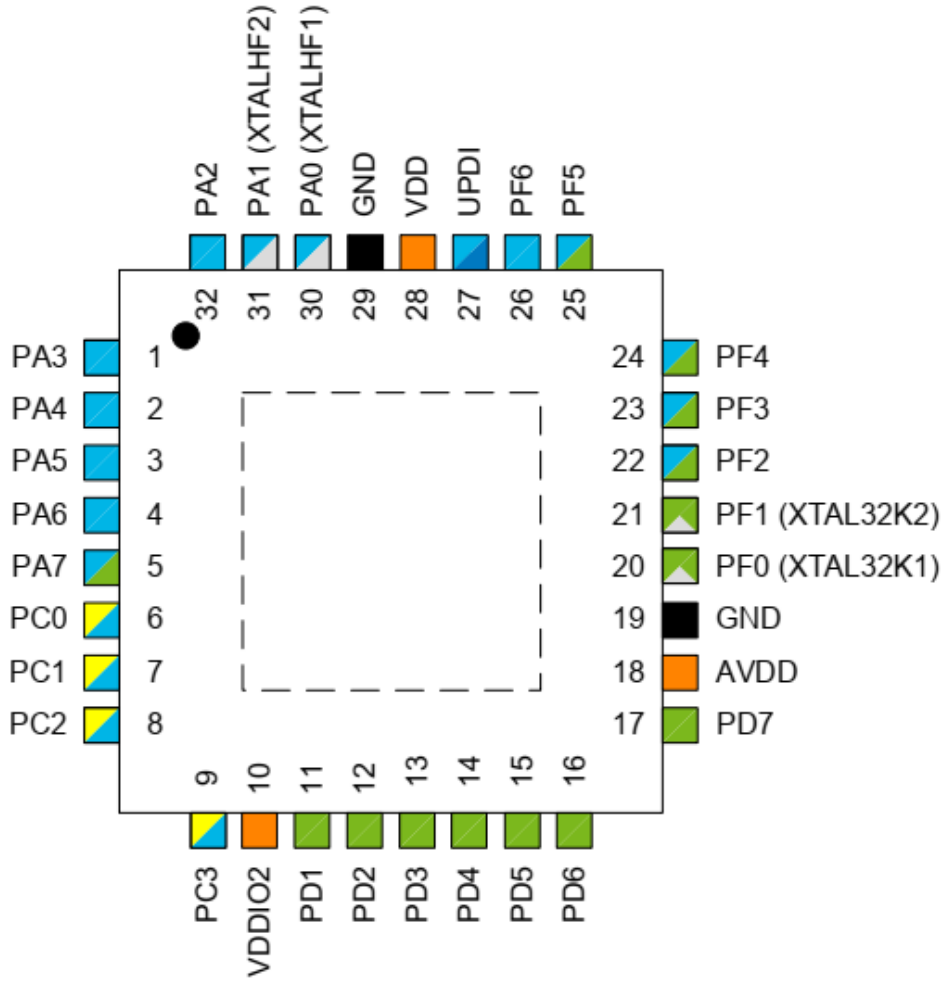


5.9.27 AVR64DB32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DB32 comes in TQFP and VQFN. It has 64KB of flash, 8 KB SRAM and 512 bytes EEPROM

32-pin VQFN and TQFP



Note: For the AVR® DBFamily of devices, AVDD is internally connected to VDD (not separate power domains).

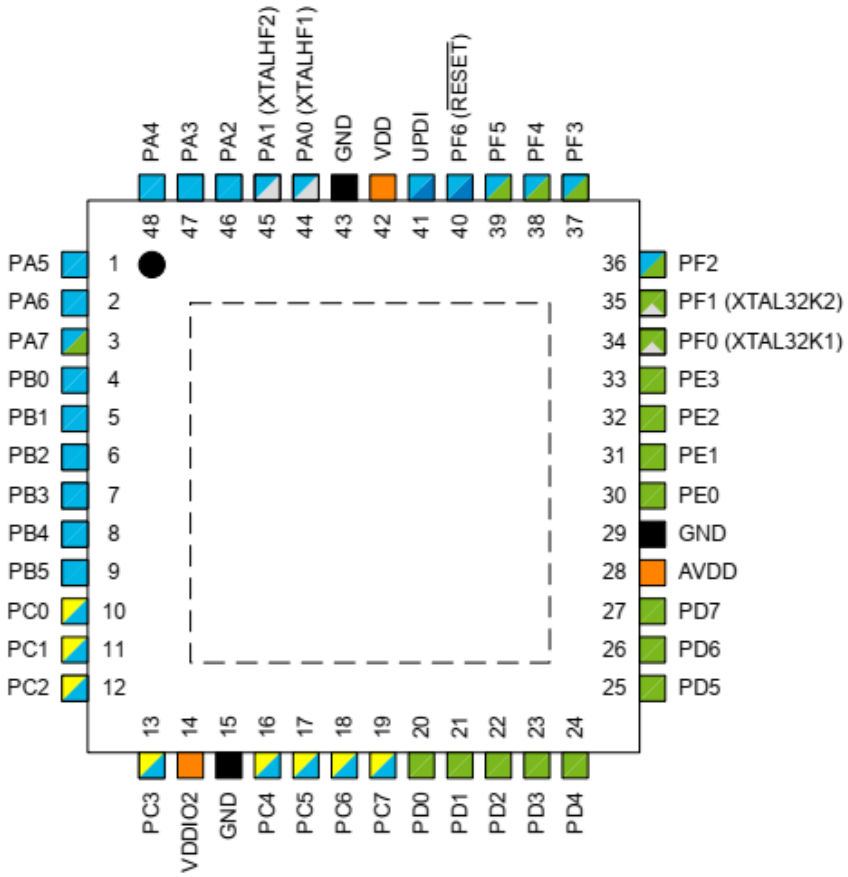
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

5.9.28 AVR64DB48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DB48 comes in TQFP and VQFN. It has 64KB of flash, 8 KB SRAM and 512 bytes EEPROM

48-pin VQFN and TQFP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

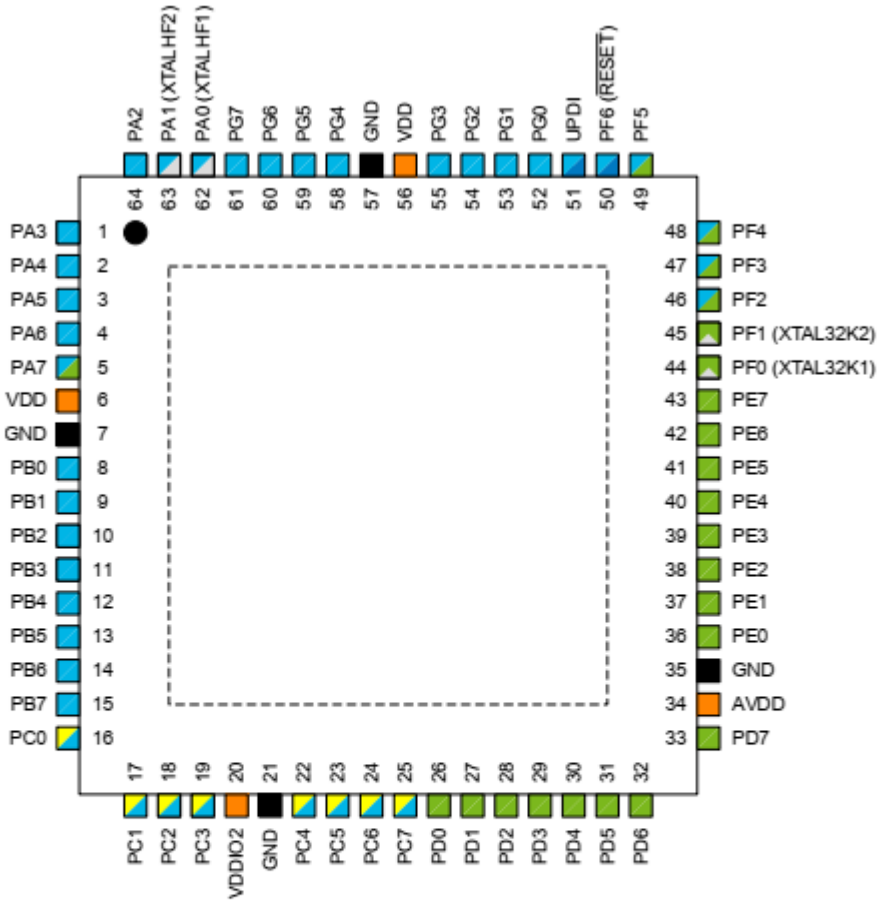
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

5.9.29 AVR64DB64

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DB64 comes in 64 TQFP and VQFN. It has 64KB of flash, 8KB of SRAM and 512 bytes EEPROM.

2.4 64-pin VQFN and TQFP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

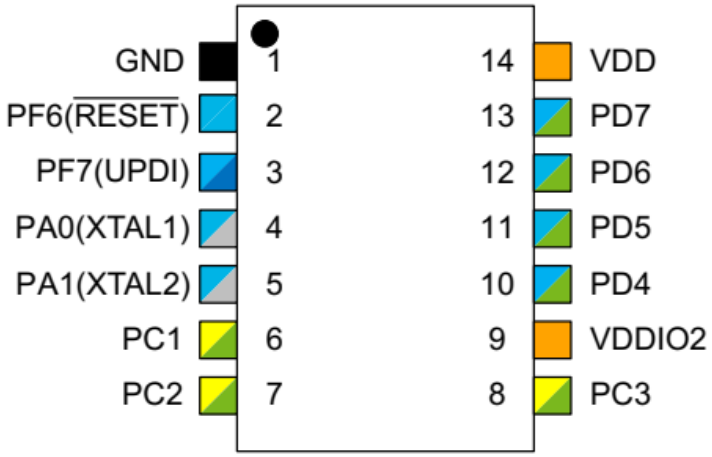
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

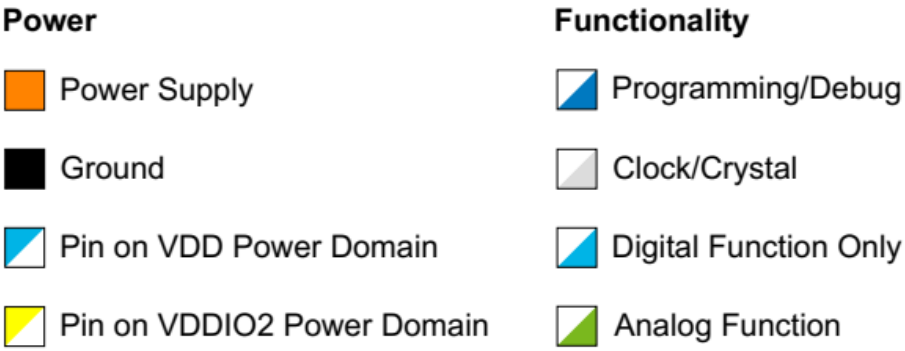
5.9.30 AVR64DD14

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DD14 comes in SOIC. It has 64KB of flash, 8KB SRAM and 256 bytes EEPROM.

14-Pin SOIC



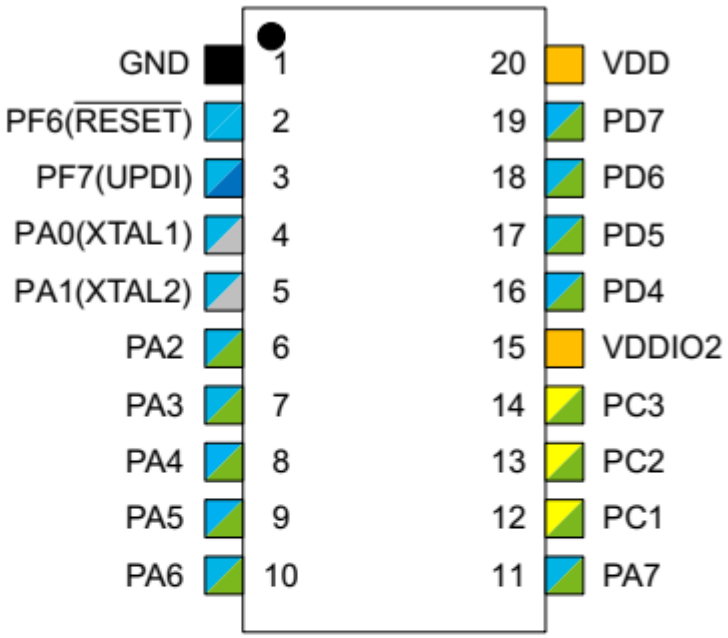


5.9.31 AVR64DD20

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DD20 comes in SOIC and VQFN. It has 64KB of flash and 20 pins. It has 8KB SRAM and 256 bytes EEPROM.

20-Pin SOIC



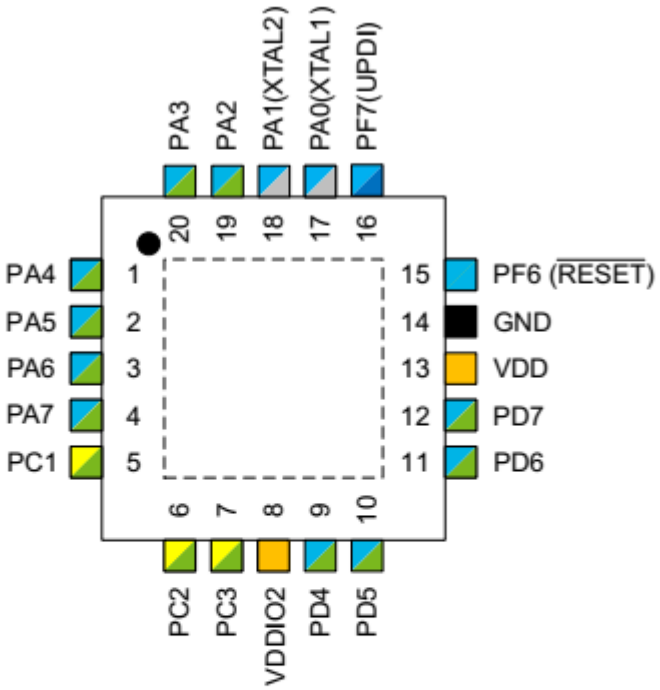
Power

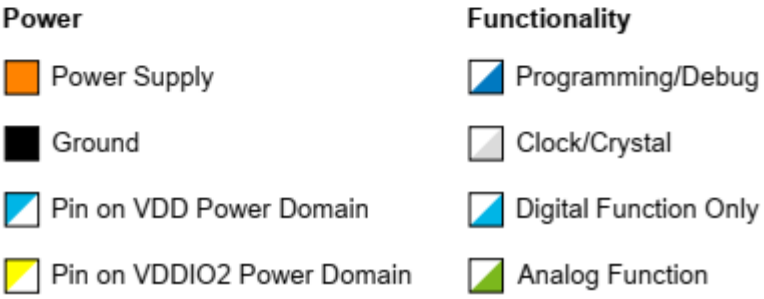
- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

20-Pin VQFN



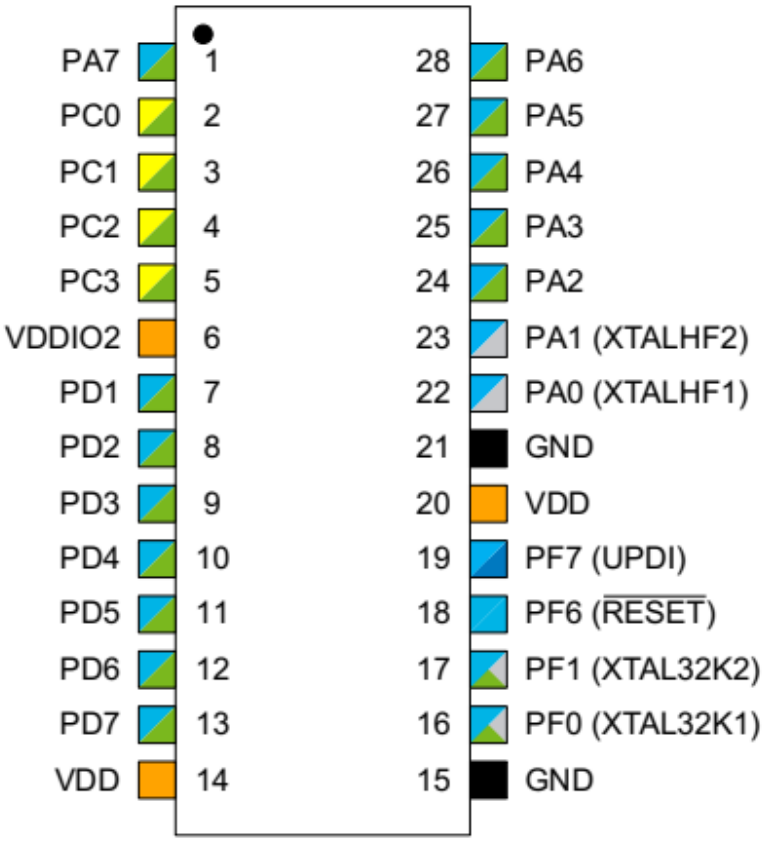


5.9.32 AVR64DD28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DD28 comes SPDIP, SSDOP and SOIC. It has 64KB of flash, 8KB SRAM and 256 bytes EEPROM.

28-Pin SPDIP, SSOP and SOIC



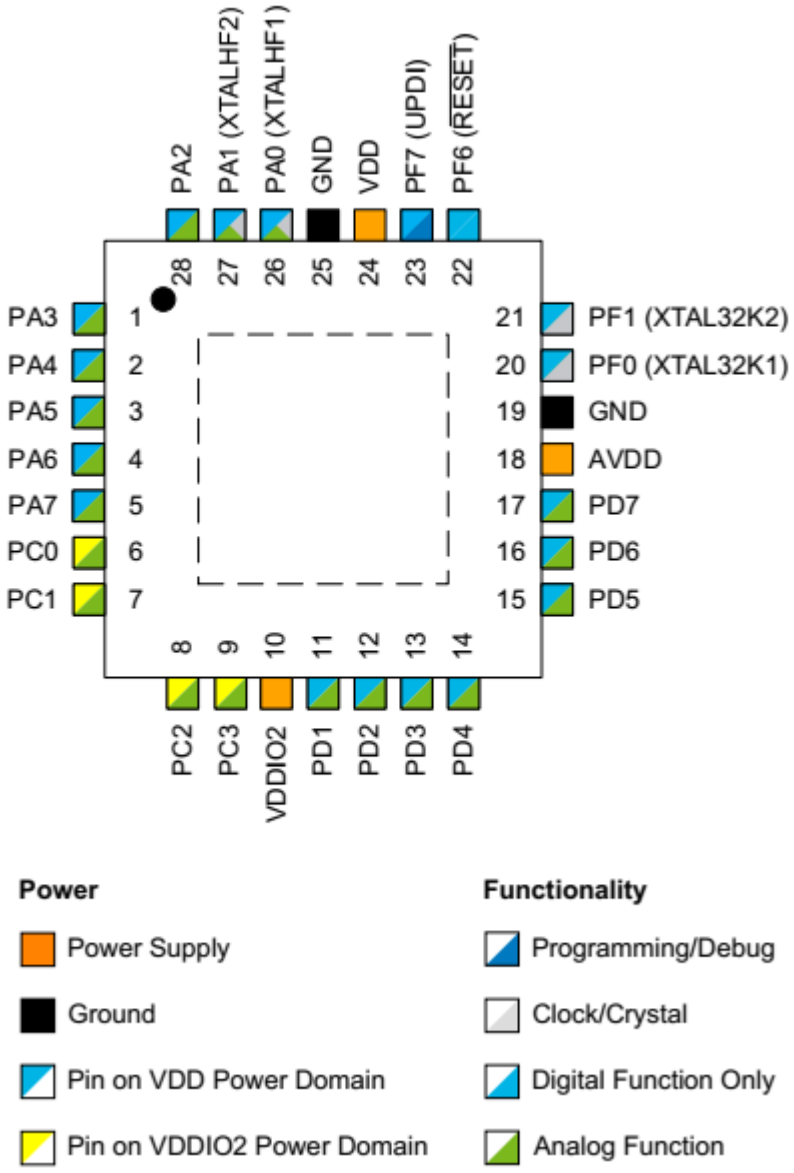
Power

- Power Supply
- Ground
- Pin on VDD Power Domain
- Pin on VDDIO2 Power Domain

Functionality

- Programming/Debug
- Clock/Crystal
- Digital Function Only
- Analog Function

28-Pin VQFN

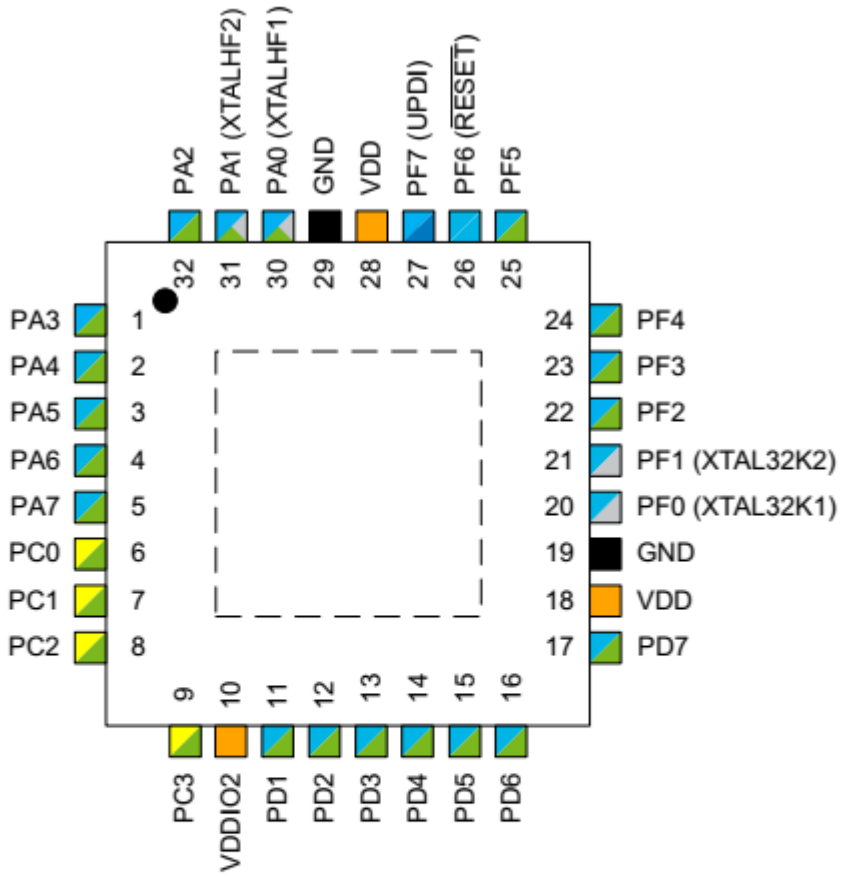


5.9.33 AVR64DD32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64DD32 comes in VQFN and TQFP. It has 64KB of flash and 32 pins. It has 8KB SRAM and 256 bytes EEPROM.

32-Pin VQFN and TQFP



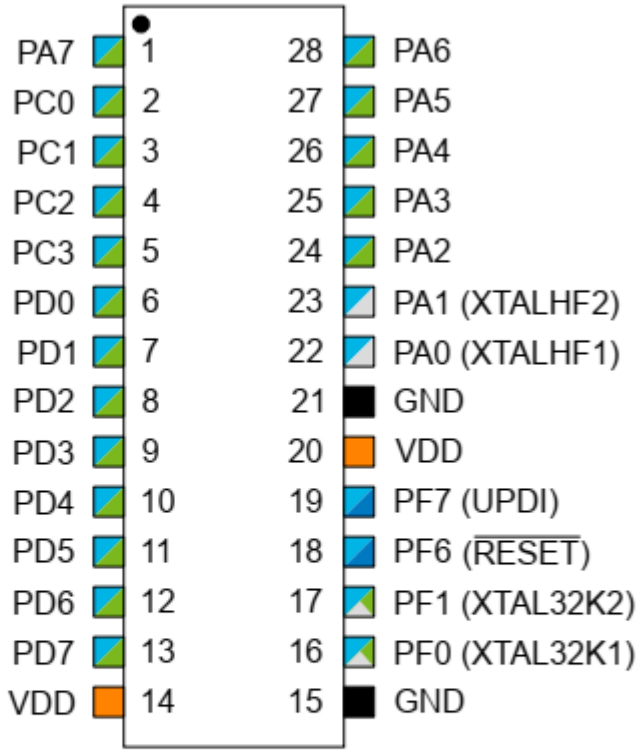
5.9.34 AVR64EA28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

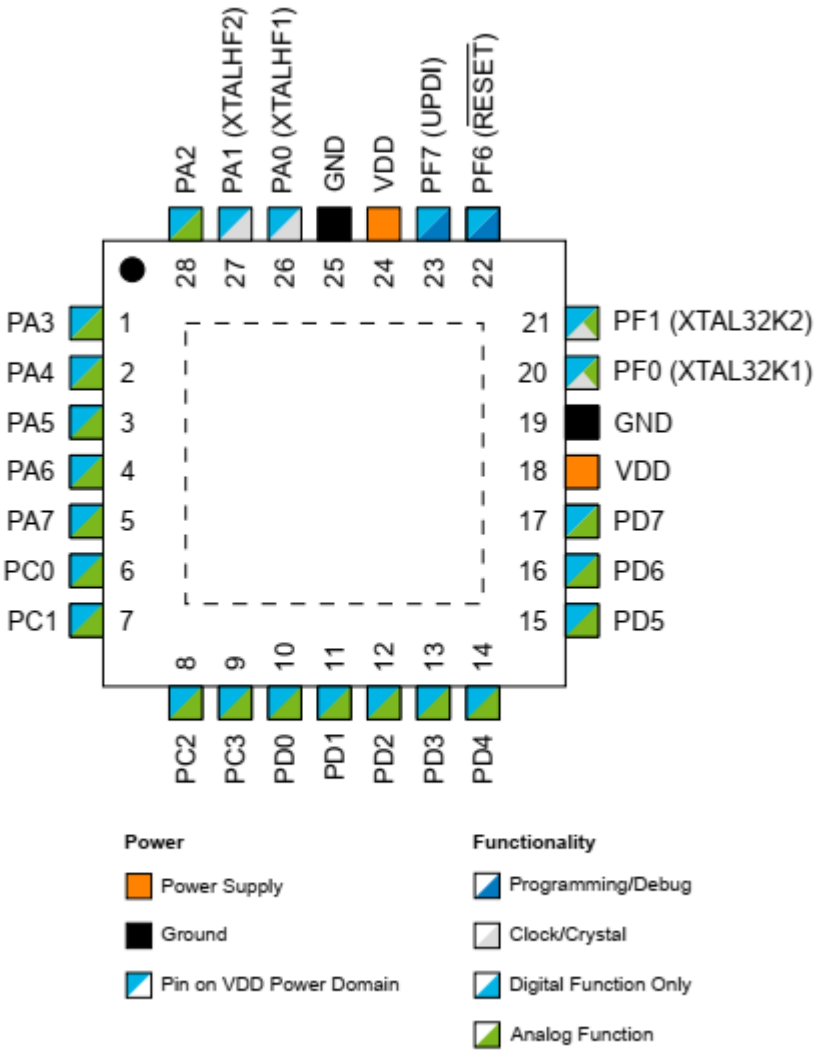
The AVR64EA28 comes in SPDIP, SSOP and VQFN. It has 64KB of flash and 28 pins. It has 6KB SRAM and 512 bytes EEPROM.

Pinout

28-pin SPDIP and SSOP



28-pin VQFN

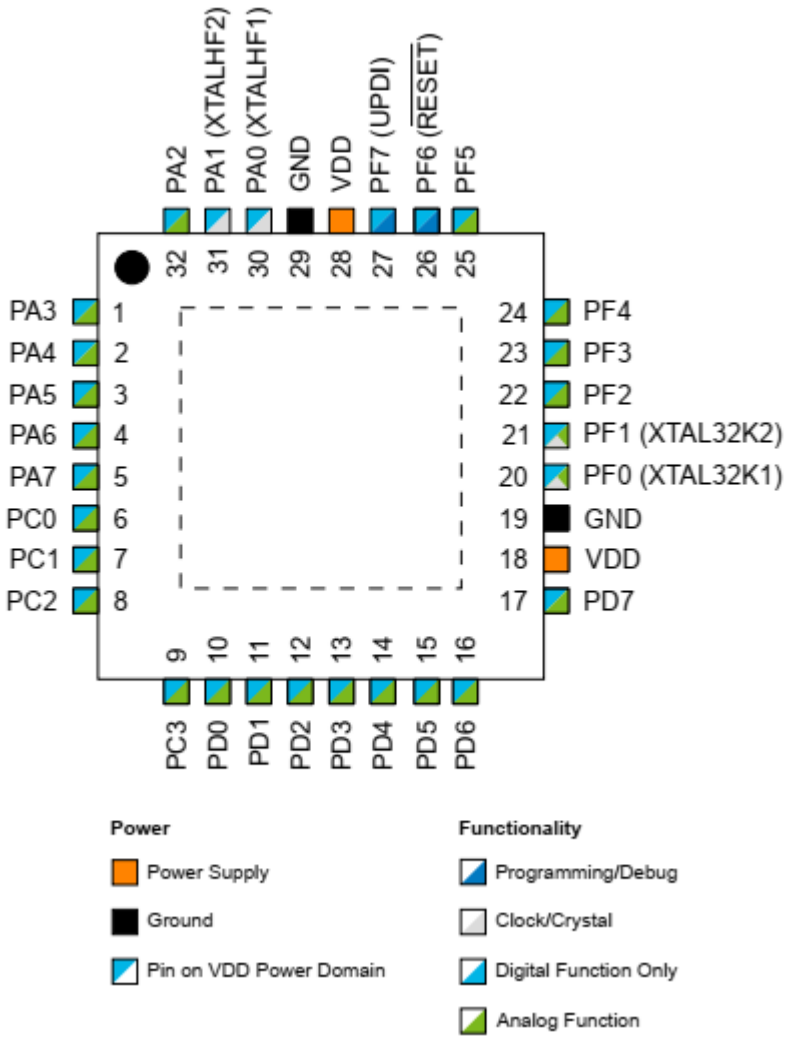


5.9.35 AVR64EA32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64EA32 comes in VQFN and TQFP. It has 64KB of flash and 32 pins. It has 6KB SRAM and 512 bytes EEPROM.

32-pin VQFN and TQFP

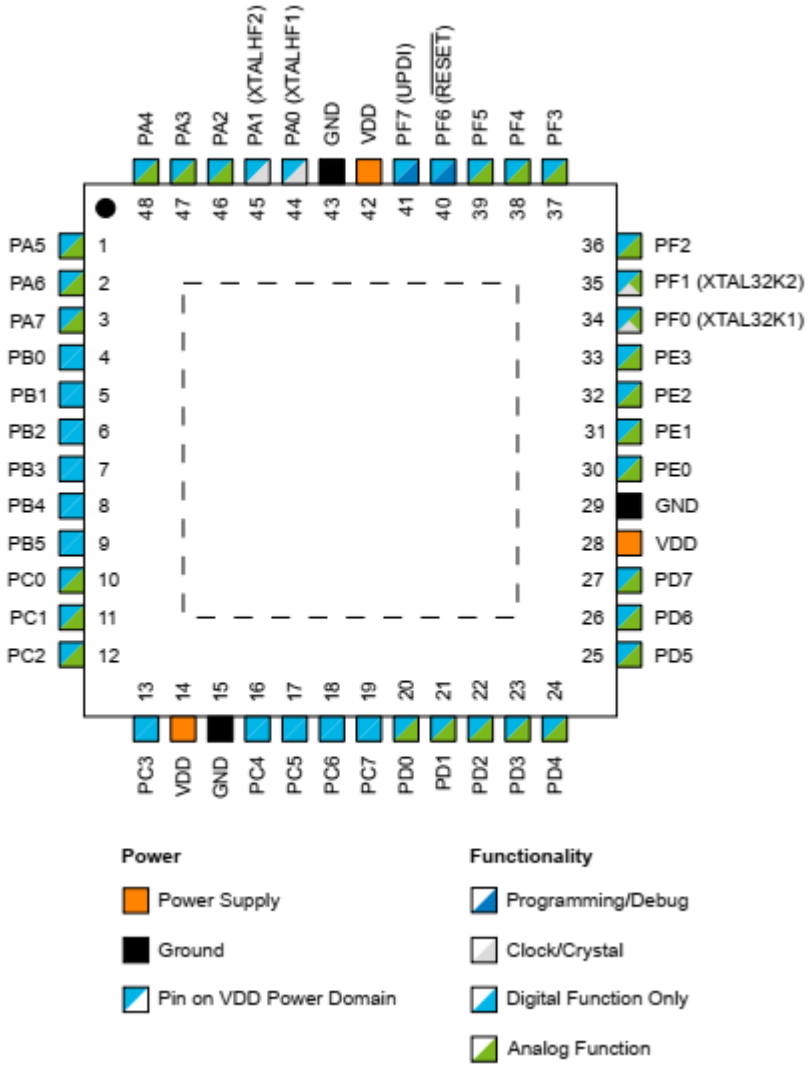


5.9.36 AVR64EA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR64EA48 comes as VQFN and TQFP. It has 64KB of flash, 8KB SRAM and 512 bytes EEPROM.

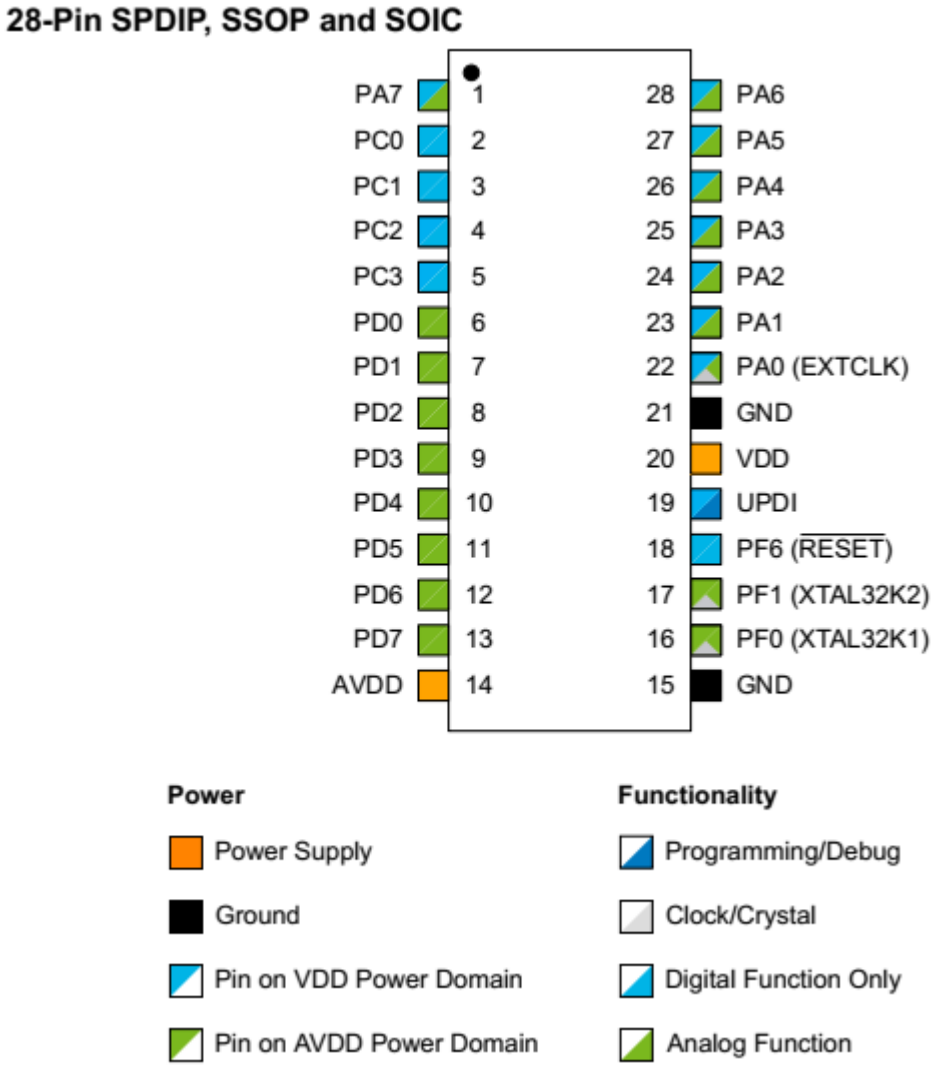
48-pin VQFN and TQFP



5.9.37 AVR128DA28

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR128DA28 comes in 28 pins SPDIP, SSOP and SOIC. It has 128KB of flash and 28 pins.
The PDIP is ideal for hobbyists. The difference with the DB series is that there is no MVIO (multi voltage IO).

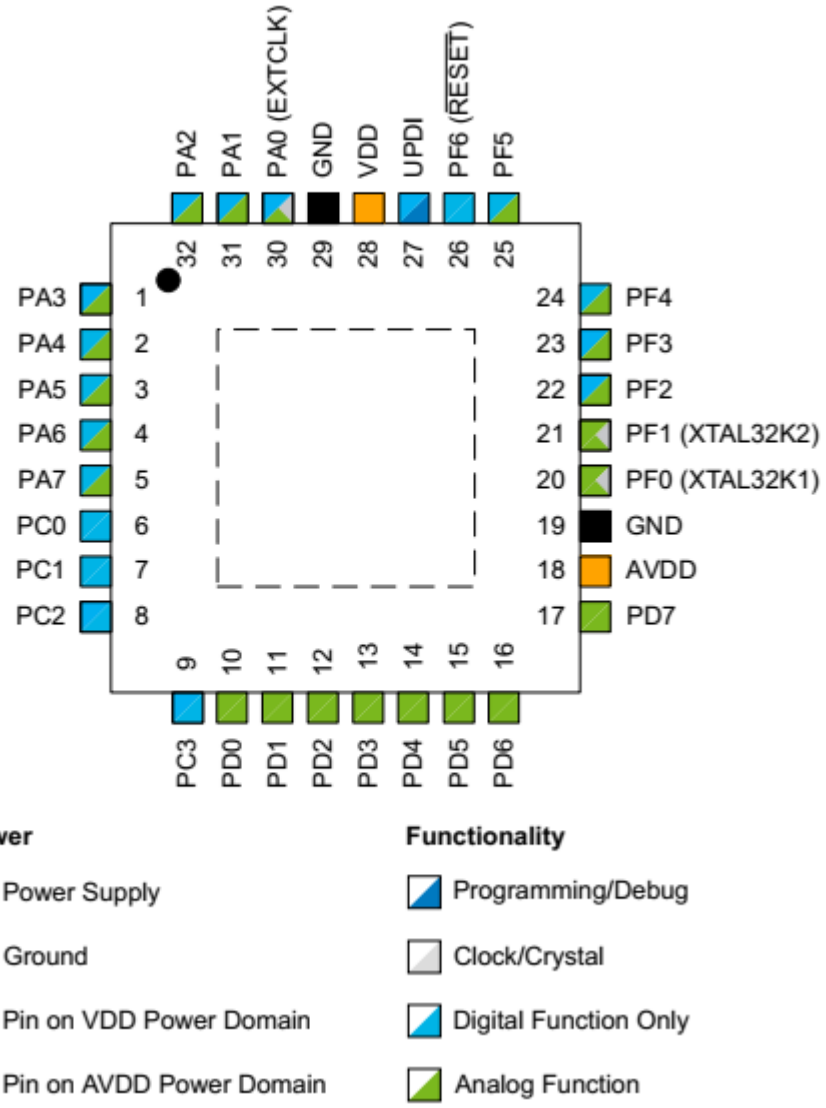


5.9.38 AVR128DA32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR128DA32 comes in 32 pins VQFN and TQFP. It has 128KB of flash and 32 pins.
The difference with the DB series is that there is no MVIO (multi voltage IO).

32-Pin VQFN and TQFP

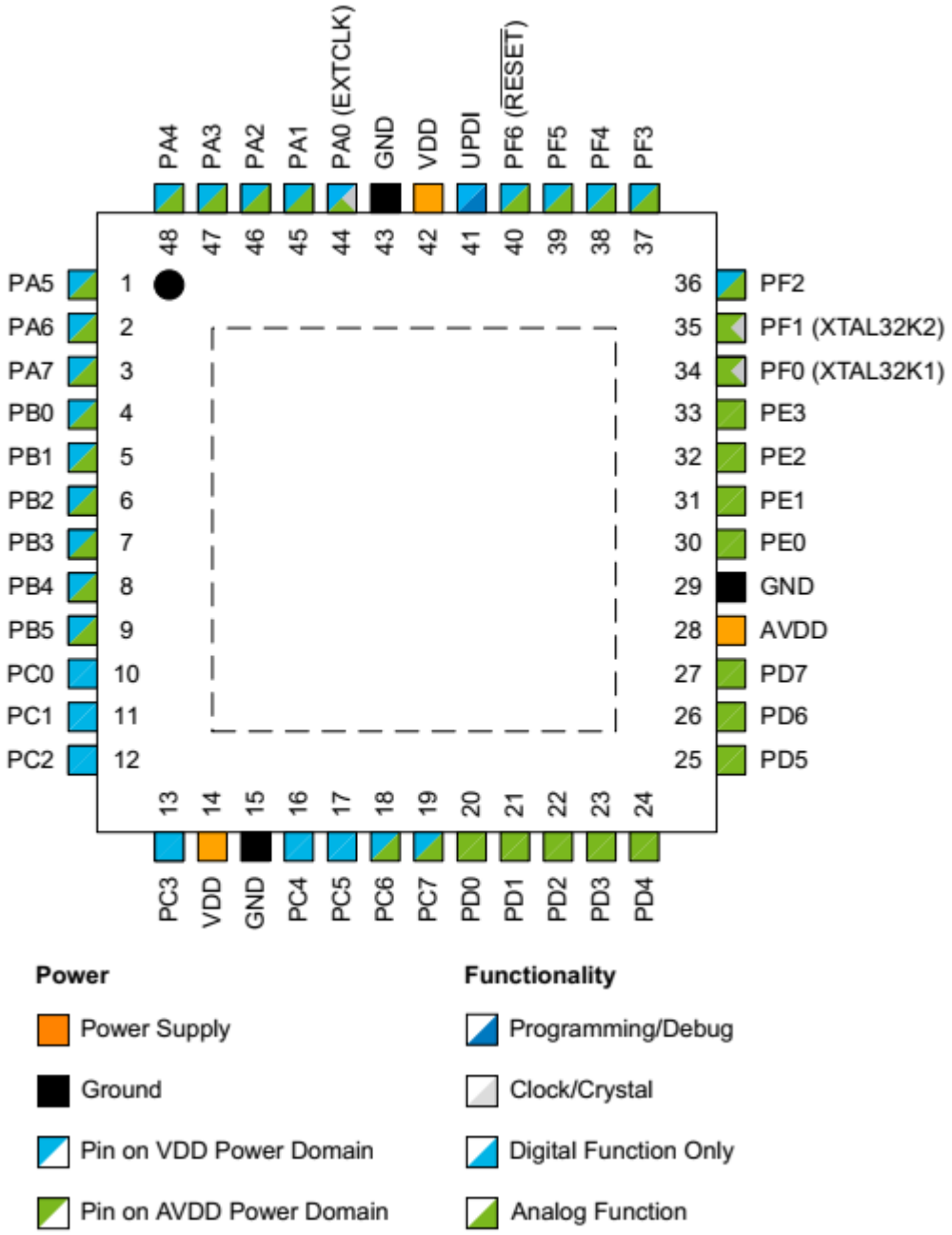


5.9.39 AVR128DA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR128DA48 comes in 48 pins VQFN and TQFP. It has 128KB of flash and 48 pins.
The difference with the DB series is that there is no MVIO (multi voltage IO).

48-Pin VQFN and TQFP

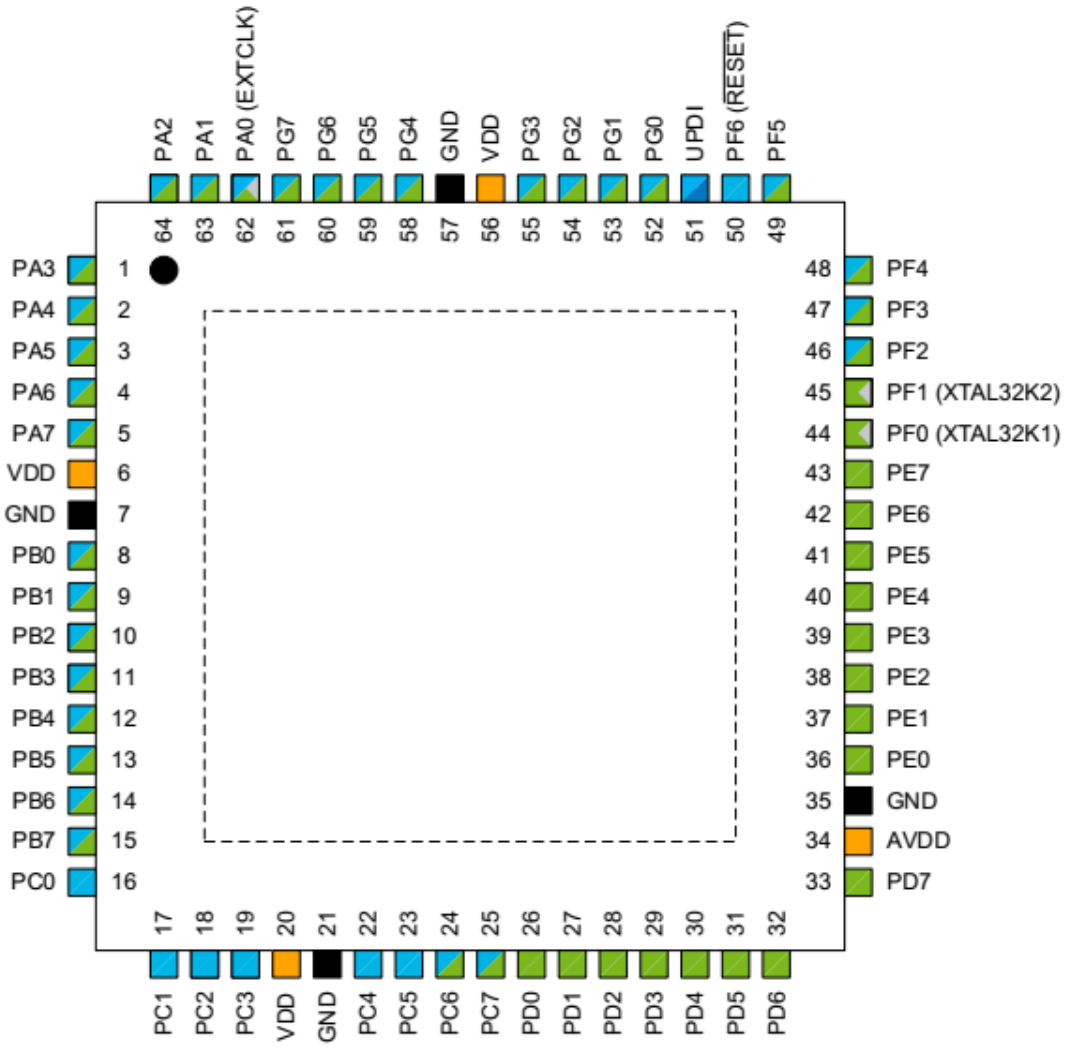


5.9.40 AVR128DA64

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVR128DA64 comes in TQFP and VQFN. It has 128KB of flash and a maximum of 64 pins. It has 8KB SRAM and 512 bytes EEPROM.

64-Pin VQFN and TQFP

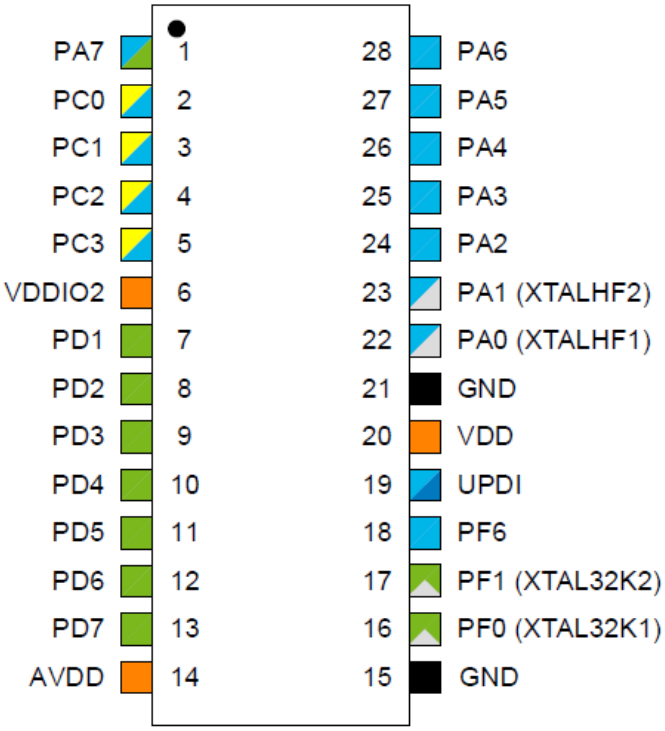


5.9.41 AVR128DB28

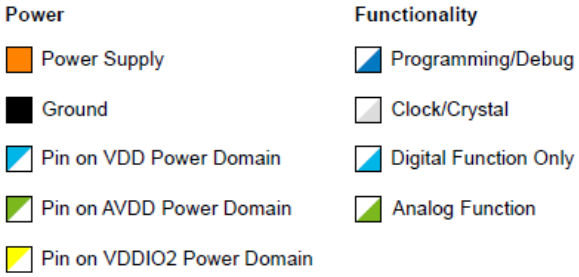
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVRDB128DB28 comes in 28 DIP/SOIC/SSOP. It has 128KB of flash and 28 pins.

2.1 28-pin SSOP, SOIC and SPDIP



Note: For the AVR® DBFamily of devices, AVDD is internally connected to VDD (not separate power domains).

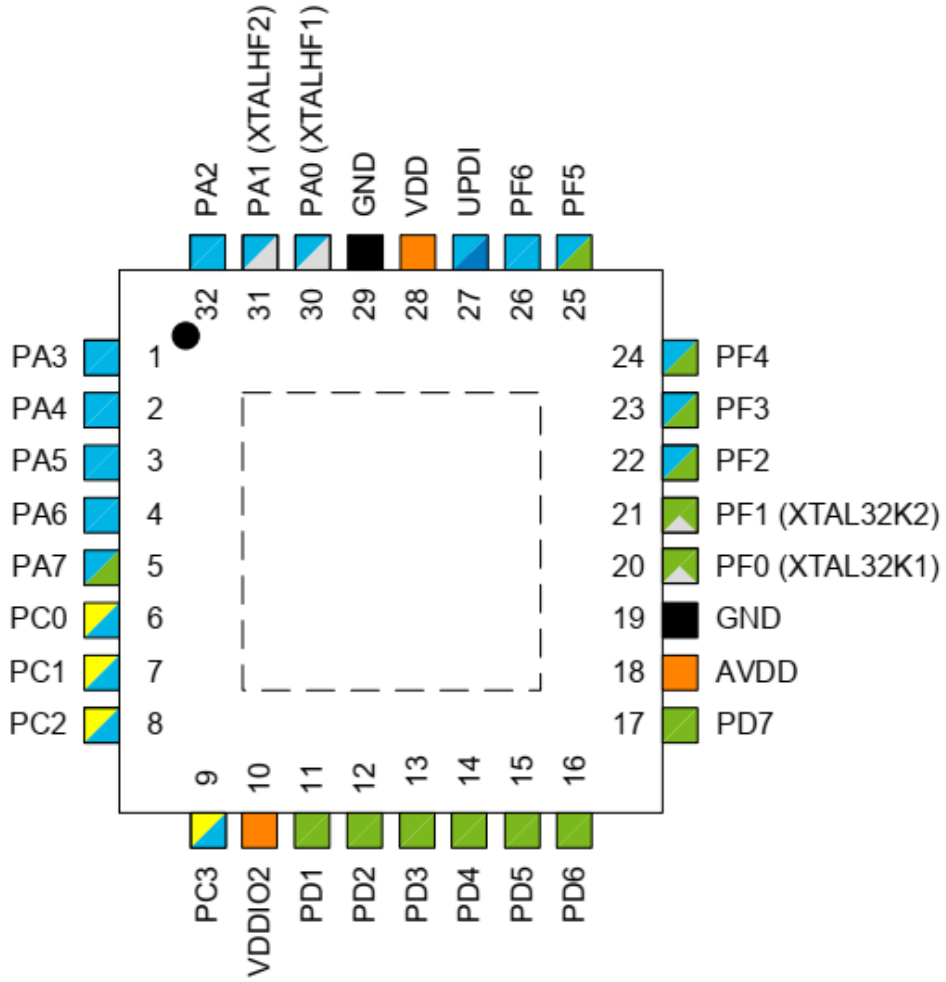


5.9.42 AVR128DB32

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVRDB128DB32 comes in 32 pin VQFN and TQFP. It has 128KB of flash and 32 pins.

32-pin VQFN and TQFP



Note: For the AVR® DBFamily of devices, AVDD is internally connected to VDD (not separate power domains).

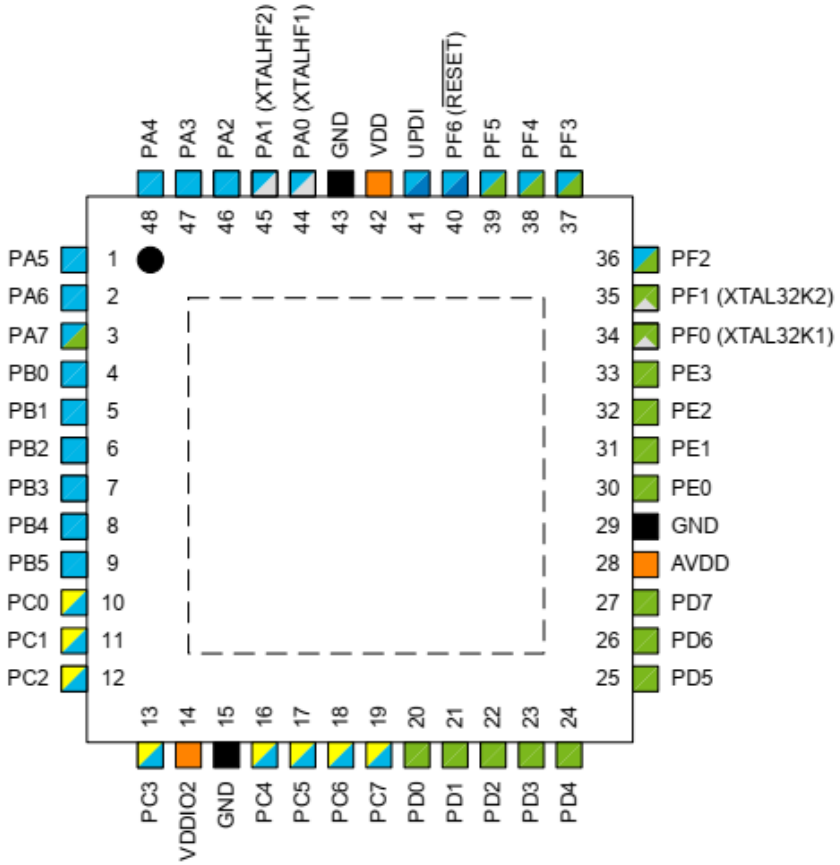
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

5.9.43 AVR128DB48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVRDB128DB48 comes in 48 pin VQFN and TQFP. It has 128KB of flash and 48 pins.

48-pin VQFN and TQFP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

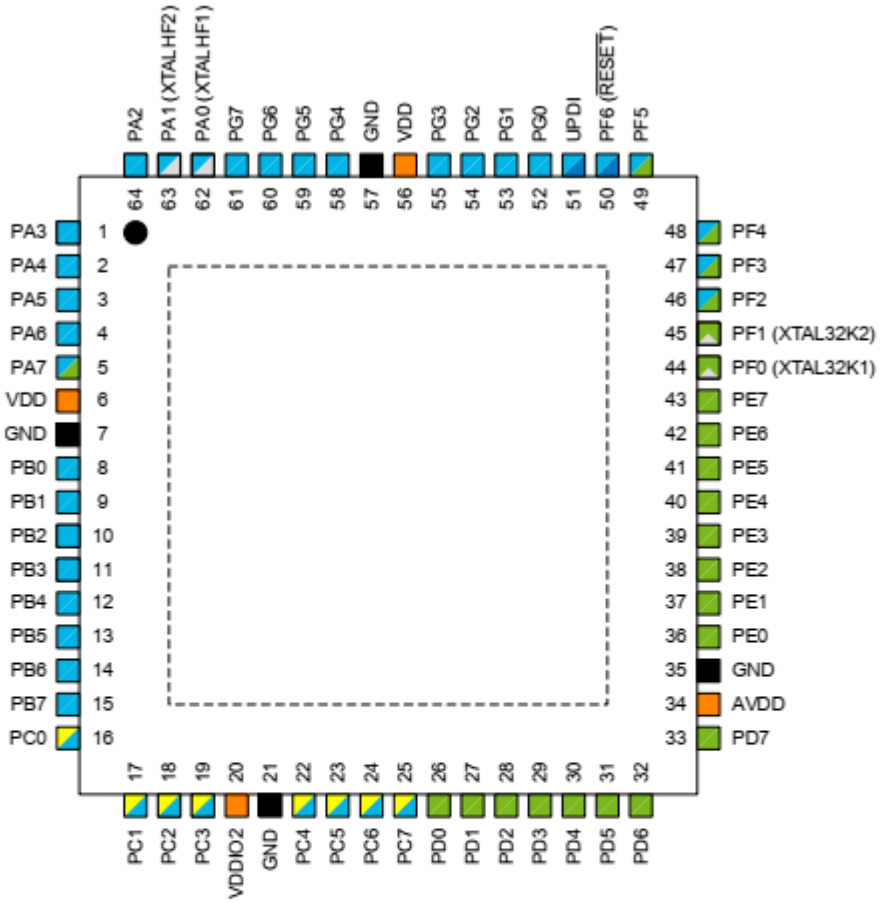
Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

5.9.44 AVR128DB64

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.
Read the generic info about Xtiny 460 , MEGAX 490 and AVRX 503

The AVRDB128DB64 comes in 64 TQFP and VQFN. It has 128KB of flash and 64 pins.

2.4 64-pin VQFN and TQFP



Note: For the AVR® DB Family of devices, AVDD is internally connected to VDD (not separate power domains).

Power	Functionality
Power Supply	Programming/Debug
Ground	Clock/Crystal
Pin on VDD Power Domain	Digital Function Only
Pin on AVDD Power Domain	Analog Function
Pin on VDDIO2 Power Domain	

Part VI

6 BASCOM Language Fundamentals

6.1 Changes compared to BASCOM-8051

The design goal was to make BASCOM-AVR compatible with BASCOM-8051.

For the AVR compiler some statements had to be removed.
New statements were also added. And some statements were changed.

They need specific attention, but the changes to the syntax will be made available to BASCOM-8051 too in the future.

Statements that were removed

STATEMENT	DESCRIPTION
$LARGE	Not needed anymore.
$ROMSTART	Code always starts at address 0 for the AVR. Added again in 1.11.6.2
$LCDHEX	Use LCD Hex(var) instead.
$NOINIT	Not needed anymore. Added in 1.11.6.2
$NOSP	Not needed anymore
$NOBREAK	Can't be used anymore because there is no object code that can be used for it.
$OBJ	Removed.
BREAK	Can't be used anymore because there is no object code that can be used for it.
PRIORITY	AVR does no allow setting priority of interrupts
PRINTHEX	You can use Print Hex(var) now
LCDHEX	You can use Lcd Hex(var) now

Statements that were added

STATEMENT	DESCRIPTION
FUNCTION	You can define your own user FUNCTIONS.
LOCAL	You can have LOCAL variables in SUB routines or FUNCTIONS.
^	New math statement. Var = 2 ^ 3 will return 2*2*2
SHIFT	Because ROTATE was changed, I added the SHIFT statement. SHIFT works just like ROTATE, but when shifted left, the LS BIT is cleared and the carry doesn't go to the LS BIT.
LTRIM	LTRIM, trims the leftmost spaces of a string.
RTRIM	RTRIM, trims the rightmost spaces of a string.
TRIM	TRIM, trims both the leftmost and rightmost spaces of a string.

Statements that behave differently

STATEMENT	DESCRIPTION
ROTATE	Rotate now behaves like the ASM rotate, this means that the carry will go to the most significant bit of a variable or the least significant bit of a variable.
CONST	String were added to the CONST statement. I also changed it to be compatible with QB.
DECLARE	BYVAL has been added since real subprograms are now supported.

DIM	You can now specify the location in memory of the variable. Dim v as byte AT 100, will use memory location 100.

6.2 Language Fundamentals

Characters from the BASCOM character set are put together to form labels, keywords, variables and operators.

These in turn are combined to form the statements that make up a program.

This chapter describes the character set and the format of BASCOM program lines. In particular, it discusses:

- The specific characters in the character set and the special meanings of some characters.
- The format of a line in a BASCOM program.
- Line labels.
- Program line length.

Character Set

The BASCOM BASIC character set consists of alphabetic characters, numeric characters, and special characters.
The alphabetic characters in BASCOM are the uppercase letters (A-Z) and lowercase letters (a-z) of the alphabet.

The BASCOM numeric characters are the digits 0-9.
The letters A-H can be used as parts of hexadecimal numbers.
The following characters have special meanings in BASCOM statements and expressions:

Character	Name
ENTER	Terminates input of a line
	Blank (or space)
'	Single quotation mark (apostrophe)
*	Asterisks (multiplication symbol)
+	Plus sign
,	Comma
-	Minus sign
.	Period (decimal point)
/	Slash (division symbol) will be handled as \
:	Colon
"	Double quotation mark
;	Semicolon
<	Less than
=	Equal sign (assignment symbol or relational operator)
>	Greater than
\	Backslash (integer/word division symbol)
^	Exponent

The BASCOM program line

BASCOM program lines have the following syntax:

[[line-identifier]] [[statement]] [[:statement]] ... [[comment]]

Using Line Identifiers

BASCOM support one type of line-identifier; alphanumeric line labels:

An alphabetic line label may be any combination of from 1 to 32 letters and digits, starting with a letter and ending with a colon.
BASCOM keywords are not permitted.

The following are valid alphanumeric line labels:

Alpha:
ScreenSUB:
Test3A:

Case is not significant. The following line labels are equivalent:

alpha:
Alpha:
ALPHA:

Line labels may begin in any column, as long as they are the first characters other than blanks on the line.
Blanks are not allowed between an alphabetic label and the colon following it.
A line can have only one label. When there is a label on the line, no other identifiers may be used on the same line. So the label is the sole identifier on a line.

BASCOM Statements

A BASCOM statement is either "executable" or " non-executable".
An executable statement advances the flow of a programs logic by telling the program what to do next.
Non executable statement perform tasks such as allocating storage for variables, declaring and defining variable types.

The following BASCOM statements are examples of non-executable statements:

- REM or (starts a comment)
- DIM

A "comment" is a non-executable statement used to clarify a programs operation and purpose.
A comment is introduced by the REM statement or a single quote character(').
The following lines are equivalent:

```
PRINT " Quantity remaining" : REM Print report label.
PRINT " Quantity remaining" ' Print report label.
```

More than one BASCOM statement can be placed on a line, but colons(:) must

separate statements, as illustrated below.

```
FOR I = 1 TO 5 : PRINT "  Gday,  mate." : NEXT I
```

Comment

Comment is intended to clarify your code. Describe what the code is supposed to do.
You can use single line comment using the **REM** statement. By default, comment is shown in green.
Since REM is a lot of type work, you can also use the **'** sign
When you want to comment multiple lines, you can also use block comment.
Block comment starts with **'(**
It ends with **')**
Please notice that block comment must be the first non white space on the line.

```
Rem some comment
Print                                                    'also
comment
'( block
comment
multiple lines
') print "ok"
```

BASCOM LineLength

If you enter your programs using the built-in editor, you are not limited to any line length, although it is advised to shorten your lines to 80 characters for clarity.

Data Types

Every variable in BASCOM has a data type that determines what can be stored in the variable. The next section summarizes the elementary data types.

Elementary Data Types

Type	Bytes used	Range	Description
Bit	1/8 Byte	0-1	A bit can hold only the value 0 or 1. A group of 8 bits is called a byte
Byte	1 Byte	0 to 255	Bytes are stored as unsigned 8-bit binary numbers
Integer	2 Bytes	-32,768 to +32,767	Integers are stored as signed sixteen-bit binary numbers
Word	2 Bytes	0 to 65535	Words are stored as unsigned sixteen-bit binary numbers
Dword	4 Bytes	0 to 4294967295	Dwords are stored as unsigned 32-bit binary numbers
Long	4 Bytes	-2147483648 to 2147483647	Longs are stored as signed 32-bit binary numbers
Single	4 Bytes	1.5 x 10^−45 to 3.4	Singles are stored as signed 32 bit binary

Type	Bytes used	Range	Description
		x 10^38	numbers
Double	8 Bytes	5.0 x 10^−324 to 1.7 x 10^308	Doubles are stored as signed 64 bit binary numbers
String	up to 254 Bytes		Strings are stored as bytes and are terminated with a chr(0) byte. A string dimensioned with a length of 10 bytes will occupy 11 bytes

Variables can be stored internal (default) , external or in EEPROM.

Variables

A variable is a name that refers to an object--a particular number.
A numeric variable, can be assigned only a numeric value (either integer, byte, long, single or bit).
The following list shows some examples of variable assignments:

- A constant value:
  A = 5
  C = 1.1
- The value of another numeric variable:
  abc = def
  k = g
- The value obtained by combining other variables, constants, and operators: Temp = a + 5
  Temp = C + 5
- The value obtained by calling a function:
  Temp = Asc(S)

Constants

A constant is a placeholder for a fixed value : you can assign it with a value only once : CONST Something = 100
Constants can be assigned with a numeric or string value. To assign a string use double quotes : CONST SomeString = "BASCOM"
You can also use expressions with constants : CONST SomeThing = 1 + 2 / (3+4)
When you keep the SHIFT key pressed and hover the mouse cursor over a constant, a tooltip/hint will show the value.

When using numeric constants in DATA 11177 lines, you need to inform the compiler about the data type. This is done by ending the constant value with a suffix. See the help for DATA.

Variable Names

A BASCOM variable name may contain up to 32 characters.

The characters allowed in a variable name are letters and numbers.
The first character in a variable name must be a letter.

A variable name cannot be a reserved word, but embedded reserved words are allowed.
For example, the following statement is illegal because AND is a reserved word.

```
AND = 8
```

However, the following statement is legal:

```
ToAND = 8
```

Reserved words include all BASCOM commands, statements, function names, internal registers and operator names.
(see BASCOM Reserved Words 583 , for a complete list of reserved words).

You can specify a hexadecimal or binary number with the prefix &H or &B.
a = &HA , a = &B1010 and a = 10 are all the same.

Before assigning a variable, you must tell the compiler about it with the DIM 1228 statement.

```
Dim b1 As Bit, I as Integer, k as Byte , s As String * 10
```

The STRING type needs an additional parameter to specify the length.

You can also use DEFINT 1227, DEFBIT 1227, DEFBYTE 1227 ,DEFWORD 1227 ,DEFLNG 1227 or DEFSNG 1227.

For example,DEFINT c tells the compiler that all variables that are not dimensioned and that are beginning with the character c are of the Integer type.

BITS and Interrupts

Bits are stored in bytes. A write to a bit/boolean variable is non-atomic. Which means that multiple operations are required to update the bit value in the byte. When interrupts are used that update bits in the same byte, you can have the effect that a change becomes lost.
To prevent this you can disable interrupts and enable them after you have updated the bit variable. Or you can use a byte instead which is recommended since it would use less code.

Expressions and Operators

This chapter discusses how to combine, modify, compare, or get information about expressions by using the operators available in BASCOM.

Anytime you do a calculation you are using expressions and operators.

This chapter describes how expressions are formed and concludes by describing the following kind of operators:

- Arithmetic operators, used to perform calculations.
- Relational operators, used to compare numeric or string values.
- Logical operators, used to test conditions or manipulate individual bits.
- Functional operators, used to supplement simple operators.

Expressions and Operators

An expression can be a numeric constant, a variable, or a single value obtained by combining constants, variables, and other expressions with operators.

Operators perform mathematical or logical operations on values.
The operators provided by BASCOM can be divided into four categories, as follows:

1. Arithmetic
2. Relational
3. Logical
4. Functional

Arithmetic

Arithmetic operators are +, - , * , \, / and ^.

- Integer
  Integer division is denoted by the backslash (\).
  Example: Z = X \ Y

- Modulo Arithmetic
  Modulo arithmetic is denoted by the modulus operator MOD.
  Modulo arithmetic provides the remainder, rather than the quotient, of an integer division.

  Example: X = 10 \ 4 : remainder = 10 MOD 4

- Overflow and division by zero
  Division by zero, produces an error.
  At the moment no message is produced, so you have to make sure yourself that this won't happen.

Relational Operators

Relational operators are used to compare two values as shown in the table below.
The result can be used to make a decision regarding program flow.

Operator	Relation Tested	Expression
=	Equality	X = Y
<>	Inequality	X <> Y
<	Less than	X < Y
>	Greater than	X > Y
<=	Less than or equal to	X <= Y
>=	Greater than or equal to	X >= Y

Logical Operators

Logical operators perform tests on relations, bit manipulations, or Boolean operators.
There four operators in BASCOM are :

Operator	Meaning
NOT	Logical complement
AND	Conjunction
OR	Disjunction
XOR	Exclusive or

It is possible to use logical operators to test bytes for a particular bit pattern.
For example the AND operator can be used to mask all but one of the bits of a status byte, while OR can be used to merge two bytes to create a particular binary value.

Example

```
A = 63 And 19
PRINT A
A = 10 Or 9
PRINT A
```

Output

```
19
11
```

Floating point SINGLE (4 BYTE)(ASM code used is supplied by Jack Tidwell)
Single numbers conforming to the IEEE binary floating point standard.
An eight bit exponent and 24 bit mantissa are supported.
Using four bytes the format is shown below:

31 30_____23 22_____0

s exponent mantissa

The exponent is biased by 128. Above 128 are positive exponents and below are negative. The sign bit is 0 for positive numbers and 1 for negative. The mantissa is stored in hidden bit normalized format so that 24 bits of precision can be obtained.

All mathematical operations are supported by the single.
You can also convert a single to an integer or word or vise versa:

```
Dim I as Integer, S as Single

S = 100.1 'assign the single
I = S  'will convert the single to an integer
```

Here is a fragment from the Microsoft knowledge base about FP:

Floating-point mathematics is a complex topic that confuses many programmers. The tutorial below should help you recognize programming situations where floating-point errors are likely to occur and how to avoid them. It should also allow you to recognize cases that are caused by inherent floating-point math limitations as opposed to actual compiler bugs.

Decimal and Binary Number Systems

Normally, we count things in base 10. The base is completely arbitrary. The only reason that people have traditionally used base 10 is that they have 10 fingers, which have made handy counting tools.

The number 532.25 in decimal (base 10) means the following:

(5 * 10^2) + (3 * 10^1) + (2 * 10^0) + (2 * 10^-1) + (5 * 10^-2)

500 + 30 + 2 + 2/10 + 5/100

_____

= 532.25

In the binary number system (base 2), each column represents a power of 2 instead of 10. For example, the number 101.01 means the following:
(1 * 2^2) + (0 * 2^1) + (1 * 2^0) + (0 * 2^-1) + (1 * 2^-2)
4 + 0 + 1 + 0 + 1/4

_____
= 5.25 Decimal

How Integers Are Represented in PCs

Because there is no fractional part to an integer, its machine representation is much simpler than it is for floating-point values. Normal integers on personal computers (PCs) are 2 bytes (16 bits) long with the most significant bit indicating the sign. Long integers are 4 bytes long.

Positive values are straightforward binary numbers. For example:

1 Decimal = 1 Binary
2 Decimal = 10 Binary
22 Decimal = 10110 Binary, etc.

However, negative integers are represented using the two's complement scheme. To get the two's complement representation for a negative number, take the binary representation for the number's absolute value and then flip all the bits and add 1. For example:

4 Decimal = 0000 0000 0000 0100

1111 1111 1111 1011 Flip the Bits

-4 = 1111 1111 1111 1100 Add 1

Note that adding any combination of two's complement numbers together

using ordinary binary arithmetic produces the correct result.

Floating-Point Complications

Every decimal integer can be exactly represented by a binary integer; however, this is not true for fractional numbers. In fact, every number that is irrational in base 10 will also be irrational in any system with a base smaller than 10.

For binary, in particular, only fractional numbers that can be represented in the form p/q, where q is an integer power of 2, can be expressed exactly, with a finite number of bits.

Even common decimal fractions, such as decimal 0.0001, cannot be represented exactly in binary. (0.0001 is a repeating binary fraction with a period of 104 bits!)

This explains why a simple example, such as the following

```
SUM = 0
FOR I% = 1 TO 10000
  SUM = SUM + 0.0001
NEXT I%
PRINT SUM ' Theoretically = 1.0.
```

will PRINT 1.000054 as output. The small error in representing 0.0001 in binary propagates to the sum.

For the same reason, you should always be very cautious when making comparisons on real numbers. The following example illustrates a common programming error:

```
item1# = 69.82#
item2# = 69.20# + 0.62#
IF item1# = item2# then print "Equality!"
```

This will NOT PRINT "Equality!" because 69.82 cannot be represented exactly in binary, which causes the value that results from the assignment to be SLIGHTLY different (in binary) than the value that is generated from the expression. In practice, you should always code such comparisons in such a way as to allow for some tolerance.

General Floating-Point Concepts

It is very important to realize that any binary floating-point system can represent only a finite number of floating-point values in exact form. All other values must be approximated by the closest represent able value. The IEEE standard specifies the method for rounding values to the "closest" represent able value. BASCOM supports the standard and rounds according to the IEEE rules.

Also, keep in mind that the numbers that can be represented in IEEE are spread out over a very wide range. You can imagine them on a number line. There is a high density of represent able numbers near 1.0 and -1.0 but fewer and fewer as you go towards 0 or infinity.

The goal of the IEEE standard, which is designed for engineering calculations, is to maximize accuracy (to get as close as possible to the actual number). Precision refers to the number of digits that you can represent. The IEEE standard attempts to balance the number of bits dedicated to the exponent with the number of bits used for the fractional part of the number, to keep both accuracy and precision within acceptable limits.

IEEE Details

Floating-point numbers are represented in the following form, where [exponent] is the binary exponent:

X = Fraction * 2^(exponent - bias)

[Fraction] is the normalized fractional part of the number, normalized because the exponent is adjusted so that the leading bit is always a 1. This way, it does not have to be stored, and you get one more bit of precision. This is why there is an implied bit. You can think of this like scientific notation, where you manipulate the exponent to have one digit to the left of the decimal point, except in binary, you can always manipulate the exponent so that the first bit is a 1, since there are only 1s and 0s.

[bias] is the bias value used to avoid having to store negative exponents.

The bias for single-precision numbers is 127 and 1023 (decimal) for double-precision numbers.

The values equal to all 0's and all 1's (binary) are reserved for representing special cases. There are other special cases as well, that indicate various error conditions.

Single-Precision Examples

2 = 1 * 2^1 = 0100 0000 0000 0000 ... 0000 0000 = 4000 0000 hex
Note the sign bit is zero, and the stored exponent is 128, or

100 0000 0 in binary, which is 127 plus 1. The stored mantissa is (1.) 000 0000 ... 0000 0000, which has an implied leading 1 and binary point, so the actual mantissa is 1.

-2 = -1 * 2^1 = 1100 0000 0000 0000 ... 0000 0000 = C000 0000 hex
Same as +2 except that the sign bit is set. This is true for all IEEE format floating-point numbers.

4 = 1 * 2^2 = 0100 0000 1000 0000 ... 0000 0000 = 4080 0000 hex
Same mantissa, exponent increases by one (biased value is 129, or 100 0000 1 in binary.

6 = 1.5 * 2^2 = 0100 0000 1100 0000 ... 0000 0000 = 40C0 0000 hex
Same exponent, mantissa is larger by half -- it's

(1.) 100 0000 ... 0000 0000, which, since this is a binary fraction, is 1-1/2 (the values of the fractional digits are 1/2, 1/4, 1/8, etc.).

1 = 1 * 2^0 = 0011 1111 1000 0000 ... 0000 0000 = 3F80 0000 hex
Same exponent as other powers of 2, mantissa is one less than 2 at 127, or 011 1111 1 in binary.

.75 = 1.5 * 2^-1 = 0011 1111 0100 0000 ... 0000 0000 = 3F40 0000 hex

The biased exponent is 126, 011 1111 0 in binary, and the mantissa is (1.) 100 0000 ... 0000 0000, which is 1-1/2.

2.5 = 1.25 * 2^1 = 0100 0000 0010 0000 ... 0000 0000 = 4020 0000 hex
Exactly the same as 2 except that the bit which represents 1/4 is set in the mantissa.

0.1 = 1.6 * 2^-4 = 0011 1101 1100 1100 ... 1100 1101 = 3DCC CCCD hex

1/10 is a repeating fraction in binary. The mantissa is just shy of 1.6, and the biased exponent says that 1.6 is to be divided by 16 (it is 011 1101 1 in binary, which is 123 n decimal). The true exponent is 123 - 127 = -4, which means that the factor by which to multiply is 2**-4 = 1/16. Note that the stored mantissa is rounded up in the last bit. This is an attempt to represent the un-representable number as accurately as possible. (The reason that 1/10 and 1/100 are not exactly representable in binary is similar to the way that 1/3 is not exactly representable in decimal.)

0 = 1.0 * 2^-128 = all zeros -- a special case.

Other Common Floating-Point Errors

The following are common floating-point errors:

1. Round-off error
This error results when all of the bits in a binary number cannot be used in a calculation.
Example: Adding 0.0001 to 0.9900 (Single Precision)
Decimal 0.0001 will be represented as:
(1.)10100011011011100010111 * 2^(-14+Bias) (13 Leading 0s in Binary!)

0.9900 will be represented as:
(1.)11111010111000010100011 * 2^(-1+Bias)

Now to actually add these numbers, the decimal (binary) points must be aligned. For this they must be Unnormalized. Here is the resulting addition:

.00000000000011010001101 * 2^0 <- Only 11 of 23 Bits retained
+.11111010111000010100011 * 2^0
_____
.11111010111011100110000 * 2^0

This is called a round-off error because some computers round when shifting for addition. Others simply truncate. Round-off errors are important to consider whenever you are adding or multiplying two very different values.

2. Subtracting two almost equal values

.1235
-.1234
_____
.0001

This will be normalized. Note that although the original numbers each had four significant digits, the result has only one significant digit.

3. Overflow and underflow
This occurs when the result is too large or too small to be represented by the data type.

4. Quantizing error
This occurs with those numbers that cannot be represented in exact form by the floating-point standard.

Rounding
When a Long is assigned to a single, the number is rounded according to the rules of the IEEE committee.

For explanation: 1.500000 is exact the middle between 1.00000 and 2.000000. If x.500000 is always rounded up, than there is trend for higher values than the average of all numbers. So their rule says, half time to round up and half time to round down, if value behind LSB is exact ..500000000.

The rule is, round this .500000000000 to next even number, that means if LSB is 1 (half time) to round up, so the LSB is going to 0 (=even), if LSB is 0 (other half time) to round down, that means no rounding.

This rounding method is best since the absolute error is 0.

You can override the default IEEE rounding method by specifying the $LIB LONG2FLOAT.LBX library which rounds up to the next number. This is the method used up to 1.11.7.4 of the compiler.

Double
The double is essential the same as a single. Except the double consist of 8 bytes instead of 4. The exponent is 11 bits leaving 52 bits for the mantissa.

Arrays
An array is a set of sequentially indexed elements having the same type. Each element of an array has a unique index number that identifies it. Changes made to an element of an array do not affect the other elements.

The index must be a numeric constant, a byte, an integer, word or long.
The maximum number of elements is 65535. For Xmega with huge memory it is 8MB!
The first element of an array is always one by default. This means that elements are 1-based.
You can change this with CONFIG BASE=0. In this case, the first element will be element 0.

Arrays can be used on each place where a 'normal' variable is expected.

You can add an offset to the index too. This could be used to emulate a 2 dimensional array.
```
row_index = row  :  shift row_index, left,4
value = parameter_array(column+row_index)
```

Example:

```
'create an array named a, with 10 elements (1 to 10)
```

```
Dim A(10) As Byte
'create an integer
Dim C As Integer
'now fill the array
For C = 1 To 10
'assign array element
A(c)= C
' print it
Print A(c)
Next
'you can add an offset to the index too
C = 0
A(c + 1)= 100
Print A(c + 1)
End
```

Strings

A string is used to store text. A string must be dimensioned with the length specified.

```
DIM S as STRING * 5
```

Will create a string that can store a text with a maximum length of 5 bytes.
The space used is 6 bytes because a string is terminated with a null byte.

To assign the string:

```
Ds = "abcd"
```

To insert special characters into the string :

```
s= "AB{027}cd"
```

The {ascii} will insert the ASCII value into the string.

The number of digits must be **3**.
s = "{27}" will assign "{27}" to the string instead of escape character 27!

Because the null byte (ASCII 0) is used to terminate a string, you can not embed a null byte into a string.

Casting

In BASCOM-AVR when you perform operations on variables they all must be of the same data type.
long = long1 * long2 ' for example

The assigned variables data type determines what kind of math is performed.
For example when you assign a long, long math will be used.

If you try to store the result of a LONG into a byte, only the LSB of the LONG will be stored into the BYTE.
Byte = LONG

When LONG = 256 , it will not fit into a BYTE. The result will be 256 AND 255 = 0.

Of course you are free to use different data types. The correct result is only guaranteed when you are using data types of the same kind or that result always can fit into the target data type.

When you use strings, the same rules apply. But there is one exception:

```
Dim b as Byte

b = 123 ' ok this is normal
b = "A" ' b = 65
```

When the target is a byte and the source variable is a string constant denoted by "", the ASCII value will be stored in the byte. This works also for tests :

```
IF b = "A" then ' when b = 65

END IF
```

This is different compared to QB/VB where you can not assign a string to a byte variable.

SINGLE CONVERSION

When you want to convert a SINGLE into a byte, word, integer or long the compiler will automatic convert the values when the source string is of the SINGLE data type.

integer = single

You can also convert a byte, word, integer or long into a SINGLE by assigning this variable to a SINGLE.

single = long

6.3 Mixing ASM and BASIC

BASCOM allows you to mix BASIC with assembly.
This can be very useful in some situations when you need full control of the generated code.

In order to use ASM you **must** start the line with the character !
Optional you can create a block of ASM using $ASM end $END ASM
Use CTRL + SPACE to get a list of ASM mnemonics.

For example :

```
Dim a As Byte At &H60   ' A is stored at location &H60
!Ldi R27 , $00    ' Load R27 with MSB of address
!Ldi R26 , $60    ' Load R26 with LSB of address
!Ld R1, X              ' load memory location $60 into R1
!SWAP R1          ' swap nibbles
```

As you can see the SWAP mnemonic is preceded by a **!** sign. Without it, it would be the BASIC SWAP statement.

Another option is to use the assembler block directives:

```
$ASM
Ldi R27 , $00     ' Load R27 with MSB of address
Ldi R26 , $60     ' Load R26 with LSB of address
Ld R1, X          ' load memory location $60 into R1
SWAP R1           ' swap nibbles
$END ASM
```

A special assembler helper function is provided to load the address into the register X or Z. Y can may not be used because it is used as the soft stack pointer.

```
Dim A As Byte           ' reserve space
LOADADR a, X            ' load address of variable named A into register
pair X

This has the same effect as :
Ldi R26, $60            ' for example !
Ldi R27, $00            ' for example !
```

Some registers are used by BASCOM
R4 and R5 are used to point to the stack frame or the temp data storage
R6 is used to store some bit variables:
R6 bit 0 = flag for integer/word conversion
R6 bit 1 = temp bit space used for swapping bits
R6 bit 2 = error bit (ERR variable)
R6 bit 3 = show/noshow flag when using INPUT statement
R8 and R9 are used as a data pointer for the READ statement.

All other registers are used depending on the used statements.

To Load the address of a variable you must enclose them in brackets.

```
Dim B As Bit
Lds R16, {B} 'will replace {B} with the address of variable B
```

To refer to the bit number you must precede the variable name by BIT.

```
Sbrs R16 , BIT.B  'notice the point!
```

Since this was the first dimensioned bit the bit number is 7. Bits are stored in bytes and the first dimensioned bit goes in the MS (most significant) bit.

To load an address of a label you must use :

```
LDI ZL, Low(lbl  * 1)
LDI ZH, High(lbl * 1)
```

Where ZL = R30 and may be R24, R26, R28 or R30

And ZH = R31 and may be R25, R27, R29 or R31.

These are so called register pairs that form a pointer.

When you want to use the LPM instruction to retrieve data you must multiply the address with 2 since the AVR object code consist of words.

```
LDI ZL, Low(lbl * 2)
LDI ZH, High(lbl * 2)
LPM ; get data into R0
Lbl:
```

Atmel mnemonics must be used to program in assembly.
You can download the pdf from www.atmel.com that shows how the different mnemonics are used.

Some points of attention :
* All instructions that use a constant as a parameter only work on the upper 16 registers (r16-r31)
So LDI R15,12 WILL NOT WORK

* The instruction SBR register, K
will work with K from 0-255. So you can set multiple bits!

The instruction SBI port, K will work with K from 0-7 and will set only ONE bit in a IO-port register.

The same applies to the CBR and CBI instructions.

You can use constants too:

```
.equ myval = (10+2)/4
ldi r24,myval+2 '5
ldi r24,asc("A")+1 ; load with 66
```

Or in BASIC with CONST :

```
CONST Myval = (10+2) / 4
Ldi r24,myval
```

How to make your own libraries and call them from BASIC?
The files for this sample can be found as libdemo.bas in the SAMPLES dir and as mylib.lib in the LIB dir.

First determine the used parameters and their type.
Also consider if they are passed by reference or by value

For example the sub test has two parameters:
x which is passed by value (copy of the variable)
y which is passed by reference(address of the variable)

In both cases the address of the variable is put on the soft stack which is indexed by the Y pointer.

The first parameter (or a copy) is put on the soft stack first
To refer to the address you must use:

```
ldd r26 , y + 0
ldd r27 , y + 1
```

This loads the address into pointer X
The second parameter will also be put on the soft stack so :
The reference for the x variable will be changed :

To refer to the address of x you must use:

```
ldd r26 , y + 2
ldd r27 , y + 3
```

To refer to the last parameter y you must use

```
ldd r26 , y + 0
ldd r27 , y + 1
```

Write the sub routine as you are used too but include the name within brackets []

```
[test]
test:
ldd r26,y+2             ; load address of x
ldd r27,y+3
ld r24,x                ; get value into r24
inc r24                 ; value + 1
```

```
st x,r24                ; put back
ldd r26,y+0             ; address of y
ldd r27,y+1
st x,r24                ; store
ret                     ; ready
[end]
```

To write a function goes the same way.
A function returns a result so a function has one additional parameter.
It is generated automatic and it has the name of the function.
This way you can assign the result to the function name

For example:

```
Declare Function Test(byval x as byte , y as byte) as byte
```

A virtual variable will be created with the name of the function in this case test.
It will be pushed on the soft stack with the Y-pointer.

To reference to the result or name of the function (test) the address will be:
y + 0 and y + 1
The first variable x will bring that to y + 2 and y + 3

And the third variable will cause that 3 parameters are saved on the soft stack

To reference to test you must use :

```
ldd r26 , y + 4
ldd r27 , y + 5
```

To reference variable x

```
ldd r26 , y + 2
ldd r27 , y + 3
```

And to reference variable y

```
ldd r26 , y + 0
ldd r27 , y + 1
```

When you use exit sub or exit function you also need to provide an additional label. It starts with sub_ and must be completed with the function / sub routine name. In our example:

```
sub_test:
```

LOCALS

When you use local variables thing become more complicated.

Each local variable address will be put on the soft stack too

When you use 1 local variable its address will become

```
ldd r26, y+0
ldd r27 , y + 1
```

All other parameters must be increased with 2 so the reference to y variable changes from

```
ldd r26 , y + 0 to ldd r26 , y + 2
ldd r27 , y + 1 to ldd r27 , y + 3
```

And of course also for the other variables.

When you have more local variables just add 2 for each.

Finally you save the file as a .lib file
Use the library manager to compile it into the lbx format.
The declare sub / function must be in the program where you use the sub / function.

The following is a copy of the libdemo.bas file :

```
' define the used library
$lib "mylib.lib"

'also define the used routines
$external Test

'this is needed so the parameters will be placed correct on the stack
Declare Sub Test(byval X As Byte , Y As Byte)

'reserve some space
Dim Z As Byte

'call our own sub routine
Call Test(1 , Z)

'z will be 2 in the used example
End
```

When you use ports in your library you must use .equ to specify the address:

```
.equ EEDR=$1d
In R24, EEDR
```

This way the library manager knows the address of the port during compile time.

As an alternative precede the mnemonic with a * so the code will not be compiled into the lib. The address of the register will be resolved at run time in that case.

This chapter is not intended to teach you ASM programming. But when you find a topic is missing to interface BASCOM with ASM send me an email.

Translation

In version 1.11.7.5 of the compiler some mnemonics are translated when there is a need for.

For example, SBIC will work only on normal PORT registers. This because the address may not be greater then 5 bits as 3 bits are used for the pin number(0-7).

SBIC worked well in the old AVR chips(AT90Sxxxx) but in the Mega128 where PORTG is on a high address, it will not work.

You always needs a normal register when you want to manipulate the bits of an external register.

For example :

```
LDS r23, PORTG    ; get value of PORTG register
SBR r23,128       ; set bit 7
STS PORTG, R23
```

The mnemonics that are translated by the compiler are : IN, OUT, SBIC, SBIS, SBI

and CBI.

The compiler will use register R23 for this. So make sure it is not used.

Special instructions

```
ADR Label   ; will create a word with the address of the label name
ADR2 Label  ; will create a word with the address of the label name,
multiplied by 2 to get the byte address
            ; since word addresses are used. This is convenient when
loading the Z-pointer to use (E)LPM.

.align      ; This directive will align the code to a 256 byte page so
that the address LSB becomes 0.
            ; When storing data at an address where the LSB is zero, you
can test for an overflow of the MSB only.
```

6.4 Assembler mnemonics

BASCOM supports the mnemonics as defined by Atmel.

The Assembler accepts mnemonic instructions from the instruction set.

A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Mnemonics	Operands	Description	Operation	Flags	Clock
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H	1
SUB	Rd, Rr	Subtract without Carry	Rd = Rd − Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Immediate	Rd = Rd − K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd = Rd - K - C	Z,C,N,V,H	1
AND	Rd, Rr	Logical AND	Rd = Rd · Rr	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	Rd = Rd · K	Z,N,V	1
OR	Rd, Rr	Logical OR	Rd = Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR with Immediate	Rd = Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR	Rd = Rd Å Rr	Z,N,V	1
COM	Rd	Ones Complement	Rd = $FF - Rd	Z,C,N,V	1
NEG	Rd	Twos Complement	Rd = $00 - Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd = Rd v K	Z,N,V	1

CBR	Rd,K	Clear Bit(s) in Register	Rd = Rd · ($FFh - K)	Z,N,V	1
INC	Rd	Increment	Rd = Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd = Rd - 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd = Rd · Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd = Rd Å Rd	Z,N,V	1
SER	Rd	Set Register	Rd = $FF	None	1
ADIW Adiw r24, K6	Rdl, K6	Add Immediate to Word	Rdh:Rdl = Rdh: Rdl + K	Z,C,N,V, S	2
SBIW Sbiw R24,K6	Rdl, K6	Subtract Immediate from Word	Rdh:Rdl = Rdh: Rdl - K	Z,C,N,V, S	2
MUL	Rd,Rr	Multiply Unsigned	R1, R0 = Rd * Rr	C	2 *
BRANCH INSTRUCTIONS					
RJMP	K	Relative Jump	PC = PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC = Z	None	2
JMP	K	Jump	PC = k	None	3
RCALL	K	Relative Call Subroutine	PC = PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC = Z	None	3
CALL	K	Call Subroutine	PC = k	None	4
RET		Subroutine Return	PC = STACK	None	4
RETI		Interrupt Return	PC = STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC = PC + 2 or 3	None	1 / 2
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V, H,	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V, H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V, H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	If (Rr(b)=0) PC = PC + 2 or 3	None	1 / 2
SBRS	Rr, b	Skip if Bit in Register Set	If (Rr(b)=1) PC = PC + 2 or 3	None	1 / 2
SBIC	P, b	Skip if Bit in I/O Register Cleared	If(I/O(P,b)=0) PC = PC + 2 or 3	None	2 / 3
SBIS	P, b	Skip if Bit in I/O Register Set	If(I/O(P,b)=1) PC = PC + 2 or 3	None	2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC=PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC=PC+k + 1	None	1 / 2
BREQ	K	Branch if Equal	if (Z = 1) then PC = PC + k + 1	None	1 / 2
BRNE	K	Branch if Not Equal	if (Z = 0) then PC = PC + k + 1	None	1 / 2
BRCS	K	Branch if Carry Set	if (C = 1) then PC	None	1 / 2

			= PC + k + 1		
BRCC	K	Branch if Carry Cleared	if (C = 0) then PC = PC + k + 1	None	1 / 2
BRSH	K	Branch if Same or Higher	if (C = 0) then PC = PC + k + 1	None	1 / 2
BRLO	K	Branch if Lower	if (C = 1) then PC = PC + k + 1	None	1 / 2
BRMI	K	Branch if Minus	if (N = 1) then PC = PC + k + 1	None	1 / 2
BRPL	K	Branch if Plus	if (N = 0) then PC = PC + k + 1	None	1 / 2
BRGE	K	Branch if Greater or Equal, Signed	if (N V= 0) then PC = PC+ k + 1	None	1 / 2
BRLT	K	Branch if Less Than, Signed	if (N V= 1) then PC = PC + k + 1	None	1 / 2
BRHS	K	Branch if Half Carry Flag Set	if (H = 1) then PC = PC + k + 1	None	1 / 2
BRHC	K	Branch if Half Carry Flag Cleared	if (H = 0) then PC = PC + k + 1	None	1 / 2
BRTS	K	Branch if T Flag Set	if (T = 1) then PC = PC + k + 1	None	1 / 2
BRTC	K	Branch if T Flag Cleared	if (T = 0) then PC = PC + k + 1	None	1 / 2
BRVS	K	Branch if Overflow Flag is Set	if (V = 1) then PC = PC + k + 1	None	1 / 2
BRVC	K	Branch if Overflow Flag is Cleared	if (V = 0) then PC = PC + k + 1	None	1 / 2
BRIE	K	Branch if Interrupt Enabled	if (I = 1) then PC = PC + k + 1	None	1 / 2
BRID	K	Branch if Interrupt Disabled	if (I = 0) then PC = PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Copy Register	Rd = Rr	None	1
LDI	Rd, K	Load Immediate	Rd = K	None	1
LDS	Rd, k	Load Direct	Rd = (k)	None	2
LD	Rd, X	Load Indirect	Rd = (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd = (X), X = X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X = X - 1, Rd = (X)	None	2
LD	Rd, Y	Load Indirect	Rd = (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd = (Y), Y = Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y = Y - 1, Rd = (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd = (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd = (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd = (Z), Z = Z+1	None	2

LD	Rd, -Z	Load Indirect and Pre-Decrement	Z = Z - 1, Rd = (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd = (Z + q)	None	2
STS	k, Rr	Store Direct	(k) = Rr	None	2
ST	X, Rr	Store Indirect	(X) = Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) = Rr, X = X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	X = X - 1, (X) = Rr	None	2
ST	Y, Rr	Store Indirect	(Y) = Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) = Rr, Y = Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y = Y - 1, (Y) = Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) = Rr	None	2
ST	Z, Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) = Rr, Z = Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z = Z - 1, (Z) = Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) = Rr	None	2
LPM		Load Program Memory	R0 =(Z)	None	3
IN	Rd, P	In Port	Rd = P	None	1
OUT	P, Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push Register on Stack	STACK = Rr	None	2
POP	Rd	Pop Register from Stack	Rd = STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
LSL	Rd	Logical Shift Left	Rd(n+1) =Rd(n), Rd(0)= 0,C=Rd (7)	Z,C,N,V, H	1
LSR	Rd	Logical Shift Right	Rd(n) = Rd(n+1), Rd(7) =0, C=Rd (0)	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) =C, Rd (n+1) =Rd(n), C=Rd(7)	Z,C,N,V, H	1
ROR	Rd	Rotate Right Through Carry	Rd(7) =C,Rd(n) =Rd(n+1),C¬Rd (0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) = Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) « Rd (7..4)	None	1
BSET	S	Flag Set	SREG(s) = 1	SREG(s)	1
BCLR	S	Flag Clear	SREG(s) = 0	SREG(s)	1
SBI	P, b	Set Bit in I/O Register	I/O(P, b) = 1	None	2

CBI	P, b	Clear Bit in I/O Register	I/O(P, b) = 0	None	2
BST	Rr, b	Bit Store from Register to T	T = Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) = T	None	1
SEC		Set Carry	C = 1	C	1
CLC		Clear Carry	C = 0	C	1
SEN		Set Negative Flag	N = 1	N	1
CLN		Clear Negative Flag	N = 0	N	1
SEZ		Set Zero Flag	Z = 1	Z	1
CLZ		Clear Zero Flag	Z = 0	Z	1
SEI		Global Interrupt Enable	I = 1	I	1
CLI		Global Interrupt Disable	I = 0	I	1
SES		Set Signed Test Flag	S = 1	S	1
CLS		Clear Signed Test Flag	S = 0	S	1
SEV		Set Twos Complement Overflow	V = 1	V	1
CLV		Clear Twos Complement Overflow	V = 0	V	1
SET		Set T in SREG	T = 1	T	1
CLT		Clear T in SREG	T = 0	T	1
SHE		Set Half Carry Flag in SREG	H = 1	H	1
CLH		Clear Half Carry Flag in SREG	H = 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1
		XMEGA ONLY			
LAC		Load and clear RAM loc		None	2
LAT		Load and toggle RAM loc		None	2
LAS		Load and set RAM loc		None	2
XCH		Exchange RAM loc		None	2

*) Not available in base-line microcontrollers

The Assembler is not case sensitive.
The operands have the following forms:

Rd: R0-R31 or R16-R31 (depending on instruction)
Rr: R0-R31
b: Constant (0-7)
s: Constant (0-7)
P: Constant (0-31/63)
K: Constant (0-255)
k: Constant, value range depending on instruction.
q: Constant (0-63)

Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions

6.5 Reserved Words

See Keyword Reference 29

Additional, there are also these reserved words :
LBYTE , HBYTE, TYPE

6.6 Error Codes

The following table lists errors that can occur.

Error	Description
1	Unknown statement
2	Unknown structure EXIT statement
3	WHILE expected
4	No more space for IRAM BIT
5	No more space for BIT
6	. expected in filename
7	IF THEN expected
8	BASIC source file not found
9	Maximum 128 aliases allowed
10	Unknown LCD type
11	INPUT, OUTPUT, 0 or 1 expected
12	Unknown CONFIG parameter
13	CONST already specified
14	Only IRAM bytes supported
15	Wrong data type
16	Unknown Definition
17	9 parameters expected
18	BIT only allowed with IRAM or SRAM
19	STRING length expected (DIM S AS STRING * 12 ,for example)
20	Unknown DATA TYPE
21	Out of IRAM space
22	Out of SRAM space
23	Out of XRAM space
24	Out of EPROM space
25	Variable already dimensioned
26	AS expected
27	parameter expected
28	IF THEN expected
29	SELECT CASE expected
30	BIT's are GLOBAL and can not be erased
31	Invalid data type
32	Variable not dimensioned
33	GLOBAL variable can not be ERASED
34	Invalid number of parameters
35	3 parameters expected

36	THEN expected
37	Invalid comparison operator
38	Operation not possible on BITS
39	FOR expected
40	Variable can not be used with RESET
41	Variable can not be used with SET
42	Numeric parameter expected
43	File not found
44	2 variables expected
45	DO expected
46	Assignment error
47	UNTIL expected
50	Value doesn't fit into INTEGER
51	Value doesn't fit into WORD
52	Value doesn't fit into LONG
60	Duplicate label
61	Label not found
62	SUB or FUNCTION expected first
63	Integer or Long expected for ABS()
64	, expected
65	device was not OPEN
66	device already OPENED
68	channel expected
70	BAUD rate not possible
71	Different parameter type passed then declared
72	Getclass error. This is an internal error.
73	Printing this FUNCTION not yet supported
74	3 parameters expected
80	Code does not fit into target chip
81	Use HEX(var) instead of PRINTHEX
82	Use HEX(var) instead of LCDHEX
85	Unknown interrupt source
86	Invalid parameter for TIMER configuration
87	ALIAS already used
88	0 or 1 expected
89	Out of range : must be 1-4
90	Address out of bounds
91	INPUT, OUTPUT, BINARY, or RANDOM expected
92	LEFT or RIGHT expected
93	Variable not dimensioned
94	Too many bits specified
95	Falling or rising expected for edge
96	Pre scale value must be 1,8,64,256 or 1024
97	SUB or FUNCTION must be DECLARED first
98	SET or RESET expected
99	TYPE expected
100	No array support for IRAM variables
101	Can't find HW-register

102	Error in internal routine
103	= expected
104	LoadReg error
105	StoreBit error
106	Unknown register
107	LoadnumValue error
108	Unknown directive in device file
109	= expected in include file for .EQU
110	Include file not found
111	SUB or FUNCTION not DECLARED
112	SUB/FUNCTION name expected
113	SUB/FUNCTION already DECLARED
114	LOCAL only allowed in SUB or FUNCTION
115	#channel expected
116	Invalid register file
117	Unknown interrupt
126	NEXT expected.
129	(or) missing.
200	.DEF not found
201	Low Pointer register expected
202	.EQU not found, probably using functions that are not supported by the selected chip
203	Error in LD or LDD statement
204	Error in ST or STD statement
205	} expected
206	Library file not found
207	Library file already registered
210	Bit definition not found
211	External routine not found
212	LOW LEVEL, RISING or FALLING expected
213	String expected for assignment
214	Size of XRAM string 0
215	Unknown ASM mnemonic
216	CONST not defined
217	No arrays allowed with BIT/BOOLEAN data type
218	Register must be in range from R16-R31
219	INT0-INT3 are always low level triggered in the MEGA
220	Forward jump out of range
221	Backward jump out of range
222	Illegal character
223	* expected
224	Index out of range
225	() may not be used with constants
226	Numeric of string constant expected
227	SRAM start greater than SRAM end
228	DATA line must be placed after the END statement
229	End Sub or End Function expected
230	You can not write to a PIN register

231	TO expected
232	Not supported for the selected micro
233	READ only works for normal DATA lines, not for EPROM data
234	') block comment expected first
235	'(block comment expected first
236	Value does not fit into byte
238	Variable is not dimensioned as an array
239	Invalid code sequence because of AVR hardware bug
240	END FUNCTION expected
241	END SUB expected
242	Source variable does not match the target variable
243	Bit index out of range for supplied data type
244	Do not use the Y pointer
245	No arrays supported with IRAM variable
246	No more room for .DEF definitions
247	. expected
248	BYVAL should be used in declaration
249	ISR already defined
250	GOSUB expected
251	Label must be named SECTIC
252	Integer or Word expected
253	ERAM variable can not be used
254	Variable expected
255	Z or Z+ expected
256	Single expected
257	"" expected
258	SRAM string expected
259	- not allowed for a byte
260	Value larger than string length
261	Array expected
262	ON or OFF expected
263	Array index out of range
264	Use ECHO OFF and ECHO ON instead
265	offset expected in LDD or STD like Z+1
266	TIMER0, TIMER1 or TIMER2 expected
267	Numeric constant expected
268	Param must be in range from 0-3
269	END SELECT expected
270	Address already occupied
322	Data type not supported with statement
323	Label too long
324	Chip not supported by I2C slave library
325	Pre-scale value must be 1,8,32,128,256 or 1024
326	#ENDIF expected
327	Maximum size is 255
328	Not valid for SW UART
329	FileDateTime can only be assigned to a variable
330	Maximum value for OUT is &H3F

332	$END ASM expected
334	') blockcomment end expected
335	Use before DIM statements
336	Could not set specified CLOCK value
337	No more space for labels
338	AS expected
339	Bytes to read may not be 0.
340	Variable is used as CONSTANT
341	OFFSET Error, contact MCS
342	OFFSET not allowed, too many locals used
343	Variable not supported with this function/statement
344	Program will overwrite bootloader
345	UART not available for the selected micro
346	External interrupt not supported or no settings found in DAT file
347	External interrupt mode not supported or found in DAT file
349	Setting not supported or not found in DAT file
350	Interrupt needs return
351	Not supported yet.
352	ALIAS can not be CONST or DIMMED variable
353	Reserved word may not be used
354	Previous Macro definition must be ended first
355	Macro previously defined
356	String constant size exceeded
357	Too many constants, increase resource languages
358	.DEF error, already defined
359	Operation not allowed on register
360	PRESCALE can not be used in COUNTER mode
361	Member expected
362	SBIC or SBIS was used followed by IN, OUT, SBIC, SBIS, SBI or CBI that also need to be converted.
363	No more room for EPROM DATA Index
364	Name not allowed, is used by constant/variable
365	Function not allowed in PRINT
366	Bit value out or range
367	Function name not allowed
368	Name used by label
369	Duplicate label name used by const or variable
370	Out of Flash memory
371	Function not allowed
372	SE entry missing in DAT file
373	Re-Configuration not allowed
374	. not allowed.
375	Duplicate definition
376	Config not found
377	Unexpected non numeric characters found
378	CAN BAUD not possible
379	Syntax error
380	Array<>Non Array mismatch

381	CONFIG RC5 not found
382	variable does not match FOR
383	Register range must be within [R16-R23]
384	Register range must be within [R16-R31]
385	Register must be even within [R0-R30]
386	Register R0 expected
387	IO address must be in range [0-31]
388	Bit number must be in range [0-7]
389	Constant out of range [0-65535]
390	Float not allowed for index
391	JTAG can not be disabled
392	Invalid operator
393	UART is fixed
394	Unsupported data type for BYREG
395	Index out of range
396	Delay not possible with selected frequency. Use WAITMS
397	.ORG exceeds PC
398	Single or Double expected
399	ASM reserved word not allowed
400	No structure found for REDO
401	No structure found for CONTINUE
402	MULTI-DIM not supported for specified page address
403	Integer Numeric constant expected
404	Mode not possible
405	SRAM DAT FILE ERROR
406	Insufficient $FRAMESPACE
407	Channel not defined
408	Invalid channel
409	END TYPE expected
410	TYPE expected
411	MEMBER already exists
500	XTINY LICENSE Required
998,999	DEMO/BETA only supports 4096 bytes of code
9999	Illegal version. Please remove this illegal crack. You will not always get this message or error. When bascom finds traces of an illegal version it will generated random bugs in your code which are hard to find. It can also show this error. Only download software from the mcselec.com server.

Other error codes are internal ones. Please report them to support@ when you encounter them.

The Code explorer can give different errors. Here is a table with errors and how you can modify your code.

Config Lcd = 16 * 2	Config Lcd = 16X2	Change the * into an X
Cursor Off Noblink	Cursor Off , Noblink	Add a comma

6.7 Newbie problems

When you are using the AVR like ATTINY, ATMEGA, ATXMEGA without knowledge of the architecture you can experience some problems as a Newbie.
Regarding XMEGA see also ATXMEGA 425

 As a newbie always use stack and framesize (until you know what you do) !

```
$hwstack = 24
$swstack = 10
$framesize = 30
```

 When you encounter problems always try to increase the values behind the stack's and framesize and test the program again.

If you want to learn more about hwstack, swstack and framesize start with Memory usage 267

 Do not include too much in Interrupt Service Routines (ISR). Keep the ISR as short as possible !
Avoid something like print function in ISR (temporarily for debugging this is OK).

See also Language Fundamentals 560

FAQ:

Question: What can I use as the first "Hello World" Bascom-AVR program ?
Answer: Following a "Hello World" example:

```
$regfile = "m16def.dat"                                    ' specify the used
AVR
$crystal = 8000000                                         ' used crystal
frequency
$hwstack = 32                                              ' default use 32
for the hardware stack
$swstack = 10                                              ' default use 10
for the SW stack
$framesize = 40                                            ' default use 40
for the frame space

$baud = 19200                                              ' use baud rate
19200 baud

Do
```

```
  Print "Hello World"                                    ' Print Hello
World
  Waitms 1000                                           ' Wait 1000ms = 1
second
Loop

End                                                     ' end program
```

With ATTINY and ATMEGA you need to check if the fuse bits are set correct for the 8MHz (for this example). Some chips will be shipped by the manufacturer (Atmel) with 1MHz frequency fuse bit settings.

If you want to change the UART Interface (like stopbits) use this here in addition to **$baud.**
(Dummy is used because the baudrate is already configured with **$baud** = 19200)

```
Config Com1 = Dummy, Synchrone = 0, Parity = None, Stopbits = 1,
Databits = 8, Clockpol = 0
```

Q: How can I program (flash) the AVR with Bascom ?
A: You can use an external programmer. See Supported Programmers 164 (For ATTINY you need to use an external hardware programmer)
You can also use the MCS bootloader MCS Bootloader 184 (ATMEGA or ATXMEGA)
See also Application Note: 143
http://www.mcselec.com/index.php?option=com_content&task=view&id=159&Itemid=57

Instead of using the BASCOM-AVR build in programmer you can also use our stand alone Bootloader application (for Windows):
http://www.mcselec.com/index.php?option=com_docman&task=doc_download&gid=153&Itemid=54

Q: I'm using an Arduino hardware with Bascom-AVR. How can I program it ?
A: See ARDUINO 197

Q: I can not set a pin high or low ? I can not read the input on a pin ?
A: The AVR has 3 registers for each port. A port normally consists of 8 pins. A port is named with a letter from A-F (ATMEGA) and even more with ATXMEGA. All parts have PORTB. When you want to set a single pin high or low you can use the SET and RESET statements. But before you use them the AVR chip must know in which direction you are going to use the pins.

Therefore there is a register named **DDRx** for each port. In our sample it is named DDRB.
When you write a 0 to the bit position of the pin you can use the pin as an input.
When you write a 1 you can use it as output.
You can also use `CONFIG PORTX.Y = INPUT|OUTPUT`
After the direction bit is set you must use either the **PORTx** register to set a logic level or the **PINx** register to READ a pin level.
Yes the third register is the PINx register. In our sample, PINB.

For example we like to use PORTB.7 as an OUTPUT pin:

```
CONFIG PORTB.7=OUTPUT          ' will write a '1' to DDRB.7
SET PORTB.7                ' will set the MS bit to +5V
RESET PORTB.7              ' will set MS bit to 0 V
```

When using a PIN in INPUT mode, you can also activate an internal pull up resistor.
Pull up means that the pin is connected with an internal resistor to VCC.
To enable the pull up resistor, you need to write a '1' to the PORT register.

Example to read PORTB.0 pin :
```
CONFIG PORTB.0=INPUT          ' clears DDRB.0
PORTB.0=1                     ' activate pull up
Print PINB.0                  ' will read LS bit and send it to the
RS-232
```

You may also read from **PORTx** but it will return the value that was last written to it and not the input of the pin.

To read or write whole bytes use :
```
PORTB = 0                     ' write 0 to register making all pins low
PRINT PINB              ' print input on pins
```

Config a Pin as output:
```
Config Porte.0 = Output
```

which is the same as:
```
DDRE = &B00000001
```

or can be written as:
```
set DDRE.0
```

Set Output:
```
Set porte.0
```

which is the same as:
```
porte.0 = 1
```

Reset Output:
```
Reset porte.0
```

which is the same as:
```
porte.0 = 0
```

Config a Pin as Input:
```
Config Pine.0 = Input
```

which is the same as:
```
DDRE.0 = 0
```

or can be written as:
```
DDRE = &B00000000
```

Read Input:
```
Variabel = PINE.0
```

To check one pin for status in an if .... statement:

```
If Pine.0 = 1 Then
' do someting....
End If
```

Q: I want to write a special character but they are not printed correct ?
A: Well this is not a newbie problem but I put it here so you could find it.
Some ASCII characters above 127 are interpreted wrong depending on country settings. To print the right value use : `PRINT "Test{123}?"`

The {xxx} will be replaced with the correct ASCII character.

You must use **3** digits otherwise the compiler will think you want to print {12} for example. This should be {012}

Q: My application was working but with a new micro it is slow and print funny ?
A: Most new micro's have an internal oscillator that is enabled by default. As it runs on 1 or 2 or 4 or 8 or 32 MHz, this might be slower or faster then your external or internal crystal. This results in slow operation.
As the baud rate is derived from the clock, it will also result in wrong baud rates.

Solution : change frequency with **$crystal** so the internal clock will be used.
Or change the fuse bits (or change config with XMEGA) so correct clock source like external xtal will be used.

Q: Some bits on Port C are not working ?
A: Some chips have a JTAG interface. Disable it with the proper fuse bit . Or use DISABLE JTAG in your code.

Q: Can I use an ATTINY or ATMEGA as TWI/I2C Slave ?
A: Yes, there is a commercial add on Bascom library available
Here the link: I2CSLAVE Library (Download version)
See also: CONFIG TWISLAVE 1123

Q: What is Overlay ?
A: See DIM 1228

Q: Is there a way to use a buffer with software UART ?
A: No, this is not supported.

Q: I have an ATTINY without UART or I need an additional UART on ATMEGA. Is there a "Software UART" in Bascom-AVR ?
A: See Using the UART 278 and scroll down to SOFTWARE UART

Q: How can I start with ATXMETGA and Bascom-AVR ?
A: See ATXMEGA 425

Q: How to declare a subroutine or function ?
A: See DECLARE SUB 1221 or DECLARE FUNCTION 1215

Q: I have a number like 1234.888999 but I just want to have one digit after decimal point (1234.8). How can I do that ?
A: See CONFIG SINGLE 1059

Q: How can I set or reset single bits in byte/integer/long variables ?
A: There are several ways to write or read a single bit:

1. You can use NBITS 1377 or BITS 804 to set or reset one or more bits

2. You can use it following way:
Example on how to set/reset single bits in a variable.

```
Dim my_long_var As Long
My_long_var.0 = 1
```

You can also use SET or RESET

```
Set My_long_var.31
Reset My_long_var.31
```

You even can use a variable as index

```
Dim Idx As Byte
Idx = 3
Reset My_long_var.idx
```

For a long variable Idx can be from 0.....31
For an integer Idx can be from 0....15
For a byte Idx can be from 0.....7

3. You can use BITWAIT to wait until a bit is set (1) or reset (0).

Example:

```
Dim A As Bit
Bitwait A , Set                                 ' wait until bit a is
set
'the above will never continue because it is not set i software
'it could be set in an ISR routine

Bitwait Pinb.7 , Reset                          ' wait until bit 7 of
Port B is 0.
```

4. You can use TOGGLE 1595 to invert the state of a bit

```
Dim my_long_var As Long
Toggle My_long_var.31
```

Q: Can I create BIT ARRAYS larger then a LONG variable ?
A: Yes, here is a way to do it:

```
$regfile = "m162def.dat"                                    ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Dim Byte_arr(32) As Byte
Dim Idx As Byte

Byte_arr(1).8 = 1
Print "Byte_arr(2) = "  ; Bin(byte_arr(2))

Byte_arr(1).15 = 1
Print "Byte_arr(2) = "  ; Bin(byte_arr(2))

Byte_arr(1).29 = 1
Print "Byte_arr(4) = "  ; Bin(byte_arr(4))

Idx = 63
Byte_arr(1).idx = 1
Print "Byte_arr(8) = "  ; Bin(byte_arr(8))

Idx = 255
Byte_arr(1).idx = 1
Print "Byte_arr(32) = " ; Bin(byte_arr(32))
```

```
'(
Bascom Simulator Output =

Byte_arr(2)  = 00000001
Byte_arr(2)  = 10000001
Byte_arr(4)  = 00100000
Byte_arr(8)  = 10000000
Byte_arr(32) = 10000000

')

End
```

Q: Can I pass a BIT variable to SUB routines or user FUNCTION ?
A: You can **not** pass BIT variables to SUB routines or user FUNCTION
Use a BYTE for that.

Here is one of many workarounds for that:

```
$regfile = "m162def.dat"                                    ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Submode = New

Dim C As Byte

Sub Test(byref B As Byte)
    Local C As Byte
    C = B And &B00000001

    If C = 1 Then
          Print "B = 1"
    Else
          Print "B = 0"
    End If
End Sub

'Main program

   Set C.0
   Call Test(c)

   Reset C.0
   Call Test(c)

End                                                         ' end
program
```

Q: Can I dimension a LOCAL variable in a function or sub routine as BIT ?
A: You can **not.** BIT variables are not possible because they are GLOBAL to the system.

Q: I still have a problem. What to do ?

A: Here is the link to the Bascom-AVR forum: http://www.mcselec.com/index2.php?option=com_forum&Itemid=59

At first please try to search the forum (often you can find users with the same problem) . The search page is here:
http://www.mcselec.com/index2.php?option=com_forum&Itemid=59&page=search

If the forum can not help you, here is the Email address for support:
support@mcselec.com

PLEASE provide as much as possible information in your post or Email:

- Include the Bascom-AVR version number and your serial number in the **Email** to support

Do not post your serial number in the Forum !!!

- Always test with the latest available version, support is only available for the latest version
- Include a small sample that will demonstrate the error.
- Make sure you include all required files for compilation or for showing the problem.
- Be clear if the problem exist in the simulator or the hardware and what kind of hardware you use

6.8 Tips and tricks

Tips & Tricks:

1. You can specify a binary number with the &B and you can use underscore "_" like:

```
Dim Var As Byte

Var = &B00_110000
Var = &B0000_1111
Var = &B00_11_00_11
```

2. How to use longer formulas:

```
Dim A As Byte
Dim B As Byte
Dim C As Byte

' Now you want to use following formula: a = B / 4 + C
' In Bascom you write

A = B / 4
A = A + C
```

3. You can use more than one Bascom statement in one line with colons ":"

```
Dim A As Byte
Dim B As Byte
Dim C As Byte

' Now you want to use following formula: a = B / 4 + C
' In Bascom you write

A = B / 4 : A = A + C
```

4. You can use overlay to have easy access to the low byte and high byte of a WORD (the same approach also work for e.g. LONG)

```
Dim My_word As Word
Dim Low_byte As Byte At My_word Overlay
Dim High_byte As Byte At My_word + 1 Overlay

Low_byte  = &B0000_1111
High_byte = &B1111_0000

' This is how it will be stored in SRAM
' <-------my_word-------->
' +-----------+----------+
' | Low_byte  |High_byte |
' +-----------+----------+
```

5. To split a word into High byte and Low byte you can also use HIGH and LOW

6. Here is a way to print the content of a variable or AVR register:

Use **Print Bin(X)**

Example:

```
$regfile = "m88def.dat"                                    ' we use the
M88
$crystal = 8000000
$baud = 19200
$hwstack = 32
$swstack = 8
$framesize = 24


Dim A As Byte

A = &B00000001
A = A * 2
Print Bin(a)

End                                                        ' end
program
```

7. If you do not want that Bascom-AVR is sending Carriage + Return after a print command use semi-colon ";" after the print funtion:

Example:

```
$regfile = "m88def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Print "Hello World" ;

End
```

8. For the user who want to use external editors:

The **bascomp.exe** has been updated. It can be downloaded from the download section.

It now supports a simpler way to be called.

The utility has been updated and now will retrieve all info from the source file, but only when your main program contains these directive :
$regfile, $hwstack, $swstack, $framesize

Example :
bascomp.exe "c:\my folder\source\sample.bas" auto

The 'auto' is a switch so the utility will retrieve the settings from your code.

9. You can use $initmicro if you want to run special tasks at startup:

See $INITMICRO 656

10. You can use $include to make larger projects better readable: See $INCLUDE 654

11. Your LCD is not working and you need a list of steps what do check:

a. Check fuse bit settings
b. Are the AVR pins are OK ?

To test the AVR pins you can do following:
Write a program that toggles all the lcd pins and then measure the logic level. Then check with a DVM or led-series resistor if all pins change level. if they do, there is a problem with the lcd

If the pin do not toggle:
- pin defect
- track or solder problem.

Here the test program:

```
$regfile = "m328pdef.dat"                                   ' Specify
The Used Micro
$crystal = 16000000                                         ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for thehardware stack
$swstack = 10                                               ' default
use 10 for theSW stack
$framesize = 40                                             ' default
use 40 for theframe space

Config Clockdiv = 1                                         ' divide
xtal clock by 1, default fuse bit is set
                                          ' to 8 by elektor

Config Portc.3 = Output                                     ' RW
Config Portd.4 = Output                                     ' Db4
Config Portd.5 = Output                                     ' Db5
Config Portd.6 = Output                                     ' Db6
Config Portd.6 = Output                                     ' Db7
Config Portc.1 = Output                                     ' E
Config Portc.2 = Output                                     ' RS

do
   toggle portc
   toggle portd
   waitms 1000
Loop

End                                                         ' end
program
```

12. With the Lib Manager you can compile a Library (*.lib) into an *.lbx file.

See here: Tools LIB Manager 132

13. There is a timeout function for hardware and software UART

See $TIMEOUT 708

14. How to use the Powerdown function:

See also: CONFIG POWERMODE 1017

If you can not measure the same power down current as written in the data sheet you also need to use a
Low Quiescent Current LDO Regulator to meet that specs (if you measure the current including the Current LDO Regulator).

Examples for 3.3Volt Low Quiescent Current LDO Regulator :

- MCP1702 --> typical 2µA
- MCP1700 --> typical 1.6µA
- AS1375 low power LDO --> 1µA (typ.) of quiescent current
- TPS78233 3,3V --> 0.4µA

```
' Using the new config powermode = PowerDown function with ATTINY13

' Used Bascom-AVR Version 2.0.7.3

' Fuse Bits:
' Disable DWEN (Debug Wire) Fuse Bit
' Disable Brown-Out Detection in Fuse Bits
' Disable Watchdog in Fuse Bits

' You can also just use  Config Powermode = Powerdown

' But this example here also considers what the data sheet write under
"MINIMIZING POWER CONSUMPTION"
' You need to follow this when you want to achieve the current
consumption which you find in the
' data sheet under Powerdown Mode.

' 1. Disable/Switch off ADC
' 2. Disable/Switch off Analog Comparator
' 3. Disable Brown-out Detection when not needed
' 4. Disable internal voltage reference
' 5. Disable Watchdog Timer when not needed
' 6. Disable the digital input buffer
' 7. Enable Pull-up or pull-down an all unused pins

$regfile = "attiny13.dat"
$crystal = 9600000                                          ' 9.6MHz
$hwstack = 10
$swstack = 0
$framesize = 24

On Int0 Int0_isr                                            ' INT0 will be the
wake-up source for Powerdown Mode
Config Int0 = Low Level
Enable Int0

' Prepare Powerdown:
' To minimize power consumption, enable pull-up or -down on all unused
pins, and
' disable the digital input buffer on pins that are connected to analog
sources
Config Portb.0 = Input
Set Portb.0
Config Portb.1 = Input                                      ' INT0 -->
external 47K pull-up
'Set Portb.1
Config Portb.2 = Input
Set Portb.2
Config Portb.3 = Input
Set Portb.3
Config Portb.4 = Input
Set Portb.4
```

```
Config Portb.5 = Input                                  ' External Pull-Up
(Reset)

Didr0 = Bits(ain1d , Ain0d)                             ' Disable digital
input buffer on the AIN1/0 pin

Set Acsr.acd                                            ' Switch off  the
power to the Analog Comparator
' alternative:
' Stop Ac

Reset Acsr.acbg                                         ' Disable Analog
Comparator Bandgap Select

Reset Adcsra.aden                                       ' Switch off ADC
' alternative:
' Stop Adc

'#########################################################################
########
Do
      Wait 3                                            ' now we have 3
second to measure the Supply Current
                                                        ' in Active Mode
      Enable Interrupts
      ' Now call Powerdown function
      Config Powermode = Powerdown
      ' Here you have time to measure PowerDown current consumption
until a Low Level
      ' on Portb.1 which is the PowerDown wake-up
Loop
'#########################################################################
########
End

Int0_isr:
' wake_up
Return
```

6.9 ASCII chart

Decimal	Octal	Hex	Binary	Value	
000	000	000	00000000	NUL	(Null char)
001	001	001	00000001	SOH	(Start of Header)
002	002	002	00000010	STX	(Start of Text)
003	003	003	00000011	ETX	(End of Text)
004	004	004	00000100	EOT	(End of Transmission)
005	005	005	00000101	ENQ	(Enquiry)
006	006	006	00000110	ACK	(Acknowledgment)
007	007	007	00000111	BEL	(Bell)
008	010	008	00001000	BS	(Backspace)
009	011	009	00001001	HT	(Horizontal Tab)
010	012	00A	00001010	LF	(Line Feed)
011	013	00B	00001011	VT	(Vertical Tab)
012	014	00C	00001100	FF	(Form Feed)
013	015	00D	00001101	CR	(Carriage Return)
014	016	00E	00001110	SO	(Shift Out)
015	017	00F	00001111	SI	(Shift In)
016	020	010	00010000	DLE	(Data Link Escape)
017	021	011	00010001	DC1	(XON) (Device Control 1)

018	022	012	00010010	DC2	(Device Control 2)
019	023	013	00010011	DC3	(XOFF)(Device Control 3)
020	024	014	00010100	DC4	(Device Control 4)
021	025	015	00010101	NAK	(Negative Acknowledgement)
022	026	016	00010110	SYN	(Synchronous Idle)
023	027	017	00010111	ETB	(End of Trans. Block)
024	030	018	00011000	CAN	(Cancel)
025	031	019	00011001	EM	(End of Medium)
026	032	01A	00011010	SUB	(Substitute)
027	033	01B	00011011	ESC	(Escape)
028	034	01C	00011100	FS	(File Separator)
029	035	01D	00011101	GS	(Group Separator)
030	036	01E	00011110	RS	(Request to Send)(Record Separator)
031	037	01F	00011111	US	(Unit Separator)
032	040	020	00100000	SP	(Space)
033	041	021	00100001	!	(exclamation mark)
034	042	022	00100010	"	(double quote)
035	043	023	00100011	#	(number sign)
036	044	024	00100100	$	(dollar sign)
037	045	025	00100101	%	(percent)
038	046	026	00100110	&	(ampersand)
039	047	027	00100111	'	(single quote)
040	050	028	00101000	(	(left/opening parenthesis)
041	051	029	00101001	)	(right/closing parenthesis)
042	052	02A	00101010	*	(asterisk)
043	053	02B	00101011	+	(plus)
044	054	02C	00101100	,	(comma)
045	055	02D	00101101	-	(minus or dash)
046	056	02E	00101110	.	(dot)
047	057	02F	00101111	/	(forward slash)
048	060	030	00110000	0	
049	061	031	00110001	1	
050	062	032	00110010	2	
051	063	033	00110011	3	
052	064	034	00110100	4	
053	065	035	00110101	5	
054	066	036	00110110	6	
055	067	037	00110111	7	
056	070	038	00111000	8	
057	071	039	00111001	9	
058	072	03A	00111010	:	(colon)
059	073	03B	00111011	;	(semi-colon)
060	074	03C	00111100	<	(less than)
061	075	03D	00111101	=	(equal sign)
062	076	03E	00111110	>	(greater than)
063	077	03F	00111111	?	(question mark)
064	100	040	01000000	@	(AT symbol)
065	101	041	01000001	A	
066	102	042	01000010	B	
067	103	043	01000011	C	
068	104	044	01000100	D	
069	105	045	01000101	E	
070	106	046	01000110	F	
071	107	047	01000111	G	
072	110	048	01001000	H	
073	111	049	01001001	I	
074	112	04A	01001010	J	

075	113	04B	01001011	K	
076	114	04C	01001100	L	
077	115	04D	01001101	M	
078	116	04E	01001110	N	
079	117	04F	01001111	O	
080	120	050	01010000	P	
081	121	051	01010001	Q	
082	122	052	01010010	R	
083	123	053	01010011	S	
084	124	054	01010100	T	
085	125	055	01010101	U	
086	126	056	01010110	V	
087	127	057	01010111	W	
088	130	058	01011000	X	
089	131	059	01011001	Y	
090	132	05A	01011010	Z	
091	133	05B	01011011	[	(left/opening bracket)
092	134	05C	01011100	\	(back slash)
093	135	05D	01011101	]	(right/closing bracket)
094	136	05E	01011110	^	(caret/circumflex)
095	137	05F	01011111	_	(underscore)
096	140	060	01100000	`	
097	141	061	01100001	a	
098	142	062	01100010	b	
099	143	063	01100011	c	
100	144	064	01100100	d	
101	145	065	01100101	e	
102	146	066	01100110	f	
103	147	067	01100111	g	
104	150	068	01101000	h	
105	151	069	01101001	i	
106	152	06A	01101010	j	
107	153	06B	01101011	k	
108	154	06C	01101100	l	
109	155	06D	01101101	m	
110	156	06E	01101110	n	
111	157	06F	01101111	o	
112	160	070	01110000	p	
113	161	071	01110001	q	
114	162	072	01110010	r	
115	163	073	01110011	s	
116	164	074	01110100	t	
117	165	075	01110101	u	
118	166	076	01110110	v	
119	167	077	01110111	w	
120	170	078	01111000	x	
121	171	079	01111001	y	
122	172	07A	01111010	z	
123	173	07B	01111011	{	(left/opening brace)
124	174	07C	01111100	|	(vertical bar)
125	175	07D	01111101	}	(right/closing brace)
126	176	07E	01111110	~	(tilde)
127	177	07F	01111111	DEL	(delete)

Part VII

7 BASCOM Language Reference

7.1 #AUTOCODE

Action
Informs the IDE that code can be maintained by the IDE.

Syntax
#AUTOCODE

CONFIG STATEMENTS

#ENDAUTOCODE

Remarks
Auto code informs the IDE that it may alter the code. A new IDE uses a property editor for the configuration. It will only update, add or delete, CONFIG statements that are enclosed in an #AUTOCODE block.
#AUTOCODE must be closed with a matching #ENDAUTOCODE

You can still use CONFIG statements in other places of your code. But the property editor will only work on the ones inside the block.
The compiler will ignore #AUTOCODE and #ENDAUTOCODE.

7.2 #IF ELSE ELSEIF ENDIF

Action
Conditional compilation directives intended for conditional compilation.

Syntax
#IF condition

#ELSEIF condition

#ELSE

#ENDIF

Remarks
Conditional compilation is supported by the compiler.
What is conditional compilation?
Conditional compilation will only compile parts of your code that meet the criteria of the condition.

By default all your code is compiled.

Conditional compilation needs a constant 1170 to test.
So before a condition can be tested you need to define a constant.

```
CONST test = 1
```

```
#IF TEST
    Print "This will be compiled"
#ELSE
    Print "And this not"
#ENDIF
```

 Note that there is no THEN and that you need to use #ENDIF which has no space between END and IF , so #END IF is wrong!

You can nest the conditions and the use of #ELSE and #ELSEIF is optional.

There are a few internal constants that you can use. These are generated by the compiler:

```
_CHIP = 0
_RAMSIZE = 128
_ERAMSIZE = 128
_SIM = 0
_XTAL = 4000000
_BUILD = 11162

_CHIP is an integer that specifies the chip, in this case the 2313
_RAMSIZE is the size of the SRAM
_ERAMSIZE is the size of the EEPROM
_SIM is set to 1 when the $SIM directive is used
_XTAL contains the value of the specified crystal
_BUILD is the build number of the compiler.
```

The build number can be used to write support for statements that are not available in a certain version :

```
#IF _BUILD >= 11162
   s = Log(1.1)
#ELSE
  Print "Sorry, implemented in 1.11.6.2"
#ENDIF
```

Conditional compilation allows you to create different versions of your program but that you keep one source file.
For example you could make a multi lingual program like this :

```
CONST LANGUAGE=1

'program goes here

#IF LANGUAGE=1
  DATA "Hello"
#ENDIF
#IF LANGUAGE=2
  DATA "Guten tag"
#ENDIF
```

By changing just one constant you can have for example English or German data lines.

Conditional compilation does work with the $REGFILE directive but you need to set the option 'Use New Method' in Environment IDE options.

VAREXIST

A special check was added to 1.11.8.1 to test for existence of dimmed/defined constants or variables.

```
#IF varexist("S")
```

```
      ' the variable S was dimensioned so we can use it here
#ELSE
      ' when it was not dimmed and we do need it, we can do it here
    DIM S as BYTE
#ENDIF
```

The Editor can show non included code with a different font color. This makes it more clear which code is included.

See Also

CONST 1170 , Edit Show Excluded Code 90

7.3 Compiler Directives

7.3.1 $AESKEY

Action

This directive accepts a 16 byte AES key and informs the compiler to encrypt the binary image.

Syntax

$AESKEY 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

Remarks

$AESKEY accepts 16 parameters. These are the 16 bytes which form a 128 bit key. When your code is compiled, the resulting binary code will be encrypted with the provided key.
A boot loader could then use AES and decrypt the binary file before writing to flash memory.

 Only the binary image is encrypted, the HEX file is not encrypted!
You can not simulate an encrypted program. Add this option when your project is ready.

See also

$XTEAKEY 715 , AESENCRYPT 1288 , AESDECRYPT 1290

Example

See the Samples\boot\xmega_dos_boot_AES.zip , an Xmega boot loader with AES decryption.

7.3.2 $ASM

Action

Start of inline assembly code block.

Syntax
$ASM

Remarks
Use $ASM together with $END ASM to insert a block of assembler code in your BASIC code. You can also precede each line with the **!** sign.

See also the chapter Mixing BASIC and Assembly 573 and assembler mnemonics 578

Example
```
Dim C As Byte

Loadadr C , X 'load address of variable C into register X

$asm
  Ldi R24,1 ; load register R24 with the constant 1
  St X,R24  ; store 1 into variable c
$end Asm
Print C
End
```

7.3.3 $BAUD

Action
Instruct the compiler to override the baud rate setting from the options menu.

Syntax
$BAUD = var

Remarks

Var	The baud rate that you want to use. This must be a numeric constant.

The baud rate is selectable from the Compiler Settings 146. It is stored in a configuration file. The $BAUD directive overrides the setting from the Compiler Settings.

In the generated report, you can view which baud rate is actually generated. The generated baud rate does depend on the used micro and crystal.

When you simulate a program you will not notice any problems when the baud rate is not set to the value you expected. In real hardware a wrong baud rate can give weird results on the terminal emulator screen. For best results use a crystal that is a multiple of the baud rate.

In the simulator you need to select the UART0-TAB to view the output of the UART0, or to send data to this UART.

See also

$CRYSTAL 625 , BAUD 1488

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200

$hwstack = 32
$swstack = 8
$framesize = 24

Config Com1 = Dummy, Synchrone = 0, Parity = None, Stopbits = 1,
Databits = 8, Clockpol = 0

Print "Hello"

'Now change the baud rate in a program
Baud = 9600
Print "Did you change the terminal emulator baud rate too?"
End
```

7.3.4 $BAUD1

Action

Instruct the compiler to set the baud rate for the second hardware UART.

Syntax

$BAUD1 = var

Remarks

Var	The baud rate that you want to use. This must be a numeric constant.

In the generated report, you can view which baud rate is actually generated.

When you simulate a program you will not notice any problems when the baud rate is not set to the value you expected. In real hardware a wrong baud rate can give weird results on the terminal emulator screen. For best results use a crystal that is a multiple of the baud rate.

Some AVR chips have 2 UARTS. For example the Mega161, Mega162, Mega103 and Mega128. There are several other's and some new chips even have 4 UARTS.

In the simulator you need to select the UART1-TAB to view the output of the UART1, or to send data to this UART.

See also

$CRYSTAL 625 , BAUD 1488 , $BAUD 607

Example

```
'-------------------------------------------------------------------------
--------
```

```
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : Mega162
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates BAUD1 directive and BAUD1
statement

'-------------------------------------------------------------------------
--------
$regfile = "M162def.dat"
$baud1 = 2400
$crystal= 14000000 ' 14 MHz crystal
$hwstack = 32
$swstack = 8
$framesize = 24
Open "COM2:" For BINARY As #1

Print #1 , "Hello"
'Now change the baud rate in a program
Baud1 = 9600                                                    '
Print #1 , "Did you change the terminal emulator baud rate too?"
Close #1
End
```

7.3.5 $BGF

Action

Includes a BASCOM Graphic File.

Syntax

$BGF "file"

Remarks

file	The file name of the BGF file to include.

Use SHOWPIC to display the BGF file. $BGF only task is to store the picture into the compressed **B**ASCOM **G**raphics **F**ormat(BGF).

See also

SHOWPIC 1355 , PSET 1348 , CONFIG GRAPHLCD 964

Example

```
'-------------------------------------------------------------------
-
'                    (c) 1994-2025 MCS Electronics
'                T6963C graphic display support demo
'-------------------------------------------------------------------
-

'The connections of the LCD used in this demo
'LCD pin                  connected to
' 1        GND            GND
```

```
 '2         GND             GND
 '3         +5V             +5V
 '4         -9V             -9V potmeter
 '5         /WR             PORTC.0
 '6         /RD             PORTC.1
 '7         /CE             PORTC.2
 '8         C/D             PORTC.3
 '9         NC              not conneted
 '10        RESET           PORTC.4
 '11-18     D0-D7            PA
 '19        FS              PORTC.5
 '20        NC              not connected

$crystal = 8000000
$regfile = "m32def.dat"
$hwstack = 40
$swstack = 40
$framesize = 40


'First we define that we use a graphic LCD

Config Graphlcd = 240x128 , Dataport = Porta , Controlport =
Portc , Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 ,
Mode = 8
'The dataport is the portname that is connected to the data lines
of the LCD
'The controlport is the portname which pins are used to control
the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40
columns

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT   to clear only the text display
' CLS GRAPH  to clear only the graphical part

Cursor Off

Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30
```

```
Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "12345678901234567890123456789012345678901234567890"

Wait 2

Cls Text
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to
turn it on
For X = 0 To 140
   Pset X , 20 , 255                                          ' set
the pixel
Next

Wait 2


'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Showpic 0 , 0 , Plaatje

Wait 2
Cls Text                                                      '
clear the text
End



'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

7.3.6 $BIGSTRINGS

Action

Instruct the compiler to use big strings.

Syntax

$BIGSTRINGS

Remarks

By default each string has a maximum length of 254 bytes. A null character is used to mark the end of a string.
When a longer string is needed, the compiler can not use bytes for passing the length. A word is needed to hold the length.
The $BIGSTRINGS directive will include the bigstrings.lbx and will handle all string routines different when parameters are passed which influence the length.

The alternative library contains modified(overloaded) routines for code not compatible with big strings.

The following string routines support $BIGSTRINGS:

ASC 819
CHARPOS 1519
CRC8 809
DELCHAR 1520
DELCHARS 1521
GET 1262
INPUT LCD , INPUT SERIAL 1493
INSERTCHAR 1522
INSTR 1526
LCASE 1527
LEFT 1528
LEN 1529
LTRIM 1529
MID 1530 function
MID 1530 statement
PUT 1402
QUOTE 1532
RIGHT 1531
RTRIM 1532
SPACE 1533
STRING 1536
TRIM 1536
UCASE 1537

See also

DIM 1228

Example

$BIGSTRINGS

7.3.7 $BOOT

Action

Instruct the compiler to include boot loader support.

Syntax

$BOOT = address

Remarks

address	The boot loader address. This is a WORD address.

Some new AVR chips have a special boot section in the upper memory of the flash.
By setting some fuse bits you can select the code size of the boot section.
The code size also determines the address of the boot loader.

With the boot loader you can reprogram the chip when a certain condition occurs.
The sample checks a pin to see if a new program must be loaded.
When the pin is low there is a jump to the boot address.

The boot code must always be located at the end of your program.
It must be written in ASM since the boot loader may not access the application flash rom. This because otherwise you could overwrite your running code!

The example is written for the M163. You can use the Upload file option of the terminal emulator to upload a new hex file. The terminal emulator must have the same baud rate as the chip. Under Options, Monitor, set the right upload speed and set a monitor delay of 20. Writing the flash take time so after every line a delay must be added while uploading a new file.

 The $BOOT directive is replaced by $LOADER. $LOADER works much simpler. $BOOT is however still supported.

See also

$LOADER 667 , $LOADERSIZE 683

Example

See BOOT.BAS from the samples dir. But better look at the $LOADER directive.

7.3.8 $BOOTVECTOR

Action

This compiler directive will force the compiler to create an interrupt vector table(IVR).

Syntax

$BOOTVECTOR

Remarks

By default an IVR is always created for normal applications. There is no good reason not to create an IVR for a normal application.
When making a boot loader application things are different. A boot loader application resides in upper flash memory inside the boot area. And when the boot loader applications runs, it has special rights so it can update the flash memory which resides in the lower flash memory.
The boot loader area size depends on the processor but is usual small. An interrupt vector table can use up to 250 bytes or more and it would be a waste of space in

many cases. So by default the $LOADER directive which is used to create a boot loader application, will not create an IVR. The downside is that when you do not have an IVR you can not use interrupts.

The $BOOTVECTOR directive will force the compiler to create an IVR when the $LOADER directive is used. This way your boot loader application will include an IVR and you can use interrupts in your code.

 The $BOOTVECTOR directive will only work when the processor has an option to move the IVR to the boot area using the IVSEL bit.

By default the interrupts are located after address 0. Address 0 is the reset vector and usually contains a jump to the real code. Behind the reset address, a table with jumps to the interrupt routines is located. That the code contains an IVR is not enough : in case of a boot loader the interrupt table must be moved to the boot area. For this purpose most processors have a register and bit to switch the IVR between the normal address 0 and the boot loader address.

In BASCOM you can use : **Config Intvectorselection = Enabled** to set the selection to the boot area.
When the boot loader application finishes, it is best to use a watchdog timeout to reset the processor so the intvector selection is set to the default address 0.
Or you can use **Config Intvectorselection = Disabled** in your main (normal) application before you enable the interrupts.

So in short you only need to add the $BOOTVECTOR directive and Config Intvectorselection = Enabled to your code. And do not forget to switch back the intvectorselection in the main application!

See also

$LOADER 667, CONFIG INTVECTORSELECTION 987, $REDUCEIVR 692

Example

```
'-----------------------------------------------------------------
-
'                          (c) 1995-2025, MCS
'                        BootEDB-IVSEL.bas
'  This Bootloader is for the BASCOM-EDB
'  VERSION 4 of the BOOTLOADER.
'                IMPORTANT :
' When changing the vector table in the boot loader you MUST
' reset the vector table in your code using :
'       Config Intvectorselection = Disabled
' otherwise your code points to the wrong table
'-----------------------------------------------------------------
-
'The loader is supported by the IDE

$prog &HFF , &HE2 , &HDF , &HF8                              '
generated. Take care that the chip supports all fuse
```

```
bytes.'-------------------------------------------------------
------
$hwstack = 40
$swstack = 40
$framesize = 40

$crystal = 8000000
$baud = 38400                                          'this
loader uses serial com
'It is VERY IMPORTANT that the baud rate matches the one of the
boot loader
'do not try to use buffered com as we can not use interrupts

'This bootloader uses buffers serial input
Config Serialin = Buffered , Size = 250

'in order to use interrupts in a bootloader, the processor must
support IVSEL
'since the vector table occupies space some processors will not
support it.
$bootvector                                            ' put
int table into bootloader section so we can use interrupts
Config Intvectorselection = Enabled                    '
enabled means that the vector table points to the boot section

'since this boot loader uses interrupts we need to activate them
but :
'AFTER the interrupt vector table is enabeld
Enable Interrupts

'$regfile = "m8def.dat"
'Const Loaderchip = 8

'$regfile = "m168def.dat"
'Const Loaderchip = 168

'$regfile = "m16def.dat"
'Const Loaderchip = 16

'$regfile = "m32def.dat"
'Const Loaderchip = 32

$regfile = "m88def.dat"
Const Loaderchip = 88

'$regfile = "m162def.dat"
'Const Loaderchip = 162

'$regfile = "m128def.dat"
```

```
'Const Loaderchip = 128

'$regfile = "m64def.dat"
'Const Loaderchip = 64

#if Loaderchip = 88
'Mega88
  $loader = $c00                                          'this
address you can find in the datasheet
  'the loader address is the same as the boot vector address
  Const Maxwordbit = 5
  Const Maxpages = 96 - 1                                 ' total
WORD pages  available for program
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 168
'Mega168
  $loader = $1c00                                         'this
address you can find in the datasheet
  'the loader address is the same as the boot vector address
  Const Maxwordbit = 6
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 32                                         '
Mega32
  $loader = $3c00                                         ' 1024
words
  Const Maxwordbit = 6                                    'Z6 is
maximum bit                                  '
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif
#if Loaderchip = 8                                          '
Mega8
  $loader = $c00                                          ' 1024
words
  Const Maxwordbit = 5                                    'Z5 is
maximum bit                                  '
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif
#if Loaderchip = 161                                        '
Mega161
  $loader = $1e00                                         ' 1024
words
```

```
  Const Maxwordbit = 6                                      'Z6 is
maximum bit                                                 '
#endif
#if Loaderchip = 162                                          '
Mega162
  $loader = $1c00                                           ' 1024
words
  Const Maxwordbit = 6                                      'Z6 is
maximum bit                                                 '
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 64                                           '
Mega64
  $loader = $7c00                                           ' 1024
words
  Const Maxwordbit = 7                                      'Z7 is
maximum bit                                                 '
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 128                                          '
Mega128
  $loader = &HFC00                                          ' 1024
words
  Const Maxwordbit = 7                                      'Z7 is
maximum bit                                                 '
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 16                                           '
Mega16
  $loader = $1c00                                           ' 1024
words
  Const Maxwordbit = 6                                      'Z6 is
maximum bit                                                 '
  Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits
= 1 , Databits = 8 , Clockpol = 0
#endif



Const Maxword =(2 ^ Maxwordbit) * 2                         '128
Const Maxwordshift = Maxwordbit + 1
Const Cdbg = 0                                              '
leave this to 0
```

```
#if Cdbg
  Print Maxword
  Print Maxwordshift
  '   Print Maxpages
#endif



'Dim the used variables
Dim Bstatus As Byte , Bretries As Byte , Bblock As Byte ,
Bblocklocal As Byte
Dim Bcsum1 As Byte , Bcsum2 As Byte , Buf(128) As Byte , Csum As
Byte
Dim J As Byte , Spmcrval As Byte                                '
self program command byte value

Dim Z As Long                                                   'this
is the Z pointer word
Dim Vl As Byte , Vh As Byte                                     '
these bytes are used for the data values
Dim Wrd As Word , Page As Word
'these vars contain the page and word address
Dim Bkind As Byte , Bstarted As Byte
'Mega 88 : 32 words, 128 pages


'in this loader we may not disable interrupts !
'Disable Interrupts                                              'we
do not use ints


'Waitms 100
'wait 100 msec sec
'We start with receiving a file. The PC must send this binary
file

'some constants used in serial com
Const Nak = &H15
Const Cack = &H06
Const Can = &H18

'we use some leds as indication in this sample , you might want
to remove it
Config Pind.7 = Output
Portd.7 = 0

$timeout = 200000                                               'we
use a timeout
```

```
'When you get LOADER errors during the upload, increase the
timeout value
'for example at 16 Mhz, use 200000

Bretries = 5                                                'we
try 5 times
Testfor123:
  #if Cdbg
    Print "Try " ; Bretries
    Print "Wait"
  #endif
  Bstatus = Waitkey()
'wait for the loader to send a byte
  #if Cdbg
    Print "Got "
  #endif

  Print Chr(bstatus);

  If Bstatus = 123 Then
'did we received value 123 ?
    Bkind = 0
'normal flash loader
    Goto Loader
  Elseif Bstatus = 124 Then                                  '
EEPROM
    Bkind = 1                                               '
EEPROM loader
    Goto Loader
  Elseif Bstatus <> 0 Then
    Decr Bretries
    If Bretries <> 0 Then Goto Testfor123                   'we
test again
  End If

  For J = 1 To 10
'this is a simple indication that we start the normal reset
vector
    Toggle Portd.7 : Waitms 100
  Next

  #if Cdbg
    Print "RESET"
  #endif
  Goto _reset
'goto the normal reset vector at address 0


  'this is the loader routine. It is a Xmodem-checksum reception
```

```
routine
Loader:
  #if Cdbg
    Print "Clear buffer"
  #endif
  Do
    Bstatus = Waitkey()
  Loop Until Bstatus = 0


  For J = 1 To 3                                              'this
is a simple indication that we start the normal reset vector
    Toggle Portd.7 : Waitms 250
  Next

  If Bkind = 0 Then
    Spmcrval = 3 : Gosub Do_spm                               '
erase  the first page
    Spmcrval = 17 : Gosub Do_spm                              '
re-enable page
  End If


  Bretries = 10
'number of retries

  Do
    Bblocklocal = 1
    Bstarted = 0                                                '
we were not started yet
    Csum = 0
'checksum is 0 when we start
    Print Chr(nak);                                             '
firt time send a nack
    Do

      Bstatus = Waitkey()
'wait for statuse byte

      Select Case Bstatus
        Case 1:                                                 '
start of heading, PC is ready to send
          Csum = 1
'checksum is 1
          Bblock = Waitkey() : Csum = Csum + Bblock       'get
block
          Bcsum1 = Waitkey() : Csum = Csum + Bcsum1       'get
checksum first byte
          For J = 1 To 128                                'get
```

```
128 bytes
            Buf(j) = Waitkey() : Csum = Csum + Buf(j)
          Next
          Bcsum2 = Waitkey()                              'get
second checksum byte
          If Bblocklocal = Bblock Then                    'are
the blocks the same?
            If Bcsum2 = Csum Then                        'is the
checksum the same?
              Gosub Writepage                           'yes go
write the page
              Print Chr(cack);
'acknowledge
              Incr Bblocklocal                          'increase
local block count
            Else                                         'no
match so send nak
              Print Chr(nak);
            End If
          Else
            Print Chr(nak);                              'blocks
do not match
          End If
        Case 4:                                             '
end of transmission , file is transmitted
          If Wrd > 0 Then                                'if
there was something left in the page
            Wrd = 0                                     'Z pointer
needs wrd to be 0
            Spmcrval = 5 : Gosub Do_spm                 'write
page
            Spmcrval = 17 : Gosub Do_spm                '
re-enable page
          End If
          Print Chr(cack);                               ' send
ack and ready

          Portd.7 = 0                                    ' simple
indication that we are finished and ok
          Waitms 20
          Goto _reset                                    ' start
new program
        Case &H18:                                          ' PC
aborts transmission
          Goto _reset                                    ' ready
        Case 123 : Exit Do                                  'was
probably still in the buffer
        Case 124 : Exit Do
        Case Else
```

```
          Exit Do                                                ' no
valid data
      End Select
    Loop
    If Bretries > 0 Then
'attempte left?
      Waitms 1000
      Decr Bretries
'decrease attempts
    Else
      Goto _reset
'reset chip
    End If
  Loop



  'write one or more pages
Writepage:
  If Bkind = 0 Then
    For J = 1 To 128 Step 2                                   'we
write 2 bytes into a page
      Vl = Buf(j) : Vh = Buf(j + 1)                           'get
Low and High bytes
      !      lds r0, {vl}
 'store them into r0 and r1 registers
      !      lds r1, {vh}
      Spmcrval = 1 : Gosub Do_spm
'write value into page at word address
      Wrd = Wrd + 2                                            '
word address increases with 2 because LS bit of Z is not used
      If Wrd = Maxword Then                                    '
page is full
        Wrd = 0                                              'Z
pointer needs wrd to be 0
        Spmcrval = 5 : Gosub Do_spm                          'write
page
        Spmcrval = 17 : Gosub Do_spm                         '
re-enable page


        If Page < Maxpages Then                              'only
if we are not erasing the bootspace
          Page = Page + 1                                   'next
page
          Spmcrval = 3 : Gosub Do_spm                       ' erase
next page
          Spmcrval = 17 : Gosub Do_spm                      '
re-enable page
```

```
        Else
          Portd.7 = 0 : Waitms 200
        End If
      End If
    Next

  Else
'eeprom
    For J = 1 To 128
      Writeeeprom Buf(j) , Wrd
      Wrd = Wrd + 1
    Next
  End If
  Toggle Portd.7 : Waitms 10 : Toggle Portd.7
'indication that we write
Return


Do_spm:
  Bitwait Spmcsr.0 , Reset                                    '
check for previous SPM complete
  Bitwait Eecr.1 , Reset                                      'wait
for eeprom

  Z = Page                                                    'make
equal to page
  Shift Z , Left , Maxwordshift
'shift to proper place
  Z = Z + Wrd                                                 'add
word
  !  lds r30,{Z}
  !  lds r31,{Z+1}

  #if Loaderchip = 128
    !      lds r24,{Z+2}
    !      sts rampz,r24                                          '
we need to set rampz also for the M128
  #endif

  Spmcsr = Spmcrval
'assign register
  !  spm
'this is an asm instruction
  !  nop
  !  nop
Return


'Sub Isr_urx()
```

```
'End Sub



'How you need to use this program:
'1- compile this program
'2- program into chip with sample elctronics programmer
'3- select MCS Bootloader from programmers
'4- compile a new program for example M88.bas
'5- press F4 and reset your micro
' the program will now be uploaded into the chip with Xmodem
Checksum
' you can write your own loader.too
'A stand alone command line loader is also available


'How to call the bootloader from your program without a reset ???
'Do
'   Print "test"
'   Waitms 1000
'   If Inkey() = 27 Then
'      Print "boot"
'      Goto &H1C00
'   End If
'Loop

'The GOTO will do the work, you need to specify the correct
bootloader address
'this is the same as the $LOADER statement.
```

7.3.9 $CRYPT

Action

This directive marks encrypted BASIC code.

Syntax

$CRYPT data

Remarks

In some cases you might want to share only portions of your code. The IDE can encrypt your code, and the compiler can process this encrypted code.
AES encryption is used. You do need a commercial add on to use the encryption.
The $crypt command can be processed by all bascom editions starting from version 2.0.5.0. So you only need an add on when you want to encrypt the code.

 Once encrypted, you can NOT DECRYPT into source code! Thus make a BACKUP

of your source code before you encrypt the code.

See also
Edit Encrypt Selected Code 88

Example
```
$CRYPT 6288E522B4A1429A6F16D639BFB7405B
$CRYPT 7ABCF89E7F817EB166E03AFF2EB64C4B
$CRYPT 645C88E996A87BF94D34726AA1B1BCCC
$CRYPT 9405555D91FA3B51DEEC4C2186F09ED1
$CRYPT 6D4790DA2ADFF09DE0DA97C594C1B074
```

7.3.10 $CRYSTAL

Action
Instruct the compiler to override the crystal frequency options setting.

Syntax
$CRYSTAL = var

Remarks

var	A numeric constant with the Frequency of the crystal.

The frequency is selectable from the Compiler Settings 146. It is stored in a configuration file. The $CRYSTAL directive overrides this setting.
It is best to use the $CRYSTAL directive as the used crystal frequency is visible in your program that way.

 The $CRYSTAL directive only informs the compiler about the used frequency. It does not set any fuse bit. The frequency must be know by the compiler for a number of reasons. First when you use serial communications, and you specify $BAUD 607, the compiler can calculate the proper settings for the UBR register. And second there are a number of routines like WAITMS 1607, that use the execution time of a loop to generate a delay. When you specify $CRYSTAL = 1000000 (1 MHz) but in reality, connect a 4 MHz XTAL, you will see that everything will work 4 times as quick.

 Most new AVR chips have an internal oscillator that is enabled by default. Check the data sheet for the default value.
Most new AVR chips have an option to divide the oscillator frequency by a number of values. If these options are used you need to take this into account.
For example, you connect a 16 MHz crystal and select the external oscillator fuse byte, this would result in a 16 MHz clock for most old processors.

Most new processors have an internal divider which can be enabled. This is an 8-divider in most cases. So in such a case, the resulting frequency would be 2 MHz. $crystal should have a value of 2 MHz in that case.
Instead of changing the divider fusebyte you can also use the CONFIG CLOCKDIV statement to select the division factor.
In case you have a crystal with 16 MHz and you code has code like : CONFIG

CLOCKDIV=4 , you would use $CRYSTAL=4000000
Thus $crystal is the clock value used to clock the processor.

Here follows some more info :
What might not be clear : $crystal value does not reflect the value of the xtal you put in your circuit. But it reflects the value of the system clock.
Early older processors just had a provision for using an external oscillator. For example you could connect a 24 MHz xtal. There was no divider or option to divide this frequency. So the oscillator frequency was directly connected to the processor. This meant : xtal=processor clock
For this reason there was the $crystal directive. There were simply no internal oscillators.

But later series got options with an internal oscillator. And also options to set a fuse to divide the clock by 8 or some times 16.
Later series also had a divisor/pre scaler between the xtal oscillator and the system clock input.
So you start with an oscillator frequency of 16 Mhz. The divider is by default 1 so it will result in a 16 Mhz clock. For this reason you use $crystal=16000000

Now when you want to run on 8 Mhz, you still have 16 Mhz osc. so you divide by 2. This means your system clock will be 8 Mhz. And for this reason you set $crystal to a value of 8000000
In short, $crystal value must reflect the system clock.
The reason is that many options need to know the system clock. For example for waitms, waitus, etc. But also when setting a baud rate.
When designed when all AVR chips were known the $crystal directive probably was named $systemclock or something like that.

See also
$BAUD 607 , BAUD 1488 , CONFIG CLOCKDIV 909

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Print "Hello world"
End
```

7.3.11 $DATA

Action
Instruct the compiler to store the data in the DATA lines following the $DATA directive, in code memory.

Syntax
$DATA

Remarks

The AVR has built-in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write to and read from the EEPROM.
To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM.
A separate file is generated with the EEP extension. This file can be used to program the EEPROM.

The compiler must know which DATA must go into the code memory and which into the EEPROM memory and therefore two compiler directives were added.

$EEPROM and $DATA.

$EEPROM tells the compiler that the DATA lines following the compiler directive must be stored in the EEP file.
To switch back to the default behavior of the DATA lines, you must use the $DATA directive.

The READ statement that is used to read the DATA info may only be used with normal DATA lines. It does not work with DATA stored in EEPROM.

 Do not confuse $DATA directive with the DATA statement.

So while normal DATA lines will store the specified data into the code memory of the micro which is called the flash memory, the $EEPROM and $DATA will cause the data to be stored into the EEPROM. The EEP file is a binary file.

See also
$EEPROM 631 , READEEPROM 1415 , WRITEEEPROM 1611 , DATA 1177

ASM
NONE

Example
```
'-----------------------------------------------------------------------
--------
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : AT90S2313
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates $DATA directive

'-----------------------------------------------------------------------
--------
$regfile = "2313def.dat"
$baud = 19200
$crystal = 4000000                          ' 4 MHz crystal

$hwstack = 16
$swstack = 16
$framesize = 16

Dim B As Byte
Readeeprom B , 0                            ' now B will be 1
End

Dta:
```

```
$eeprom
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
$data
End
```

7.3.12 $DBG

Action
Enables debugging output to the hardware UART.

Syntax
$DBG

Remarks
Calculating the hardware, software and frame space can be a difficult task.
With $DBG the compiler will insert characters for the various spaces.

To the Frame space 'F' will be written. When you have a frame size of 4, FFFF will be written.
To the Hardware space 'H' will be written. If you have a hardware stack space of 8, HHHHHHHH will be written to this space.
To the software space 'S' will be written. If you have a software stack space of 6, SSSSSS will be written.
The idea is that when a character is overwritten, it is being used. So by watching these spaces you can determine if the space is used or not.
With the DBG statement a record is written to the HW UART. The record must be logged to a file so it can be analyzed by the stack analyzer.

Make the following steps to determine the proper values:

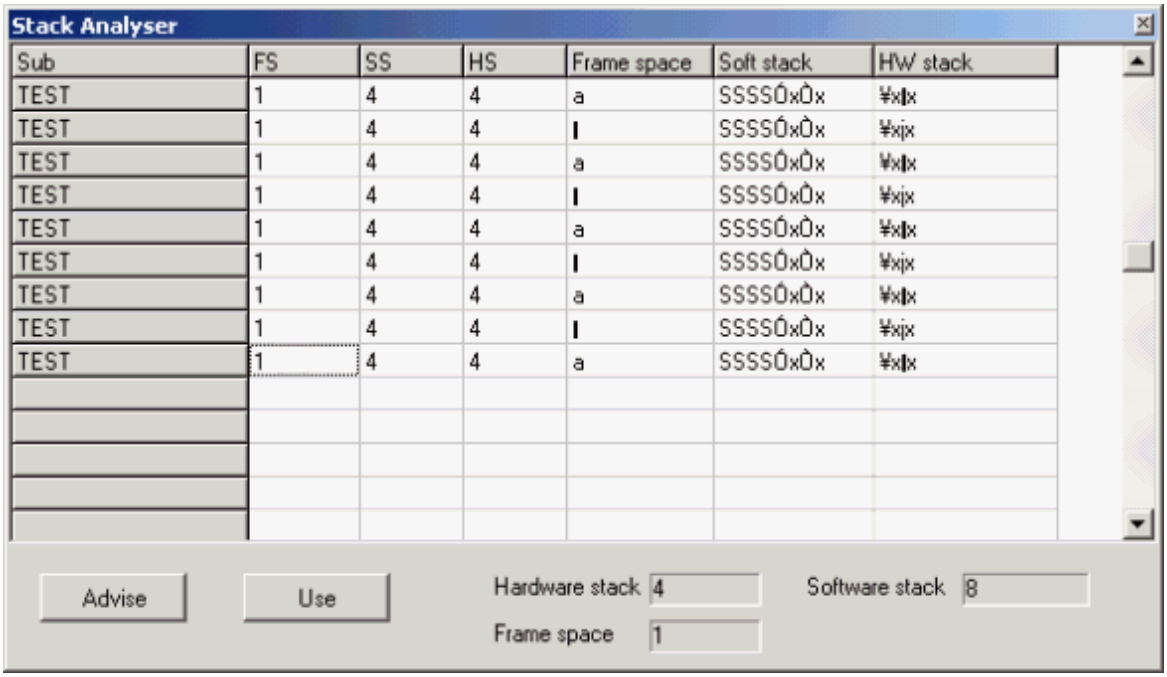
- Make the frame space 40, the soft stack 20 and the HW stack 50
- Add $DBG to the top of your program
- Add a DBG statement to every Subroutine or Function
- Open the terminal emulator and open a new log file. By default it will have the name of your current program with the .log extension
- Run your program and notice that it will dump information to the terminal emulator
- When your program has executed all sub modules or options you have build in, turn off the file logging and turn off the program
- Choose the Tools Stack analyzer option
- A window will be shown with the data from the log file
- Press the Advise button that will determine the needed space. Make sure that there is at least one H, S and F in the data. Otherwise it means that all the data is overwritten and that you need to increase the size.
- Press the Use button to use the advised settings.

As an alternative you can watch the space in the simulator and determine if the characters are overwritten or not.

The DBG statement will assign an internal variable named ___SUBROUTINE
Because the name of a SUB or Function may be 32 long, this variable uses 33 bytes!

___SUBROUTINE will be assigned with the name of the current SUB or FUNCTION.

When you first run a SUB named Test1234 it will be assigned with Test1234
When the next DBG statement is in a SUB named Test, it will be assigned with Test.
The 234 will still be there so it will be shown in the log file.



Every DBG record will be shown as a row.
The columns are:

Column	Description
Sub	Name of the sub or function from where the DBG was used
FS	Used frame space
SS	Used software stack space
HS	Used hardware stack space
Frame space	Frame space
Soft stack	Soft stack space
HW stack	Hardware stack space

The Frame space is used to store temp and local variables.
It also stores the variables that are passed to subs/functions by value.
Because PRINT , INPUT and the FP num<>String conversion routines require a buffer, the compiler always is using 24 bytes of frame space.

When the advise is to use 2 bytes of frame space, the setting will be 24+2=26.

For example when you use : print var, var need to be converted into a string before it can be printed or shown with LCD.

An alternative for the buffer would be to setup a temp buffer and free it once finished. This gives more code overhead.
In older version of BASCOM the start of the frame was used for the buffer but that gave conflicts when variables were printed from an ISR.

See also
DBG 1210

7.3.13 $DEFAULT

Action
Set the default for data types dimensioning to the specified type.

Syntax
$DEFAULT var

Remarks

Var	SRAM, XRAM, ERAM

Each variable that is dimensioned will be stored into SRAM, the internal memory of the chip. You can override it by specifying the data type.
Dim B As XRAM Byte , will store the data into external memory.

When you want all your variables to be stored in XRAM for example, you can use the statement : $DEFAULT XRAM
Each Dim statement will place the variable in XRAM in that case.

To switch back to the default behavior, use $END $DEFAULT

See also
NONE

ASM
NONE

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

$default Xram
Dim A As Byte , B As Byte , C As Byte
'a,b and c will be stored into XRAM

$default Sram
Dim D As Byte
'D will be stored in internal memory, SRAM
```

7.3.14 $EEPLEAVE

Action
Instructs the compiler not to recreate or erase the EEP file.

Syntax
$EEPLEAVE

Remarks
When you want to store data in the EEPROM, and you use an external tool to create the EEP file, you can use the $EEPLEAVE directive.
Normally the EEP file will be created or erased, but this directive will not touch any existing EEP file.
Otherwise you would erase an existing EEP file, created with another tool.

See also
$EEPROMHEX 632

Example
NONE

7.3.15 $EEPROM

Action
Instruct the compiler to store the data in the DATA lines following the $EEPROM directive in an EEP file.

Syntax
$EEPROM

Remarks
The AVR has built-in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write to and read from the EEPROM.
To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM.
A separate file is generated with the EEP extension. This file can be used to program the EEPROM.

The compiler must know which DATA must go into the code memory and which into the EEPROM memory and therefore two compiler directives were added.

$EEPROM and $DATA.

$EEPROM tells the compiler that the DATA lines following the compiler directive must be stored in the EEP file.
To switch back to the default behavior of the DATA lines, you must use the $DATA directive.

The READ statement that is used to read the DATA info may only be used with normal DATA lines. It does not work with DATA stored in EEPROM.

 Do not confuse $DATA directive with the DATA statement.

So while normal DATA lines will store the specified data into the code memory of the

micro which is called the flash memory, the $EEPROM 631 and $DATA will cause the data to be stored into the EEPROM. The EEP file is a binary file. The $EEPROMHEX 632 directive can be used to create Intel HEX records in the EEP file

See also
$EEPROM 631 , READEEPROM 1415 , WRITEEEPROM 1611 , DATA 1177 , $EEPROMHEX 632

ASM
NONE

Example
```
'-----------------------------------------------------------------------
--------
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : AT90S2313
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates $DATA directive

'-----------------------------------------------------------------------
--------
$regfile = "2313def.dat"
$baud = 19200
$crystal = 4000000                                  ' 4 MHz crystal
$hwstack = 16
$swstack = 16
$framesize = 16

Dim B As Byte
Readeeprom B , 0                                    'now B will be 1
End

Dta:
$eeprom
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
$data
End
```

7.3.16 $EEPROMHEX

Action
Instruct the compiler to store the data in the EEP file in Intel HEX format instead of binary format.

Syntax
$EEPROMHEX

Remarks
The AVR has build in EEPROM. With the WRITEEEPROM and READEEPROM

statements, you can write and read to the EEPROM.

To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM. $EEPROM must be used to create a EEP file that holds the data.

The EEP file is by default a binary file. When you use the STK500 you need an Intel HEX file. Use $EEPROMHEX to create an Intel Hex EEP file.

 $EEPROMHEX must be used together with $EEPROM.

See also
$EEPROMLEAVE 630

Example
```
$eeprom'the following DATA lines data will go to the EEP file
Data 200 , 100,50
$data
```

This would create an EEP file of 3 bytes. With the values 200,100 and 50.
Add $eepromhex in order to create an Intel Hex file.

This is how the EEP file content looks when using $eepromhex

```
:0A00000001020304050A141E283251
:00000001FF
```

7.3.17 $EEPROMSIZE

Action
Instruct the compiler to override the EEPROM size of the micro processor.

Syntax
$EEPROMSIZE = size

size	The size in bytes of the EEPROM.

Remarks
The AVR has build in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write and read to the EEPROM. You can also use the ERAM pseudo variables to read/write EEPROM.

When you use an external EEPROM and an alternative EEPROM library such as FM24C16 or FM25C256 you can override the internal EEPROM. All EEPROM routines will use the external EEPROM then. This way you are able to use a bigger EEPROM than internal available. Or you can use a quicker EEPROM such as a RAMTRON FRAM EEPROM. These EEPROM's are as quick as SRAM and also can be written to almost unlimited times.

 When using an external EEPROM and $EEPROMSIZE , take care that the supported programmers can not write to this EEPROM. They assume the internal EEPROM.

See also
FM24C16 1743, FM25C256 1748

Example
```
$eepromsize = &H8000
```

7.3.18 $EXTERNAL

Action
Instruct the compiler to include ASM routines from a library.

Syntax
$EXTERNAL Myroutine [, myroutine2]

Remarks
You can place ASM routines in a library file. With the $EXTERNAL directive you tell the compiler which routines must be included in your program.

See also
$LIB 665

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
$hwstack = 16
$swstack = 16
$framesize = 16
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits =
1 , Databits = 8 , Clockpol = 0
'In order to let this work you must put the mylib.lib file in the
LIB dir
'And compile it to a LBX
'-----------------------------------------------------------------
---------
'define the used library
$lib"mylib.lbx"
'you can also use the original ASM :
'$LIB "mylib.LIB"

'also define the used routines
```

```
$external Test
'this is needed so the parameters will be placed correct on the
stack
Declare Sub Test(byval X As Byte , Y As Byte)
'reserve some space
Dim Z As Byte
'call our own sub routine
Call Test(1 , Z)
'z will be 2 in the used example
End
```

7.3.19 $FILE

Action
Change name of generated files.

Syntax
$FILE = "myname.bin"

Remarks
In some cases it is desired to change the name of the output file. By default, the generated files have the same base name as the opened project file. So if your program name is "mytest.bas" , all generated files will start with the base "mytest". The $FILE directive let you change this base name.

 Simulating and programming will NOT work since the IDE uses the base name of your project. If you change it with $FILE, the files can not be located.

See also
NONE

Example
```
$FILE = "mytest.bin"
```

7.3.20 $FORCESOFTI2C

Action
The $forcesofti2c directive force the ATXMEGA/ATXTINY to use software I2C/TWI Library instead of the hardware I2C registers of ATXMEGA/XTINY.

Syntax
$forcesofti2c

Remarks
ATXMEGA have usually enough I2C interfaces with fixed SDA and SCL pins but if you

want to use other pins as SDA/SCL you can use this directive.
Required Library: $lib "i2c.lbx"

 You can not combine the soft mode with the hardware TWI. Thus when using **$forcesofti2c**, you can not add an additional TWI channel.

```
$forcesofti2c                  ' with this the software I2C/TWI commands
are used when inlcuding i2c.lbx
$lib "i2c.lbx"                 ' override the normal xmega i2c lib
```

Then you need to configure the SDA and SCL Pin and initialize the pins:

```
Config Scl = Port0.1     ' Pin to use as SCL (The hardware pin is Pinb.1)
Config Sda = Port0.0     ' Pin to use as SDA (The hardware pin is Pinb.0)
I2cinit                  ' Bring the Pin's in the proper state
```

 It is important that you include the i2c library name after the $forcesofti2c directive. It is also important that you do that early in your code and that you do not use the hardware TWI registers. Thus CONFIG TWI should not be used.
The variable named Twi_start is not required when using soft TWI/I2C. So you can remove it to see if the right code is used. The Xmega/Xtiny code need this variable and will give error messages when you do not include it. The software implementation does not need it and should not gives error messages when you remove this variable.

See also

Using the I2C protocol 297

Example

```
' Using ATXMEGA with software I2C routines to use also pins which
are no hardware SDA/SCL pins
' Needed Library: $lib "i2c.lbx"
' The $forcesofti2c directive force the ATXMEGA to use software
I2c/TWI Library
' The hardware for this example is XMEGA-A3BU XPlained board from
Atmel
' Don't forget the pull-ups on SDA/SCL pin !
' Bascom Version 2.0.7.6 or higher needed
$regfile = "XM256A3BUDEF.DAT"
$crystal = 32000000                          '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80
$forcesofti2c                  ' with this the software I2C/TWI
commands are used when inlcuding i2c.lbx
$lib "i2c.lbx"                 ' override the normal xmega i2c
lib

Config Osc = Enabled , 32mhzosc = Enabled
```

```
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Portr.0 = Output

Led0 Alias Portr.0                                          'LED
0 (XMEGA-A3BU XPlained board from Atmel )

Config Portr.1 = Output
Led1 Alias Portr.1              'LED 1 (XMEGA-A3BU XPlained board
from Atmel )

Dim B As Byte
'We use here  Virtual port 0
Config Vport0 = B                                           ' map
portB to virtual port0
Config Scl = Port0.1                                        ' Pin
to use as SCL (The hardware pin is Pinb.1)
Config Sda = Port0.0                                        ' Pin
to use as SDA (The hardware pin is Pinb.0)
I2cinit                                                     '
Bring the Pin's in the proper state

Do
   Waitms 500
   Set Led1
   Reset Led0
   Waitms 500
   Reset Led1
   Set Led0
   Incr B
   I2cstart
   I2cwbyte &H24                                            '
address of I2C Slave
   I2cwbyte B                                               '
databyte to send to slave
   I2cstop
Loop
End                                                         'end
program
```

7.3.21 $FRAMEPROTECT

Action

This directive will enable or disable interrupt frame protection.

Syntax

$FRAMEPROTECT = value

Remarks

Value must be a constant expression that evaluates to false (0) or true (<>0).
By **default** the frame protection is **off**.

When a user function/sub passes parameters with byval, a copy is created and passed to the user sub/function.
When an interrupt is executed, and it calls user sub/functions with parameters passed with byval, the values can get corrupted.

When activated, the compiler disables interrupts before passing variables, and enables interrupts (when they were enabled) inside the user sub/function. This ensures that the values can not get corrupted from an interrupt which is calling other user sub/functions.
When you do not call user sub/functions from inside your interrupt you can omit the $frameprotect directive or set it to 0 in order to reduce code.
In version 2075 the compiler had frame protection as a default, and the $NOFRAMEPROTECT served as an override. While you can still use $NOFRAMEPROTECT, it is off by default in 2076 to the preferred switch is $FRAMEPROTECT = 0|1

When you activate frame protection the internal constant named _FPROTECT will be set to 1.

When you have a user function that calls an ASM library, you must include code to restore the I-flag.
The bcd.lib user lib sample demonstrates this with this code :

```
#IF _FPROTECT
  Out sreg,r3                   ; restore I flag
#ENDIF
```

See also

$NOFRAMEPROTECT 685

Example

```
'************************************************
'        TESTING THE FRAME PARAMETER PASSING
'        UNDER HEAVY INTERRUPT LOAD
'************************************************

' file:  frame_pass_test.bas

$regfile = "m88def.dat"
$crystal = 8000000
$hwstack = 100
$swstack = 100
$framesize = 100

$noframeprotect   ' in this sample, disabling the frame protection will
result in errors
$frameprotect=0   ' from version 2076, this is the preferred method

Dim Ww As Word , Www As Word , Wwww As Word
```

```
Declare Sub Stack_checking(byval Identifier As Integer )

$baud = 19200
Open "com1:" For Binary As #1

Const T0_idozito = 100
Config Timer0 = Timer , Prescale = 1024   '256 --> 4.096 msec egység,
1024 --> 16.384 msec
On Ovf0 Timer0_interrupt
Enable Timer0
Start Timer0
Load Timer0 , T0_idozito

'   These routines are called under the timer interrupt
Declare Sub Under_it_pass_1(byval Inpar1_uit As Word )
Declare Sub Under_it_pass_2(byval Inpar2_uit As Word )
Declare Sub Test()
'   These routines are called in the main loop
Declare Sub Inmain_test_routine_1(byval Im1_par1 As Word , Byval
Im1_par2 As Word , Byval Im1_par3 As Word , Byval Im1_par4 As Word ,
Byval Im1_par5 As Word , Byval Im1_par6 As Word )
Declare Sub Inmain_test_routine_2(byval Im2_par1 As Word , Byval
Im2_par2 As Word , Byval Im2_par3 As Word , Byval Im2_par4 As Word ,
Byval Im2_par5 As Word , Byval Im2_par6 As Word )
Declare Sub Inmain_test_routine_3(byval Im3_par1 As Word , Byval
Im3_par2 As Word , Byval Im3_par3 As Word , Byval Im3_par4 As Word ,
Byval Im3_par5 As Word , Byval Im3_par6 As Word )

'   Routine-1 parameters are stored here
Dim Dim1_p1 As Word
Dim Dim1_p2 As Word
Dim Dim1_p3 As Word
Dim Dim1_p4 As Word
Dim Dim1_p5 As Word
Dim Dim1_p6 As Word

'   Routine-3 parameters are stored here
Dim Dim3_p1 As Word
Dim Dim3_p2 As Word
Dim Dim3_p3 As Word
Dim Dim3_p4 As Word
Dim Dim3_p5 As Word
Dim Dim3_p6 As Word

Program_begins_here:
    Enable Interrupts
    Print #1 , "PROGRAM BEGIN"
    Do
       Call Inmain_test_routine_1(&Haaaa , &HAAAA , &HAAAA , &HAAAA , &
HAAAA , &HAAAA )

       Call Inmain_test_routine_2(&Haaaa , &HAAAA , &HAAAA , &HAAAA , &
HAAAA , &HAAAA )

       Call Inmain_test_routine_3(&Haaaa , &HAAAA , &HAAAA , &HAAAA , &
HAAAA , &HAAAA )
    Loop

'   All the three routines always gets all parameters as &hAAAA, if they
see anything else, they print an error
'   routine_1 stores to DIM area and checks the stored values
'   routine 2 check immediately the incoming parameters
'   routine_3 completely identical to routine_1, except the parameter
passing protection
```

```
'

Sub Inmain_test_routine_1(byval Im1_par1 As Word , Byval Im1_par2 As
Word , Byval Im1_par3 As Word , Byval Im1_par4 As Word , Byval Im1_par5
As Word , Byval Im1_par6 As Word )
    Dim1_p1 = Im1_par1 : Dim1_p2 = Im1_par2 : Dim1_p3 = Im1_par3 :
Dim1_p4 = Im1_par4 : Dim1_p5 = Im1_par5 :
    Dim1_p6 = Im1_par6
    If Dim1_p1 <> &HAAAA Or Dim1_p2 <> &HAAAA Or Dim1_p3 <> &HAAAA Or
Dim1_p4 <> &HAAAA Or Dim1_p5 <> &HAAAA _
    Or Dim1_p6 <> &HAAAA Then
        Print #1 , " PAR ERROR R1 " ; Hex(dim1_p1 ) ; " " ; Hex(dim1_p2
) ; " " ; Hex(dim1_p3 ) ; " " ;
        Print #1 , Hex(dim1_p4 ) ; " " ; Hex(dim1_p5 ) ; " " ; Hex(
dim1_p6 )
    End If
End Sub

Sub Inmain_test_routine_2(byval Im2_par1 As Word , Byval Im2_par2 As
Word , Byval Im2_par3 As Word , Byval Im2_par4 As Word , Byval Im2_par5
As Word , Byval Im2_par6 As Word )
    If Im2_par1 <> &HAAAA Or Im2_par2 <> &HAAAA Or Im2_par3 <> &HAAAA Or
Im2_par4 <> &HAAAA Or Im2_par5 <> &HAAAA Or _           Im2_par6 <> &
HAAAA Then
        Print #1 , " PAR ERROR R2 " ; Hex(im2_par1 ) ; " " ; Hex(
im2_par2 ) ; " " ; Hex(im2_par3 ) ; " " ;
        Print #1 , Hex(im2_par4 ) ; " " ; Hex(im2_par5 ) ; " " ; Hex(
im2_par6 )
    End If
End Sub

Sub Inmain_test_routine_3(byval Im3_par1 As Word , Byval Im3_par2 As
Word , Byval Im3_par3 As Word , Byval Im3_par4 As Word , Byval Im3_par5
As Word , Byval Im3_par6 As Word )
    Dim3_p1 = Im3_par1 : Dim3_p2 = Im3_par2 : Dim3_p3 = Im3_par3 :
Dim3_p4 = Im3_par4 : Dim3_p5 = Im3_par5 :
    Dim3_p6 = Im3_par6
    If Dim3_p1 <> &HAAAA Or Dim3_p2 <> &HAAAA Or Dim3_p3 <> &HAAAA Or
Dim3_p4 <> &HAAAA Or Dim3_p5 <> &HAAAA Or _
    Dim3_p6 <> &HAAAA Then
        Print #1 , " PAR ERROR R3 " ; Hex(dim3_p1 ) ; " " ; Hex(dim3_p2
) ; " " ; Hex(dim3_p3 ) ; " " ;
        Print #1 , Hex(dim3_p4 ) ; " " ; Hex(dim3_p5 ) ; " " ; Hex(
dim3_p6 )
    End If
End Sub

Dim Under_it_store_1 As Word
Dim Under_it_store_2 As Word

'   these two routines are called under timer IT
'   They don't do much, except use the frame for parameter passing

Sub Under_it_pass_1(byval Inpar1_uit As Word )
    Under_it_store_1 = Inpar1_uit
End Sub

Sub Under_it_pass_2(byval Inpar2_uit As Word )
    Under_it_store_2 = Inpar2_uit
End Sub

'   Timer IT calling two routines which use the frame

Timer0_interrupt:
```

```
    Load Timer0 , T0_idozito
    Call Under_it_pass_1(&H5555 )
    Call Under_it_pass_2(&H3333 )
Return


End
```

7.3.22 $FRAMESIZE

Action
Sets the available space for the frame.

Syntax
$FRAMESIZE = var

Remarks

Var	A numeric decimal value.

While you can configure the Frame Size in Options, Compiler, Chip, it is good practice to put the value into your code. This way you do no need the cfg(configuration) file.

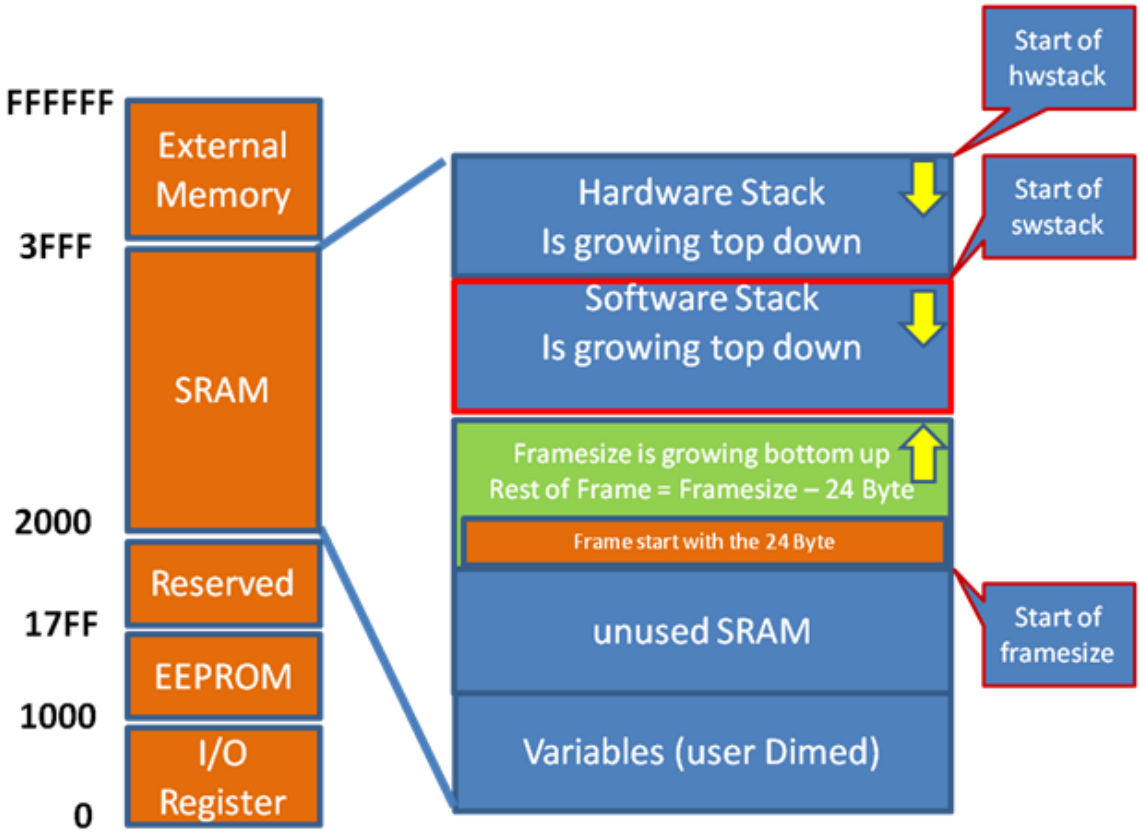
The $FRAMESIZE directive overrides the value from the IDE Options.

It is important that the $FRAMESIZE directive occurs in your main project file. It may not be included in an $include file as only the main file is parsed for $FRAMESIZE. $FRAMESIZE only accepts numeric values.

 Functions like PRINT 1501, LCD 657, INPUT 1493 and the FP num <> FORMAT 828 String conversion routines require a buffer in SRAM. Because of that **the compiler always is using 24 bytes of frame space**. This 24 Byte start at the beginning of the Frame which act as the conversion buffer within the frame (See also picture). Because the FRAME is growing bottom up and this 24 Byte start at the beginning of the FRAME this 24 Byte conversion buffer start at the lowest FRAME Address (See picture). Here you also see that a too small $framesize causes an overwriting of Software Stack and/or Hardware Stack which lead to malfunction. If you use Print numVar, then the numeric variable "numvar" is converted into a string representation of the binary number. The framespace buffer is also used for that.

When there is not enough room inside the frame, the ERR variable will be set to 1.

See also
$SWSTACK 705, $HWSTACK 648, Memory usage 267



Picture: Memory of ATXMEGA128A1

A LOCAL variable is a temporary variable that is stored in frame.
There can be only LOCAL variables of the type BYTE, INTEGER, WORD, LONG, SINGLE, DOUBLE or STRING.

A LOCAL Integer will use 2 Bytes of Frame ,
A LOCAL Long will use 4 Bytes.
A LOCAL string * 20 will use 20 + 1 = 21 Byte (this additional 1 Byte is because every String is terminated with a 0-Byte)

When the SUB or FUNCTION is terminated, the memory will be released back to the frame but the FRAME will not be cleared ! Therefore a LOCAL variable is not initialized. So you can not assume the variable is 0. If you like it to be 0, you need to assign it !

BIT variables are not possible as LOCAL because they are always GLOBAL to the system.
Arrays can NOT be used as LOCAL (but arrays can be passed by REFERENCE as parameter to SUB and FUNCTIONS which just need 2 Bytes Software Stack of the Address of Array start)

See following example for frame calculation:

Example

```
$regfile = "xm128a1def.dat"
$crystal = 32000000 '32MHz
$hwstack = 64
```

```
$swstack = 128
$framesize = 288

Config Osc = Enabled , 32mhzosc = Enabled '32MHz

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 '32MHz

'Config Interrupts
Config Priority = Static , Vector = Application , Lo = Enabled 'Enable
Lo Level Interrupts
Config Com1 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8


Declare Sub My_sub()

Call My_sub()

End 'end program

Sub My_sub()
 Local A1 As Byte , A2 As Byte , A3 As Byte , A4 As Byte , A5 As Byte
 Local S As String * 254

 For A1 = 1 To 254
   S = S + "1"
 Next A1

 A1 = 1
 A2 = 2
 A3 = 3
 A4 = 4
 A5 = 5
 Print A1
End Sub
```

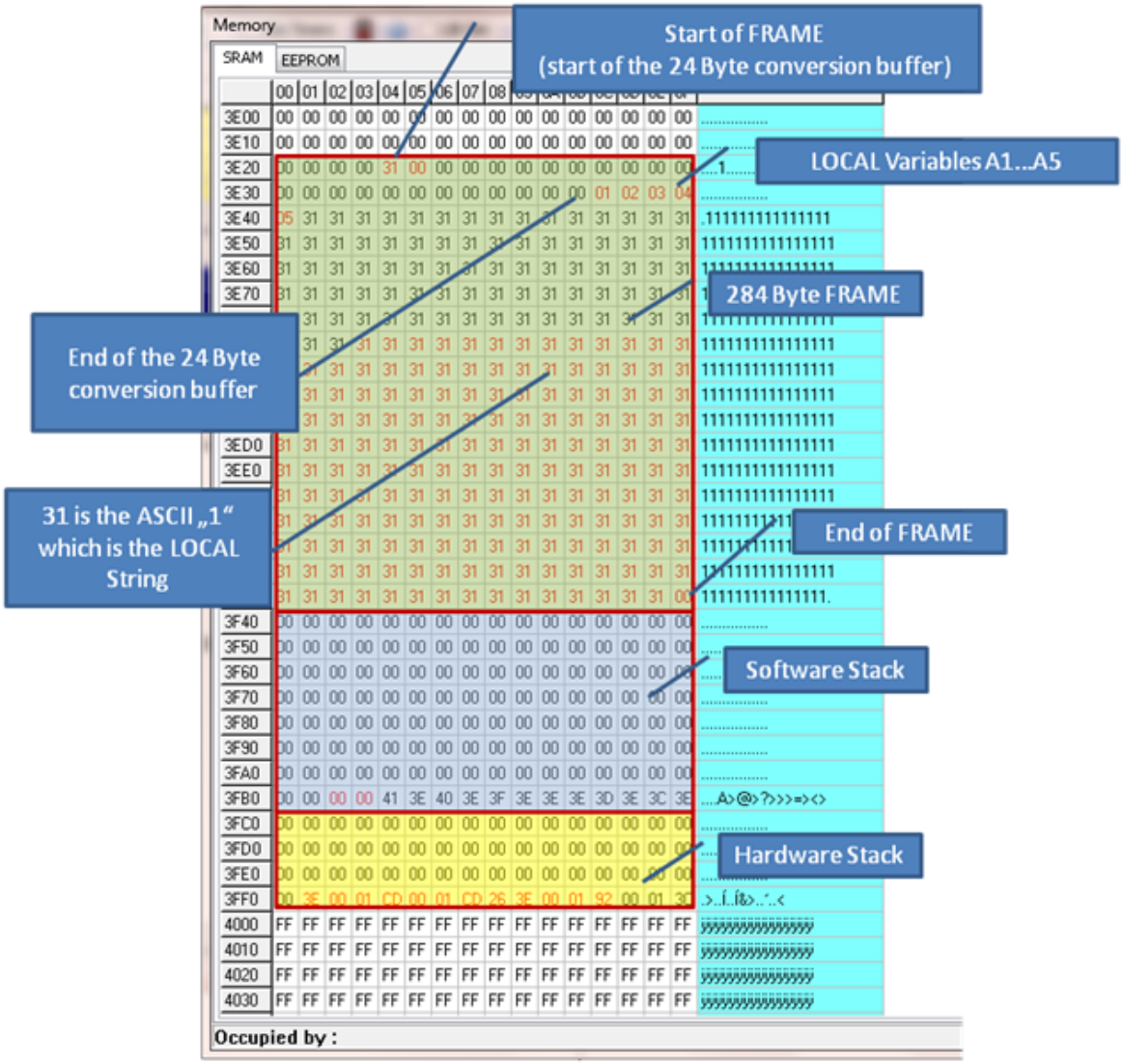
Now we calculate the FRAME:

The Print A1 will be placed in the first frame-Byte of the 24 Byte conversion buffer.
5 LOCAL Byte (A1 … A5) = 5 Byte of FRAME
LOCAL String: 254 Byte + 1 Byte = 255 Byte
Frame needed = 24Byte Frame conversion Buffer + 5 Byte + 255 Byte = 284 Byte
This can be easy double checked with BASCOM-AVR Simulator (see following picture).

In following picture you see the start of FRAME which start with the 24Byte conversion buffer. The 31 in the first Frame Byte is from Print A1. After the 24 Byte conversion buffer follow the 5 Local Byte variables (A1 …. A5) and then the 255 Byte for the LOCAL String.

As with Software Stack you need to calculate the Framesize needed by the SUB or FUNCTION with the most LOCAL Variables and parameter passed by REFERENCE etc..

Take care when calling a SUB within a SUB. In this case you need to add the FRAME needed by both SUB !
When both SUB need 284 Byte you need to use:
24 Byte conversion Buffer + 2* 5 Byte (A1…A5) + 2*255 Byte (String) = **544 Byte**
(the conversion buffer is needed only once !)



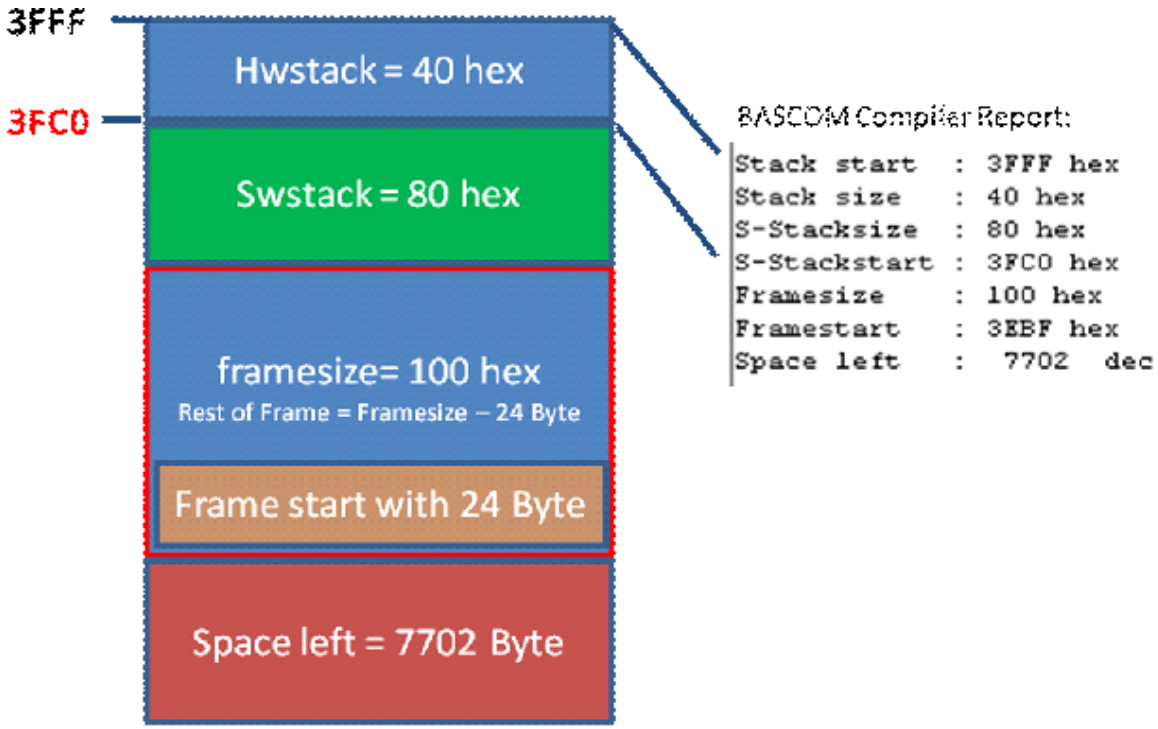
Picture: Memory window of BASCOM-AVR Simulator (Frame calculation example)

For further investigation of Stacks and Frame we use a SUB with 5 LOCAL Byte Variables and a PRINT function within the SUB. We start with hwstack, swstack and framesize defined and in second step we set swstack to 0. In addition we will lower the framesize to a not recommended value to force overwriting of other stack bytes.

```
$regfile = "xm128a1def.dat"
$crystal = 32000000 '32MHz
$hwstack = 64
$swstack = 128
$framesize = 256

Config Osc = Enabled , 32mhzosc = Enabled '32MHz

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 '32MHz

'Config Interrupts
Config Priority = Static , Vector = Application , Lo = Enabled 'Enable
Lo Level Interrupts
Config Com1 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
```

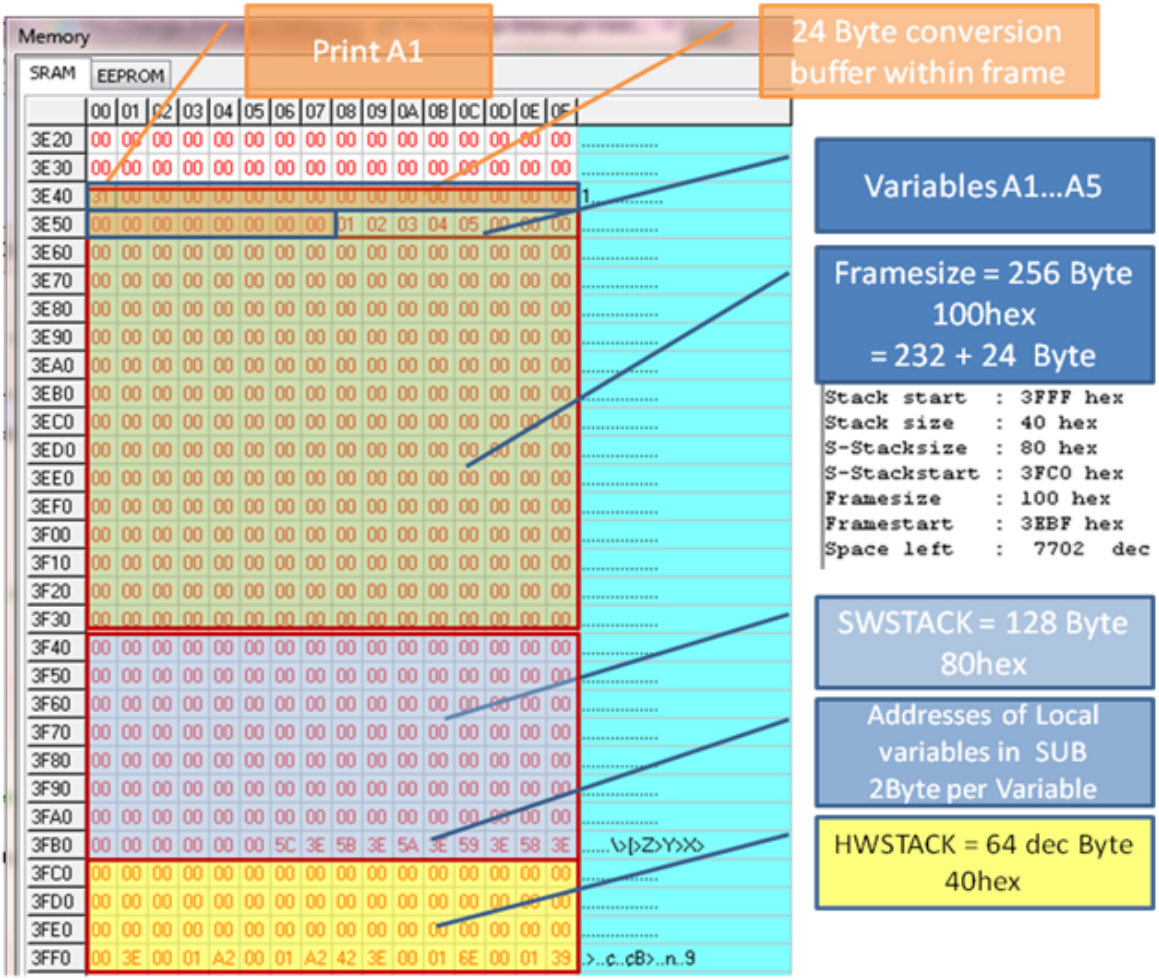
```
Declare Sub My_sub()

Call My_sub()

End 'end program

Sub My_sub()
 Local A1 As Byte , A2 As Byte , A3 As Byte , A4 As Byte , A5 As Byte

 A1 = 1
 A2 = 2
 A3 = 3
 A4 = 4
 A5 = 5
 Print A1

End Sub
```

Here we see the 64 Byte Hardware Stack followed by 128 Byte Software Stack and then 256 Byte Frame. As always the Frame is the 24 Byte conversion buffer + rest of frame.



Picture : SRAM for Example with $hwstack = 64, $swstack = 128, $framesize = 256

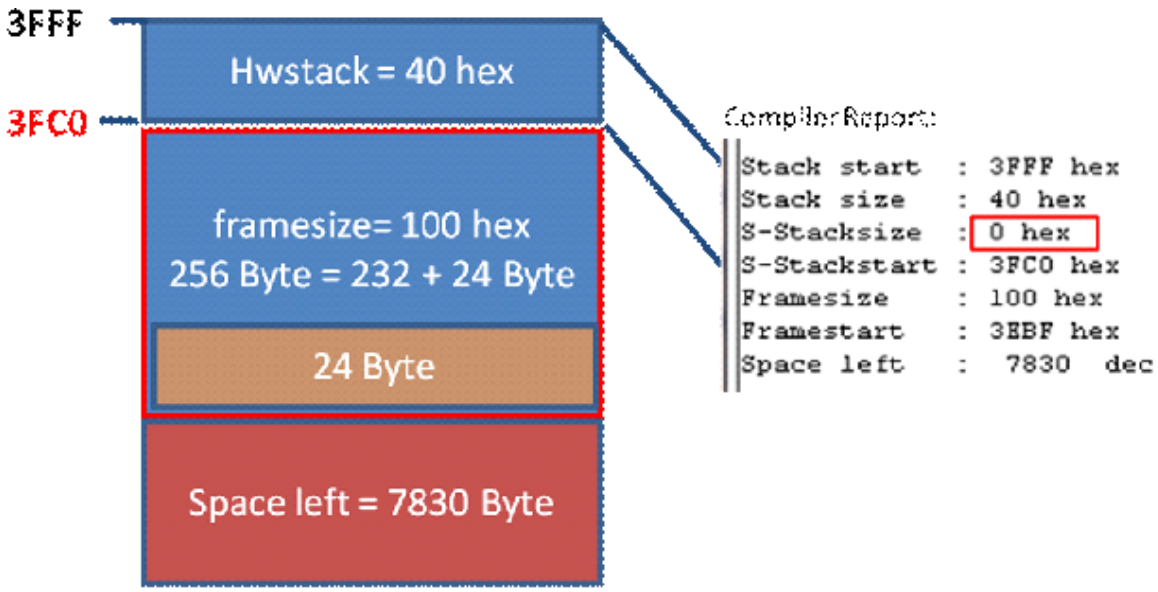
The Simulator Memory Window show give us the details:



Picture: Simulator Memory Window for Example with $hwstack = 64, $swstack = 128, $framesize = 256

The second example use $hwstack = 64, $swstack = 0, $framesize = 256

Without defining a software Stack or with $swstack = 0 the Frame follows direct after the Hardware Stack. The Frame is as always 24 Byte conversion buffer + Rest of Frame.
Rest of Frame is in this case: 256 Byte – 24 Byte = 232 Byte

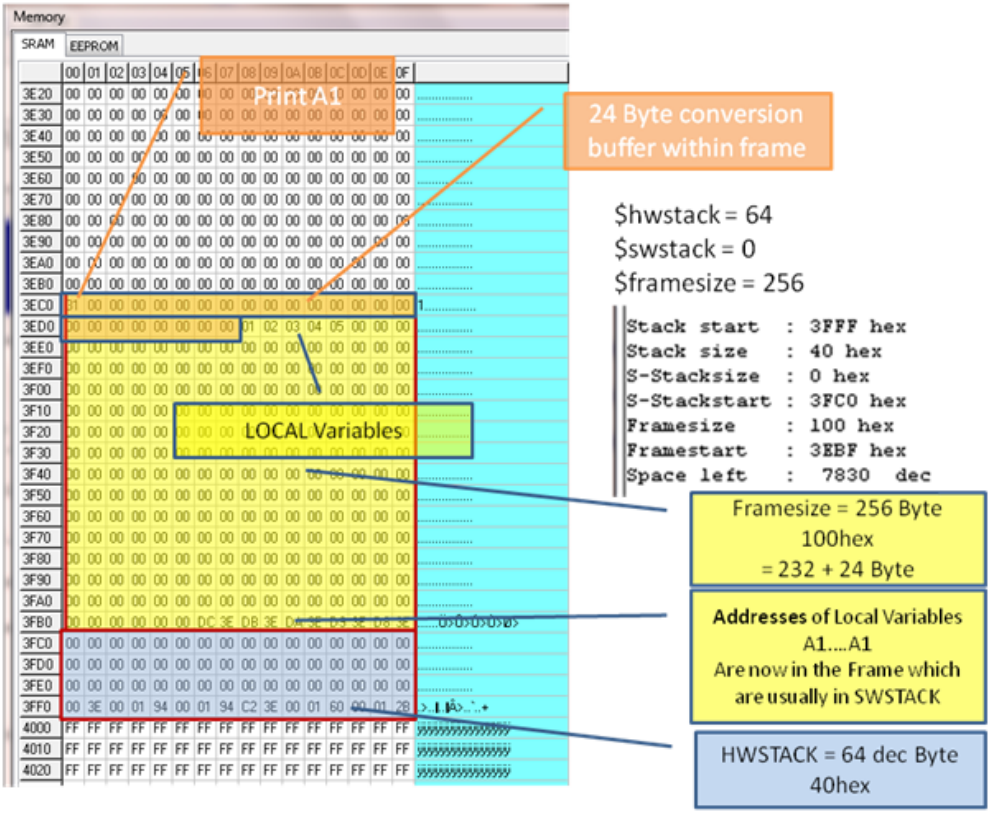


Picture: SRAM for example with $hwstack = 64, $swstack = 0, $framesize = 256

In the BASCOM Simulator Window you now see the addresses of the LOCAL variables are now stored in FRAME (which are usually in the Software Stack). This is not a problem as long as the Frame is big enough not to overwrite these addresses of the LOCAL variables.

(Remember: Address of LOCAL variables are stored in Software Stack (when Software Stack is defined) . The LOCAL Variables itself are stored in FRAME)

And here you see also with the 24 Byte conversion buffer the absolute minimum you need to define for software Stack and Framesize together is 24 Byte !
But this is not the recommendation. The recommendation is always define values for all Stack and Frame !



Picture: Simulator Memory Window for Example with $hwstack = 64, $swstack = 0, $framesize = 256

7.3.23 $HWSTACK

Action

Sets the available space for the Hardware stack.

Syntax

$HWSTACK = var

Remarks

Var	A numeric decimal value.

While you can configure the HW Stack in Options, Compiler, Chip, it is good practice to put the value into your code. This way you do no need the cfg(configuration) file.

The $HWSTACK directive overrides the value from the IDE Options.

It is important that the $HWSTACK directive occurs in your main project file. It may not be included in an $include file as only the main file is parsed for $HWSTACK. $HWSTACK only accepts numeric values.

The Hardware stack is room space in SRAM that is needed by your program. Each time you call a SUB or FUNCTION, or use GOSUB, the processor need to know at which address to return after returning from the call. Also for RETURN Address after Interrupt this is needed by the program. For this purpose, the processor saves this address on the hardware stack.
When you use GOSUB label, the microprocessor pushes the return address on the

hardware stack and will use 2 Bytes for that. When you use RETURN, the Hardware stack is popped back and the program can continue at the proper address. When you nest GOSUB, CALL or functions, you will use more stack space. Most statements use HW stack because a machine language routine is called.
The Hardware Stack is growing top down. The Hardware Stack start at the highest available SRAM Address and therefore is located before Software Stack and/or Frame.

See also

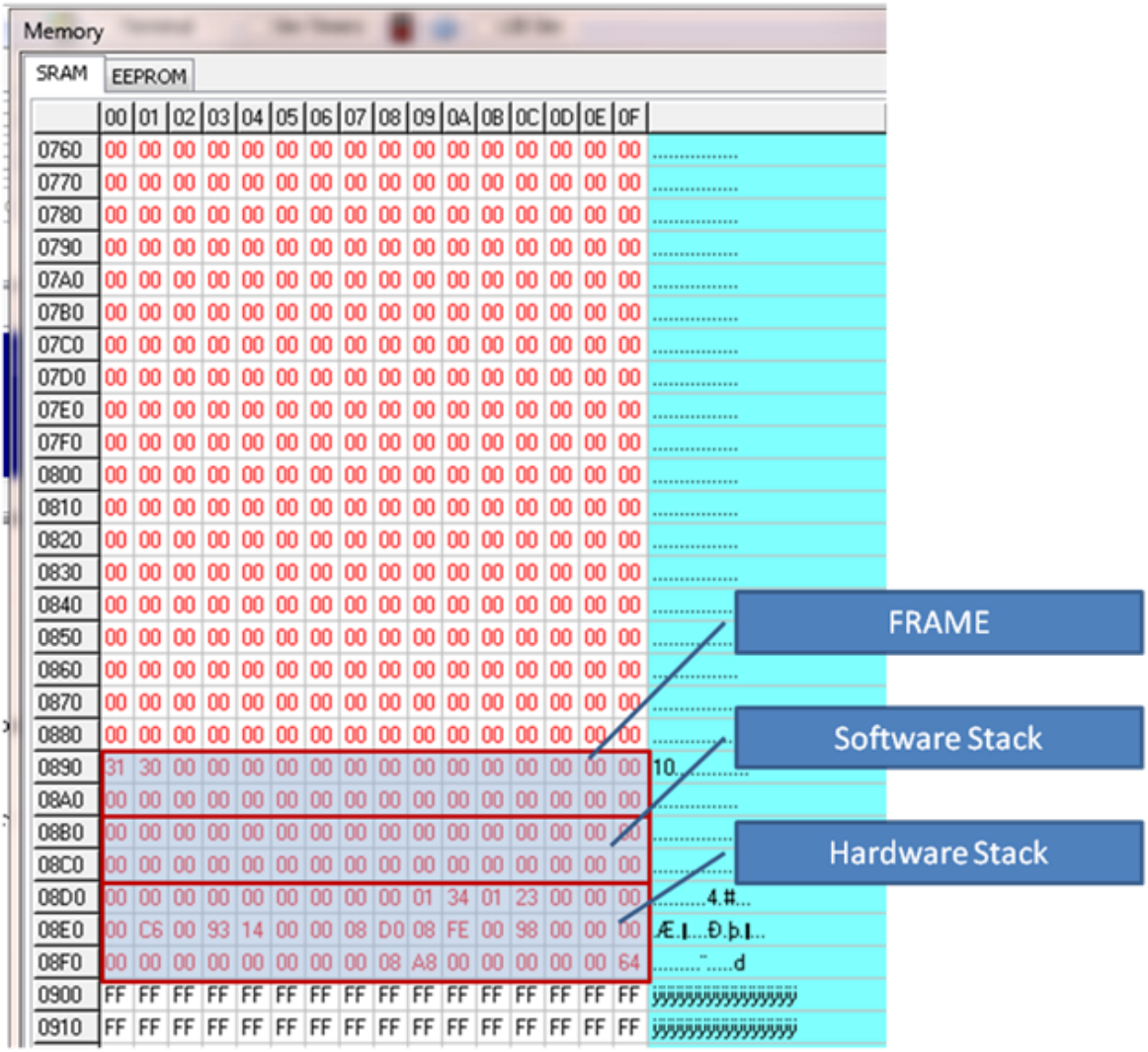
$SWSTACK 705 , $FRAMESIZE 641, Memory Usage 267

Example for using an Interrupt and examine Hardware Stack:

With the following example we just define and enable the Receive Interrupt of the UART and examine when clicking on Interrupt button within the Bascom-AVR Simulator Interrupts Tab how many Hardware Stack is needed.

```
$regfile = "m328pdef.dat"
$crystal = 16000000
$hwstack = 48
$swstack = 32
$framesize = 32

$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Dim Rs232 As Byte

'Enable Receive Interrupt for COM1
On Urxc Rxc_isr
Enable Urxc
Enable Interrupts

Do
 !nop
Loop

End

Rxc_isr:
  Rs232 = Inkey()
  Print Rs232
Return
```

Bascom-AVR Simulator output of the example above:



Picture : The Hardware Stack will be filled by clicking the Bascom-AVR Simulator Interrupt

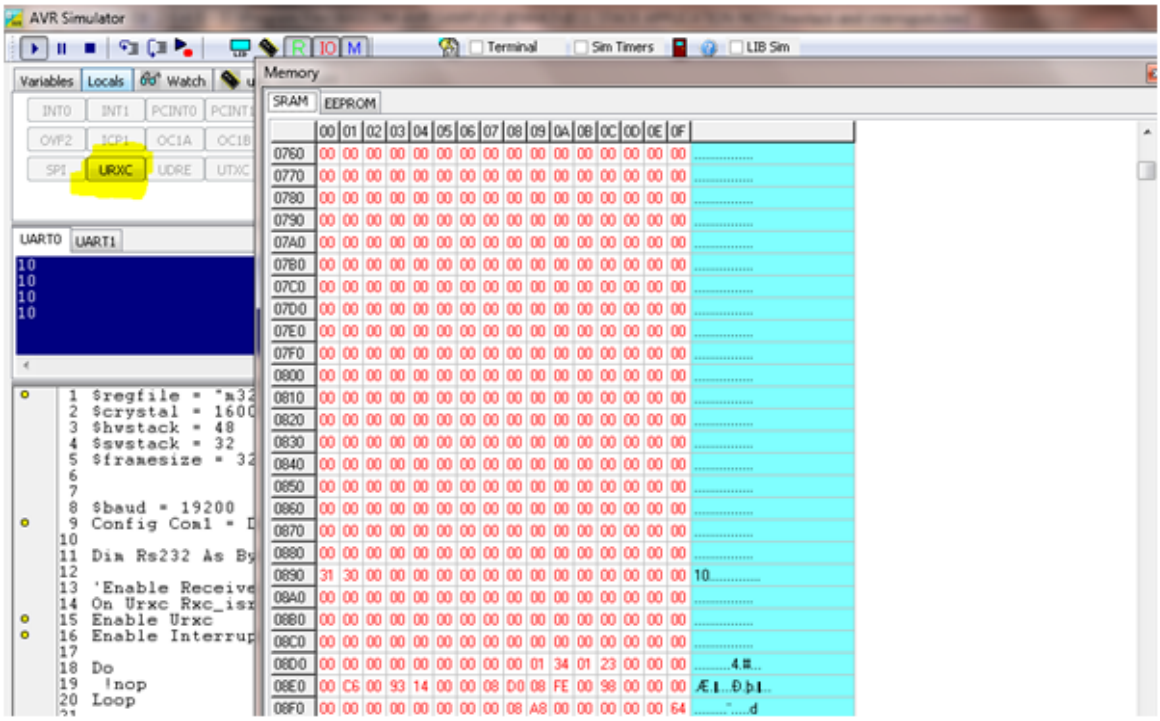
With this example we see (by counting the changed SRAM Bytes in Bascom Simulator Memory Window) that Software Stack is NOT needed but at least **39 Byte of Hardware Stack** and the Frame with the 24 Byte conversion buffer because of PRINT.

Most of the 39 Bytes are the saved Registers when jumping in Interrupt Service Routine. These are SREG , R31 to R16 and R11 to R0 with exception of R6,R8 and R9.

The following should be considered in any case (not only when using NOSAVE):
Take care when using floating point math in the ISR because the Register R12 to R15 are not saved in the regular process of processor register backup. Using floating point math in ISR is not recommended anyway.

When you try the same example with NOSAVE (`On Urxc Rxc_isr Nosave`) you will see the example will need less Hardware Stack but **you are responsible then to save all of the Registers** with PUSH and POP in the Interrupt Service Routine that are needed or changed during the Interrupt Service Routine.
The easier, and above all safer way is not using NOSAVE which is also the default way.

By clicking on the Interrupts Button will fire an interrupt in Simulator



7.3.24 $HWCHECK, $SWCHECK, $SOFTCHECK

Action

This directive can be used to determine the required stack space.

Syntax

$HWCHECK
$FRAMECHECK
$SOFTCHECK

Remarks

All variables you DIM in your application require RAM or SRAM space. But an application needs more RAM space.
Each time you call a sub or function, or us gosub, the processor need to know at which address to return after returning from the call. For this purpose, the processor saves this address on the hardware stack. There is noting you can do about this. This hardware stack grows downwards. Some basic statements compile into code that do not need any calls. But some call a machine language function which in turn can call other functions. Which and how many other calls will be made depend on the selected processor and other options. sometimes it also depends on variable parameters.

When parameters are passed to a sub or function, the address is passed of the variables. These are word addresses thus using 2 bytes for each variable. This passing is being done via the so called soft stack. This area is located below the HW stack space. And it also grows down.
All LOCAL variables you use also need 2 bytes of the soft stack.

When you pass a parameter with BYVAL or when you create a LOCAL variable, some temporarily space is need.

Consider this example : somestring = "abc" + somestring
When the compiler assigns "abc" to somestring, the somestring variable will become "abc" and it will overwrite the content making it impossible to add it's content after the "abc".
So we first need to store the content of somestring before we can start assigning new data to this string.
This copy also requires space.

This space is created dynamically and is taken from the so called frame space. This space is located below the soft stack.
Now you can use $DBG or some default values for most projects to determine the values.
But when you have a problem and have absolutely no idea how the settings must be made, you can use the $HWCHECK option.

You start with including a special library named "stackcheck.lib" to your code.
Then you run your application and somewhere in your code you print the value of the generated **_hw_lowest** variable.
This variable is set to &HFFFF and each time a call is made, the stack is compared to this value. If the hardware stack (SPL and SPH registers) are lower then the _hw_lowest value, _hw_lowest is assigned with the new lowest stack value.
This way you determine the lowest possible hardware stack value that occurred during the runtime of your application.
Of course it is important that your application runs all code.
You can print the value or show it on LCD. To determine the actual needed space you subtract it from the stacktop value.

For the softstack the same applies. It will store the lowest Y-pointer value to the variable named _sw_lowest.

For the framespace the the variable _fw_highest is used and this variables is increasing.

The stackcheck.bas example demonstrates how to retrieve the values when a recursive sub is used.

See also
NONE

Example
```
$regfile = "m88def.dat"
$hwstack = 40
$swstack = 80
$framesize = 80
$lib "stackcheck.lib"

Declare Sub Test(byval Prm As Byte)

Print "stack test"

Dim G As Byte , W As Word
Dim P As Byte

$hwcheck                                                    'hw stack
check on
```

```
$framecheck
$softcheck

Test P
Print _hw_lowest
W = _hwstackstart - _hw_lowest
Print "HW stack needed : " ; W

Print _fw_highest
If _fw_highest > 0 Then
   W = _frame_high - _fw_highest
   Print "Frame space needed : " ; W
End If


Print _sw_lowest
W = _hwstack_low - _sw_lowest
Print "SW stack needed : " ; W


End


Sub Test(byval Prm As Byte)
   Local L As Byte
   Print "HWSTACK:" ; _hw_lowest
   Print "Frame:" ; _fw_highest
   Print "SWSTACK:" ; _sw_lowest

   G = G + 1                                                ' global var
   If G >= 5 Then
     Exit Sub
   Else
      Test P                                                'recursive
call
   End If
End Sub
```

7.3.25 $INC

Action

Includes a binary file in the program at the current position.

Syntax

$INC label , size | nosize , "file"

Remarks

Label	The name of the label you can use to refer to the data.
Nosize	Specify either nosize or size. When you use size, the size of the data will be included. This way you know how many bytes you can retrieve.
File	Name of the file which must be included.

Use RESTORE to get a pointer to the data. And use READ, to read in the data.

The $INC statement is an alternative for the DATA statement.
While DATA works ok for little data, it is harder to use on large sets of data.

See Also
RESTORE 1429, DATA 1177 , READ 1413

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
$hwstack = 16
$swstack = 16
$framesize = 16

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Dim Size As Word , W As Word , B As Byte

Restore L1                                                  ' set
pointer to label
Read Size                                                   ' get size
of the data

Print Size ; " bytes stored at label L1"
For W = 1 To Size
  Read B : Print Chr(b);
Next

End

'include some data here
$inc L1 , Size , "c:\test.bas"
'when you get an error, insert a file you have on your system
```

7.3.26 $INCLUDE

Action
Includes an ASCII file in the program at the current position.

Syntax
$INCLUDE "file"

Remarks

File	Name of the ASCII file, which must contain valid BASCOM statements. This option can be used if you make use of the same routines in many programs. You can write modules and include them into your program. If there are changes to make you only have to change the module file, not all your BASCOM programs. You can only include ASCII files! An include file will only be included once, even if you include it multiple times.

Use $INC 653 when you want to include binary files.

You can specify an absolute file name (with a drive and full path) like : $INCLUDE "c:\folder\myfile.bas"
Or you can specify a relative file name like : $INCLUDE "myfile.bas"
The main program path will be used to determine the absolute file name.
If your main file is stored under c:\abc\main.bas , and you include a file named "test.inc" , the compiler expects a file named "c:\abc\test.inc"

You can include a path too. The path is relative to the main file.
When used in sub folders use " \ " (back slash). The path uses the DOS/Windows convention. A forward slash will work too since windows does not seem to be bothered with it.
Example with sub folder Test: `$include "Test\my_functions.bas"`

When you include sub procedures and functions before the actual code, your code will run into this code. You can use a GOTO to jump over the included code or you can use CONFIG SUBMODE=NEW so that the compiler will only include the used functions. See Example2

See Also

$INC 653 , CONFIG SUBMODE=NEW 1079

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$hwstack = 10
$swstack = 10
$framesize = 26
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'-----------------------------------------------------------
Print "INCLUDE.BAS"
'Note that the file 123.bas contains an error
$include "123.bas"                          'include file that prints
Hello
Print "Back in INCLUDE.BAS"
End
```

Example2

```
$regfile = "m48def.dat"
$crystal = 4000000
$hwstack = 10
$swstack = 10
$framesize = 26
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'-----------------------------------------------------------
$include "mysubs.bas"                         'include file with sub
procedures
  'this is the included code :  Sub Test()
  '                                print "Test"
```

```
  '                                End Sub
  'Without a GOTO to jump over the included code the code will run into
the sub without a call
 ' Or use CONFIG SUBMODE=NEW so you do not need to change a thing

Print "Back in INCLUDE.BAS"
End
```

7.3.27 $INITMICRO

Action

Calls a user routine at startup to perform important initialization functions such as setting ports.

Syntax

$INITMICRO

Remarks

This directive will call a label named _INIT_MICRO just after the most important initialization is performed. You can put the _INIT_MICRO routine into your program, or you can put it in a library. Advantage of a library is that it is the same for all programs, and advantage of storing the code into your program is that you can change it for every program.

It is important that you end the routine with a RETURN as the label is called and expects a return.
The $initmicro can be used to set a port direction or value as it performs before the memory is cleared which can take some mS.
The best solution for a defined logic level at startup remains the usage of pull up/pull down resistors.

See Also

NONE

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$hwstack = 10
$swstack = 10
$framesize = 26
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

$initmicro

Print Version()                                          'show date
and time of compilation

Print Portb
Do
   nop
Loop
End
```

```
'do not write a complete application in this routine.
'only perform needed init functions
_init_micro:
  Config Portb = Output
  Portb = 3
Return
```

7.3.28 $LCD

Action

Instruct the compiler to generate code for 8-bit LCD displays attached to the data bus.

Syntax

$LCD = [&H]address

Remarks

Address	The address where must be written to, to enable the LCD display and the RS line of the LCD display. The db0-db7 lines of the LCD must be connected to the data lines D0-D7. (or is 4 bit mode, connect only D4-D7) The RS line of the LCD can be configured with the LCDRS statement. On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

Do not confuse $LCD with the LCD statement.
The compiler will create a constant named ___LCD_ADR which you could use in an alternative LCD library.

See also

$LCDRS 662 , CONFIG LCD 998 , LCD 1335

Example

```
'-----------------------------------------------------------
'                (c) 1995-2025 MCS Electronics
'-----------------------------------------------------------
'  file: LCD.BAS
'  demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
'        CURSOR, DISPLAY
'-----------------------------------------------------------

'note : tested in bus mode with 4-bit on the STK200
'LCD   -   STK200
'------------------
'D4         D4
'D5         D5
'D6         D6
'D7         D7
'WR         WR
'E          E
'RS         RS
```

```
'+5V        +5V
'GND        GND
'V0         V0
'D0-D3 are not connected since 4 bit bus mode is used!

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Rem with the config lcdpin statement you can override the compiler
settings

$regfile = "8515def.dat"
$lcd = &HC000
$lcdrs = &H8000
Config Lcdbus = 4

Dim A As Byte
Config Lcd = 16x2                                          'configure lcd
screen
'other options are 16 * 2 , 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                         'clear the
LCD display
Lcd "Hello world."                                          'display
this at the top line
Wait 1
Lowerline                                                   'select the
lower line
Wait 1
Lcd "Shift this."                                           'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                           'shift the
text to the right
   Wait 1                                                   'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                            'shift the
text to the left
   Wait 1                                                   'wait a
moment
Next

Locate 2 , 1                                                'set cursor
position
Lcd "*"                                                     'display
this
Wait 1                                                      'wait a
moment

Shiftcursor Right                                           'shift the
cursor
Lcd "@"                                                     'display
this
Wait 1                                                      'wait a
```

```
moment

Home Upper                                                    'select line
1 and return home
Lcd "Replaced."                                               'replace the
text
Wait 1                                                        'wait a
moment

Cursor Off Noblink                                            'hide cursor
Wait 1                                                        'wait a
moment
Cursor On Blink                                               'show cursor
Wait 1                                                        'wait a
moment
Display Off                                                   'turn
display off
Wait 1                                                        'wait a
moment
Display On                                                    'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                    'goto home
on line three
Home Fourth
Home F                                                        'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                           'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                           'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                    'value into
ACC
!rCall _write_lcd                                             'put it on
LCD
End
```

7.3.29 $LCDPUTCTRL

Action

Specifies that LCD control output must be redirected.

Syntax
$LCDPUTCTRL = label

Remarks

Label	The name of the assembler routine that must be called when a control byte is printed with the LCD statement. The character must be placed in register R24.

With the redirection of the LCD statement, you can use your own routines.

See also
$LCDPUTDATA 661

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'dimension used variables
Dim S As String* 10
Dim W As Long

'inform the compiler which routine must be called to get serial
'characters
$lcdputdata= Myoutput
$lcdputctrl= Myoutputctrl
'make a never ending loop
Do
  Lcd "test"
Loop

End

'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and 'restore
'all registers so we can use all BASIC statements
'$LCDPUTDATA requires that the character is passed in R24

Myoutput:
  Pushall                                                  'save all
registers
   'your code here
  Popall                                                   'restore
registers
Return

MyoutputCtrl:
  Pushall                                                  'save all
registers
   'your code here
  Popall                                                   'restore
registers
Return
```

7.3.30 $LCDPUTDATA

Action
Specifies that LCD data output must be redirected.

Syntax
$LCDPUTDATA = label

Remarks

Label	The name of the assembler routine that must be called when a character is printed with the LCD statement. The character must be placed in R24.

With the redirection of the LCD statement, you can use your own routines.

See also
$LCDPUTCTRL 659

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'dimension used variables
Dim S As String* 10
Dim W As Long

'inform the compiler which routine must be called to get serial
'characters
$lcdputdata= Myoutput
$lcdputctrl= Myoutputctrl
'make a never ending loop
Do
  Lcd "test"
Loop

End

'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and 'restore
'all registers so we can use all BASIC statements
'$LCDPUTDATA requires that the character is passed in R24

Myoutput:
  Pushall                                                    'save all
registers
   'your code here
  Popall                                                     'restore
registers
Return

MyoutputCtrl:
  Pushall                                                    'save all
registers
```

```
   'your code here
  Popall                                                    'restore
registers
Return
```

7.3.31 $LCDRS

Action

Instruct the compiler to generate code for 8-bit LCD displays attached to the data bus.

Syntax

$LCDRS = [&H]address

Remarks

Address	The address where must be written to, to enable the LCD display. The db0-db7 lines of the LCD must be connected to the data lines D0-D7. (or is 4 bit mode, connect only D4-D7) On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

The compiler will create a constant named ___LCDRS_ADR which you could use in an alternative LCD library.

See also

$LCD 657 , CONFIG LCDBUS 998

Example

```
'-----------------------------------------------------------
'                 (c) 1995-2025 MCS Electronics
'-----------------------------------------------------------
'  file: LCD.BAS
'  demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
'        CURSOR, DISPLAY
'-----------------------------------------------------------

'note : tested in bus mode with 4-bit on the STK200
'LCD   -   STK200
'------------------
'D4         D4
'D5         D5
'D6         D6
'D7         D7
'WR         WR
'E          E
'RS         RS
'+5V        +5V
'GND        GND
'V0         V0
' D0-D3 are not connected since 4 bit bus mode is used!

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
```

```
Rem with the config lcdpin statement you can override the compiler
settings

$regfile = "8515def.dat"
$lcd = &HC000
$lcdrs = &H8000
Config Lcdbus = 4

Dim A As Byte
Config Lcd = 16 * 2                                          'configure
lcd screen
'other options are 16 * 2 , 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                          'clear the
LCD display
Lcd "Hello world."                                           'display
this at the top line
Wait 1
Lowerline                                                    'select the
lower line
Wait 1
Lcd "Shift this."                                            'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                            'shift the
text to the right
   Wait 1                                                    'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                             'shift the
text to the left
   Wait 1                                                    'wait a
moment
Next

Locate 2 , 1                                                 'set cursor
position
Lcd "*"                                                      'display
this
Wait 1                                                       'wait a
moment

Shiftcursor Right                                            'shift the
cursor
Lcd "@"                                                      'display
this
Wait 1                                                       'wait a
moment

Home Upper                                                   'select line
1 and return home
Lcd "Replaced."                                              'replace the
text
Wait 1                                                       'wait a
```

```
moment

Cursor Off Noblink                                          'hide cursor
Wait 1                                                      'wait a
moment
Cursor On Blink                                             'show cursor
Wait 1                                                      'wait a
moment
Display Off                                                 'turn
display off
Wait 1                                                      'wait a
moment
Display On                                                  'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                  'goto home
on line three
Home Fourth
Home F                                                      'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                         'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                         'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                  'value into
ACC
!rCall _write_lcd                                           'put it on
LCD
End
```

7.3.32 $LCDVFO

Action

Instruct the compiler to generate very short Enable pulse for VFO displays.

Syntax

$LCDVFO

Remarks

VFO based displays need a very short Enable pulse. Normal LCD displays need a longer pulse. To support VFO displays this compiler directive has been added.

The display need to be instruction compatible with normal HD44780 based text displays.
Noritake is the biggest manufacturer of VFO displays.

The $LCDVFO directive is intended to be used in combination with the LCD routines.

ASM
NONE

See also
NONE

Example
NONE

7.3.33 $LIB

Action
Informs the compiler about the used libraries.

Syntax
$LIB "libname1" [, "libname2"]

Remarks
Libname1 is the name of the library that holds ASM routines that are used by your program. More filenames can be specified by separating the names by a comma.

The specified libraries will be searched when you specify the routines to use with the $EXTERNAL directive.

The search order is the same as the order you specify the library names.

The MCS.LBX will be searched last and is always included so you don't need to specify it with the $LIB directive.

Because the MCS.LBX is searched last you can include duplicate routines in your own library. These routines will be used instead of the ones from the default MCS.LBX library. This is a good way when you want to enhance the MCS.LBX routines. Just copy the MCS.LIB to a new file and make the changes in this new file. When we make changes to the library your changes will be preserved.

Creating your own LIB file

A library file is a simple ASCII file. It can be created with the BASCOM editor, notepad or any other ASCII editor.
When you use BASCOM, make sure that the LIB extension is added to the Options, Environment, Editor, "No reformat extension".

This will prevent the editor to reformat the LIB file when you open it.

The file must include the following header information. It is not used yet but will be later.
copyright = Your name
www = optional location where people can find the latest source
email = your email address
comment = AVR compiler library
libversion = the version of the library in the format : 1.00
date = date of last modification
statement = A statement with copyright and usage information

The routine must start with the name in brackets and must end with the [END].

The following ASM routine example is from the MYLIB.LIB library.

```
[test]
Test:
ldd r26,y+2 ; load address of X
ldd r27,y+3
ld r24,x ; get value into r24
Inc r24 ; value + 1
St x,r24 ; put back
ldd r26,y+0 ; address of Y
ldd r27,y+1
st x,r24 ; store
ret ; ready
[END]
```

After you have saved your library in the **LIB** subdirectory you must compile it with the LIB Manager 132. Or you can include it with the LIB extension in which case you don't have to compile it.

About the assembler.

When you reference constants that are declared in your basic program you need to put a star(*) before the line.

```
' Basic Program
CONST myconst = 7

' asm lib
* sbi portb, myconst
```

By adding the *, the line will be compiled when the basic program is compiled. It will not be changed into object code in the LBX file.
When you use constants you need to use valid BASIC constants:

```
Ldi r24,12
Ldi r24, 1+1
Ldi r24, &B001  ; binary basic
Ldi r24,0b001   ; binary
Ldi r24,&HFF    ; hex basic
Ldi r24,$FF     ; hex
Ldi r24,0xFF    ; hex
```

Other syntax is NOT supported.

See also
$EXTERNAL 634

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


'In order to let this work you must put the mylib.lib file in the LIB
dir
'And compile it to a LBX
'-----------------------------------------------------------------------
--
'define the used library
$lib"mylib.lbx"
'you can also use the original ASM :
'$LIB "mylib.LIB"

'also define the used routines
$external Test

'this is needed so the parameters will be placed correct on the stack
Declare Sub Test(byval X Asbyte , Y Asbyte)

'reserve some space
Dim Z As Byte

'call our own sub routine
Call Test(1 , Z)

'z will be 2 in the used example
End
```

7.3.34 $LOADER

Action
Instruct the compiler to create a boot loader at the specified address.
Can be used for all AVR that support a boot loader like ATMEGA and ATXMEGA chips.

Syntax
$LOADER = address [,BOOTONLY]

Remarks

address	The address where the boot loader is located. You can find this address in the data sheet. In version 2081 a constant named _LOADER_PAGE is created that holds the 64 KB page number.
BOOTONLY	Normally when the BIN and HEX files are created, they are an image of the processor flash memory. The BOOTONLY option will write only the code for the BOOT area to the BIN file. This option has no effect on the HEX file. Writing just the boot portion to a binary file allows to include the boot

	loader code using the $INC directive to a normal program.

A lot of AVR microcontrollers are configured such that it is possible to use a boot loader able to receive firmware updates and to reprogram the Flash memory on demand.
These AVR which support boot loader have a so called boot section.
Normally a chip will start at address 0 when it resets.
This is also called the reset vector.
Chips that have a boot section, split the flash memory in two parts. The boot section is a small part of the normal flash and by setting a fuse bit you select that the chip runs code at the boot sector when it resets instead of the normal reset vector.

The Program Flash memory space of ATXMEGA chips is also divided into Application and Boot sections. Both sections
have dedicated Lock Bits for setting restrictions on write or read/write operations.

ATXMEGA Program Flash memory parts:
1. Application Section for application code
2. Application Table Section for application code or data storage
3. **Boot Section** for application code or bootloader code

 You need to set the fuse bits so the chip jump to the boot loader address at reset (BOOTRST) !

Some chips also have fuse bits to select the size of the boot loader (e.g. 1024 words, 2048 words, 4096 words)

The boot loader start address depends also on the boot size.
You can find following information in the data sheet of the device (example for **ATMEGA644**):

Boot Size	Boot Loader Flash Section	Boot Reset Address (Start Boot Loader Section)
512 words	0x7E00 - 0x7FFF	**$loader** = $7E00
1024 words	0x7C00 - 0x7FFF	**$loader** = $7C00
2048 words	0x7800 - 0x7FFF	**$loader** = $7800
4096 words	0x7000 - 0x7FFF	**$loader** = $7000

For ATXMEGA chips like ATXMEGA32A4 the boot section is part of the Flash Program Memory.
You can find following information in the data sheet of the ATXMEGA device under Flash Program Memory
(example for ATxmega16A4 .....ATxmega128A4):

Chip	Boot Loader Flash Section	Boot Reset Address (Start Boot Loader Section)
ATxmega16A4	0x2000 - 0x7FFF	**$loader** = &H2000
ATxmega32A4	0x4000 - 0x47FF	**$loader** = &H4000
ATxmega64A4	0x8000 - 0x87FF	**$loader** = &H8000
ATxmega128A4	0x10000 - 0x10FFF	**$loader** = &H10000

 An external programmer is needed to program the boot loader into the chip. After the fuse bits are set and the boot loader is programmed you do not need the external programmer anymore for this chip (except you want to change the fuse bits).

The MCS boot loader sample is a serial boot loader that uses the serial port (USART).

With ATXMEGA or with ATMEGA with more then one USART you can choose which USART (COM port) should be used with the boot loader.

For example you can use COM7 with an ATXMEGA:

```
Config Com7 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8          '
Open "COM7:" For Binary As #7
```

 When using another UART as COM1 do not forget to add the Interface number (in this example **#7**) to all the Serial IO functions like `Waitkey(#7)` or `Print #7 , Chr(bstatus);` in the boot loader example

The boot loader uses the X-modem checksum protocol to receive the data. (XModem protocol (packet size = 128))
Most terminal emulators can send X-modem checksum.

The Boot loader sample can upload both normal flash programs and EEPROM images. The Boot loader sends a byte with value of 123 to the AVR Boot loader. This boot loader program then enter the boot loader or will jump to the reset vector (0000) to execute the normal flash program.

When it receives 124 instead of 123, it will upload the EEPROM.
When you select a BIN file the flash will be uploaded. When you select an EEP file, the EEPROM will be uploaded.

The following sample is written so it supports all chips with a boot section.

How you need to use this ATMEGA boot loader example program:

1. Uncomment the Chip type and `Const Loaderchip` you want to use (for example ATMEGA644)

```
$regfile = "m644def.dat"
'$regfile = "m644Pdef.dat"
Const Loaderchip = 644
```

2. Double check the baud rate and COM port you want to use
3. Compile the boot loader example
4. Program it into the chip with an external programmer like AVR ISP MKII
5. Select MCS Bootloader 184 from programmer (select the right COM Port and baud rate)
6. compile a new program or example for this chip
7. reset the chip

Ways to reset the AVR chip:

Hardware reset:
1. Hardware Reset switch/button to GND (manual)
2. MCS Bootloader can set and reset the DTR or RTS line of serial COM port which can be used to reset the AVR (automatic)

Software Reset:
1. Reset with Watchdog Timer (e.g. setting the Watchdog to 16ms, start it and let it time out)

2. With GOTO command (e.g. when ATMEGA644 is used the boot loader start at $7c00 ($loader = $7c00).
   Then you can use:
   ```
   GOTO &H7c00
   ```
   to jump to the boot loader start.
3. With ATXMEGA there is a special register to reset the ATXMEGA via software. See also topic ATXMEGA
4. With MCS Bootloader you can send one or several ASCII character to reset the chip like with string "boot_me". In this case the "boot_me" must be detected in your main application on the AVR and then use for example Watchdog or GOTO to reset the chip.

The boot loader is written to work at a baud rate of 57600. This baud rate works for most chips that use the internal oscillator. But it is best to check it first with a simple program. When you use a crystal you might even use a higher baud rate.
You can change this by changing the baud rate in the boot loader example (take care to use also the same baud rate in the boot loader application (e.g. MCS Bootloader 184) on the PC side)

Now make a new test program and compile it. Press **F4** to start the MCS bootloader 184. You now need to reset the chip so that it will start the boot loader section. The boot loader will send a byte with value of 123 and the Bascom boot loader receives this and thus starts the loader process.

There will be a stand alone boot loader available too. And the sample will be extended to support other AVR chips with boot section too.

 There is a $BOOT directive too. It is advised to use $LOADER as it allows you to write the boot loader in BASIC.

 You can not use interrupts in your boot loader program as the interrupts will point to the reset vector which is located in the lower section of the flash. When you start to writing pages, you overwrite this part.

 Take care when Watchdog is enabled by fuse bits and using a boot loader. You need to reset or deactivate the Watchdog in the boot loader example otherwise the firmware upload could be terminated by watchdog reset !

 If you want to analyze the MCU Control and Status Register to know which reset source caused the reset you need to save this register already in the boot loader example because this register will be cleared and it will be always 0 when you check it at start of your application.

 When you use a boot loader it will use space from the available flash memory. The compiler does not know if you use a boot loader or not. When your program exceeds the available space and runs into the boot sector space, it will overwrite the boot loader.
The $LOADERSIZE 683 directive will take the boot loader size into account so you will get an error when the target file gets too big.

Encryption/Decryption with Bootloader:

You can use for example AES or XTEA (XTEADECODE 1298, XTEAENCODE 1297) in

combination with boot loader examples.
There is an AES with boot loader and AVR-DOS example in the ...BASCOM-AVR\SAMPLES\boot folder (xmega_dos_boot_AES.zip).

See also

$BOOT 612 , $LOADERSIZE 683, MCS Bootloader 184 , CONFIG INTVECTORSELECTION 987 , $BOOTVECTOR 613

There is an example for ATMEGA chips and for ATXMEGA Chips:

ATMEGA Example:

```
'-----------------------------------------------------------------
'                          (c) 1995-2025, MCS
'                        Bootloader.bas
'  This sample demonstrates how you can write your own bootloader
'  in BASCOM BASIC
'  VERSION 2 of the BOOTLOADER. The waiting for the NAK is stretched
'  further a bug was resolved for the M64/M128 that have a big page size
'-----------------------------------------------------------------
'This sample will be extended to support other chips with bootloader
'The loader is supported from the IDE
$crystal = 8000000
'$crystal = 14745600
$baud = 57600                                               'this loader
uses serial com
'It is VERY IMPORTANT that the baud rate matches the one of the boot
loader
'do not try to use buffered com as we can not use interrupts

'possible return codes of the PC bootloader.exe
' -6005    Cancel requested
' -6006    Fatal time out
' -6007    Unrecoverable event during protocol
' -6008    Too many errors during protocol
' -6009    Block sequence error in Xmodem
' -6016    Session aborted

'$regfile = "m8def.dat"

'Const Loaderchip = 8
'$regfile = "m168def.dat"
'Const Loaderchip = 168

'$regfile = "m16def.dat"
'Const Loaderchip = 16

'$regfile = "m32def.dat"
'Const Loaderchip = 32

'$regfile = "m88def.dat"
'Const Loaderchip = 88

'$regfile = "m162def.dat"
'Const Loaderchip = 162

'$regfile = "m8515.dat"
'Const Loaderchip = 8515

'$regfile = "m128def.dat"
'Const Loaderchip = 128
```

```
'$regfile = "m64def.dat"
'Const Loaderchip = 64

'$regfile = "m2561def.dat"
'Const Loaderchip = 2561

'$regfile = "m2560def.dat"
'Const Loaderchip = 2560

'$regfile = "m329def.dat"
'Const Loaderchip = 329

'$regfile = "m324pdef.dat"
'Const Loaderchip = 324

$regfile = "m644def.dat"
'$regfile = "m644Pdef.dat"
Const Loaderchip = 644

#if Loaderchip = 88                                          'Mega88
    $loader = $c00                                          'this
address you can find in the datasheet
    'the loader address is the same as the boot vector address
    Const Maxwordbit = 5
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 168                                         'Mega168
    $loader = $1c00                                         'this
address you can find in the datasheet
    'the loader address is the same as the boot vector address
    Const Maxwordbit = 6
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 32                                          ' Mega32
    $loader = $3c00                                         ' 1024 words
    Const Maxwordbit = 6                                    'Z6 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif
#if Loaderchip = 8                                           ' Mega8
    $loader = $c00                                          ' 1024 words
    Const Maxwordbit = 5                                    'Z5 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif
#if Loaderchip = 161                                         ' Mega161
    $loader = $1e00                                         ' 1024 words
    Const Maxwordbit = 6                                    'Z6 is
maximum bit
#endif
#if Loaderchip = 162                                         ' Mega162
    $loader = $1c00                                         ' 1024 words
    Const Maxwordbit = 6                                    'Z6 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif
```

```
#if Loaderchip = 8515                                       ' Mega8515
    $loader = $c00                                          ' 1024 words
    Const Maxwordbit = 5                                    'Z6 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
    Osccal = &HB3                                           ' the
internal osc needed a new value
#endif

#if Loaderchip = 64                                         ' Mega64
    $loader = $7c00                                         ' 1024 words
    Const Maxwordbit = 7                                    'Z7 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 128                                        ' Mega128
    $loader = &HFC00                                        ' 1024 words
    Const Maxwordbit = 7                                    'Z7 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 2561                                       ' Mega2561
    $loader = &H1FC00                                       ' 1024 words
    Const Maxwordbit = 7                                    'Z7 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 2560                                       ' Mega2560
    $loader = &H1FC00                                       ' 1024 words
    Const Maxwordbit = 7                                    'Z7 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 16                                         ' Mega16
    $loader = $1c00                                         ' 1024 words
    Const Maxwordbit = 6                                    'Z6 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 329                                        ' Mega32
    $loader = $3c00                                         ' 1024 words
    Const Maxwordbit = 6                                    'Z6 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

#if Loaderchip = 324                                        ' Mega32
    $loader = $3c00                                         ' 1024 words
    Const Maxwordbit = 6                                    'Z6 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

```
#endif


#if Loaderchip = 644                                        ' Mega644P
    $loader = $7c00                                        ' 1024 words
    Const Maxwordbit = 7                                   'Z7 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif

Const Maxword =(2 ^ Maxwordbit) * 2                        '128
Const Maxwordshift = Maxwordbit + 1
Const Cdebug = 0                                           ' leave this
to 0

#if Cdebug
   Print Maxword
   Print Maxwordshift
#endif

'Dim the used variables
Dim Bstatus As Byte , Bretries As Byte , Bblock As Byte , Bblocklocal As
 Byte
Dim Bcsum1 As Byte , Bcsum2 As Byte , Buf(128) As Byte , Csum As Byte
Dim J As Byte , Spmcrval As Byte                           ' self
program command byte value
Dim Z As Long                                              'this is the
Z pointer word
Dim Vl As Byte , Vh As Byte                                ' these
bytes are used for the data values
Dim Wrd As Word , Page As Word                             'these vars
contain the page and word address
Dim Bkind As Byte , Bstarted As Byte
'Mega 88 : 32 words, 128 pages



Disable Interrupts                                         'we do not
use ints
'Waitms 100                                                 'wait 100
msec sec
'We start with receiving a file. The PC must send this binary file

'some constants used in serial com
Const Nak = &H15
Const Ack = &H06
Const Can = &H18

'we use some leds as indication in this sample , you might want to
remove it
Config Pinb.2 = Output
Portb.2 = 1                                                'the stk200
has inverted logic for the leds
Config Pinb.3 = Output
Portb.3 = 1

$timeout = 400000                                          'we use a
timeout
'When you get LOADER errors during the upload, increase the timeout
value
'for example at 16 Mhz, use 200000

Bretries = 5                                               'we try 5
```

```
times
Testfor123:
#if Cdebug
    Print "Try " ; Bretries
    Print "Wait"
#endif
Bstatus = Waitkey()                                          'wait for
the loader to send a byte
#if Cdebug
   Print "Got "
#endif

Print Chr(bstatus);

If Bstatus = 123 Then                                       'did we received
value 123 ?
   Bkind = 0                                                'normal flash
loader
   Goto Loader
Elseif Bstatus = 124 Then                                   ' EEPROM
   Bkind = 1                                                ' EEPROM loader
   Goto Loader
Elseif Bstatus <> 0 Then
   Decr Bretries
   If Bretries <> 0 Then Goto Testfor123                    'we test again
End If

For J = 1 To 10                  'this is a simple indication that we
start the normal reset vector
   Toggle Portb.2 : Waitms 100
Next

#if Cdebug
  Print "RESET"
#endif
Goto _reset                     'goto the normal reset vector at address
0


'this is the loader routine. It is a Xmodem-checksum reception routine
Loader:
  #if Cdebug
      Print "Clear buffer"
  #endif
  Do
     Bstatus = Waitkey()
  Loop Until Bstatus = 0

  For J = 1 To 3                        his is a simple indication that
we start the normal reset vector
     Toggle Portb.2 : Waitms 50
  Next

  If Bkind = 0 Then
     Spmcrval = 3 : Gosub Do_spm             ' erase  the first page
     Spmcrval = 17 : Gosub Do_spm            ' re-enable page
  End If

Bretries = 10                                 'number of retries

Do
  Bstarted = 0                                ' we were not started yet
  Csum = 0                                    'checksum is 0 when we
start
```

```
  Print Chr(nak);                                ' first time send a nack
  Do
    Bstatus = Waitkey()                          'wait for status byte

    Select Case Bstatus
       Case 1:                                   ' start of heading, PC is
ready to send
            Incr Bblocklocal                     'increase local block count
            Csum = 1                             'checksum is 1
            Bblock = Waitkey() : Csum = Csum + Bblock     'get block
            Bcsum1 = Waitkey() : Csum = Csum + Bcsum1     'get checksum
first byte
            For J = 1 To 128                             'get 128 bytes
              Buf(j) = Waitkey() : Csum = Csum + Buf(j)
            Next
            Bcsum2 = Waitkey()                           'get second
checksum byte
            If Bblocklocal = Bblock Then                 'are the
blocks the same?
               If Bcsum2 = Csum Then                     'is the
checksum the same?
                  Gosub Writepage                        'yes go write
the page
                  Print Chr(ack);                        'acknowledge
               Else                                      'no match so
send nak
                  Print Chr(nak);
               End If
            Else
               Print Chr(nak);                           'blocks do not
match
            End If
       Case 4:                                           ' end of
transmission , file is transmitted
            If Wrd > 0 And Bkind = 0 Then                'if there was
something left in the page
              Wrd = 0                                    'Z pointer
needs wrd to be 0
               Spmcrval = 5 : Gosub Do_spm               'write page
               Spmcrval = 17 : Gosub Do_spm              ' re-enable
page
            End If
           '  Waitms 100                                ' OPTIONAL
REMARK THIS IF THE DTR SIGNAL ARRIVES TO EARLY
            Print Chr(ack);                            ' send ack and
ready
           Portb.3 = 0                                 ' simple
indication that we are finished and ok
            Waitms 20
            Goto _reset                                ' start new
program
       Case &H18:                                      ' PC aborts
transmission
            Goto _reset                                ' ready
       Case 123 : Exit Do                              'was probably
still in the buffer
       Case 124 : Exit Do
       Case Else
          Exit Do                                      ' no valid
data
    End Select
  Loop
  If Bretries > 0 Then                                 'attempte
left?
```

```
      Waitms 1000
      Decr Bretries                                    'decrease
attempts
   Else
      Goto _reset                                      'reset chip
   End If
Loop

'write one or more pages
Writepage:
 If Bkind = 0 Then
    For J = 1 To 128 Step 2                            'we write 2
bytes into a page
       Vl = Buf(j) : Vh = Buf(j + 1)                   'get Low and
High bytes
!       lds r0, {vl}                                   'store them
into r0 and r1 registers
!       lds r1, {vh}
       Spmcrval = 1 : Gosub Do_spm                     'write value
into page at word address
       Wrd = Wrd + 2                                   ' word address
increases with 2 because LS bit of Z is not used
       If Wrd = Maxword Then                           ' page is full
           Wrd = 0                                     'Z pointer
needs wrd to be 0
           Spmcrval = 5 : Gosub Do_spm                 'write page
           Spmcrval = 17 : Gosub Do_spm                ' re-enable
page

           Page = Page + 1                             'next page
           Spmcrval = 3 : Gosub Do_spm                 ' erase  next
page
           Spmcrval = 17 : Gosub Do_spm                ' re-enable
page
       End If
    Next

 Else                                                  'eeprom
     For J = 1 To 128
       Writeeeprom Buf(j) , Wrd
       Wrd = Wrd + 1
     Next
 End If
 Toggle Portb.2 : Waitms 10 : Toggle Portb.2           'indication
that we write
Return


Do_spm:
   Bitwait Spmcsr.0 , Reset                            ' check for
previous SPM complete
   Bitwait Eecr.1 , Reset                              'wait for
eeprom

   Z = Page                                            'make equal to
page
   Shift Z , Left , Maxwordshift                       'shift to
proper place
   Z = Z + Wrd                                         'add word
!   lds r30,{Z}
!   lds r31,{Z+1}

   #if _romsize > 65536
!       lds r24,{Z+2}
```

```
!      sts rampz,r24                                          ' we need to
set rampz also for the M128
  #endif

  Spmcsr = Spmcrval                                          'assign
register
!  spm                                                       'this is an asm
instruction
!  nop
!  nop
Return


'How you need to use this program:
'1- compile this program
'2- program into chip with sample elctronics programmer
'3- select MCS Bootloader from programmers
'4- compile a new program for example M88.bas
'5- press F4 and reset your micro
' the program will now be uploaded into the chip with Xmodem Checksum
' you can write your own loader.too
'A stand alone command line loader is also available

'How to call the bootloader from your program without a reset ???
'Do
'   Print "test"
'   Waitms 1000
'   If Inkey() = 27 Then
'      Print "boot"
'      Goto &H1C00
'   End If
'Loop

'The GOTO will do the work, you need to specify the correct bootloader
address
'this is the same as the $LOADER statement.
```

ATXMEGA Example:

 NOTICE that there are many Xmega processors and the page size differs.
The example is for the xmega32A4 which uses MAXWORDBIT=7
Other chips require a different value. See the table after the example.

```
'-----------------------------------------------------------------
'                          (c) 1995-2025, MCS
'                       BootloaderXmega32A4.bas
'  This sample demonstrates how you can write your own bootloader
'  in BASCOM BASIC for the XMEGA
'-----------------------------------------------------------------
'The loader is supported from the IDE
$crystal = 32000000                                           ' xmega128
is running on 32 MHz
$regfile = "xm32a4def.dat"
$lib "xmega.lib"                                              ' add a
reference to this lib

'first enabled the osc of your choice
Config Osc = Disabled , 32mhzosc = Enabled                    'internal 2
```

```
MHz and 32 MHz enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1  ' we will
use 32 MHz and divide by 1 to end up with 32 MHz

$loader = &H4000                                            ' bootloader
starts after the application

Config Com1 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8  ' use USART C0
'COM0-USARTC0, COM1-USARTC2, COM2-USARTD0. etc.
Config Portc.3 = Output                                     'define TX
as output
Config Pinc.2 = Input

Const Maxwordbit = 7                                        ' Z7 is
maximum bit                                  '
Const Maxword =(2 ^ Maxwordbit) * 2                         '128
Const Maxwordshift = Maxwordbit + 1
Const Cdebug = 0                                            ' leave this
to 0

'Dim the used variables
Dim Bstatus As Byte , Bretries As Byte , Bmincount As Byte , Bblock As
Byte , Bblocklocal As Byte
Dim Bcsum1 As Byte , Bcsum2 As Byte , Buf(128) As Byte , Csum As Byte
Dim J As Byte , Spmcrval As Byte                            ' self
program command byte value
Dim Z As Long                                               'this is the
Z pointer word
Dim Vl As Byte , Vh As Byte                                 ' these
bytes are used for the data values
Dim Wrd As Word , Page As Word                              'these vars
contain the page and word address

Disable Interrupts                                          'we do not
use ints

'We start with receiving a file. The PC must send this binary file

'some constants used in serial com
Const Nak = &H15
Const Ack = &H06
Const Can = &H18

$timeout = 300000                                           'we use a
timeout
'When you get LOADER errors during the upload, increase the timeout
value
'for example at 16 Mhz, use 200000

Bretries = 5 : Bmincount = 3                                'we try 10
times and want to get 123 at least 3 times
Do
  Bstatus = Waitkey()                                       'wait for
the loader to send a byte

  If Bstatus = 123 Then                                     'did we
received value 123 ?
     If Bmincount > 0 Then
        Decr Bmincount
     Else
        Print Chr(bstatus);
        Goto Loader                                         ' yes so run
```

```
bootloader
     End If
  Else                                                    'we received
some other data
     If Bretries > 0 Then                                 'retries
left?
        Bmincount = 3
        Decr Bretries
     Else
        Rampz = 0
        Goto Proces_reset                        'goto the normal reset
vector at address 0
     End If
  End If
Loop

'this is the loader routine. It is a Xmodem-checksum reception routine
Loader:
  Do
     Bstatus = Waitkey()
  Loop Until Bstatus = 0

  Spmcrval = &H20 : Gosub Do_spm                          ' erase  all
app pages


Bretries = 10                                             'number of
retries

Do
  Csum = 0                                                'checksum is
0 when we start
  Print Chr(nak);                                         ' first time
send a nack
  Do

    Bstatus = Waitkey()                                   'wait for
status byte

    Select Case Bstatus
       Case 1:                                            ' start of
heading, PC is ready to send
            Incr Bblocklocal                              'increase
local block count
            Csum = 1                                      'checksum is
1
            Bblock = Waitkey() : Csum = Csum + Bblock     'get block
            Bcsum1 = Waitkey() : Csum = Csum + Bcsum1     'get
checksum first byte
            For J = 1 To 128                              'get 128
bytes
              Buf(j) = Waitkey() : Csum = Csum + Buf(j)
            Next
            Bcsum2 = Waitkey()                            'get second
checksum byte

            If Bblocklocal = Bblock Then                  'are the
blocks the same?

               If Bcsum2 = Csum Then                      'is the
checksum the same?
                  Gosub Writepage                         'yes go
write the page
                  Print Chr(ack);                         'acknowledge
```

```
               Else                                          'no match so
send nak
                  Print Chr(nak);
               End If
            Else
               Print Chr(nak);                               'blocks do
not match
            End If
       Case 4:                                               ' end of
transmission , file is transmitted
            If Wrd > 0 Then                                  'if there
was something left in the page
               Wrd = 0                                       'Z pointer
needs wrd to be 0
                Spmcrval = &H24 : Gosub Do_spm               'write page
            End If
            Print Chr(ack);                                  ' send ack
and ready
            Waitms 20
            Goto Proces_reset
       Case &H18:                                            ' PC aborts
transmission
            Goto Proces_reset                                ' ready
       Case 123 : Exit Do                                    'was
probably still in the buffer
       Case 124 : Exit Do
       Case Else
          Exit Do                                            ' no valid
data
    End Select
  Loop
  If Bretries > 0 Then                                       'attempts
left?
     Waitms 1000
     Decr Bretries                                           'decrease
attempts
  Else
     Goto Proces_reset                                       'reset chip
  End If
Loop

'write one or more pages
Writepage:
   For J = 1 To 128 Step 2                                   'we write 2
bytes into a page
      Vl = Buf(j) : Vh = Buf(j + 1)                          'get Low and
High bytes
!      lds r0, {vl}                                          'store them
into r0 and r1 registers
!      lds r1, {vh}
      Spmcrval = &H23 : Gosub Do_spm                         'write value
into page at word address
      Wrd = Wrd + 2                                          ' word
address increases with 2 because LS bit of Z is not used
      If Wrd = Maxword Then                                  ' page is
full
          Wrd = 0                                            'Z pointer
needs wrd to be 0
          Spmcrval = &H24 : Gosub Do_spm                     'write page
          Page = Page + 1                                    'next page
      End If
   Next
Return
```

```
Do_spm:
  Z = Page                                                    'make equal
to page
  Shift Z , Left , Maxwordshift                               'shift to
proper place
  Z = Z + Wrd                                                 'add word
!  lds r30,{Z}
!  lds r31,{Z+1}

  #if _romsize > 65536
!      lds r24,{Z+2}
!      sts rampz,r24                                           ' we need
to set rampz also for the M128
  #endif

  Nvm_cmd = Spmcrval
  Cpu_ccp = &H9D
!  spm                                                         'this is an
asm instruction
Do_spm_busy:
!   lds r23, NVM_STATUS
!   sbrc r23,7 ;if  busy bit is cleared  skip next instruc tion
!   rjmp do_spm_busy
Return

Proces_reset:
  Rampz = 0
  Goto _reset                                                 'start at
address 0
```

PAGE SIZE

Processor	Pagesize	Maxwordbit
ATxmega128A1	512	8
ATxmega128A1U	512	8
ATxmega128A3	512	8
ATxmega128A3U	512	8
ATxmega128A4U	256	7
ATxmega128B1	256	7
ATxmega128B3	256	7
ATxmega128C3	512	8
ATxmega128D3	512	8
ATxmega128D4	256	7
ATxmega16A4	256	7
ATxmega16A4U	256	7
ATxmega16C4	256	7
ATxmega16D4	256	7
ATxmega16E5	128	6
ATxmega192A3	512	8
ATxmega192A3U	512	8
ATxmega192C3	512	8
ATxmega192D3	512	8
ATxmega256A3	512	8
ATxmega256A3B	512	8
ATxmega256A3BU	512	8
ATxmega256A3U	512	8
ATxmega256C3	512	8
ATxmega256D3	512	8
ATxmega32A4	256	7

ATxmega32A4U	256	7
ATxmega32C3	256	7
ATxmega32C4	256	7
ATxmega32D3	256	7
ATxmega32D4	256	7
ATxmega32E5	128	6
ATxmega384C3	512	8
ATxmega384D3	512	8
ATxmega64A1	256	7
ATxmega64A1U	256	7
ATxmega64A3	256	7
ATxmega64A3U	256	7
ATxmega64A4U	256	7
ATxmega64B1	256	7
ATxmega64B3	256	7
ATxmega64C3	256	7
ATxmega64D3	256	7
ATxmega64D4	256	7
ATxmega8E5	128	6

7.3.35 $LOADERSIZE

Action
Instruct the compiler that a boot loader is used so it will not overwrite the boot space.

Syntax
$LOADERSIZE = size

Remarks

size	The amount of space in bytes that is used by the boot loader.

When you use a boot loader it will use space from the available flash memory. The compiler does not know if you use a boot loader or not. It also does not know how you have set the fuse bits, so it is impossible to know how big the bootloader size is. When your program exceeds the available space and runs into the boot sector space, it will overwrite the boot loader.
The $loadersize directive will take the boot loader size into account so you will get an error when the target file gets too big.

When you select the MCS boot loader as programmer the IDE also will take into account the specified boot loader size.
The directive can be used when you have a different programmer selected. For example an external programmer that does not know about the boot size.

 Do not use this directive in the bootloader program itself. You will get an error 344 in that case. $LOADERSIZE is only intended to be used in normal applications.

See also
$LOADER 667 , $BOOT 612

ASM
NONE

Example
```
NONE
```

7.3.36 $MAP

Action
Will generate label info in the report.

Syntax
$MAP

Remarks
The $MAP directive will put an entry for each line number with the address into the report file. This info can be used for debugging purposes with other tools.

See also
NONE

ASM
NONE

Example
$MAP

The report file will now contain the following section :

```
Code map
--------------------------------------------------------------------------------
Line                                    Address(hex)
--------------------------------------------------------------------------------
1                                       0
9                                       36
26                                      39
30                                      3B
31                                      3E
32                                      48
33                                      4B
36                                      50
37                                      56
42                                      5B
43                                      6C
44                                      7D
45                                      80
46                                      81
```

7.3.37 $NOCOMPILE

Action
Instruct the compiler **not** to compile the file.

Syntax
$NOCOMPILE

Remarks
This looks like an odd directive. Since you can split your program in multiple files, and you can create configuration files, you might open a file and try to compile it. Only normal project files can be compiled and you will get a number of errors and also unwanted files like error, report, etc.
To prevent that you compile a file that is intended to be included, you can insert the $NOCOMPILE directive.
Then the file will only be compiled when it is called from your main file, or other include file.

A file that is opened as thus the main file, and which includes the $NOCOMP directive, can not be compiled.
The IDE will see it as a successful compilation. This is important for the Batch Compiler.

See also
Batch Compiler 135

Example
```
$NOCOMPILE
```

7.3.38 $NOFRAMEPROTECT

Action
This directive will disable interrupt frame protection.

Syntax
$NOFRAMEPROTECT

Remarks
See the new preferred switch : $FRAMEPROTECT 637

See also
$FRAMEPROTECT 637

Example
NONE

7.3.39 $NOINIT

Action
Instruct the compiler to generate code without initialization code.

Syntax
$NOINIT

Remarks
$NOINIT is only needed in rare situations. It will instruct the compiler not to add initialization code. But that means that you need to write your own code then. $NOINIT was added in order to support boot loaders. But the new $LOADER directive can better be used as it does not require special ASM knowledge.

See also
$LOADER 667

Example
NONE

7.3.40 $NORAMCLEAR

Action
Instruct the compiler to not generate initial RAM clear code.

Syntax
$NORAMCLEAR

Remarks
Normally the SRAM is cleared in the initialization code. When you don't want the SRAM to be cleared(set to 0) you can use this directive.

Because all variables are automatically set to 0 or ""(strings) without the $NORAMCLEAR, using $NORAMCLEAR will set the variables to an unknown value. That is, the variables will probably set to FF but you cannot count on it.

When you have a battery back upped circuit, you do not want to clear the RAM at start up. So that would be a situation when you could use $NORAMCLEAR.

See also
$NOINIT 686

7.3.41 $NORAMPZ

Action
This compiler directive disables RAMPZ clearing.

Syntax
$NORAMPZ

Remarks
Processors with more then 64 KB of memory need to set the RAMPZ register in order to point to the proper 64 KB page.
If the RAMPZ register is used, it will be cleared when it is used for something different then accessing the flash.
BASCOM uses the Z register to access flash memory or RAM memory. Since processors with external memory capability can access more then 64KB of RAM, the RAMPZ must be set/cleared when accessing this memory.
Otherwise accessing the flash code could result in a change of RAMPZ, and after this, accessing the RAM would not point to the proper place in memory.
But setting this register requires extra code. When your application just fitted into a M128 or M256 and you do not want this RAMPZ handling because your application works fine, then you can use this $NORAMPZ directive.
To see if your processor

See also
NONE

Example
NONE

7.3.42 $NOTRANSFORM

Action
This option controls transformation of unsupported ASM mnemonics.

Syntax
$NOTRANSFORM ON|OFF

Remarks
By default, assembler mnemonics that are not supported for a chip or register are transformed into different assembler mnemonics.
The IN and OUT instructions for example only work on hardware registers with an address lower then 64. Most PORT registers are located in this lower address space, but there are many chips that have more ports which are located in extended memory. For such chips, using a IN or OUT on an extended address would result in a failure.
Thus the compiler changes IN into an LDS and an OUT into an STS. When a register is required, R23 will be used except for SBIS/SBIC, these instructions use R0 when required.

When you develop some ASM code, you might want to get an error when you are using an instruction the wrong way. For this purpose you can turn off the transformation.
$NOTRANSFORM ON will turn off the transformation. And with $NOTRANSFORM OFF you can turn it back on.
You should only use this option in your own code. When you use it on your whole

program, it will not compile since the bascom libraries which use CBI, SBI, SBIS, IN, OUT, etc. will use the transformation.

See also
NONE

Example
NONE

7.3.43 $NOTYPECHECK

Action
This directive will turn off type checking

Syntax
$NOTYPECHECK

Remarks
Type checking is performed on some operations. It is turned on by default. With the $NOTYPECHECK you can turn this feature off.

See also
$TYPECHECK 710

Example
NONE

7.3.44 $PROJECTTIME

Action
This directive will keep track of time you spend on the source.

Syntax
$PROJECTTIME

Remarks
Keeping track of project time is the only purpose of this directive. It will be ignored by the compiler.
When the IDE finds the $PROJECTTIME directive, it will count the minutes you spend on the code.
Each time you save the code, the updated value will be shown.
The IDE will automatic insert the value after $PROJECTTIME.

So how does this work?
When you type, you start a timer. When there are no keystrokes for 2 minutes, this process stops. It is started automatic as soon as you start typing.
So when you type a character each minute, each minute will be counted a a full

minutes of work.

The time is counted and shown in minutes.

While you can edit the value in the source, it will be changed as soon as you save the source.

See also
NONE

Example
```
$PROJECTTIME
```

7.3.45 $PROG

Action
Directive to auto program the lock and fuse bits.

Syntax old AVR
$PROG LB, FB , FBH , FBX

Syntax Xmega
$PROG LB, F0 , F1 , F2 , F3 ,F4 , F5

Remarks
While the lock and fuse bits make the AVR customizable, the settings for your project can give some problems.
The $PROG directive will create a file with the project name and the PRG extension.

Every time you program the chip, it will check the lock and fuse bit settings and will change them if needed.
So in a new chip, the lock and fuse bits will be set automatically. A chip that has been programmed with the desired settings will not be changed.

The programmer has an option to create the PRG file from the current chip settings.

The LB, FH, FBH and FBX values are stored in hexadecimal format in the PRJ file.
You may use any notation as long as it is a numeric constant.

Some chips might not have a setting for FBH or FBX, or you might not want to set all values. In that case, do NOT specify the value. For example:

```
$PROG &H20 ,,,
```

This will only write the Lockbit settings.

```
$PROG ,,&H30,
```

This will only write the FBH settings.

LB	Lockbit settings

FB	Fusebit settings
FBH	Fusebit High settings
FBX	Extended Fusebit settings

Sometimes the data sheet refers to the Fusebit as the Fusebit Low settings.

The $PROG setting is only supported by the AVRISP, STK200/300, Sample Electronics and Universal MCS Programmer Interface. The USB-ISP programmer also supports the $PROG directive.

 When you select the wrong Fuse bit, you could lock your chip. For example when you choose the wrong oscillator option, it could mean that the micro expects an external crystal oscillator. But when you connect a simple crystal, it will not work.
In these cases where you can not communicate with the micro anymore, the advise is to apply a clock signal to X1 input of the micro.
You can then select the proper fuse bits again.
When you set the Lock bits, you can not read the chip content anymore. Only after erasing the chip, it could be reprogrammed again.
 Once the lock bits and fuse bits are set, it is best to remark the $PROG directive. This because it takes more time to read and compare the bits every time.

Xmega
The Xmega has one lock byte and 6 fuse bytes. For an Xmega the Write PRG option will write the correct code.

Xtiny
The Xtiny has way more fuses and has a special CONFIG FUSES statement code.

See also
Programmers 164 , CONFIG FUSES 962 , $PROGRAMMER 690

7.3.46 $PROGRAMMER

Action
Will set the programmer from the source code.

Syntax
$PROGRAMMER = number [,"COMx"]
$PROGRAMMER = number [,"serial"]

Remarks

Number	A numeric constant that identifies the programmer.
COMx	An optional parameter in double quotes that specifies the COM port to use. Future versions might offer additional options. Example : "COM3"

serial	For MCSEDBG this can be the product name with serial number. Use this when you have multiple programmers connected.

The $PROGRAMMER directive requires that you set the option 'Use New Method' in Options, Environment, IDE.

The $PROGRAMMER directive will set the programmer just before it starts programming.
When you press **F4** to program a chip, the selected programmer will be made active. This is convenient when you have different projects open and use different programmers.
But it can also lead to frustration as you might think that you have the 'STK200' selected, and the directive will set it to USB-ISP.

The following values can be used :

Value	Programmer
0	AVR-ISP programmer(old AN 910) *
1	STK200/STK300 *
2	PG302 *
3	External programmer
4	Sample Electronics *
5	Eddie Mc Mullen *
6	KITSRUS K122 *
7	STK500
8	Universal MCS Interface *
9	STK500 extended
10	Lawicel Bootloader *
11	MCS USB
12	USB-ISP I *
13	MCS Bootloader
14	Proggy *
15	FLIP (Atmel)
16	USBprog Programmer/ AVR ISP mkII (Atmel)
17	Kamprog for AVR
18	MyAVR MKII/AVR910
19	USBASP
20	JTAG MKII
21	STK600
22	ARDUINO (using stk500v1 protocol)
23	ARDUINO V2 (using stk500v2 protocol)
24	MINI-MAX/AVR-C (BIPOM)
25	mySmart USB light STK500 mode
26	MSC UPDI programmer
27	MCS EDBG programmer

* - not recommended for purchase/use

The file **prog.inc** in the \INC subfolder contains constants for all programmers.
We recommend to use the constant since it is more clear which programmer is used.

An additional parameter can be used to specify the COM port to use. Like :

$PROGRAMMER 26, "COM45"
Notice that there is no double point after the COM port number.

For the MCS EDBG programmer you can specify an optional product name with serial number.
For example :
$programmer = prgMCSEDBG,"mEDBG CMSIS-DAP:SNAP"

See also
$PROG 689

ASM
NONE

Example
```
$regfile = "AVRX64EA28.dat"
$crystal = 20000000
$hwstack = 64
$swstack = 64
$framesize = 64

$programmer = Prgmcsedbg , "nEDBG CMSIS-DAP:MC020059804RYN000236"
'               use MCS EDBG programmer. The USB product name is
nEDBG CMSIS-DAP and the serial is MC020059804RYN000236
```

7.3.47 $REDUCEIVR

Action
This directive will inform the compiler to reduce the IVR (interrupt vector table) to the smallest possible size.

Syntax
$REDUCEIVR

Remarks
The flash memory of the processor always starts with the IVR (interrupt vector table). The user code is placed after this table.
So what is this IVR ?
This table contains an address for each interrupt. When an interrupt occurs, the processor will jump to a specific and fixed address in code memory.
The address depends on the interrupt itself, and on the processor. For example the MEGA88 has 25 interrupt sources. You can find them in the dat file :

INTname1=INT0,$001,EIMSK.INT0,EIFR.INTF0
INTname2=INT1,$002,EIMSK.INT1,EIFR.INTF1
INTname3=PCINT0,$003,PCICR.PCIE0,PCIFR.PCIF0
INTname4=PCINT1,$004,PCICR.PCIE1,PCIFR.PCIF1
INTname5=PCINT2,$005,PCICR.PCIE2,PCIFR.PCIF2
INTname6=WDT@WATCHDOG,$006,WDTCSR.WDIE,WDTCSR.WDIF

```
INTname7=OC2A@COMPARE2A,$007,TIMSK2.OCIE2A,TIFR2.OCF2A
INTname8=OC2B@COMPARE2B,$008,TIMSK2.OCIE2B,TIFR2.OCF2B
INTname9=OVF2@TIMER2,$009,TIMSK2.TOIE2,TIFR2.TOV2
INTname10=ICP1@CAPTURE1,$00A,TIMSK1.TICIE1,TIFR1.ICF1
INTname11=OC1A@COMPARE1A,$00B,TIMSK1.OCIE1A,TIFR1.OCF1A
INTname12=OC1B@COMPARE1B,$00C,TIMSK1.OCIE1B,TIFR1.OCF1B
INTname13=OVF1@TIMER1,$00D,TIMSK1.TOIE1,TIFR1.TOV1
INTname14=OC0A@COMPARE0A,$00E,TIMSK0.OCIE0A,TIFR0.OCF0A
INTname15=OC0B@COMPARE0B,$00F,TIMSK0.OCIE0B,TIFR0.OCF0B
INTname16=OVF0@TIMER0,$010,TIMSK0.TOIE0,TIFR0.TOV0
INTname17=SPI,$011,SPCR.SPIE,SPSR.SPIF
INTname18=URXC@SERIAL,$012,UCSR0B.RXCIE0,UCSR0A.RXC0
INTname19=UDRE,$013,UCSR0B.UDRIE0,UCSR0A.UDRE0
INTname20=UTXC,$014,UCSR0B.TXCIE0,UCSR0A.TXC0
INTname21=ADCC@ADC,$015,ADCSRA.ADIE,ADCSRA.ADIF
INTname22=ERDY,$016,EECR.EERIE
INTname23=ACI,$017,ACSR.ACIE,ACSR.ACI
INTname24=TWI,$018,TWCR.TWIE,TWCR.TWINT
INTname25=SPM,$019,SPMCSR.SPMIE
```

You can see that INT0 comes first. And that the address is $001 which is &H0001. So when INT0 occurs, the processor will jump to address 1.
When you did not define an ISR (ON INT0) , the compiler will insert a RETI instruction. So nothing bad will happen to your code.
When you did define an ISR , the compiler will insert a JUMP to your interrupt routine. When your interrupt ends, the RETI will let the processor continue where it was when the interrupt occurred.
In the example when we only use the ISR with the lowest address all other addresses in the table would get a RETI instruction. And the user code could start at &H1A (one address after $19).
Now that is not so bad but there are also processors with bigger tables and with tables that require 2 words for a JUMP. You waste a lot of space this way.

So what does $REDUCEIVR do? It will determine which interrupt you have used has the highest address. And it will use the address after that as the user code start.
This means that if we use only INT0 and we use $REDUCEIVR, the user code will start at address &H2 ($2). So you will save a lot of code space this way.
Ok so why isn't this enabled by default? There is a catch : when your code has an interrupt enabled and there is no matching ON <INT> the processor will jump into the user code and this will create a crash almost for sure.
So our advise : use this when you understand what this option does, and use it when your application is finished. In any case, retest the complete application when the option is enabled.

See also

$LOADER 667 , CONFIG INTVECTORSELECTION 987 , $BOOTVECTOR 613

Example

$REDUCEIVR

7.3.48 $REGFILE

Action
Instruct the compiler to use the specified register file instead of the selected dat file.

Syntax
$REGFILE = "name"

Remarks

Name	The name of the register file. The register files are stored in the BASCOM-AVR application directory and they all have the DAT extension. The register file holds information about the chip such as the internal registers and interrupt addresses. The register file info is derived from atmel definition files.

The $REGFILE statement overrides the setting from the Options, Compiler, Chip menu.
The settings are stored in a <project>.CFG file.

The $REGFILE directive must be the first statement in your program. It may not be put into an included file since only the main source file is checked for the $REGFILE directive.

 It is good practice to use the $REGFILE directive. It has the advantage that you can see in the source which chip it was written for. The $REGFILE directive is also needed when the PinOut 93 viewer or the PDF 97 viewer is used.

The register files contain the hardware register names from the micro. They also contain the bit names. These are constants that you may use in your program. But the names can not be used to dim a variable for example.

Example :
DIM PORTA As Byte
This will not work since PORTA is a register constant.

See also
$SWSTACK 705 , $HWSTACK 648 , $FRAMESIZE 641, Memory usage 267

ASM
NONE

Example
```
$REGFILE = "8515DEF.DAT"
```

7.3.49 $RESOURCE

Action
Instruct the compiler to use a special resource file for multi language support.

Syntax

$RESOURCE [DUMP] "lang1" [, "lang2"]
$RESOURCE ON | OFF

Remarks

lang1	This is the name of the first and default language. You can add a maximum of 8 languages. The names will be used in the resource editor. But they are only intended as a reference. The resource names will not end up in your application. They are used for the column names in the resource editor.
lang2	The second language. You can add multiple languages separated by a comma. The language must be specified within double quotes.
ON	This will turn on the languages resource handling. In some cases you need to turn the language handling ON or OFF which is explained later
OFF	This will turn OFF the language handling
DUMP	This mode will create a <project>.BCS file which contains all used string constants

Some applications require that the interface is available in multiple languages. You write your application the same way as you always do.
When it is ready, you can add the $RESOURCE directive to make the application suited for multiple languages.
The $RESOURCE option will generate a BYTE variable named LANGUAGE. You can change the value in your application. The compiler will take care that the proper string is shown.
But first you need to translate the strings into the languages of your choice.
For this purpose you can use the Resource Editor. The Resource Editor 140 can import a BCS file (BASCOM String file) which contains the languages and the strings.
You can then add a string for all languages.
So first make sure your application works. Then compile using the $RESOURCE DUMP option.

When you test the languages.bas sample the content will look like this :

```
"English" , "Dutch" , "German" , "Italian"
"Multi language test"
"This"
" is a test"
"Name "
"Hello "
```

As you can see, the first line contains the languages. The other lines only contain a string. Each string is only stored once in BASCOM. So even while "Mark" can have multiple meanings, it will only end up once in the BCS file.
After you have translated the strings, the content of the BCR (BASCOM Resource) file will look like :

```
"English","Dutch","German","Italian"
"This","Dit","Dies","Questo"
"Name ","Naam","Name","Nome"
"Multi language test","Meertalen test","","Test multilingua"
"Hello ","Hallo","Hallo","Ciao"
" is a test"," is een test","ist ein test","è un test"
"mark","Mark","Marcus","Marco"
```

You may edit this file yourself, using Notepad or you can use the Resource Editor.

Untranslated strings will be stored as "". Untranslated strings will be shown in the original language !

Now recompile your project and the compiler will handle every string it will find in the resource file (BCR) in a special way. Strings that are not found in the BCR file, are not processed and handled like normal. For example when you have a PRINT "check this out" , and you did not put that in the BCR file, it will show the same no matter which value the LANGUAGE variable has.

But for each string found in the BCR file, the compiler will show the string depending on the LANGUAGE variable. When one of the languages is not translated, it will show as the original language.
When LANGUAGE is 0, it will show the first string (the string from the first column).
When languages is 1, it will show the string from the second column, and so on.
You must take care that the LANGUAGE variables has a valid value.

So by switching/changing 1 variable, you can change the language in the entire application. Strings are used for PRINT, LCD and other commands. It will work on every string that is in the BCR file. But that also brings us to the next option.

Image this code :

```
   If S = "mark" Then
      Print "we can not change names"
   End If
```

As you can see, we use a string. The code will fail if the string is translated (and is different in each language). You can simply remove the this string from the Resource file. But when you also need the word "mark" in the interface, you have a problem.
For this purpose you can turn off the resource handling using $RESOURCE OFF
The compiler will then not process the code following the directive with the special resource handling.
And when you are done, you can turn the resource handling on again
using $RESOURCE ON.

See also

Resource Editor 140

Example

```
'-------------------------------------------------------------------------------
'                          language.bas
'                 (c) 1995-2025 , MCS Electronics
'This example will only work with the resource add on
'resources are only needed for multi language applications
'By changing the LANGUAGE variable all strings used will be shown in the proper lan
'-------------------------------------------------------------------------------
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200

'a few steps are needed to create a multi language application
'STEP 1, make your program as usual
'STEP 2, generate a file with all string resources using the $RESOURCE DUMP directi
'$resource Dump , "English" , "Dutch" , "German" , "Italian" 'we will use 4 languag
'STEP 3, compile and you will find a file with the BCS extesion
```

```
'STEP 4, use Tools, Resource Editor and inport the resources
'STEP 5, add languages, translate the original strings
'STEP 6, compile your program this time with specifying the languages without the D

$resource "English" , "Dutch" , "German" , "Italian"
'this must be done before you use any other resource !
'in this sample 4 languages are used
'this because all resources found are looked up in the BCR file(BasCom Resource)
Dim S As String * 20
Dim B As Byte

Print "Multi language test"
Do
   Print "This" ;
   S = " is a test" : Print S
   Input "Name " , S
   Print "Hello " ; S


   'now something to look out for !
   'all string data not found in the BCR file is not resourced. so there is no prob
   If S = "mark" Then
      Print "we can not change names"
   End If

   'but if you want to have "mark" resourced for another sentence you have a proble
   'the solution is to turn off resourcing
   $resource Off
   Print "mark"
   If S = "mark" Then
      Print "we can not change names"
   End If
   $resource On

   Language = Language + 1
   If Language > 3 Then Language = 0
Loop
```

7.3.50 $ROMSTART

Action

Instruct the compiler to generate a hex/bin file that starts at the specified address.

Syntax

$ROMSTART = address

Remarks

Address	The address where the code must start. By default the first address is 0.

The $ROMSTART directive is an inheritance from BASCOM-8051. In the AVR it did not have any meaning.
In the 8051 you can use and relocate external memory. This is not possible in the AVR and hence there is no practical usage.
For a bootloader used with AVR and XMega you can use the $LOADER directive.
Only use $ROMSTART with the Xtiny platform and when you have a boot loader !

In version 2083 where XTINY is supported the $ROMSTART has a new purpose. Since the boot loading mechanism in the XTINY differs from the other AVR processors, the $ROMSTART can be used to relocate the address.
The memory map starts with the BOOT area, followed by the application area. This means that a boot loader starts at &H0000 and a normal application will start after that.
So in order to use a bootloader, your normal code need to be compiled with the $ROMSTART directive. When the boot loader uses 1024 bytes, it means that the normal application starts at the WORD address which is halve of the byte address and in this case would be &H200 (1024 dec=400 hex).

The simulator supports $ROMSTART relocated code.

See also
$LOADER 667 , Using a BOOTLOADER 285

ASM
NONE

Example
```
$ROMSTART = &H200 'xtiny boot loader
```

7.3.51 $SERIALINPUT

Action
Specifies that serial input must be redirected.

Syntax
$SERIALINPUT = label

Remarks

Label	The name of the assembler routine that must be called when a character is needed by the INPUT routine. The character must be returned in R24.

With the redirection of the INPUT command, you can use your own input routines.

This way you can use other devices as input devices.
Note that the INPUT statement is terminated when a RETURN code (13) is received.

By default when you use INPUT or INKEY(), the compiler will expect data from the COM port. When you want to use a keyboard or remote control as the input device you can write a custom routine that puts the data into register R24 once it needs this data.

See also
$SERIALOUTPUT 701

Example

```
'-----------------------------------------------------------------------
---------
'name                     : $serialinput.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates $SERIALINPUT redirection of
serial input
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"

'define used crystal
$crystal = 4000000

$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                'default use
10 for the SW stack
$framesize = 40                                              'default use
40 for the frame space

'dimension used variables
Dim S As String * 10
Dim W As Long

'inform the compiler which routine must be called to get serial
characters
$serialinput = Myinput

'make a never ending loop
Do
  'ask for name
  Input "name " , S
  Print S
  'error is set on time out
  Print "Error " ; Err
Loop

End

'custom character handling  routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and restore
'all registers so we can use all BASIC statements
'$SERIALINPUT requires that the character is passed back in R24
Myinput:
  Pushall                                                    'save all
registers
  W = 0                                                      'reset
counter
Myinput1:
   Incr W                                                    'increase
counter
!   Sbis USR, 7                                             ' Wait for
character
!   Rjmp myinput2                                           'no charac
waiting so check again
   Popall                                                    'we got
something
   Err = 0                                                   'reset error
```

```
!   In _temp1, UDR                                          ' Read
character from UART
   Return                                                  'end of
routine
Myinput2:
   If W > 1000000 Then                                     'with 4 MHz
ca 10 sec delay
!      rjmp Myinput_exit                                    'waited too
long
   Else
      Goto Myinput1                                        'try again
   End If
Myinput_exit:
  Popall                                                   'restore
registers
  Err = 1                                                  'set error
variable
!  ldi R24, 13                                             'fake enter
so INPUT will end
Return
```

7.3.52 $SERIALINPUT1

Action

Specifies that serial input of the second UART must be redirected.

Syntax

$SERIALINPUT1 = label

Remarks

Label	The name of the assembler routine that must be called when a character is needed from the INPUT routine. The character must be returned in R24.

With the redirection of the INPUT command, you can use your own input routines.

This way you can use other devices as input devices.
Note that the INPUT statement is terminated when a RETURN code (13) is received.

By default when you use INPUT or INKEY(), the compiler will expect data from the COM2 port. When you want to use a keyboard or remote control as the input device you can write a custom routine that puts the data into register R24 once it asks for this data.

See also

$SERIALOUTPUT1 702 , $SERIALINPUT 698 , $SERIALOUTPUT 701

Example

See the $SERIALINPUT 698 sample

7.3.53 $SERIALINPUT2LCD

Action
This compiler directive will redirect all serial input to the LCD display instead of echoing to the serial port.

Syntax
$SERIALINPUT2LCD

Remarks
You can also write your own custom input or output driver with the $SERIALINPUT 698 and $SERIALOUTPUT 701 statements, but the $SERIALINPUT2LCD is handy when you use a LCD display. By adding only this directive, you can view all output form routines such as PRINT, PRINTBIN, on the LCD display.

See also
$SERIALINPUT 698 , $SERIALOUTPUT 701 , $SERIALINPUT1 700 , $SERIALOUTPUT1 702

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.5 , Db6 = Portb.6 ,
Db7 = Portb.7 , E = Portc.7 , Rs = Portc.6

$serialinput2lcd
Dim V As Byte
Do
  Cls
  Input "Number " , V                                    'this will
go to the LCD display
Loop
```

7.3.54 $SERIALOUTPUT

Action
Specifies that serial output must be redirected.

Syntax
$SERIALOUTPUT = label

Remarks

Label	The name of the assembler routine that must be called when a character is send to the serial buffer (UDR). The character is placed into R24.

With the redirection of the PRINT and other serial output related commands, you can use your own routines.
This way you can use other devices as output devices.

See also
$SERIALINPUT 698 , $SERIALINPUT2LCD 701 , $SERIALINPUT1 700 , $SERIALOUTPUT1 702

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

$serialoutput = Myoutput
'your program goes here
Do
  Print "Hello"
Loop
End

myoutput:
'perform the needed actions here
'the data arrives in R24
'just set the output to PORTB
  !outportb,r24
!ret
```

7.3.55 $SERIALOUTPUT1

Action
Specifies that serial output of the second UART must be redirected.

Syntax
$SERIALOUTPUT1 = label

Remarks

Label	The name of the assembler routine that must be called when a character is send to the serial buffer (UDR1). The character is placed into R24.

With the redirection of the PRINT and other serial output related commands, you can use your own routines.
This way you can use other devices as output devices.

See also
$SERIALINPUT1 700 , $SERIALINPUT 698 , $SERIALINPUT2LCD 701 , $SERIALOUTPUT 701

Example
See the $SERIALOUTPUT 701 example

7.3.56 $SIM

Action
Instructs the compiler to generate empty wait loops for the WAIT and WAITMS statements. This to allow faster simulation.

Syntax
$SIM

Remarks
Simulation of a WAIT statement can take a long time especially when memory view windows are opened.
The $SIM compiler directive instructs the compiler to not generate code for WAITMS and WAIT. This will of course allows faster simulation.

When your application is ready you must remark the $SIM directive or otherwise the WAIT and WAITMS statements will not work as expected.
When you forget to remove the $SIM option and you try to program a chip you will receive a warning that $SIM was used.

See also
NONE

ASM
NONE

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

$sim
Do
  Wait 1
  Print "Hello"
Loop
```

7.3.57 $STACKDUMP

Action
Makes the compiler hook up the reset vector and includes code, which allows to get a dump of the stack residing in SRAM.

Syntax
$stackdump

Preface

Using $stackdump presumes certain knowledge of assembler code, i.e. reading and understanding disassembled code. On the other hand it's possible that an user, who has little to no experience in assembly reading, simply uses $stackdump, while an assembly-experienced user evaluates the dumped result. This allows sharing of experience, knowledge and active debugging of difficult code via Internet, without having actual hardware available.

Remarks

Additional code in Bascom-Basic is used to put out the content of the saved stack to whatever target, in the provided example code the dump is written to the serial interface, however any other reasonable target for receiving the dump is feasible. For example, a dump can be saved to EEProm also, the user is free to modify the target himself.

Function

After each reset an AVR micrcontroller executes the reset-vector, the $stackdump code hooks this vector and executes a small routine, which saves a certain amount of stack to a protected memory range. This is possible, as SRAM memory keeps its content even after a reset. After saving the stack, a routine is executed which clears SRAM, excluding the previously saved range. In the following it's save to put out the saved stack content by
regular Bascom code. Without $stackdump this can't be done, as a) the stack would be destroyed by normal SRam-clearing code, and b) because every Bascom-code modifies, i.e overwrite the stack itself.

Usage

Stack can contain two types of data, 1) data, i.e. saved registers and 2) return addresses, which were pushed on
the stack by previous calls. The most interesting is the latter, as it can point to faulty code. If followed these
return addresses (which of course needs also some guesswork to distinguish it apart from saved registers), it's
possible to find out interrupting code, and this way difficult to find bugs.

Options

Depending whether the stack pointer is intact at reset, one of the two options can be used:

```
Ignore_SP = 1
Ignore_SP = 0
```

If a hard-rest occurs, for example by a watchdog reset, the stack pointer is reset to its default values, and this way can't be used to determine the stack pointers last position. For this case Ignore_SP = 1 is useful.
In this mode the amount of bytes given by Stck_siz_sav beginning from stack end is saved. This can be used for tracking down randomly occurring resets by whatever reasons. Be aware that without knowing the stack pointers last position, it's much harder to find out the last executed call, but it's still possible.

In contrary, if a soft reset occurs, the stack pointer is likely intact and the the option Ignore_SP = 0 is useful.
Here the stack is saved from the stack pointers last position to the amount of Stck_siz_sav bytes till ramend/stack-end. In case ram-end comes first, only the stack

range between stack pointer and ram-end is saved.
The method using Ignore_SP = 0 is useful to redirect any interrupt to the reset vector by writing:

```
On interrupt_xy my_isr NOSAVE
Enable interrupt_xy
Enable Interrupts
'...
my_isr:
!jmp 0
return
```

If using an external interrupt, for example INT0 for my_isr, a signal on INT0 will create the stack dump, pointing to code executed at occurrence of the signal. This works like an on-chip hardware debugger. In certain chips a watchdog timer interrupt is available, this interrupt can be used and a watchdog timeout will then create a dump.

Notice: Previous mentioned functionality for Ignore_SP = 0 needs enabled interrupts. In case these special, or also global interrupts get disabled by code, it will fail. But also a disabled interrupt can point to the source of a bug. Using Ignore_SP = 1 will work in any case, but with said restrictions.

Closing note

$stackdump can only increase the chance to trap down a nasty bug or do some special type debugging. It's for sure no cure-all type of tool. Because of certain restrictions given by AVR hardware it can't be universal.

7.3.58 $SWSTACK

Action

Sets the available space for the software stack.

Syntax

$SWSTACK = var

Remarks

Var	A numeric decimal value.

While you can configure the SW Stack in Options, Compiler, Chip, it is good practice to put the value into your code. This way you do no need the cfg(configuration) file.

The $SWSTACK directive overrides the value from the IDE Options.

 It is important that the $SWSTACK directive occurs in your main project file. It may not be included in an $include file as only the main file is parsed for $SWSTACK. $SWSTACK only accepts numeric values.

Software Stack stores the parameter addresses passed to a subroutine and LOCAL variable addresses.
So the Software stack stores the addresses of variables where each passed variable

and local variable use 2 bytes per respective addresses.

When using SUB or FUNCTION there are 3 ways for parameters:
- Using BYREF pass a variable by reference with its ADDRESS (so it is pointing to an existing variable which is already in SRAM)
- Using BYVAL the value is stored in FRAME (during the SUB is processed) so it is pointing to the address in FRAME.
- Using BYLABEL pass the address of a label

When nothing is specified the parameter will be passed BYREF.

See also

$HWSTACK 648 , $FRAMESIZE 641, Memory Usage 267

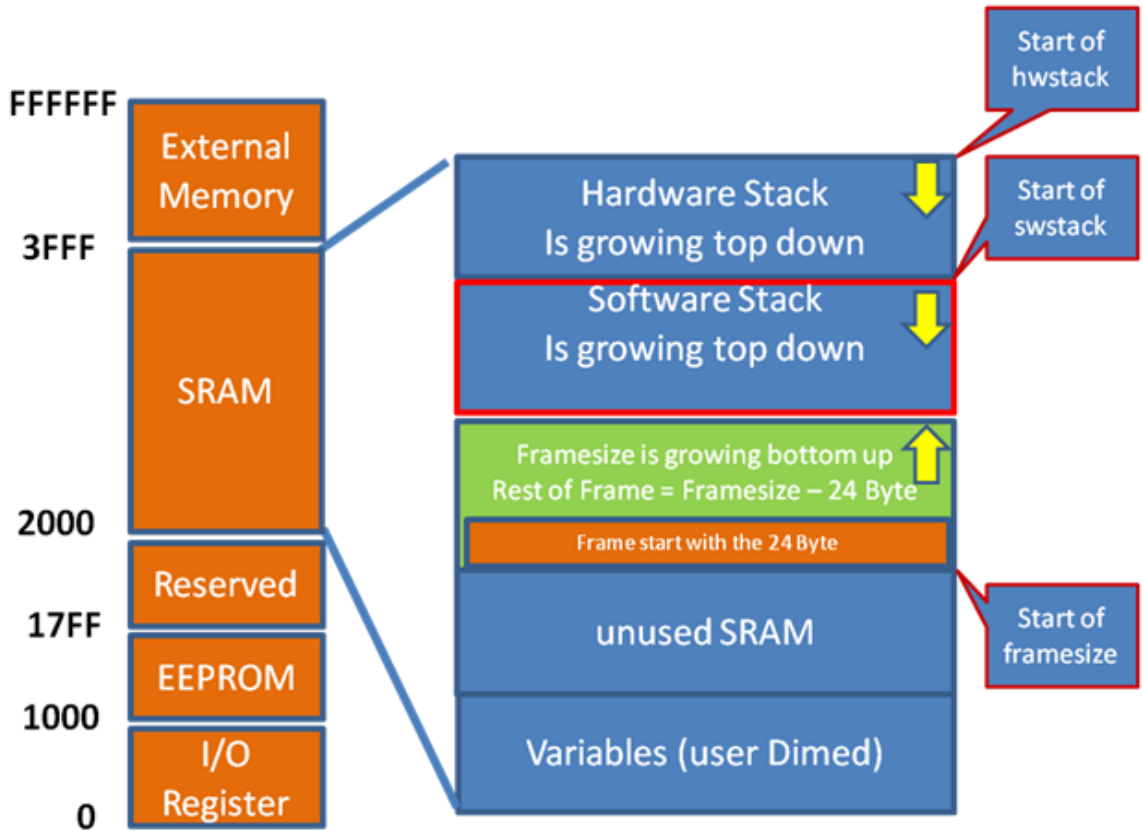
For example if you have used 10 locals in a SUB and there are 3 parameters passed to it, you need:
(10 * 2 Byte) + (3 * 2 Byte) = **26 Byte Software Stack**.

The following SUB need 10 Byte of Software Stack:
5* 2 Byte = **10 Byte**

```
Sub My_sub()
  Local A1 As Byte , A2 As Byte , A3 As Byte , A4 As Byte , A5 As Byte
  A1 = 1
  A2 = 2
  A3 = 3
  A4 = 4
  A5 = 5
End Sub
```

So the software stack size can be calculated by taking the maximum number of parameter passed to a SUB routine, adding the number of LOCAL variables and multiplying the result by 2. To be safe, add 4 more bytes for internally used LOCAL variables.

If you have several SUB or FUNCTIONS search for the SUB or FUNCTION with the most parameters and LOCAL variables and use that calculated maximum numbers for defining the Software Stack ($swstack).
The Software Stack is growing top down (see picture) and start direct after the Hardware Stack. The Software Stack grows against the FRAME.



Picture: Example Memory of ATXMEGA128A1

[****] 267

Example

```
$regfile = "xm128a1def.dat"
$crystal = 32000000 '32MHz
$hwstack = 64
$swstack = 128
$framesize = 288


Config Osc = Enabled , 32mhzosc = Enabled '32MHz
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 '32MHz
'Config Interrupts
Config Priority = Static , Vector = Application , Lo = Enabled 'Enable
Lo Level Interrupts
Config Com1 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

Declare Sub My_sub()

Call My_sub()

End 'end program

Sub My_sub()
 Local A1 As Byte , A2 As Byte , A3 As Byte , A4 As Byte , A5 As Byte
```

```
Local S As String * 254

For A1 = 1 To 254
  S = S + "1"
Next A1

A1 = 1
A2 = 2
A3 = 3
A4 = 4
A5 = 5
Print A1

End Sub                                    'default use 40 for the frame
space
```

7.3.59 $TIMEOUT

Action

Enable timeout on the hardware UART and software UART.

Syntax

$TIMEOUT = value

Remarks

Value	A constant that fits into a LONG , indicating how much time must be waited before the waiting is terminated.

All RS-232 serial statements and functions(except INKEY) that use the hardware UART or software UART, will halt the program until a character is received. Only with buffered serial input you can process your main program while the buffer receives data on the background.

 $TIMEOUT is an alternative for normal serial reception. It is not intended to be used with buffered serial reception. As of version 2077, the first (and only the first) UART supports the $TIMEOUT feature. The latest version supports timeout on ALL UARTS.

When you assign a constant to $TIMEOUT, you actual assign a value to the internal created value named ___*TIMEOUT*.

This value will be decremented in the routine that waits for serial data. When it reaches zero, it will terminate.

So the bigger the value, the longer the wait time before the timeout occurs. The timeout is not in seconds or microseconds, it is a relative number. Only the speed of the oscillator has effect on the duration. And the value of the number of course.

When the time out is reached, a zero/null will be returned to the calling routine. Waitkey() will return 0 when used with a byte. When you use INPUT with a string, the timeout will be set for every character. And when 3 characters are received but no CR and/or LF, these 3 characters will be returned.

 When you activate $TIMEOUT, and your micro has two UARTS(Mega128 for example) it will be active for both UART0 and UART1. And for an ATMEGA2560 with 4

UARTS, it will be enabled for all 4 UARTS, but only when no serial input buffer is configured.

$TIMEOUT is also supported by the software UART. In fact, when you enable it for the hardware UART, you enable it for the software UART as well.
Note that for the SW UART the maximum timeout value is lower. The maximum is &HFF_FF_FF. This because 1 byte less is used.

See Also

INPUT 1493 , WAITKEY 1511 , INKEY 1492

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : timeout.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstration of the $timeout option
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'most serial communication functions and routines wait until a character
'or end of line is received.
'This blocks execution of your program. SOmething you can change by
using buffered input
'There is also another option : using a timeout
'$timeout Does Not Work With Buffered Serial Input


Dim Sname As String * 10
Dim B As Byte
Do
   $timeout = 1000000
   Input "Name : " , Sname
   Print "Hello " ; Sname

   $timeout = 5000000
   Input "Name : " , Sname
   Print "Hello " ; Sname
Loop

'you can re-configure $timeout
```

7.3.60 $TINY

Action
Instruct the compiler to generate initialize code without setting up the stacks.

Syntax
$TINY

Remarks
The tiny11 for example is a powerful chip. It only does not have SRAM. BASCOM depends on SRAM for the hardware stack and software stack.
When you like to program in ASM you can use BASCOM with the $TINY directive.

Some BASCOM statements will also already work but the biggest part will not work. A future version will support a subset of the BASCOM statements and function to be used with the chips without SRAM.

Note that the generated code is not yet optimized for the tiny parts. Some used ASM statements for example will not work because the chip does not support it.

See also
NONE

ASM
NONE

Example
```
$regfile = "attiny15.dat"
$tiny
$crystal = 1000000
$noramclear
$hwstack = 0
$swstack = 0
$framesize = 0

Dim A As Iram Byte
Dim B As Iram Byte
A = 100 : B = 5
A = A + B
End
```

7.3.61 $TYPECHECK

Action
This directive will turn on type checking

Syntax
$TYPECHECK

Remarks
Type checking is performed on some operations. It is turned on by default. With the $NOTYPECHECK you can turn this feature off. And with $TYPECHECK you can turn it on again.

See also
$NOTYPECHECK 688

Example
```
NONE
```

7.3.62 $USER

Action
This directive will let the compiler create an **.usr** file that contains the data following this directive. Switch back to normal DATA statements with $DATA

Syntax
```
$USER
```

Remarks
Some new UPDI processors have a large User Signature data area. While smaller User Signature areas can be changed by the UPDI programmer, it is not practical when the size is larger than 64 bytes.
The MCS EDBG programmer has a new TAB : **User Signature** where you can change data similar as the **EEPROM** TAB.
You can create an **.USR** file using the $USER directive followed by DATA statements.
Switch back to normal Flash DATA using $DATA, the default.
The file the compiler creates is a binary file. The MCS EDBG programmer will load the data automatically when it exists.

See also
READUSERSIG 1427 , $DATA 1177 , DATA 626

Partial Example
```
'create .usr data file
$user
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 ,
15 , 16
'    ^  address 0
'switch back to normal DATA lines
$data
Data 10 , 20
```

7.3.63 $VERSION

Action
This compiler directive stores version information.

Syntax
$VERSION V,S,R

Remarks
Version info is important information. If you need to maintain source code, it will make it easy to identify the code.
$VERSION has 3 parameters. These must be numeric digits. Each time you compile your code, the release number is increased.

You can use Version(2) to print this information. $version 1,2,3 will be printed as 1.2.3

The compiler will create three internal constants named _VERSION_MAJOR, _VERSION_MINOR and _VERSION_BUILD with the specified values.
For example when $version is set to : $VERSION 1,2,3
_VERSION_MAJOR will become 1 , _VERSION_MINOR will become 2 and _VERSION_BUILD will become 3.

See also
VERSION 1606

Example
```
$version 1,2,3
Print Version(2)
```

7.3.64 $WAITSTATE

Action
Compiler directive to activate external SRAM and to insert a WAIT STATE for a slower ALE signal.

 CONFIG XRAM 1161 should be used instead.

Syntax
$WAITSTATE

Remarks
The $WAITSTATE can be used to override the Compiler Chip Options setting.
Wait states are needed for slow external components that can not handle the fast ALE signal from the AVR chip.

See also
$XA 713 , CONFIG XRAM 1161

Example
```
$WAITSTATE
```

7.3.65 $XA

Action
Compiler directive to activate external memory access.

 CONFIG XRAM 1161 should be used instead.

Syntax
$XA

Remarks
The $XA directive can be used to override the Compiler Chip Options setting.
This way you can store the setting in your program code. It is strongly advised to do this.

See also
$WAITSTATE 712 , CONFIG XRAM 1161

Example
```
$XA
```

7.3.66 $XRAMSIZE

Action
Specifies the size of the external RAM memory.

Syntax
$XRAMSIZE = [&H] size

Remarks

Size	A constant with the size of the external RAM memory chip.

The size of the chip can be selected from the Options Compiler Chip 143 menu.
The $XRAMSIZE overrides this setting. It is important that $XRAMSTART precedes $XRAMSIZE

See also
$XRAMSTART 714 , CONFIG XRAM 1161

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : m128.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrate using $XRAM directive
'micro                    : Mega128
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m128def.dat"                                    ' specify
the used micro
$crystal = 1000000                                          ' used
crystal frequency
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

$xramstart = &H1000

$xramsize = &H1000
Dim X As X
```

7.3.67 $XRAMSTART

Action

Specifies the location of the external RAM memory.

Syntax

$XRAMSTART = [&H]address

Remarks

Address	The (hex)-address where the data is stored. Or the lowest address that enables the RAM chip. You can use this option when you want to run your code in systems with external RAM memory. Address must be a constant.

By default the extended RAM will start after the internal memory so the lower addresses of the external RAM can't be used to store information.

When you want to protect an area of the chip, you can specify a higher address for the compiler to store the data. For example, you can specify &H400. The first dimensioned variable will be placed in address &H400 and not in &H260.

It is important that when you use $XRAMSTART and $XRAMSIZE that $XRAMSTART comes before $XRAMSIZE.

See also

$XRAMSIZE 713

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : m128.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrate using $XRAM directive
'micro                    : Mega128
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m128def.dat"                                    ' specify
the used micro
$crystal = 1000000                                          ' used
crystal frequency
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

$xramstart = &H1000

$xramsize = &H1000
Dim X As X
```

7.3.68 $XTEAKEY

Action

This directive accepts a 16 byte XTEA key and informs the compiler to encrypt the binary image.

Syntax

$XTEAKEY 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

Remarks

$XTEAKEY accepts 16 parameters. These are the 16 bytes which together form a 128 bit key.
When your code is compiled, the resulting binary code will be encrypted with the provided key.
A boot loader could then use XTEA and decrypt the binary file before writing to flash memory.
The XTEADECODE statement can be used inside a boot loader to decrypt the encrypted blocks.

The XTEA encoder uses 32 rounds. The same as used in the xtea.lib

 Only the binary image is encrypted, the HEX file is not encrypted!
You can not simulate an encrypted program. Add this option when your project is ready.

See also
$AESKEY 606 , XTEAENCODE 1297 , XTEADECODE 1298

Example
NONE

7.4 1WIRE

7.4.1 1WIRECOUNT

Action
This statement reads the number of 1wire devices attached to the bus.

Syntax
var2 = **1WIRECOUNT**()
var2 = **1WIRECOUNT**(port , pin)

Remarks

var2	A WORD variable that is assigned with the number of devices on the bus.
port	The PIN port name like PINB or PIND.
pin	The pin number of the port. In the range from 0-7. May be a numeric constant or variable.

The variable must be of the type word or integer.
You can use the 1wirecount() function to know how many times the 1wsearchNext() function should be called to get all the Id's on the bus.

The 1wirecount function will take 4 bytes of SRAM.

```
___1w_bitstorage , Byte used for bit storage :
lastdeviceflag bit 0
id_bit bit 1
cmp_id_bit bit 2
search_dir bit 3
___1wid_bit_number, Byte
___1wlast_zero, Byte
___1wlast_discrepancy , Byte
```

When there is no 1WIRE device on the bus, the ERR bit will be set. When devices are found, ERR will be cleared.

ASM
The following asm routines are called from mcs.lib.

```
_1wire_Count : (calls _1WIRE, _1WIRE_SEARCH_FIRST , _1WIRE_SEARCH_NEXT)
```

Parameters passed : R24 : pin number, R30 : port , Y+0,Y+1 : 2 bytes of soft stack,

X : pointer to the frame space

Returns Y+0 and Y+1 with the value of the count. This is assigned to the target variable.

See also

1WWRITE 729 , 1WRESET 718 , 1WREAD 720, 1WSEARCHFIRST 723, 1WSEARCHNEXT 725 , Using the 1wire protocol 310

Example

```
'-----------------------------------------------------------------------
---------
'name                     : 1wireSearch.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates 1wsearch
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 4000000

$hwstack = 32                           ' default use 32 for the
hardware stack
$swstack = 10                           'default use 10 for the SW stack
$framesize = 40                         'default use 40 for the frame
space

Config 1wire = Portb.0                  'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
'      lastdeviceflag bit 0
'      id_bit         bit 1
'      cmp_id_bit     bit 2
'      search_dir     bit 3
'___1wid_bit_number, Byte
'___1wlast_zero,  Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8                                          'print the
number
    Print Hex(reg_no(i));
Next
```

```
Print

Do
  'Now search for other devices
  Reg_no(1) = 1wsearchnext()
  For I = 1 To 8
     Print Hex(reg_no(i));
  Next
  Print
Loop Until Err = 1



'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                    'for
example look at pin PINB.1
End
```

7.4.2 1WRESET

Action

This statement brings the 1wire pin to the correct state, and sends a reset to the bus.

Syntax

1WRESET
1WRESET PORT , PIN

Remarks

1WRESET	Reset the 1WIRE bus. The error variable ERR will return 1 if an error occurred
Port	The register name of the input port. Like PINB, PIND.
Pin	The pin number to use. In the range from 0-7. May be a numeric constant or variable.

The global variable ERR is set when an error occurs.
There is also support for multi 1-wire devices on different pins.

To use this you must specify the port and pin that is used for the communication.

The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement.

The syntax for additional 1-wire devices is :

```
1WRESET port , pin
1WWRITE var/constant ,bytes] , port, pin
var = 1WREAD( bytes) , for the configured 1 wire pin
var = 1WREAD(bytes, port, pin) ,for reading multiple bytes
```

See also

1WREAD 720 , 1WWRITE 729

Example

```
'-----------------------------------------------------------------------
---------
'name                    : 1wire.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates 1wreset, 1wwrite and 1wread()
'micro                   : Mega48
'suited for demo         : yes
'commercial addon needed  : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 4000000

$hwstack = 32                              ' default use 32 for the
hardware stack
$swstack = 10                              'default use 10 for the SW
stack
$framesize = 40                            'default use 40 for the frame
space

'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"


Config 1wire = Portb.0                                      'use this
pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte


Do
  Wait 1
  1wreset                                                   'reset the
device
  Print Err                                                 'print error
1 if error
  1wwrite &H33                                              'read ROM
```

```
command
  For I = 1 To 8
    Ar(i) = 1wread()                                        'place into
array
  Next

'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)                                         'read 8
bytes

  For I = 1 To 8
     Print Hex(ar(i));                                      'print
output
  Next
  Print                                                     'linefeed
Loop


'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!

'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
  Ar(i) = 0                                                 'clear array
to see that it works
Next

1wreset Pinb , 2                                            'use this
port and  pin for the second device
1wwrite &H33 , 1 , Pinb , 2                                 'note that
now the number of bytes must be specified!
'1wwrite Ar(1) , 5,pinb,2

'reading is also different
Ar(1) = 1wread(8 , Pinb , 2)                                'read 8
bytes from portB on pin 2

For I = 1 To 8
  Print Hex(ar(i));
Next

'you could create a loop with a variable for the bit number !
For I = 0 To 3                                              'for pin 0-3
  1wreset Pinb , I
  1wwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8 , Pinb , I)
  For A = 1 To 8
    Print Hex(ar(a));
  Next
  Print
Next
End
```

7.4.3 1WREAD

Action

This statement reads data from the 1wire bus into a variable.

Syntax

var2 = **1WREAD**([bytes])
var2 = **1WREAD**(bytes , port , pin)

Remarks

var2	Reads a byte from the bus and places it into variable var2.
bytes	Optional parameter to specify the number of bytes to read.
Port	The **PIN** port name like PINB or PIND.
Pin	The pin number of the port. In the range from 0-7. Must be a numeric constant or variable.

Multi 1-wire devices on different pins are supported.
To use this you must specify the port pin that is used for the communication.

The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement 857.

The syntax for additional 1-wire devices is :
```
1WRESET port, pin
1WWRITE var/constant , bytes, port, pin
var = 1WREAD(bytes, port, pin) for reading multiple bytes
```

See also

1WWRITE 729 , 1WRESET 718

Example

```
'-----------------------------------------------------------------------
---------
'name                     : 1wire.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates 1wreset, 1wwrite and 1wread()
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 4000000

$hwstack = 32                              ' default use 32 for the
hardware stack
$swstack = 10                              'default use 10 for the SW
stack
$framesize = 40                            'default use 40 for the frame
space

'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"


Config 1wire = Portb.0                     'use this pin
```

```
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte


Do
  Wait 1
  1wreset                                                 'reset the
device
  Print Err                                               'print error
1 if error
  1wwrite &H33                                            'read ROM
command
  For I = 1 To 8
    Ar(i) = 1wread()                                      'place into
array
  Next

'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)                                        'read 8
bytes

  For I = 1 To 8
     Print Hex(ar(i));                                    'print
output
  Next
  Print                                                   'linefeed
Loop


'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!

'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
  Ar(i) = 0                                               'clear array
to see that it works
Next

1wreset Pinb , 2                                          'use this
port and  pin for the second device
1wwrite &H33 , 1 , Pinb , 2                               'note that
now the number of bytes must be specified!
'1wwrite Ar(1) , 5,pinb,2

'reading is also different
Ar(1) = 1wread(8 , Pinb , 2)                              'read 8
bytes from portB on pin 2

For I = 1 To 8
  Print Hex(ar(i));
Next

'you could create a loop with a variable for the bit number !
For I = 0 To 3                                            'for pin 0-3
  1wreset Pinb , I
  1wwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8 , Pinb , I)
  For A = 1 To 8
    Print Hex(ar(a));
  Next
  Print
Next
```

```
End
```

7.4.4 1WSEARCHFIRST

Action

This statement reads the first ID from the 1wire bus into a variable(array).

Syntax

var2 = **1WSEARCHFIRST**()
var2 = **1WSEARCHFIRST**(port , pin)

Remarks

var2	A variable or array that should be at least 8 bytes long that will be assigned with the 8 byte ID from the first 1wire device on the bus.
port	The PIN port name like PINB or PIND.
pin	The pin number of the port. In the range from 0-7. Maybe a numeric constant or variable.

The 1wireSearchFirst() function must be called once to initiate the ID retrieval process. After the 1wireSearchFirst() function is used you should use successive function calls to the 1wSearchNext 725 function to retrieve other ID's on the bus.

A string can not be assigned to get the values from the bus. This because a null may be returned as a value and the null is also used as a string terminator.

I would advice to use a byte array as shown in the example.

The 1wirecount function will take 4 bytes of SRAM.

```
___1w_bitstorage , Byte used for bit storage :
lastdeviceflag bit 0
id_bit bit 1
cmp_id_bit bit 2
search_dir bit 3
___1wid_bit_number, Byte
___1wlast_zero, Byte
___1wlast_discrepancy , Byte
```

ASM

The following asm routines are called from mcs.lib.

```
_1wire_Search_First : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS)
Parameters passed : R24 : pin number, R30 : port , X : address of target array
```

Returns nothing.

See also

1WWRITE 729 , 1WRESET 718 , 1WREAD 720, 1WSEARCHNEXT 725, 1WIRECOUNT 716

Example

```
'-----------------------------------------------------------------------
---------
'name                    : 1wireSearch.bas
```

```
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates 1wsearch
'micro                   : Mega48
'suited for demo         : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 4000000

$hwstack = 32                         ' default use 32 for the hardware
stack
$swstack = 10                         'default use 10 for the SW stack
$framesize = 40                       'default use 40 for the frame space

Config 1wire = Portb.0                'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
'      lastdeviceflag bit 0
'      id_bit         bit 1
'      cmp_id_bit     bit 2
'      search_dir     bit 3
'___1wid_bit_number, Byte
'___1wlast_zero,  Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8                                          'print the
number
    Print Hex(reg_no(i));
Next
Print

Do
  'Now search for other devices
  Reg_no(1) = 1wsearchnext()
  For I = 1 To 8
     Print Hex(reg_no(i));
  Next
  Print
Loop Until Err = 1



'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
```

```
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                    'for
example look at pin PINB.1
End
```

7.4.5 1WSEARCHNEXT

Action

This statement reads the next ID from the 1wire bus into a variable(array).

Syntax

var2 = **1WSEARCHNEXT**()
var2 = **1WSEARCHNEXT**(port , pin)

Remarks

var2	A variable or array that should be at least 8 bytes long that will be assigned with the 8 byte ID from the next 1wire device on the bus.
Port	The PIN port name like PINB or PIND.
Pin	The pin number of the port. In the range from 0-7. May be a numeric constant or variable.

The 1wireSearchFirst() function must be called once to initiate the ID retrieval process. After the 1wireSearchFirst() function is used you should use successive function calls to the 1wireSearchNext function to retrieve other ID's on the bus.

A string can not be assigned to get the values from the bus. This because a null may be returned as a value and the null is also used as a string terminator.

I would advice to use a byte array as shown in the example.

The 1wirecount function will take 4 bytes of SRAM.

```
___1w_bitstorage , Byte used for bit storage :
lastdeviceflag bit 0
id_bit bit 1
cmp_id_bit bit 2
search_dir bit 3
___1wid_bit_number, Byte
___1wlast_zero, Byte
___1wlast_discrepancy , Byte
```

ASM

The following asm routines are called from mcs.lib.

```
_1wire_Search_Next : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS)
Parameters passed : R24 : pin number, R30 : port , X : address of target array
Returns nothing.
```

See also

1WWRITE 729 , 1WRESET 718 , 1WREAD 720, 1WSEARCHFIRST 723, 1WIRECOUNT 716

Example

```
'-----------------------------------------------------------------------
---------
'name                     : 1wireSearch.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates 1wsearch
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 4000000

$hwstack = 32                           ' default use 32 for the hardware
stack
$swstack = 10                           'default use 10 for the SW stack
$framesize = 40                         'default use 40 for the frame
space

Config 1wire = Portb.0                  'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
'      lastdeviceflag bit 0
'      id_bit         bit 1
'      cmp_id_bit     bit 2
'      search_dir     bit 3
'___1wid_bit_number, Byte
'___1wlast_zero,  Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8                                              'print the
```

```
number
    Print Hex(reg_no(i));
Next
Print

Do
  'Now search for other devices
  Reg_no(1) = 1wsearchnext()
  For I = 1 To 8
     Print Hex(reg_no(i));
  Next
  Print
Loop Until Err = 1



'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                    'for
example look at pin PINB.1
End
```

7.4.6 1WVERIFY

Action

This verifies if an ID is available on the 1wire bus.

Syntax

1WVERIFY ar(1)
1WVERIFY ar(1) , port, pin

Remarks

Ar(1)	A byte array that holds the ID to verify.
port	The name of the PORT PINx register like PINB or PIND.
pin	The pin number in the range from 0-7. May be a numeric constant or variable.

Returns ERR set to 0 when the ID is found on the bus otherwise it will be 1.

ASM

The following asm routines are called from mcs.lib.
_1wire_Search_Next : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS)

See also

1WWRITE 729 , 1WRESET 718 , 1WREAD 720, 1WSEARCHFIRST 723, 1WIRECOUNT 716

Example

```
'-----------------------------------------------------------------------
---------
'name                     : 1wireSearch.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates 1wsearch
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 4000000

$hwstack = 32                           ' default use 32 for the
hardware stack
$swstack = 10                           'default use 10 for the SW
stack
$framesize = 40                         'default use 40 for the frame
space

Config 1wire = Portb.0                  'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
'      lastdeviceflag bit 0
'      id_bit         bit 1
'      cmp_id_bit     bit 2
'      search_dir     bit 3
'___1wid_bit_number, Byte
'___1wlast_zero,  Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8                                          'print the
number
    Print Hex(reg_no(i));
Next
```

```
Print

Do
  'Now search for other devices
  Reg_no(1) = 1wsearchnext()
  For I = 1 To 8
     Print Hex(reg_no(i));
  Next
  Print
Loop Until Err = 1



'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optional call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                    'for
example look at pin PINB.1
End
```

7.4.7 1WWRITE

Action

This statement writes a variable to the 1wire bus.

Syntax

1WWRITE var1
1WWRITE var1, bytes
1WWRITE var1 , bytes , port , pin

Remarks

var1	Sends the value of var1 to the bus. The number of bytes can be specified too but this is optional.
bytes	The number of bytes to write. Must be specified when port and pin are used.
port	The name of the PORT PINx register like PINB or PIND.
pin	The pin number in the range from 0-7. May be a numeric constant or variable.

Multiple 1-wire devices on different pins are supported.
To use this you must specify the port and pin that are used for the communication.

The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE 857 statement.

The syntax for additional 1-wire devices is :
```
1WRESET port , pin
1WWRITE var/constant, bytes, port , pin
var = 1WREAD(bytes, port, pin) ,for reading multiple bytes
```

See also
1WREAD 720 , 1WRESET 718

Example
```
'-----------------------------------------------------------------------
---------
'name                     : 1wire.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates 1wreset, 1wwrite and 1wread()
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 4000000

$hwstack = 32                           ' default use 32 for the hardware
stack
$swstack = 10                           'default use 10 for the SW stack
$framesize = 40                         'default use 40 for the frame
space

'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"


Config 1wire = Portb.0                   'use this pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte


Do
  Wait 1
  1wreset                                                    'reset the
device
  Print Err                                                  'print error
1 if error
  1wwrite &H33                                               'read ROM
command
  For I = 1 To 8
    Ar(i) = 1wread()                                         'place into
```

```
array
  Next

'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)                                          'read 8
bytes

  For I = 1 To 8
     Print Hex(ar(i));                                      'print
output
  Next
  Print                                                     'linefeed
Loop


'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!

'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
  Ar(i) = 0                                                 'clear array
to see that it works
Next

1wreset Pinb , 2                                            'use this
port and  pin for the second device
1wwrite &H33 , 1 , Pinb , 2                                 'note that
now the number of bytes must be specified!
'1wwrite Ar(1) , 5,pinb,2

'reading is also different
Ar(1) = 1wread(8 , Pinb , 2)                                'read 8
bytes from portB on pin 2

For I = 1 To 8
  Print Hex(ar(i));
Next

'you could create a loop with a variable for the bit number !
For I = 0 To 3                                              'for pin 0-3
  1wreset Pinb , I
  1wwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8 , Pinb , I)
  For A = 1 To 8
    Print Hex(ar(a));
  Next
  Print
Next
End
```

7.5 ADR , ADR2

Action

Create label address.

Syntax

ADR label
ADR2 label

Remarks

label	The name of a label.

The AVR uses WORD addresses. ADR will create the word address. To find a byte in memory, you need to multiply by 2. For this purpose ADR2 is available. It will create the address of the label multiplied by 2.

Using ADR2 you can use tables. The sample program demonstrates this together with some more advanced ASM code.

The sample includes ADR2.LIB. This lib contains a special version of _MoveConst2String .
The normal routine in MCS.LIB will stop printing once a null byte (zero) is encountered that indicates the end of a string.
But for the sample program, we may not change the address, so the address is restored when the null byte is found.

See Also

NONE

Example

```
'=================================================================================
' This is an example of how to create an interactive menu system supporting
' sub-menus and support routines using the !ADR and !ADR2 statements
'=================================================================================

$regfile = "M644def.dat"
$crystal = 8000000

$hwstack = 64                                ' specify the hardware stack depth
$swstack = 64                                ' specify the software stack depth
$framesize = 64                              ' specify the framesize (local stack dept

$lib "adr2.lib"

'---------------------------------------------------------------------------------

Dim Menupointer As Word
Dim Actionpointer As Word

Dim Entries As Byte
Dim Dummy As Byte
Dim Message As String * 32

Dim Local1 As Byte
Dim Local_loop1 As Byte

Const Menu_id = &HAA                                          ' sub-menu ID byte
Const Routine_id = &H55                                       ' service routine ID by

'---------------------------------------------------------------------------------

 Restore Main_menu                                            ' point to the start of
! sts {MenuPointer}, R8                                       ' }
! sts {MenuPointer + 1}, R9                                   ' } store the pointer
```

```
Display_new_menu:

! lds R8, {MenuPointer}                                          ' }
! lds R9, {MenuPointer + 1}                                      ' } restore the pointe

 Read Entries                                                    ' get the number of ent
 Print
 For Local_loop1 = 1 To Entries
  Read Message                                                   ' read the message
  Print Message                                                  ' send it to the consol
 Next

 Read Dataptr                                                    ' get the pointer to th
! sts {ActionPointer}, R8                                        ' }
! sts {ActionPointer + 1}, R9                                    ' } store the pointer

 Input "Entry ? " , Local1                                       ' ask the user which me
 If Local1 = 0 Then                                              ' is it valid ?
  Goto Display_new_menu                                          ' if not, re-display th
 End If
 If Local1 => Entries Then                                       ' is it valid ?
  Goto Display_new_menu                                          ' if not, re-display th
 End If

! lds R8,{ActionPointer}                                         ' }
! lds R9,{ActionPointer + 1}                                     ' } restore the pointe

 If Local1 <> 1 Then
     For Local_loop1 = 2 To Local1                               '
!    ldI R30,4                                                  ' }
!    clr R1                                                     ' }
!    add R8,R30                                                 ' }
!    adc R9,R1                                                  ' }
     Next                                                       ' } calculate the locat
 End If

 Read Local1                                                     ' get the menu entry's
 Read Dummy                                                      ' to handle the uP expe

 If Local1 = Menu_id Then                                        ' did the user select a
  Read Dataptr
!  sts {MenuPointer}, R8                                        ' }
!  sts {MenuPointer + 1}, R9                                    ' } store the start of
  Goto Display_new_menu
 End If

 Read Dataptr                                                    ' get the address of th
! movw R30,R8
! icall                                                          ' pass control to the

 Goto Display_new_menu                                           ' re-display the last m

'-----------------------------------------------------------------------------
'   Test support routines
'-----------------------------------------------------------------------------

Hello_message:

 Print
 Print "You asked to print 'Hello'"                              ' confirmation that Men
 Return

2nd_menu_1st_entry_routine:
```

```
 Print
 Print "You selected Entry 1 of the 2nd menu"               ' confirmation that Men
 Return

2nd_menu_2nd_entry_routine:

 Print
 Print "You selected Entry 2 of the 2nd menu"               ' confirmation that Men
 Return

3rd_menu_1st_entry_routine:

 Print
 Print "You selected Entry 1 of the 3rd menu"               ' confirmation that Men
 Return

3rd_menu_2nd_entry_routine:

 Print
 Print "You selected Entry 2 of the 3rd menu"               ' confirmation the Menu
 Return

 End

'=============================================================================
' Data Statements
'=============================================================================

$data

'-----------------------------------------------------------------------------
' Main Menu
'-----------------------------------------------------------------------------

Main_menu:

 Data 4                                                    '  number of entries in

 Data "MAIN MENU"                                          ' } menu title
 Data "1. Go to Menu 2"                                    ' } 1st menu entry
 Data "2. Go to Menu 3"                                    ' } 2nd menu entry
 Data "3. Print 'Hello' message"                           ' } 3rd menu entry

 Adr2 Mainmenu_supporttable                                ' point to this menu su

'-----------------------------------------------------------------------------

Mainmenu_supporttable:

 Data Menu_id                                              ' identify this menu en
 Adr2 Second_menu                                          ' address of next menu

 Data Menu_id                                              ' identify this menu en
 Adr2 Third_menu                                           ' address of next menu

 Data Routine_id                                           ' identify this menu en
 Adr Hello_message                                         ' address of the suppor

'-----------------------------------------------------------------------------
' Second Menu
'-----------------------------------------------------------------------------

Second_menu:
```

```
 Data 4                                                    ' number of entries in

 Data "SECOND MENU"                                        ' } menu title
 Data "1. 2nd Menu Entry #1"                               ' } 1st menu entry
 Data "2. 2nd Menu Entry #2"                               ' } 2nd menu entry
 Data "3. Go to previous menu"                             ' } 3rd menu entry

 Adr2 Secondmenu_supporttable                              ' point to this menu su

'-----------------------------------------------------------------------------

Secondmenu_supporttable:

 Data Routine_id                                           ' identify this menu en
 Adr 2nd_menu_1st_entry_routine                            ' support routine for 1

 Data Routine_id                                           ' identify this menu en
 Adr 2nd_menu_2nd_entry_routine                            ' support routine for 2

 Data Menu_id                                              ' identify this menu en
 Adr2 Main_menu                                            ' support routine for 3

'-----------------------------------------------------------------------------
' Third Menu
'-----------------------------------------------------------------------------

Third_menu:

 Data 4                                                    ' number of entries in

 Data "THIRD MENU"                                         ' } menu title
 Data "1. 3rd Menu Entry #1"                               ' } 1st menu entry
 Data "2. 3rd Menu Entry #2"                               ' } 2nd menu entry
 Data "3. Go to previous menu"                             ' } 3rd menu entry

 Adr2 Thirdmenu_supporttable                               ' point to this menu su

'-----------------------------------------------------------------------------

Thirdmenu_supporttable:

 Data Routine_id                                           ' identify this menu en
 Adr 3rd_menu_1st_entry_routine                            ' support routine for 1

 Data Routine_id                                           ' identify this menu en
 Adr 3rd_menu_2nd_entry_routine                            ' support routine for 2

 Data Menu_id                                              ' identify this menu en
 Adr2 Main_menu                                            ' support routine for 3
```

7.6 ALIAS

Action

Indicates that the variable can be referenced with another name.

Syntax

newvar **ALIAS** oldvar

Remarks

oldvar	Name of the variable such as PORTB.1
newvar	New name of the variable such as direction

Aliasing port pins can give the pin names a more meaningful name. For example, when your program uses 4 different pins to control 4 different relays, you could name them portb.1, portb.2, portb.3 and portb.4.
But it would be more convenient to refer to them as relais1, relais2, relais3 and realais4.

When you later on change your PCB and decide that relays 4 must be connected to portD.4 instead of portb.4, you only need to change the ALIAS line, and not your whole program.

See also
CONST

Example
```
'-----------------------------------------------------------------------
--------
'copyright               : (c) 1995-2025, MCS Electronics
'micro                   : Mega48
'suited for demo         : yes
'commercial addon needed : no
'purpose                 : demonstrates ALIAS

'-----------------------------------------------------------------------
--------
$regfile = "m48def.dat"
$crystal = 4000000                                          ' 4 MHz
crystal

Const On = 1
Const Off = 0

Config Portb = Output
Relais1 Alias Portb.1
Relais2 Alias Portb.2
Relais3 Alias Portd.5
Relais4 Alias Portd.2

Set Relais1
Relais2 = 0
Relais3 = On
Relais4 = Off

End
```

7.7 Math

7.7.1 ABS

Action
Returns the absolute value of a numeric signed variable.

Syntax
var = **ABS**(var2)

Remarks

Var	Variable that is assigned with the absolute value of var2.
Var2	The source variable to retrieve the absolute value from.

var : Integer , Long, Single or Double.
var2 : Integer, Long, Single or Double.

 The absolute value of a number is always positive.

See also
NONE

ASM
Calls: _abs16 for an Integer and _abs32 for a Long
Input: R16-R17 for an Integer and R16-R19 for a Long
Output:R16-R17 for an Integer and R16-R19 for a Long

Calls _Fltabsmem for a single from the fp_trig library.

Example
```
Dim a as Integer, c as Integer
a =-1000
c = Abs(a)
Print c
End
```

7.7.2 ACOS

Action
Returns the arccosine of a float in radians.

Syntax
var = **ACOS**(x)

Remarks

Var	A floating point variable such as single or double, that is assigned with the ACOS of variable x.
X	The float to get the ACOS of. Input is valid from –1 to +1 and returns p to 0. If Input is < -1 than p and input is > 1 than 0 will returned.

If Input is cause of rounding effect in float-operations a little bit over 1 or -1, the value for 1.0 (-1.0) will be returned. This is the reason to give the value of the limit-point back, if Input is beyond limit. Generally the user have to take care, that Input to this function lies within –1 to +1.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
RAD2DEG 764 , DEG2RAD 747 , COS 746 , SIN 769 , TAN 767 , ATN 742 , ASIN 741 , ATN2 743

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As Single , X As Single
x= 0.5 : S = Acos(x)
Print S
End
```

7.7.3 AND

Action
This logical operator returns the AND of two numeric variables.

Syntax
target = source1 **AND** source2

Remarks
The AND operator works on two bits. It returns a '1' if both inputs are '1'.

A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

The truth table above shows all possible values. A and B represent the 2 inputs. R is the Return or output value. As you can see, you will only get a '1' when both inputs are '1'
It is like having 2 switches in series. You have to switch them both on in order to have a closed circuit.

While you can use AND on bits, you can also perform the same operation on bytes, integers, etc. In such a case, all bits of the variables will be AND-ed.

Example :
Dim A as Byte, B as Byte, R as byte
A=&B**1**100_0001
B=&B**1**001_0000

R=A AND B

R=&B**1**000_0000

As you can see, only bit 7 of both variables is '1'. So in the result, only bit 7 is set. This makes the AND operation perfect for isolating or clearing bits. If you want a value to be in a range of say 0-7 you can set the value to 7 : result= var AND &B111

See also

OR 755 , XOR 770, NOT 753

Example

```
'-----------------------------------------------------------------------
----------
'name                     : boolean.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: AND, OR, XOR, NOT, BIT, SET, RESET
and MOD
'suited for demo          : yes
'commercial add on needed : no
'use in simulator         : possible
'-----------------------------------------------------------------------
----------
'This very same program example can be used in the Help-files for
'      AND, OR, XOR, NOT, BIT, SET, RESET and MOD


$baud = 19200
$crystal = 8000000
$regfile = "m88def.dat"

$hwstack = 40
$swstack = 20
$framesize = 20

Dim A As Byte , B1 As Byte , C As Byte
Dim Aa As Bit , I As Integer

A = 5 : B1 = 3                                          ' assign
values
C = A And B1                                            ' and a
with b
Print "A And B1 = " ; C                                 ' print it:
result = 1


C = A Or B1
```

```
Print "A Or B1 = " ; C                                    ' print it:
result = 7

C = A Xor B1
Print "A Xor B1 = " ; C                                   ' print it:
result = 6

A = 1
C = Not A
Print "c = Not A " ; C                                    ' print it:
result = 254
C = C Mod 10
Print "C Mod 10 = " ; C                                   ' print it:
result = 4


If Portb.1 = 1 Then                                       'test a bit
from a PORT (which is not the same as testing the input state)
  Print "Bit set"
Else
  Print "Bit not set"
End If                                                    'result =
Bit not set

Config Pinb.0 = Input : Portb.0 = 1                       'configure
as input pin
Do
Loop Until Pinb.0 = 0                                     ' repeat
this loop until the logic level becomes 0

Aa = 1                                                    'use this
or ..
Set Aa                                                    'use the
set statement
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                    'result =
Bit set (aa=1)

Aa = 0                                                    'now try 0
Reset Aa                                                  'or use
reset
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                    'result =
Bit not set(aa=0)

C = 8                                                     'assign
variable to &B0000_1000
```

```
Set C                                                       'use the
set statement without specifying the bit
Print C                                                     'print it:
result = 9 ; bit0 has been set

B1 = 255                                                    'assign
variable
Reset B1.0                                                  'reset bit
0 of a byte variable
Print B1                                                    'print it:
result = 254 = &B11111110

B1 = 8                                                      'assign
variable to &B00001000
Set B1.7                                                    'set it
Print B1                                                    'print it:
result = 9 = &B00001001
End
```

7.7.4 ASIN

Action

Returns the arcsine of a float in radians.

Syntax

var = **ASIN**(x)

Remarks

Var	A float variable such as single or double that is assigned with the ASIN of variable x.
X	The float to get the ASIN of. Input is valid from −1 to +1 and returns -p/2 to +p/2. If Input is < -1 than -p/2 and input is > 1 than p/2 will returned.

If Input is cause of rounding effect in single-operations a little bit over 1 or -1, the value for 1.0 (-1.0) will be returned. This is the reason to give the value of the limit-point back, if Input is beyond limit. Generally the user have to take care, that Input to this function lies within −1 to +1.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG 764 , DEG2RAD 747 , COS 746 , SIN 769 , TAN 767 , ATN 742 , ACOS 737 , ATN2 743

Example

```
$regfile = "m48def.dat"                                   ' specify
the used micro
$crystal = 8000000                                        ' used
```

```
crystal frequency
$baud = 19200                                                ' use baud
rate
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As Single , X As Single
X = 0.5 : S = Asin(x)
Print S    '0.523595867

End
```

7.7.5 ATN

Action

Returns the Arctangent of a floating point variable in radians.

Syntax

var = **ATN**(float)

Remarks

Var	A float variable that is assigned with the arctangent of variable float.
float	The float variable to get the arctangent of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.
Floating point variables can be of the single or double data type.

See Also

RAD2DEG 764 , DEG2RAD 747 , COS 746 , SIN 769 , TAN 767 , ATN2 743

Example

```
$regfile = "m48def.dat"                                      ' specify
the used micro
$crystal = 8000000                                           ' used
crystal frequency
$baud = 19200                                                ' use baud
rate
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
```

```
Databits = 8 , Clockpol = 0


Dim S As Single , X As Single
S = Atn(1) * 4
Print S ' prints 3.141593 PI
End
```

7.7.6 ATN2

Action

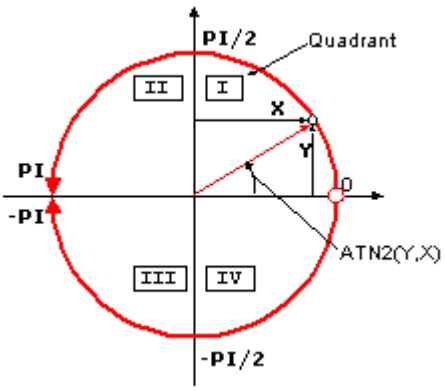
ATN2 is a four-quadrant arc-tangent.
While the ATN-function returns from -p/2 (-90°) to p/2 (90°), the ATN2 function returns the whole range of a circle from -p (-180°) to +p (180°). The result depends on the ratio of Y/X and the signs of X and Y.

Syntax

var = **ATN2**(y, x)

Remarks

Var	A floating point variable that is assigned with the ATN2 of variable y and x.
X	The float variable with the distance in x-direction.
Y	The float variable with the distance in y-direction



Quadrant	Sign Y	Sign X	ATN2
I	+	+	0 to p/2
II	+	-	p/2 to p
III	-	-	-p/2 to -p
IV	-	+	0 to –p/2

If you go with the ratio Y/X into ATN you will get same result for X greater zero (right side in coordinate system) as with ATN2. ATN2 uses X and Y and can give information of the angle of the point over 360° in the coordinates system.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
RAD2DEG 764 , DEG2RAD 747 , COS 746 , SIN 769 , TAN 767 , ATN 742

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As Single , X As Single
X = 0.5 : S = 1.1
S = Atn2(s , X)
Print S ' prints 1.144164676

End
```

7.7.7 CHECKFLOAT

Action
This function validates the value of a floating point variable.

Syntax
targ = **CHECKFLOAT(**var [,option]**)**

Remarks

targ	A numeric variable that will be assigned with the result of the validation. The following bits can be set: cBitInfinity = 0 *cmBitInfinity = 1 ;(2 ^ cBitInfinity)* cBitZero = 1 *cmBitZero = 2 ;(2 ^ cBitZero)* cBitNAN = 2 *cmBitNAN = 4 ;(2 ^ cBitNAN)* cBitSign = 7 *cmBitSign = 128 ;(2 ^ cBitSign)* The byte values are shown in italic. The bit constants are defined in the single and double libraries.

var	A floating point variable such as a single or double to validate.
option	This is an optional numeric constant that servers as a mask. This allows to test for makes it possible to test for a single error.

A floating point value may contain an illegal value as the result of a calculation. These illegal values are NAN (not a number) and INFINITY.
The two other tests which are performed are a test for zero, and a sign test.
If the result bit 0 is '1' then the number is infinity.
If the result bit 1 is '1' then the number is zero.
If the result bit 2 is '1' then the number if NAN.
If the result bit 7 is '1' then the number is negative.

If you want to test only for NAN and INFINITY you can add the bits and pass this as the optional numeric mask. For NAN and INFINITY this would be 1+4=5
The resulting value will be AND-ed and if any of the two bits is set, the result will be non-zero, indicating an error. If both values are 0, the result will be zero.

 This functions works for both the double and single data types. For the single there is however a note. When you divide a number by a real 0, the result is a zero (0).
In the double data type you actually get an INFinite number. For this reason the sample contains a trick with overlayed variables to test the function.

See also
NONE

Example
```
$regfile = "m2561def.dat"
$crystal = 8000000
$hwstack = 64
$swstack = 64
$framesize = 64
$baud = 19200


$lib "single.lbx"

Dim S1 As Single , S2 As Single , S3 As Single
dim d1 as Double , d2 as Double , d3 as Double
dim bCheck as Byte
dim bs(4) as Byte at s3 overlay
dim bd(8) as Byte at d3 overlay


S1 = 0 : Bcheck = Checkfloat(s1) : Print Bin(bcheck)

S1 = 0 : Bcheck = Checkfloat(s1 , 2) : Print Bin(bcheck)



d1 = 1: d2 = 0 : d3 = d1 / d2
    ' 1/0 should result in infinty
Bcheck = Checkfloat(d3) : Print Bin(bcheck)
Bcheck = Checkfloat(d3 , 5) : Print Bcheck                  ' test for
infinity and nan
```

```
d1 = -1
d3 = sqr(d1)            ' should produce NAN
Bcheck = Checkfloat(d3) : Print Bin(bcheck)

' single routines must be checked for returning IEEE-Rulues according
values
s1 = 1: s2 = 0 : s3 = s1 / s2
    ' 1/0 should result in infinty
Bcheck = Checkfloat(s3) : Print Bin(bcheck)

s1 = -1
s3 = sqr(s1)            ' should produce NAN
Bcheck = Checkfloat(s3) : Print Bin(bcheck)


' now check with hard-coded values for singles
bs(1) = &HFF: bs(2) = &HFF: bs(3) = &HFF: bs(4) = &H7F   ' NAN
Bcheck = Checkfloat(s3) : Print Bin(bcheck)

bs(1) = &H00: bs(2) = &H00: bs(3) = &H80: bs(4) = &H7F   ' infinity
Bcheck = Checkfloat(s3) : Print Bin(bcheck)

End
```

7.7.8 COS

Action

Returns the cosine of a floating point variable

Syntax

var = **COS**(float)

Remarks

Var	A numeric variable that is assigned with cosine of variable float.
float	The floating point variable to get the cosine of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG 764 , DEG2RAD 747 , ATN 742 , SIN 769 , TAN 767

Example

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space
```

```
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Dim S As Single , X As Single
S = 0.5 : X = Tan(s) : Print X                                    ' prints
0.546302195
S = 0.5 : X = Sin(s) : Print X                                    ' prints
0.479419108
S = 0.5 : X = Cos(s) : Print X                                    ' prints
0.877588389
End
```

7.7.9 COSH

Action
Returns the cosine hyperbole of a floating point variable

Syntax
var = **COSH**(float)

Remarks

Var	A numeric variable that is assigned with cosine hyperbole of variable float.
float	The single or double variable to get the cosine hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
RAD2DEG 764 , DEG2RAD 747 , ATN 742 , COS 746 , SIN 769 , TANH 767 , SINH 769

Example
Show sample 1827

7.7.10 DEG2RAD

Action
Converts an angle in to radians.

Syntax
var = **DEG2RAD**(angle)

Remarks

Var	A numeric variable that is assigned with the radians of variable Source.
angle	The single or double variable to get the degrees of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between

radians and angles.

Radian is the ratio between the length of an arc and its radius. The radian is the standard unit of angular measure.
You can find a good explanation at wikipedia.

See Also
RAD2DEG 764

Example
```
'-------------------------------------------------------------------------
--------
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates DEG2RAD function

'-------------------------------------------------------------------------
--------
Dim S As Single
S = 90

S = Deg2Rad(s)
Print S
S = Rad2deg(s)
Print S
End
```

7.7.11 EXP

Action
Returns e(the base of the natural logarithm) to the power of a single or double variable.

Syntax
Target = **EXP**(source)

Remarks

Target	The single or double that is assigned with the Exp() of the target.
Source	The source to get the Exp of.

See also
LOG 752 , LOG10 752

Example
```
'-------------------------------------------------------------------------
--------
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : Mega88
'suited for demo          : no, but without the DOUBLE, it works for
```

```
DEMO too in M48
'commercial addon needed  : no
'purpose                  : demonstrates EXP function
'-----------------------------------------------------------------------
--------

$regfile = "m88def.dat"                ' specify the used micro
$crystal = 8000000                     ' used crystal frequency
$baud = 19200                          ' use baud rate
$hwstack = 32                          ' default use 32 for the hardware
stack
$swstack = 40                          ' default use 10 for the SW stack
$framesize = 40                        ' default use 40 for the frame
space


Dim X As Single

X = Exp(1.1)
Print X
'prints 3.004166124
X = 1.1
X = Exp(x)
Print X
'prints 3.004164931


Dim D As Double

D = Exp(1.1)
Print D
'prints 3.00416602394643
D = 1.1
D = Exp(d)
Print D
'prints 3.00416602394638
End
```

7.7.12 FIX

Action

Returns for values greater then zero the next lower value, for values less then zero the next upper value.

Syntax

var = **FIX**(x)

Remarks

Var	A single or double variable that is assigned with the FIX of variable x.
X	The floating point variable to get the FIX of.

See Also

INT 751 , ROUND 765 , SGN 766

Example

```
'-----------------------------------------------------------------------------
'name                     : round_fix_int.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : ROUND,FIX
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------

$regfile = "m48def.dat"                         ' specify the used micro
$crystal = 4000000                              ' used crystal frequency
$baud = 19200                                   ' use baud rate
$hwstack = 32                                   ' default use 32 for the hardware stac
$swstack = 10                                   ' default use 10 for the SW stack
$framesize = 40                                 ' default use 40 for the frame space

Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
  Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

7.7.13 FRAC

Action

Returns the fraction of a single.

Syntax

var = **FRAC**(single)

Remarks

var	A numeric single variable that is assigned with the fraction of variable single.
single	The single variable to get the fraction of.

The fraction is the right side after the decimal point of a single.

See Also

INT 751

Example

```
'-----------------------------------------------------------------------------
--------
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates FRAC function
'-----------------------------------------------------------------------------
--------

$regfile = "m48def.dat"                 ' specify the used micro
$crystal = 8000000                      ' used crystal frequency
$baud = 19200                           ' use baud rate
$hwstack = 32                           ' default use 32 for the
```

```
hardware stack
$swstack = 40                           ' default use 10 for the SW
stack
$framesize = 40                         ' default use 40 for the frame
space


Dim X As Single

X = 1.123456
Print X
Print Frac(x)
End
```

7.7.14 INT

Action
Returns the integer part of a single or double.

Syntax
var = **INT**(source)

Remarks

Var	A numeric floating point variable that is assigned with the integer of variable source.
Source	The source floating point variable to get the integer part of.

The fraction is the right side after the decimal point of a single.
The integer is the left side before the decimal point.

1234.567 1234 is the integer part, .567 is the fraction


The assigned variable must be a single or double. When you want to convert a floating point data type to an integer data type, just assign the variable to a variable of that type : someLong = someDouble

See Also
FRAC 750 , FIX 749 , ROUND 765

Example
```
'--------------------------------------------------------------------
------------------
'name                    : round_fix_int.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demo : ROUND,FIX
'micro                   : Mega48
'suited for demo         : yes
'commercial addon needed  : no
'--------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                     ' specify the used micro
$crystal = 4000000                          ' used crystal frequency
$baud = 19200                               ' use baud rate
```

```
$hwstack = 32                                   ' default use 32 for the
hardware stack
$swstack = 10                                   ' default use 10 for the SW
stack
$framesize = 40                                 ' default use 40 for the
frame space

Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
  Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

7.7.15 LOG10

Action
Returns the base 10 logarithm of a floating point variable.

Syntax
Target = **LOG10**(source)

Remarks

Target	The single or double that is assigned with the base 10 logarithm of single/ double target.
Source	The source single or double to get the base 10 LOG of.

See also
EXP 748 , LOG 752

Example
Show sample 1827

7.7.16 LOG

Action
Returns the natural logarithm of a floating point variable.

Syntax
Target = **LOG**(source)

Remarks

Target	The single or double that is assigned with the LOG() of single target.
Source	The source single or doubler to get the LOG of.

See also
EXP 748 , LOG10 752

Example
Show sample 1827

7.7.17 NOT

Action
This logical operator returns the inversed value.

Syntax
target = **NOT** source2

Remarks
The NOT operator inverts the input bit. When the bit is '0' it will return a '1'. And when the bit is '1' it will return a '0'

A	R
0	1
1	0

The truth table above shows the possible values. A represent the input. R is the Return or output value.

While you can use NOT on bits, you can also perform the same operation on bytes, integers, etc. In such a case, all bits of the variables will be inverted.
Example :
Dim A as Byte, B as Byte, R as byte
A=&B1100_0001
R=NOT A
R=&B00**11_111**0

See also
AND 738 , XOR 770, OR 755

Example
```
'-----------------------------------------------------------------------
----------
'name                     : boolean.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: AND, OR, XOR, NOT, BIT, SET, RESET
and MOD
'suited for demo          : yes
'commercial add on needed : no
'use in simulator         : possible
'-----------------------------------------------------------------------
----------
'This very same program example can be used in the Help-files for
'      AND, OR, XOR, NOT, BIT, SET, RESET and MOD
```

```
$baud = 19200
$crystal = 8000000
$regfile = "m88def.dat"

$hwstack = 40
$swstack = 20
$framesize = 20

Dim A As Byte , B1 As Byte , C As Byte
Dim Aa As Bit , I As Integer

A = 5 : B1 = 3                                          ' assign
values
C = A And B1                                            ' and a
with b
Print "A And B1 = " ; C                                 ' print it:
result = 1


C = A Or B1
Print "A Or B1 = " ; C                                  ' print it:
result = 7

C = A Xor B1
Print "A Xor B1 = " ; C                                 ' print it:
result = 6

A = 1
C = Not A
Print "c = Not A " ; C                                  ' print it:
result = 254
C = C Mod 10
Print "C Mod 10 = " ; C                                 ' print it:
result = 4


If Portb.1 = 1 Then          'test a bit from a PORT (which is not the
same as testing the input state)
  Print "Bit set"
Else
  Print "Bit not set"
End If                                                  'result =
Bit not set

Config Pinb.0 = Input : Portb.0 = 1                     'configure
as input pin
Do
Loop Until Pinb.0 = 0                                   ' repeat
this loop until the logic level becomes 0

Aa = 1                                                  'use this
or ..
```

```
Set Aa                                                    'use the
set statement
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                    'result =
Bit set (aa=1)

Aa = 0                                                    'now try 0
Reset Aa                                                  'or use
reset
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                    'result =
Bit not set(aa=0)

C = 8                                                     'assign
variable to &B0000_1000
Set C                                                     'use the
set statement without specifying the bit
Print C                                                   'print it:
result = 9 ; bit0 has been set

B1 = 255                                                  'assign
variable
Reset B1.0                                                'reset bit
0 of a byte variable
Print B1                                                  'print it:
result = 254 = &B11111110

B1 = 8                                                    'assign
variable to &B00001000
Set B1.7                                                  'set it
Print B1                                                  'print it:
result = 9 = &B00001001
End
```

7.7.18 OR

Action

This logical operator returns the OR of two numeric variables.

Syntax

target = source1 **OR** source2

Remarks

The OR operator works on two bits. It returns a '1' if one of both inputs is '1'.

A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

The truth table above shows all possible values. A and B represent the 2 inputs. R is the Return or output value. As you can see, you will get a '1' when either or both inputs is '1'
It is like having 2 switches in parallel. Both switches will create a closed circuit.

While you can use OR on bits, you can also perform the same operation on bytes, integers, etc. In such a case, all bits of the variables will be OR-ed.
Example :
Dim A as Byte, B as Byte, R as byte
A=&B1100_0001
B=&B1001_0000
R=A OR B

R=&B1001_0001

See also
AND 738 , XOR 770, NOT 753

Example
```
'-----------------------------------------------------------------------
----------
'name                     : boolean.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: AND, OR, XOR, NOT, BIT, SET, RESET
and MOD
'suited for demo          : yes
'commercial add on needed : no
'use in simulator         : possible
'-----------------------------------------------------------------------
----------
'This very same program example can be used in the Help-files for
'      AND, OR, XOR, NOT, BIT, SET, RESET and MOD


$baud = 19200
$crystal = 8000000
$regfile = "m88def.dat"

$hwstack = 40
$swstack = 20
$framesize = 20

Dim A As Byte , B1 As Byte , C As Byte
Dim Aa As Bit , I As Integer

A = 5 : B1 = 3                                              ' assign
```

```
values
C = A And B1                                                ' and a
with b
Print "A And B1 = " ; C                                     ' print it:
result = 1


C = A Or B1
Print "A Or B1 = " ; C                                      ' print it:
result = 7

C = A Xor B1
Print "A Xor B1 = " ; C                                     ' print it:
result = 6

A = 1
C = Not A
Print "c = Not A " ; C                                      ' print it:
result = 254
C = C Mod 10
Print "C Mod 10 = " ; C                                     ' print it:
result = 4


If Portb.1 = 1 Then              'test a bit from a PORT (which is not
the same as testing the input state)
  Print "Bit set"
Else
  Print "Bit not set"
End If                                                      'result =
Bit not set

Config Pinb.0 = Input : Portb.0 = 1                         'configure
as input pin
Do
Loop Until Pinb.0 = 0                                       ' repeat
this loop until the logic level becomes 0

Aa = 1                                                      'use this
or ..
Set Aa                                                      'use the
set statement
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                      'result =
Bit set (aa=1)

Aa = 0                                                      'now try 0
Reset Aa                                                    'or use
reset
If Aa = 1 Then
```

```
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                    'result =
Bit not set(aa=0)

C = 8                                                     'assign
variable to &B0000_1000
Set C                                                     'use the
set statement without specifying the bit
Print C                                                   'print it:
result = 9 ; bit0 has been set

B1 = 255                                                  'assign
variable
Reset B1.0                                                'reset bit
0 of a byte variable
Print B1                                                  'print it:
result = 254 = &B11111110

B1 = 8                                                    'assign
variable to &B00001000
Set B1.7                                                  'set it
Print B1                                                  'print it:
result = 9 = &B00001001
End
```

7.7.19 POWER

Action

Returns the power of a single or double variable and its argument

Syntax

var = **POWER**(source, raise)

Remarks

Var	A numeric variable that is assigned with the power of variable source ^ raise.
Source	The single or double variable to get the power of.

The POWER function works for positive floating point variables only.
When you use a ^ b , the sign will be preserved.

While Excel does not allow raising a negative single, QB does allow it.
The Power functions uses less code compared with the code that is generated when you use ^ for floating point values.
It is important that you use single variables for both single and raise. Constants are not accepted.

In version 1.11.9.2 the power function is improved so that it returns the same result as Excel. Previously it returned the same number as QB/VB. For example : -2 ^ 2

would be returned as -4, but -2 ^ 3 would be returned as -8 which is wring since -2 ^ 3 = -2 x -2 x -2=4 x -2 = -8. Minus times a minutes makes a positive number. So it depends on the sign of the base and if the number of raise if even or odd.

The exception handling was also improved.

Base	Raise	Result
0	0	NAN
NAN	x	NAN
x	NAN	NAN
Infinity	x	NAN
x	Infinity	NAN
0	x<0	Infinity
0	x>0	0
x	0	1
x<0	x<>int(x)	NAN

See Also

EXP 748 ,LOG 752, LOG10 752 , SQR 768

Example

Show sample 1827

Example for Double Exceptions

```
$regfile = "m128def.dat"
$crystal = 4000000

Dim D1 As Double , D2 As Double , D3 As Double
Dim dInf as Double, dNAN as Double
d1 = -1: dNAN = log(d1)
d1 = 1: d2 = 0: dInf = D1 / D2
Print "POWER() - Test"
Print "=============="
D1 = 0: D2 = 0: GoSub ShowPowerTest
D1 = dNAN: D2 = 3: GoSub ShowPowerTest
D1 = 3: D2 = dNAN: GoSub ShowPowerTest
D1 = dInf: D2 = 4: GoSub ShowPowerTest
D1 = 4: D2 = dInf: GoSub ShowPowerTest
D1 = 0: D2 = -2: GoSub ShowPowerTest
D1 = 0: D2 = 3: GoSub ShowPowerTest
D1 = 5: D2 = 0: GoSub ShowPowerTest
D1 = -2: D2 = -3.5: GoSub ShowPowerTest
D1 = -2: D2 = 3.5: GoSub ShowPowerTest
D1 = -2: D2 = -3: GoSub ShowPowerTest
D1 = -2: D2 = -4: GoSub ShowPowerTest
D1 = -2: D2 = -5: GoSub ShowPowerTest
D1 = -2: D2 = 3: GoSub ShowPowerTest
D1 = -2: D2 = 4: GoSub ShowPowerTest
D1 = -2: D2 = 5: GoSub ShowPowerTest

end

ShowPowerTest:
D3 = POWER(D1, D2)
Print "POWER( " ; D1 ; " , " ; D2 ; ") = " ; D3
```

```
Return


--------------------------Simulator Output -------------------
POWER() - Test
==============
POWER( 0 , 0) = NAN
POWER( NAN , 3) = NAN
POWER( 3 , NAN) = NAN
POWER( Infinity , 4) = NAN
POWER( 4 , Infinity) = NAN
POWER( 0 , -2) = Infinity
POWER( 0 , 3) = 0
POWER( 5 , 0) = 1
POWER( -2 , -3.5) = NAN
POWER( -2 , 3.5) = NAN
POWER( -2 , -3) = -125E-3
POWER( -2 , -4) = 62.5E-3
POWER( -2 , -5) = -31.25E-3
POWER( -2 , 3) = -8
POWER( -2 , 4) = 16
POWER( -2 , 5) = -32
```

7.7.20 QSIN

Action

Returns the sinus of an integer

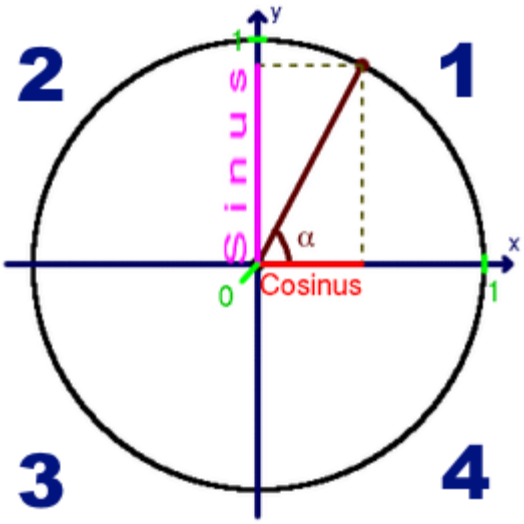
Syntax

var = **QSIN**(source)

Remarks

Var	A numeric integer variable that is assigned with sinus of variable source.
source	The integer variable to get the sinus of.

Integer SIN and COS use a lookup table to determine the Sinus or Co sinus. Qsin and Qcos are used by some of the FT800 routines.



The sinus of angle α is shown above. At 0 degrees the value on the y-ax is 0 and at 90 degrees, the value is at its maximum on the Y-ax. In the first quadrant of the circle (1) sinus will have a positive number as a result. In quadrant 2 of the circle, the sinus goes from the maximum value down to 0 and the result is a positive number as well.
In quadrant 3 and 4 of the circle, we will get a negative number as a result since the result is below the x-ax.
The QSIN works with integers which have a range from -32768 to 32767. This means that for the quadrant 1 and 2 we can use a value between 0 and 32767. In degrees we would use a value between 0 and 180. This means that each degree has a value of 182 (32767/180).
The negative values are reserved for quadrant 3 and 4.
Instead of integers you can also use a word variable.
The following simple sample will show the input and output values.

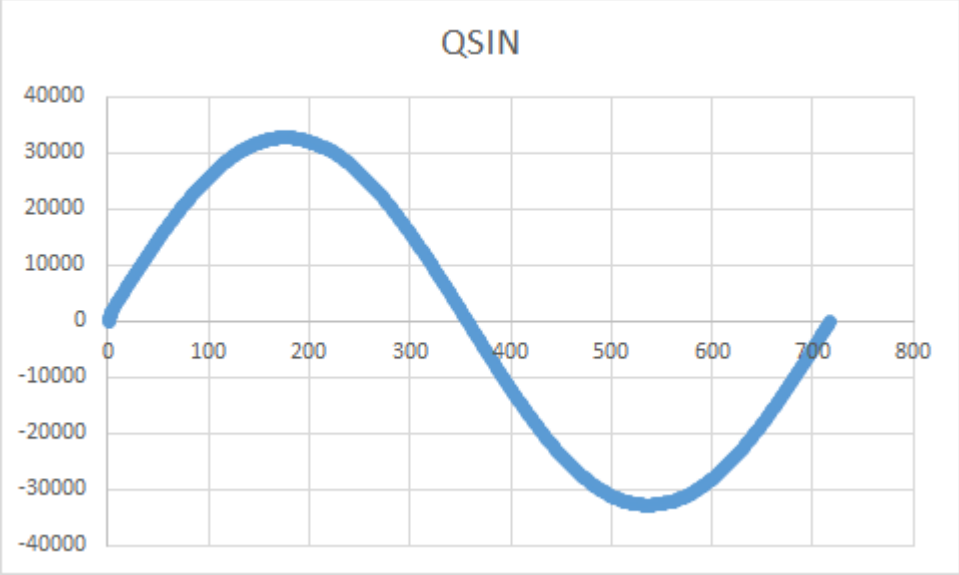
```
Dim Iii As Integer , I2 As Integer , W As Word
For W= 0 To 65535 Step 182
  Iii = W          ' for usage as an integer
  I2 = Qsin(iii)  ' get the value of W/III
  Print W; "  " ; Iii ; "  " ; I2
Next
```

This will give the output :

```
W III  QSIN
0  0  0
182  182  571
364  364  1143
546  546  1713
728  728  2284
910  910  2854
1092  1092  3423
1274  1274  3992
1456  1456  4558
1638  1638  5124
1820  1820  5687

--snip--

15470  15470  32640
15652  15652  32685
15834  15834  32720
16016  16016  32745
16198  16198  32760
16380  16380  32766
16562  16562  32761
16744  16744  32746
16926  16926  32721
17108  17108  32687
17290  17290  32643
17472  17472  32588
17654  17654  32523
17836  17836  32448
18018  18018  32364
18200  18200  32270
18382  18382  32166
18564  18564  32053
18746  18746  31929
```

```
18928  18928  31796
19110  19110  31653
19292  19292  31500
19474  19474  31339
19656  19656  31166
19838  19838  30986
20020  20020  30794
20202  20202  30595
20384  20384  30386
20566  20566  30167
20748  20748  29939
20930  20930  29702
21112  21112  29456
21294  21294  29202
21476  21476  28938
21658  21658  28666
21840  21840  28384
22022  22022  28095
22204  22204  27796
22386  22386  27489
22568  22568  27173
22750  22750  26850

--snip--


32214  32214  1738
32396  32396  1168
32578  32578  596
32760  32760  25
32942  -32594  -546
33124  -32412  -1118
33306  -32230  -1688
33488  -32048  -2259
33670  -31866  -2829
33852  -31684  -3398

--snip --

48230  -17306  -32638
48412  -17124  -32683
48594  -16942  -32719
48776  -16760  -32744
48958  -16578  -32760
49140  -16396  -32766
49322  -16214  -32761
49504  -16032  -32747
49686  -15850  -32723
49868  -15668  -32688
50050  -15486  -32645
50232  -15304  -32590
50414  -15122  -32526
50596  -14940  -32452
50778  -14758  -32368
50960  -14576  -32275
51142  -14394  -32171
51324  -14212  -32058
51506  -14030  -31934
```

```
51688  -13848  -31802
51870  -13666  -31659
52052  -13484  -31507
52234  -13302  -31346
52416  -13120  -31174
52598  -12938  -30994
52780  -12756  -30803
52962  -12574  -30604
53144  -12392  -30395

--snip--

63154  -2382  -7417
63336  -2200  -6859
63518  -2018  -6299
63700  -1836  -5737
63882  -1654  -5173
64064  -1472  -4608
64246  -1290  -4042
64428  -1108  -3473
64610  -926  -2904
64792  -744  -2334
64974  -562  -1764
65156  -380  -1193
65338  -198  -621
65520  -16  -50
```



When we feed the output in Excel we can create the graph above, showing a perfect sinus.

See Also

QCOS 764

Example

NONE

7.7.21 QCOS

Action
Returns the Co Sinus of an integer

Syntax
var = **QCOS**(source)

Remarks

Var	A numeric integer variable that is assigned with sinus of variable source.
source	The integer variable to get the Co Sinus of.

Integer SIN and COS use a lookup table to determine the Sinus or Co sinus. Qsin and Qcos are used by some of the FT800 routines.

See Also
SIN 760

Example
NONE

7.7.22 RAD2DEG

Action
Converts a value from radians to degrees.

Syntax
var = **RAD2DEG**(Source)

Remarks

Var	A numeric variable that is assigned with the angle of variable source.
Source	The single or double variable to get the angle of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
DEG2RAD 747

Example
```
'-----------------------------------------------------------------
--------
```

```
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates DEG2RAD function

'-----------------------------------------------------------------------
--------
Dim S As Single
S = 90

S = Deg2Rad(s)
Print S
S = Rad2deg(s)
Print S
End
```

7.7.23 ROUND

Action

Returns a value rounded to the nearest value.

Syntax

var = **ROUND**(x)

Remarks

Var	A single or double variable that is assigned with the ROUND of variable x.
X	The single or double to get the ROUND of.

Round(2.3) = 2 , Round(2.8) = 3
Round(-2.3) = -2 , Round(-2.8) = -3

See Also

INT 751 , FIX 749 , SGN 766

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : round_fix_int.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : ROUND,FIX
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
```

```
use 32 for the hardware stack
$swstack = 10                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space

Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
  Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

7.7.24 SGN

Action

Returns the sign of a numeric value.

Syntax

var = **SGN**(x)

Remarks

Var	A numeric variable that is assigned with the SGN() of variable x.
X	The numeric variable to get the sign of.

For values < 0, -1 will be returned
For 0, 0 will be returned
For values >0, 1 will be returned

While the SGN function can return a negative value, it can only do so for integers, longs, singles and doubles.
When a byte, word or dword is passed, only 0 or 1 can be returned since these values do not contain a sign bit.
When a byte,word or dword is passed, the returned value is a byte.
When an integer is passed, the returned value is an integer.
When a long is passed, the returned value is a long.
When a single is passed, the returned value is a single.
When a double is passed, the returned value is a double.

See Also

INT 751 , FIX 749 , ROUND 765

Example

```
Dim S As Single , X As Single , Y As Single
X = 2.3 : S = Sgn(x)
Print S
X = -2.3 : S = Sgn(x)
Print S
End
```

7.7.25 TANH

Action
Returns the hyperbole of a floating point variable

Syntax
var = **TANH**(source)

Remarks

Var	A numeric variable that is assigned with hyperbole of variable source.
Source	The single or double variable to get the hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
RAD2DEG 764 , DEG2RAD 747 , ATN 742 , COS 746 , SIN 769 , SINH 769 , COSH 747

Example
Show sample 1827

7.7.26 TAN

Action
Returns the tangent of a float

Syntax
var = **TAN**(source)

Remarks

Var	A numeric variable that is assigned with tangent of variable source.
Source	The single or double variable to get the tangent of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
RAD2DEG 764 , DEG2RAD 747 , ATN 742 , COS 746 , SIN 769 , ATN2 743

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
```

```
$hwstack = 32                                                    ' default
use 32 for the hardware stack
$swstack = 10                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Dim S As Single , X As Single
S = 0.5 : X = Tan(s) : Print X                                   ' prints
0.546302195
S = 0.5 : X = Sin(s) : Print X                                   ' prints
0.479419108
S = 0.5 : X = Cos(s) : Print X                                   ' prints
0.877588389
End
```

7.7.27 SQR

Action

Returns the Square root of a variable.

Syntax

var = **SQR**(source)

Remarks

var	A numeric single or double variable that is assigned with the SQR of variable source.
source	The single or double variable to get the SQR of.

When SQR is used with a single, the FP_TRIG library will be used.
When SQR is used with bytes, integers, words and longs, the SQR routine from MCS. LBX will be used.

See Also

POWER 758

Example

```
$regfile = "m48def.dat"                                          ' specify
the used micro
$crystal = 8000000                                               ' used
crystal frequency
$baud = 19200                                                    ' use baud
rate
$hwstack = 32                                                    ' default
use 32 for the hardware stack
$swstack = 40                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space
```

```
Dim A As Single
Dim B As Double
A = 9.0
B = 12345678.123

A =Sqr(A)
Print A                                                    ' prints 3.0
B = Sqr(b)
Print B
End
```

7.7.28 SINH

Action
Returns the sinus hyperbole of a float

Syntax
var = **SINH**(source)

Remarks

Var	A numeric variable that is assigned with sinus hyperbole of variable source.
source	The single or double variable to get the sinus hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
RAD2DEG 764 , DEG2RAD 747 , ATN 742 , COS 746 , SIN 769 , TANH 767 , COSH 747

Example
Show sample 1827

7.7.29 SIN

Action
Returns the sine of a float

Syntax
var = **SIN**(source)

Remarks

Var	A numeric variable that is assigned with sinus of variable source.
source	The single or double variable to get the sinus of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also
RAD2DEG 764 , DEG2RAD 747 , ATN 742 , COS 746

Example
```
$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As Single , X As Single
S = 0.5 : X = Tan(s) : Print X                              ' prints
0.546302195
S = 0.5 : X = Sin(s) : Print X                              ' prints
0.479419108
S = 0.5 : X = Cos(s) : Print X                              ' prints
0.877588389
End
```

7.7.30 XOR

Action
This logical operator returns the XOR of two numeric variables.

Syntax
target = source1 **XOR** source2

Remarks
The XOR operator works on two bits. It returns a '1' if both inputs are different.

A	B	R
0	0	0
0	1	1
1	0	1
1	1	0

The truth table above shows all possible values. A and B represent the 2 inputs. R is the Return or output value. As you can see, you will get a '1' when both inputs differ.

While you can use XOR on bits, you can also perform the same operation on bytes, integers, etc. In such a case, all bits of the variables will be XOR-ed.
Example :
Dim A as Byte, B as Byte, R as byte
A=&B1100_0001

B=&B1001_0000
R=A XOR B

R=&B0101_0001

See also

AND 738 , OR 755, NOT 753

Example

```
'--------------------------------------------------------------------
----------
'name                       : boolean.bas
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : demo: AND, OR, XOR, NOT, BIT, SET, RESET
and MOD
'suited for demo            : yes
'commercial add on needed   : no
'use in simulator           : possible
'--------------------------------------------------------------------
----------
'This very same program example can be used in the Help-files for
'      AND, OR, XOR, NOT, BIT, SET, RESET and MOD


$baud = 19200
$crystal = 8000000
$regfile = "m88def.dat"

$hwstack = 40
$swstack = 20
$framesize = 20

Dim A As Byte , B1 As Byte , C As Byte
Dim Aa As Bit , I As Integer

A = 5 : B1 = 3                                          ' assign
values
C = A And B1                                            ' and a
with b
Print "A And B1 = " ; C                                 ' print it:
result = 1


C = A Or B1
Print "A Or B1 = " ; C                                  ' print it:
result = 7

C = A Xor B1
Print "A Xor B1 = " ; C                                 ' print it:
result = 6
```

```
A = 1
C = Not A
Print "c = Not A " ; C                                    ' print it:
result = 254
C = C Mod 10
Print "C Mod 10 = " ; C                                   ' print it:
result = 4


If Portb.1 = 1 Then          'test a bit from a PORT (which is not the
same as testing the input state)
  Print "Bit set"
Else
  Print "Bit not set"
End If                                                    'result =
Bit not set

Config Pinb.0 = Input : Portb.0 = 1                       'configure
as input pin
Do
Loop Until Pinb.0 = 0                                     ' repeat
this loop until the logic level becomes 0

Aa = 1                                                    'use this
or ..
Set Aa                                                    'use the
set statement
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                    'result =
Bit set (aa=1)

Aa = 0                                                    'now try 0
Reset Aa                                                  'or use
reset
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                    'result =
Bit not set(aa=0)

C = 8                                                     'assign
variable to &B0000_1000
Set C                                                     'use the
set statement without specifying the bit
Print C                                                   'print it:
result = 9 ; bit0 has been set

B1 = 255                                                  'assign
```

```
variable
Reset B1.0                                                  'reset bit
0 of a byte variable
Print B1                                                    'print it:
result = 254 = &B11111110

B1 = 8                                                      'assign
variable to &B00001000
Set B1.7                                                    'set it
Print B1                                                    'print it:
result = 9 = &B00001001
End
```

7.8 AVR-DOS File I/O

7.8.1 BLOAD

Action

Writes the Content of a File into SRAM

Syntax

BLoad sFileName, wSRAMPointer

Remarks

sFileName	(String) Name of the File to be read
wSRAMPointer	(Word) Variable, which holds the SRAM Address to which the content of the file should be written

This function writes the content of a file to a desired space in SRAM. A free handle is needed for this function.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_BLoad	
Input	X: Pointer to string with filename	Z: Pointer to Long-variable, which holds the start position of SRAM
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
' THIS IS A CODE FRAGMENT, it needs AVR-DOS in order to work
'now the good old bsave and bload
Dim Ar(100)as Byte , I Asbyte
For I = 1 To 100
  Ar(i) = I                                                ' fill the
array
Next

Wait 2

W = Varptr(ar(1))
Bsave"josef.img", W , 100
For I = 1 To 100
  Ar(i) = 0                                                ' reset the
array
Next

Bload "josef.img" , W                                      ' Josef you
are amazing !

For I = 1 To 10
  Print Ar(i) ; " ";
Next
Print
```

7.8.2 BSAVE

Action

Save a range in SRAM to a File

Syntax

BSave sFileName, wSRAMPointer, wLength

Remarks

sFileName	(String) Name of the File to be written
wSRAMPointer	(Word) Variable, which holds the SRAM Address, from where SRAM should be written to a File
wLength	(Word) Count of Bytes from SRAM, which should be written to the file

This function writes a range from the SRAM to a file. A free file handle is needed for this function.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_BSave	
Input	X: Pointer to string with	Z: Pointer to Long-variable, which holds the

	filename	start position of SRAM
	r20/r21: Count of bytes to be written	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
' THIS IS A CODE FRAGMENT, it needs AVR-DOS in order to work
'now the good old bsave and bload
Dim Ar(100)as Byte , I Asbyte
For I = 1 To 100
  Ar(i) = I                                                 ' fill the
array
Next

Wait 2

W = Varptr(ar(1))
Bsave"josef.img", W , 100
For I = 1 To 100
  Ar(i) = 0                                                 ' reset the
array
Next

Bload "josef.img" , W                                       ' Josef you
are amazing !

For I = 1 To 10
  Print Ar(i) ; " ";
Next
Print
```

7.8.3 CHDIR

Action

This statement will change the current directory.

Syntax

CHDIR directory

Remarks

CHange **DIR**ectory changes the current folder or directory.
The directory name must be a valid and existing folder or directory. Like in DOS, you can use ".." to go back one directory.
And you can use "\" to go to the root directory.
You can not specify a path.

You may can have multiple open files, and you could copy from one folder to another folder using the file handles.

The DIR command and the OPEN command works in the current directory. IF you OPEN a file, the position (Sector# and position inside this sector) of the directory entry of the file is stored at the file-handle-part of that file. So you can move to another directory and OPEN there a second file an so on. In Short: Directory nesting is not limited and you can open files in multiple directories.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILELEN 788 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , WRITE 801 , INPUT 1493 , DIR 777, MKDIR 796, RMDIR 798 , NAME 797

Example

```
MKDIR "abc"
CHDIR "abc"
```

7.8.4 CLEARATTR

Action

Clears the file Attribute.

Syntax

CLEARATTR [sFile] , bFileAttribute

Remarks

sFile	The name of the file (no wildcard) which attribute need to be cleared. You may also omit the name in which case the file will be used previous found by the DIR() function.
bFileAttribute	Numeric variable holding the attribute bits to clear.

This functions clears the DOS file attributes. A file can have multiple attributes.
You should not use attributes 8(Volume) and 16(Sub Directory) on a normal file.

Return value	DOS Attribute
1	Read Only
2	Hidden
4	System File
8	Volume Label
16	Sub Directory
32	Archive
64,128	reserved

A file can have multiple bits set like 3 (hidden + read only). So you can clear multiple bits by combining the bits.

When you specify the filename, make sure it does not have a wildcard. CLEARTTR does not support wildcards.
When you omit the filename, the last found file from DIR 777() will be used for the operation.

In VB, CLEARATTR expects a new value for the attribute which replaces the old

attribute byte.
In AVR-DOS you specify the bits to clear. So old attribute bits which are not altered are kept.
In AVR-DOS you can also set the individual bits using the SETRATTR statement.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788, WRITE 801 , INPUT 1493 , FILEATTR 786 , SETATTR 798 , GETATTR 791

Example

See SETATTR 798

7.8.5 DIR

Action

Returns the filename that matches the specified file mask.

Syntax

sFile = **DIR**(mask)
sFile = **DIR**()

Remarks

SFile	A string variable that is assigned with the filename.
Mask	A file mask with a valid DOS file mask like *.TXT Use *.* to select all files.

The first function call needs a file mask. All other calls do not need the file mask. In fact when you want to get the next filename from the directory, you must not provide a mask after the first call.

Dir() returns an empty string when there are no more files or when no file name is found that matches the mask.

 You should not use directory/file manipulation functions between DIR(mask) and DIR(). This because the first use of DIR() with a mask will search the FAT and will set up a pointer to this table. The next use of DIR() will continue to search the next match. But when you change the directory, or create a file or directory the pointer will be lost.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILELEN 788 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , WRITE 801 , INPUT 1493 , MKDIR 796, RMDIR 798 ,

CHDIR 775

ASM

Calls	_Dir ; with file mask	_Dir0 ; without file mask
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Partial Example

```
'Lets have a look at the file we created
Print "Dir function demo"
S = Dir("*.*")
'The first call to the DIR() function must contain a file mask
' The * means everything.
'
While Len(s)> 0 ' if there was a file found
  Print S ;" ";Filedate();" ";Filetime();" ";Filelen()
' print file , the date the fime was created/changed , the time and the size of the
  S = Dir()' get next
Wend
```

7.8.6 DISKFREE

Action

Returns the free size of the Disk in KB.

Syntax

lFreeSize = **DISKFREE**()

Remarks

lFreeSize	A Long Variable, which is assigned with the available Bytes on the Disk in Kilo Bytes.

This functions returns the free size of the disk in KB.
With the support of FAT32, the return value was changed from byte into KB.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_GetDiskFreeSize
Input	none
Output	r16-r19: Long-Value of free Bytes

Partial Example

```
Dim Gbtemp1 As Byte                    ' scratch byte
Gbtemp1 =Initfilesystem(1)             ' we must init the filesystem once
If Gbtemp1 > 0 Then
   Print#1 ,"Error "; Gbtemp1
Else
   Print#1 ," OK"
Print "Disksize : ";Disksize()         ' show disk size in bytes
Print "Disk free: ";Diskfree()         ' show free space too
End If
```

7.8.7 DISKSIZE

Action

Returns the size of the Disk in KB.

Syntax

lSize = **DISKSIZE**()

Remarks

lSize	A Long Variable, which is assigned with the capacity of the disk in Kilo Bytes

This functions returns the capacity of the disk in KB.
With the support of FAT32, the return value was changed from byte into KB.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_GetDiskSize
Input	none
Output	16-r19: Long-Value of capacity in Bytes

Partial Example

```
Dim Gbtemp1 As Byte                    ' scratch byte
Gbtemp1 = Initfilesystem(1)            ' we must init the filesystem once
If Gbtemp1 > 0 Then
   Print#1 ,"Error "; Gbtemp1
Else
   Print#1 ," OK"
Print "Disksize : "; Disksize()        ' show disk size in bytes
Print "Disk free: "; Diskfree()        ' show free space too
End If
```

7.8.8 DriveCheck

Action
Checks the Drive, if it is ready for use

Syntax
bErrorCode = **DRIVECHECK**()

Remarks

bErrorCode	A Byte Variable, which is assigned with the return value of the function

This function checks the drive, if it is ready for use (for example, whether a compact flash card is inserted). The functions returns 0 if the drive can be used, otherwise an error code is returned. For Error code see section Error codes.

See also
DriveReset 782 , DriveInit 781 , DriveGetIdentity 780 , DriveWriteSector 784 , DriveReadSector 782

ASM

Calls	_DriveCheck	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example
```
Dim bError as Byte
bError = DriveCheck()
```

7.8.9 DriveGetIdentity

Action
Returns the Parameter information from the Card/Drive

Syntax
bErrorCode = **DRIVEGETIDENTITY**(wSRAMPointer)

Remarks

BErrorCode	A Byte Variable, which is assigned with the error code of the function
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer) , to which the information of the Drive should be written

The Identify Drive Function returns the parameter information (512 Bytes) from the Compact Flash Memory Card/Drive and writes it to SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing. This information are for example number of sectors of the card, serial number and so on. Refer to the Card/

Drive manual for further information. The functions returns 0 if no error occurred. For Error code see section Error codes.

Note: For meaning of wSRAMPointer see Note in DriveReadSector

See also

DriveCheck 780, DriveReset 782 , DriveInit 781 , DriveWriteSector 784 , DriveReadSector 782

ASM

Calls	_DriveGetIdentity	
Input		Z: SRAM-Address of buffer *)
Output	r25: Errorcode	C-Flag: Set on Error

 *) Please note: This is not the address of wSRAMPointer, it is its content, which is the starting-address of the buffer.

Partial Example

```
Dim bError as Byte
Dim aBuffer(512) as Byte' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word' Address-Pointer for write

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer =VarPtr(aBuffer(1))


' Now read the parameter Information from CF-Card
bError = DriveGetIdentity( wSRAMPointer)
```

7.8.10 DriveInit

Action

Sets the AVR-Hardware (PORTs, PINs) attached to the Drive and resets the Drive.

Syntax

bErrorCode = **DRIVEINIT**()

Remarks

BErrorCode	A Byte Variable, which is assigned with the error code of the function

Set the Ports and Pins attaching the Drive for Input/Output and give initial values to the output-pins. After that the Drive is reset.
Which action is done in this function depends of the drive and its kind of connection to the AVR. The functions returns 0 if no error occurred.
For Error code see section Error codes.

See also

DriveCheck 780, DriveReset 782 , DriveGetIdentity 780 , DriveWriteSector 784 , DriveReadSector 782, AVR-DOS File System 1794

ASM

Calls	_DriveInit	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

```
Dim bError as Byte
bError = DriveInit()
```

7.8.11 DriveReset

Action

Resets the Drive.

Syntax

bErrorCode = **DRIVERESET**()

Remarks

BErrorCode	A Byte Variable, which is assigned with the error code of the function

This function resets the drive and brings it to an initial state. The functions returns 0 if no error occurred. For Error code see section Error codes.

See also

DriveCheck 780, DriveInit 781 , DriveGetIdentity 780 , DriveWriteSector 784 , DriveReadSector 782

ASM

Calls	_DriveReset	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

```
Dim bError as Byte
bError = DriveReset()
```

7.8.12 DriveReadSector

Action

Read a Sector (512 Bytes) from the (Compact Flashcard) Drive

Syntax
bErrorCode = **DRIVEREADSECTOR**(wSRAMPointer, lSectorNumber)

Remarks

bErrorCode	A Byte Variable, which is assigned with the error code of the function
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer) , to which the Sector from the Drive should be written
lSectorNumber	A Long Variable, which give the sector number on the drive be transfer.

Reads a Sector (512 Bytes) from the Drive and write it to SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing. The functions returns 0 if no error occurred. For Error code see section Error codes.

Note: wSRAMPointer is not the variable, to which the content of the desired drive-sector should be written, it is the Word-Variable/Value which contains the SRAM address of the range, to which 512 Bytes should be written from the Drive. This gives you the flexibility to read and write every SRAM-Range to and from the drive, even it is not declared as variable. If you know the SRAM-Address (from the compiler report) of a buffer you can pass this value directly, otherwise you can get the address with the BASCOM-function VARPTR (see example).

See also
DriveCheck 780, DriveReset 782 , DriveInit 781 , DriveGetIdentity 780 , DriveWriteSector 784

ASM

Calls	_DriveReadSector	
Input	Z: SRAM-Address of buffer *)	X: Address of Long-variable with sectornumber
Output	r25: Errorcode	C-Flag: Set on Error

 This is not the address of wSRAMPointer, it is its content, which is the starting-address of the buffer.

Partial Example
```
Dim bError as Byte
Dim aBuffer(512)as Byte' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word' Address-Pointer for write
Dim lSectorNumber as Long' Sector Number

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer =VarPtr(aBuffer(1))

' Set Sectornumber, sector 32 normally holds the Boot record sector of first partit
lSectorNumber = 32

' Now read in sector 32 from CF-Card
```

```
bError = DriveReadSector( wSRAMPointer , lSectorNumber)
' Now Sector number 32 is in Byte-Array bBuffer
```

7.8.13 DriveWriteSector

Action

Write a Sector (512 Bytes) to the (Compact Flashcard) Drive

Syntax

bErrorCode = **DRIVEWRITESECTOR**(wSRAMPointer, lSectorNumber)

Remarks

bErrorCode	A Byte Variable, which is assigned with the error code of the function
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer), from which the Sector to the Drive should be written
lSectorNumber	A Long Variable, which give the sector number on the drive to transfer.

Writes a Sector (512 Bytes) from SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing to the Drive to sector number lSectornumber. The functions returns 0 if no error occurred. For Error code see section Error codes.

 For the meaning of wSRAMPointer see Note in DriveReadSector

See also

DriveCheck 780, DriveReset 782 , DriveInit 781 , DriveGetIdentity 780 , DriveReadSector 782

ASM

Calls	_DriveWriteSector	
Input	Z: SRAM-Address of buffer *)	X: Address of Long-variable with sectornumber
Output	r25: Errorcode	C-Flag: Set on Error

 This is not the address of wSRAMPointer, it is its content, which is the starting-address of the buffer.

Partial Example

```
Dim bError as Byte
Dim aBuffer(512) as Byte' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word' Address-Pointer for read
Dim lSectorNumber as Long' Sector Number


' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer =VarPtr(aBuffer(1))

' Set Sectornumber
```

```
lSectorNumber = 3

' Now Write in sector 3 from CF-Card
bError = DriveWriteSector( wSRAMPointer , lSectorNumber)
```

7.8.14 EOF

Action
Returns the End of File Status.

Syntax
bFileEOFStatus = **EOF**(#bFileNumber)

Remarks

bFileEOFStatus	(Byte) A Byte Variable, which assigned with the EOF Status
bFileNumber	(Byte) Number of the opened file

This functions returns information about the End of File Status

Return value	Status
0	NOT EOF
255	EOF

In case of an error (invalid file number) 255 (EOF) is returned too.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_FileEOF	
Input	r24: Filenumber	
Output	r24: EOF Status	r25: Error code
	C-Flag: Set on Error	

Partial Example
```
Ff =Freefile()' get file handle
Open "test.txt" For Input As #ff    ' we can use a constant for the file too
Print Lof(#ff); " length of file"
Print Fileattr(#ff); " file mode"   ' should be 1 for input
Do
  LineInput #ff , S                 ' read a line
  ' line input is used to read a line of text from a file
```

```
  Print S                              ' print on terminal emulator
Loop Until Eof(#ff)<> 0
'The EOF() function returns a non-zero number when the end of the file is reached
'This way we know that there is no more data we can read
Close #ff
```

7.8.15 FILEATTR

Action

Returns the file open mode.

Syntax

bFileAttribut = **FILEATTR**(bFileNumber)

Remarks

bFileAttribut	(Byte) File open mode, See table
bFileNumber	(Byte) Number of the opened file

This functions returns information about the File open mode

Return value	Open mode
1	INPUT
2	OUTPUT
8	APPEND
32	BINARY

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788, WRITE 801 , INPUT 1493 , GETATTR 791

ASM

Calls	_FileAttr	
Input	r24: Filenumber	
Output	24: File open mode	r25: Errorcode
	C-Flag: Set on Error	

Partial Example

```
'open the file in BINARY mode
Open "test.biN" For Binary As #2
Print Fileattr(#2); " file mode"     ' should be 32 for binary
Put #2 , Sn                          ' write a single
Put #2 , Stxt                        ' write a string
Close #2
```

7.8.16 FILEDATE

Action
Returns the date of a file

Syntax
sDate = **FILEDATE** ()
sDate = **FILEDATE** (file)

Remarks

Sdate	A string variable that is assigned with the date.
File	The name of the file to get the date of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779, GET 1262 , PUT 1402, FILELEN 788 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , WRITE 801 , INPUT 1493

ASM

Calls	_FileDateS ; with filename	_FileDateS0 ; for current file from DIR ()
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Partial Example
```
Print "File demo"
Print Filelen("josef.img");" length"      ' length of file
Print Filetime("josef.img");" time"       ' time file was changed
Print Filedate("josef.img");" date"       ' file date
```

7.8.17 FILEDATETIME

Action
Returns the file date and time of a file

Syntax
Var = **FILEDATETIME** ()
Var = **FILEDATETIME** (file)

Remarks

Var	A string variable or byte array that is assigned with the file date and time

	of the specified file
File	The name of the file to get the date time of.

When the target variable is a string, it must be dimensioned with a length of at least 17 bytes.
When the target variable is a byte array, the array size must be at least 6 bytes.

When you use a numeric variable, the internal file date and time format will be used.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , GET 1262 , PUT 1402 , FILELEN 788 , FILEDATE 787 , FILETIME 789 , DIR 777 , WRITE 801 , INPUT 1493

ASM

Calls	_FileDateTimeS	_FileDateTimeS0
Input		
Output		

Calls	_FileDateTimeB	_FileDateTimeB0
Input		
Output		

Example
See fs_subfunc_decl_lib.bas in the samples dir.

7.8.18 FILELEN

Action
Returns the size of a file

Syntax
lSize = **FILELEN** ()
lSize = **FILELEN** (file)

Remarks

lSize	A Long Variable, which is assigned with the file size in bytes of the file.
File	A string or string constant to get the file length of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME

787 , DIR 777 , WRITE 801 , INPUT 1493

ASM

Calls	_FileLen	
Input		
Output		

Partial Example

```
Print "File demo"
Print Filelen("josef.img");" length"      ' length of file
Print Filetime("josef.img");" time"       ' time file was changed
Print Filedate("josef.img");" date"       ' file date
```

7.8.19 FILETIME

Action

Returns the time of a file

Syntax

sTime = **FILETIME** ()
sTime = **FILETIME** (file)

Remarks

Stime	A string variable that is assigned with the file time.
File	The name of the file to get the time of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , GET 1262 , PUT 1402 , FILELEN 788 , FILEDATE 787 , FILEDATETIME 787 , DIR 777 , WRITE 801 , INPUT 1493

ASM

Calls	_FileTimeS ; with file param	_FileTimeS0 ; current file
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Example

```
Print "File demo"
Print Filelen("josef.img");" length"      ' length of file
Print Filetime("josef.img");" time"       ' time file was changed
Print Filedate("josef.img");" date"       ' file date
```

7.8.20 FLUSH

Action
Write current buffer of File to Card and updates Directory

Syntax
FLUSH #bFileNumber
FLUSH

Remarks

BFileNumber	Filenumber, which identifies an opened file such as #1 or #ff

This function writes all information of an open file, which is not saved yet to the Disk. Normally the Card is updated, if a file will be closed or changed to another sector.

When no file number is specified, all open files will be flushed.
Flush does not need additional buffers. You could use FLUSH to be absolutely sure that data is written to the disk. For example in a data log application which is updated infrequently. A power failure could result in a problem when there would be data in the buffer.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_FileFlush	_FilesAllFlush
Input	r24: filenumber	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example
```
$include "startup.inc"

'open the file in BINARY mode
Open "test.bin" For Binary As #2
Put #2 , B                              ' write a byte
Put #2 , W                              ' write a word
Put #2 , L                              ' write a long
Ltemp = Loc(#2) + 1                     ' get the position of the next byte
Print Ltemp ;" LOC"                     ' store the location of the file pointer
Print Lof(#2);" length of file"
Print Fileattr(#2);" file mode"         ' should be 32 for binary
Put #2 , Sn                             ' write a single
Put #2 , Stxt                           ' write a string

Flush #2                                ' flush to disk
Close #2
```

7.8.21 FREEFILE

Action
Returns a free Filenumber.

Syntax
bFileNumber = **FREEFILE**()

Remarks

bFileNumber	A byte variable , which can be used for opening next file

This function gives you a free file number, which can be used for file – opening statements. In contrast to VB this file numbers start with 128 and goes up to 255. Use range 1 to 127 for user defined file numbers to avoid file number conflicts with the system numbers from FreeFile()

This function is implemented for compatility with VB.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_GetFreeFileNumber	
Input	none	
Output	r24: Filenumber	r25: Errorcode
	C-Flag: Set on Error	

Partial Example
```
Ff =Freefile()                      ' get file handle
Open"test.txt" For Input As #ff     ' we can use a constant for the file too
Print Lof(#ff);" length of file"
Print Fileattr(#ff);" file mode"    ' should be 1 for input
Do
   LineInput #ff , S                ' read a line
  ' line input is used to read a line of text from a file
  Print S ' print on terminal emulator
Loop UntilEof(ff)<> 0
'The EOF() function returns a non-zero number when the end of the file is reached
'This way we know that there is no more data we can read
Close #ff
```

7.8.22 GETATTR

Action
Returns the file Attribute.

Syntax
bFileAttribut = **GETATTR**([sFile])

Remarks

bFileAttribut	Numeric variable which is assigned with the file attribute.
sFile	The name of the file (no wildcard) to get the attribute from. You may also omit the name in which case the file will be used previous found by the DIR() function.

This functions returns the DOS file attributes. A file can have multiple attributes.

Return value	DOS Attribute
1	Read Only
2	Hidden
4	System File
8	Volume Label
16	Sub Directory
32	Archive
64,128	reserved

A file could have an attribute of 3 (hidden+ read only).

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788, WRITE 801 , INPUT 1493 , FILEATTR 786

Partial Example
```
'open the file in BINARY mode

Print Getattr("somefile.bin")
```

7.8.23 INITFILESYSTEM

Action
Initialize the file system

Syntax
bErrorCode = **INITFILESYSTEM** (bPartitionNumber)

Remarks

bErrorCode	(Byte) Error Result from Routine, Returns 0 if no Error
bPartitionNumber	(Byte) Partition number on the Flashcard Drive (normally 1)

Reads the Master boot record and the partition boot record (Sector) from the flash card and initializes the file system.
This function must be called before any other file-system function is used.

See also

OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493, AVR-DOS File System 1794

ASM

Calls	_GetFileSystem	
Input	r24: partitionnumber (1-based)	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

```
Dim bErrorCode as Byte
bErrorCode = InitFileSystem(1)
If bErrorCode > 0 then
   Print "Error: "; bErrorCode
Else
   Print "Filesystem successfully initialized"
End If
```

7.8.24 KILL

Action

Delete a file from the Disk

Syntax

KILL sFileName

Remarks

sFileName	A String variable or string expression, which denotes the file to delete

This function deletes a file from the disk. A file in use can't be deleted. WildCards in Filename are not supported. Check the DOS-Error in variable gDOSError.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_DeleteFile	
Input	X: Pointer to string with	

	filename	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

```
'We can use the KILL statement to delete a file.
'A file mask is not supported
Print "Kill (delete) file demo"
Kill "test.txt"
```

7.8.25 LINEINPUT

Action

Read a Line from an opened File.

Syntax

LINEINPUT #bFileNumber, sLineText
LINE_INPUT #bFileNumber, sLineText

Remarks

BfileNumber	(Byte) File number, which identifies an opened file
SlineText	(String) A string, which is assigned with the next line from the file.

Only valid for files opened in mode INPUT. Line INPUT works only with strings. It is great for working on text files.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_FileLineInput	
Input	r24: filenumber	X: Pointer to String to be written from file
	r25: Stringlength	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
'Ok we want to check if the file contains the written lines
Ff = Freefile()' get file handle
Open "test.txt" For Input As #ff    ' we can use a constant for the file too
Print Lof(#ff); " length of file"
Print Fileattr(#ff); " file mode"   ' should be 1 for input
Do
  LineInput #ff , S                 ' read a line
  ' line input is used to read a line of text from a file
  Print S                           ' print on terminal emulator
Loop Until Eof(ff)<> 0
```

```
'The EOF() function returns a non-zero number when the end of the file is reached
'This way we know that there is no more data we can read
Close #ff
```

7.8.26 LOC

Action

Returns the position of last read or written Byte of the file

Syntax

lLastReadWritten = **LOC** (#bFileNumber)

Remarks

bFileNumber	(Byte) File number, which identifies an opened file
lLastReadWritten	(Long) Variable, assigned with the Position of last read or written Byte (1-based)

This function returns the position of the last read or written Byte. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError. If the file position pointer is changed with the command SEEK, this function can not be used till the next read/ write operation.

This function differs from VB. In VB the byte position is divided by 128.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_FileLoc	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
' open the file in BINARY mode
Open "test.bin" For Binary As #2
Put #2 , B                          ' write a byte
Put #2 , W                          ' write a word
Put #2 , L                          ' write a long
Ltemp = Loc(#2)+ 1                  ' get the position of the next byte
Print Ltemp ;" LOC"                 ' store the location of the file pointer
Print Lof(#2);" length of file"
Print Fileattr(#2);" file mode"     ' should be 32 for binary
Put #2 , Sn                         ' write a single
Put #2 , Stxt                       ' write a string

Flush #2                            ' flush to disk
Close #2
```

7.8.27 LOF

Action
Returns the length of the File in Bytes

Syntax
lFileLength = **LOF** (#bFileNumber)

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
LFileLength	(Long) Variable, which assigned with the Length of the file (1-based)

This function returns the length of an opened file. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Calls	_FileLOF	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Output	r25: Errorcode	C-Flag: Set on Error

Example
```
' open the file in BINARY mode
Open "test.bin" For Binary As #2
Put #2 , B                          ' write a byte
Put #2 , W                          ' write a word
Put #2 , L                          ' write a long
Ltemp = Loc(#2)+ 1                  ' get the position of the next byte
Print Ltemp ;" LOC"                 ' store the location of the file pointer
Print Lof(#2);" length of file"
Print Fileattr(#2);" file mode"     ' should be 32 for binary
Put #2 , Sn                         ' write a single
Put #2 , Stxt                       ' write a string

Flush #2                            ' flush to disk
Close #2
```

7.8.28 MKDIR

Action
This statement creates a folder or directory in the current directory.

Syntax
MKDIR directory

Remarks

MaKeDIRectory creates a folder or directory in the current directory.
The directory may not have a device name like COM1, LPT1, etc.
The directory may also not have a name like ".." or "\" since these names are reserved.
You can not create a directory using a path.

```
MKDIR "test"  ' ok
MKDIR "test\abc" ' NOT OK
MKDIR ".." ' NOT OK
MKDIR "\" ' NOT OK

MKDIR "test" : CHDIR "test" : MKDIR "abc"  ' this would create test\abc
```

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILELEN 788 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , WRITE 801 , INPUT 1493 , DIR 777, RMDIR 798 , CHDIR 775 , NAME 797

Example

```
MKDIR "test"
```

7.8.29 NAME

Action

This AVR-DOS statement renames a file or directory name.

Syntax

NAME old **AS** new

Remarks

old	The name of the file or folder that you want to rename. This file must exist in the current folder.
new	The new name of the file. The new file may not already exist. The current folder will be used.

Both old and new must be valid file names and of the string data type. Constants are not allowed.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILELEN 788 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , WRITE 801 , INPUT 1493 , DIR 777, MKDIR 796, RMDIR 798 , CHDIR 775

Example
```
Old = "file1.txt"
New = "fileNew.txt"
NAME old AS new
```

7.8.30 RMDIR

Action
This statement removes the specified directory.

Syntax
RMDIR directory

Remarks
RMDIR (**R**e**M**ove **DIR**ectory) removes the specified directory or folder.
The folder must exist. You may not use a path.
While KILL is used to remove files, RMDIR is used to remove directories.
You should remove all files before you remove the directory.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILELEN 788 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , WRITE 801 , INPUT 1493 , DIR 777, MKDIR 796, CHDIR 775 , NAME 797

Example
```
RMDIR "abc"
```

7.8.31 SETATTR

Action
Sets the file Attribute.

Syntax
SETATTR [sFile ,] bFileAttribute

Remarks

sFile	The name of the file (no wildcard) which attribute need to be set. You may also omit the name in which case the file will be used previous found by the DIR() function.

bFileAttribute	Numeric variable holding the attribute bits to set.

This statement sets the DOS file attributes. A file can have multiple attributes.
You should not use attributes 8(Volume) and 16(Sub Directory) on a normal file.

Return value	DOS Attribute
1	Read Only
2	Hidden
4	System File
8	Volume Label
16	Sub Directory
32	Archive
64,128	reserved

A file can have multiple bits set like 3 (hidden + read only). So you can combine multiple bits to set multiple bits at once.

When you specify the filename, make sure it does not have a wildcard. SETATTR does not support wildcards.
When you omit the filename, the last found file from DIR 777() will be used for the operation.

In VB, SETATTR expect a new value for the attribute which replaces the old attribute byte.
In AVR-DOS you specify the bits to set. So old attributes are kept.
In AVR-DOS you can also reset the individual bits using the CLEARATTR statement.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402, FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788, WRITE 801 , INPUT 1493 , FILEATTR 786 , CLEARATTR 776 , GETATTR 791

Example

```
'-----------------------------------------------------------------
-------------
'                          simulate-AVR-DOS.bas
'  simulate AVR-DOS using virtual XRAM drive
'
'-----------------------------------------------------------------
-------------
$regfile = "M128def.dat"
$crystal = 16000000
' Adjust HW Stack, Soft-Stack and Frame size to 128 minimum
each!!!
$hwstack=128 : $swstack=128 : $framesize=128
$xramsize = &H10000  'specify 64KB of XRAM for the file system
$sim 'for simulation only !
```

```
$baud = 19200

Config Clock = Soft
Enable Interrupts
Config Date = Mdy , Separator = dot

Dim Btemp1 As Byte ,battr1 as Byte, battr2 as Byte
$include "Config_XRAMDrive.bas"                                    '
Does drive init too
$include "Config_AVR-DOS.BAS"

Print "Wait for Drive"
If Gbdriveerror = 0 Then
   Print "Init File System ... ";
   Btemp1 = Initfilesystem(1)                                      '
Partition 1
                                          ' use 0 for drive
without Master boot record
   If Btemp1 <> 0 Then
      Print  "Error: " ; Btemp1 ; " at Init file system"
   Else
      Print  " OK"
      Print  "Filesystem: " ; Gbfilesystem
      Print  "FAT Start Sector: " ; Glfatfirstsector
      Print  "Root Start Sector: " ; Glrootfirstsector
      Print  "Data First Sector: " ; Gldatafirstsector
      Print  "Max. Cluster Nummber: " ; Glmaxclusternumber
      Print  "Sectors per Cluster: " ; Gbsectorspercluster
      Print  "Root Entries: " ; Gwrootentries
      Print  "Sectors per FAT: " ; Glsectorsperfat
      Print  "Number of FATs: " ; Gbnumberoffats
   End If
Else
   Print  "Error during Drive Init: " ; Gbdriveerror
End If

Dim strDummy as String * 12
Dim Datei As String * 12 , Attribut As Byte
Datei = "Test1.txt"

Open Datei For Output As #11
Print #11 , "Testzeile1"
Close #11

open "Test2.txt" For output as #11
Print #11, "Testzeile2"
close #11

open "Test3.txt" for output as #11
```

```
Print #11, "Testzeile3"
close #11


' Set readonly Bit in Test1.txt
Attribut = &B00000001
Setattr Datei , Attribut

' Reset Archib-Bit in test1.txt
Attribut = &B00100000
clearattr Datei , Attribut

' Check for Filename with wildcard, which is not supported
' Set readonly Bit in Test1.txt
Datei = "Test*.txt"
Attribut = &B00000001
Setattr Datei , Attribut
Print gbDOSError

Datei = DIR("Test*.txt")
Attribut = &B00000001
while Datei > ""
   SetAttr Attribut
   Datei = DIR()
wend

Datei = DIR("Test*.txt")
Attribut = &B00100000
While Datei > ""
   battr1=Getattr()
   clearattr Attribut
   battr2=Getattr()
   print datei ;" "; battr1;" " ; battr2
   Datei = DIR()
wend

End
```

7.8.32 WRITE

Action

Writes data to a sequential file

Syntax

WRITE #ch , data [,data1]

Remarks

Ch	A channel number, which

	identifies an opened file. This can be a hard coded constant or a variable.
Data , data1	A variable who's content are written to the file.

When you write a variables value, you do not write the binary representation but the ASCII representation. When you look in a file it contains readable text.

When you use PUT, to write binary info, the files are not readable or contain unreadable characters.
Strings written are surrounded by string delimiters "". Multiple variables written are separated by a comma. Consider this example :

```
Dim S as String * 10 , W as Word
S="hello" : W = 100
OPEN "test.txt" For OUTPUT as #1
WRITE #1, S , W
CLOSE #1
```

The file content will look like this : "hello",100
Use INPUT to read the values from value.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , WRITE 801 , INPUT 1493

ASM

Calls	_FileWriteQuotationMark	_FileWriteDecInt
	_FileWriteDecByte	_FileWriteDecWord
	_FileWriteDecLong	_FileWriteDecSingle
Input	Z points to variable	
Output		

Partial Example

```
Dim S As String * 10 , W As Word , L As Long

S = "write"
Open "write.dmo"for Output As #2
Write #2 , S , W , L                                    ' write is
also supported
Close #2

Open "write.dmo"for Input As #2
Input #2 , S , W , L                                    ' write is
also supported
Close #2
Print S ; " " ; W ; " " ; L
```

7.9 BITWAIT

Action
Wait until a bit is set or reset.

Syntax
BITWAIT x , SET/RESET

Remarks

X	Bit variable or internal register like PORTB.x , where x ranges from 0-7.

When using bit variables make sure that they are set/reset by software otherwise your program will stay in a loop.

When you use internal registers that can be set/reset by hardware such as PINB.0 this doesn't apply since this state can change as a result from for example a key press.

See also
NONE

ASM
Calls: NONE
Input: NONE
Output: NONE

Code : shown for address 0-31

```
label1:
Sbic PINB.0,label2
Rjmp label1
Label2:
```

Example
```
$regfile = "m48def.dat"                                    ' specify the used micr
$crystal = 8000000                                         ' used crystal frequenc
$baud = 19200                                              ' use baud rate
$hwstack = 32                                              ' default use 32 for th
$swstack = 10                                              ' default use 10 for th
$framesize = 40                                            ' default use 40 for th

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,

Dim A As Bit
Bitwait A , Set                                            'wait until bit a is se
'the code above will loop forever since the bit 'A' is not set in software
'it could be set in an ISR routine

Bitwait Pinb.7 , Reset                                     'wait until bit 7 of Po
End
```

7.10 BITS

Action
Set all specified bits to 1.

Syntax
Var = **Bits**(b1 [,bn])

Remarks

Var	The BYTE/PORT variable that is assigned with the constant.
B1 , bn	A list of bit numbers that must be set to 1.

While it is simple to assign a value to a byte, and there is special Boolean notation &B for assigning bits, the Bits() function makes it simple to assign a few bits.

B = &B1000001 : how many zero's are there?

This would make it more readable : B = Bits(0, 6)

You can read from the code that bit 0 and bit 6 are set to 1.
It does not save code space as the effect is the same.
It can only be used on bytes and port registers.

Valid bits are in range from 0 to 7.

See Also
NBITS 1377

Example
```
'-----------------------------------------------------------------------
---------
'name                     : bits-nbits.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo for Bits() AND Nbits()
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'use in simulator         : possible
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim B As Byte
```

```
'while you can use &B notation for setting bits, like B = &B1000_0111
'there is also an alternative by specifying the bits to set
B = Bits(0 , 1 , 2 , 7)                                    'set only
bit 0,1,2 and 7
Print B

'and while bits() will set all bits specified to 1, there is also Nbits
()
'the N is for NOT. Nbits(1,2) means, set all bits except 1 and 2
B = Nbits(7)                                               'do not set
bit 7
Print B
End
```

7.11 BREAK

Action
This statement will break the simulator/debugger.

Syntax
BREAK

Remarks
A number of new processors support the BREAK instruction. The break instruction will break execution of the simulator. This is support by the BASCOM simulator but also by Atmels Studio.
Processors that do not support the BREAK instruction interpret the BREAK instruction as a NOP (no operation). So it is safe to use on all processors.
The BREAK instruction uses 1 cycle just like the ASM NOP instruction.

See also
NOP 1378

Example
NONE

7.12 BYVAL

Action
Specifies that a variable will be passed by value.

Syntax
Sub Test(**BYVAL** var)

Remarks

Var	Variable name

The default for passing variables to SUBS and FUNCTIONS, is by reference(BYREF).

When you pass a variable by reference, the address is passed to the SUB or FUNCTION. When you pass a variable by Value, a temp variable is created on the frame and the address of the copy is passed.

When you pass by reference, changes to the variable will be made to the calling variable.
When you pass by value, changes to the variable will be made to the copy so the original value will not be changed.

By default passing by reference is used.
Note that calling by reference will generate less code.

See also
CALL 806 , DECLARE 1221 , SUB 1545 , FUNCTION 1215

ASM
NONE

Example
```
Declare Sub Test(Byval X As Byte, Byref Y As Byte, Z As Byte)
```

7.13 CALL

Action
Call and execute a subroutine.

Syntax
CALL Test [(var1, var-n)]

Remarks

Var1	Any BASCOM variable or constant.
Var-n	Any BASCOM variable or constant.
Test	Name of the subroutine. In this case Test.

You can call sub routines with or without passing parameters.

It is important that the SUB routine is DECLARED before you make the CALL to the subroutine. Of course the number of declared parameters must match the number of passed parameters.

It is also important that when you pass constants to a SUB routine, you must DECLARE these parameters with the BYVAL argument. This because a constant has no address, it is assigned at compile time.

With the CALL statement, you can call a procedure or subroutine.

For example: Call Test2

The call statement enables you to implement your own statements.
You don't have to use the CALL statement:

Test2 will also call subroutine test2

When you don't supply the CALL statement, you must leave out the parenthesis. So Call Routine(x,y,z) must be written as Routine x,y,z

Unlike normal SUB programs called with the GOSUB statement, the CALL statement enables you to pass variables to a SUB routine that may be local to the SUB.

 By using CONFIG SUBMODE=NEW, you do not need to DECLARE each sub/ function.

See also

DECLARE 1221 , SUB 1545 , EXIT 1257 , FUNCTION 1215 , LOCAL 1360 , CONFIG SUBMODE 1079

Example

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim A As Byte , B As Byte                                  'dimension
some variables
Declare Sub Test(b1 As Byte , Byval B2 As Byte)            'declare the
SUB program
A = 65                                                     'assign a
value to variable A
Call Test(a , 5)'call test with parameter A and constant
Test A , 5                                                 'alternative
call
Print A                                                    'now print
the new value
End

Sub Test(b1 As Byte , Byval B2 As Byte)                    'use the
same variable names as 'the declared one
  Print B1                                                 'print it
  Print Bcd(b2)
  B1 = 10                                                  'reassign
the variable
  B2 = 15                                                  'reassign
the variable
End Sub
```

 One important thing to notice is that you can change b2 but that the change will

not be reflected to the calling program!
Variable A is changed however.

This is the difference between the BYVAL and BYREF argument in the DECLARE ration of the SUB program.

When you use BYVAL, this means that you will pass the argument by its value. A copy of the variable is made and passed to the SUB program. So the SUB program can use the value and modify it, but the change will not be reflected to the calling parameter. It would be impossible too when you pass a numeric constant for example.

If you do not specify BYVAL, BYREF will be used by default and you will pass the address of the variable. So when you reassign B1 in the above example, you are actually changing parameter A.

7.14 CHECKSUM

7.14.1 CHECKSUM CHECKSUMXOR

Action

Returns a checksum of a string.

Syntax

PRINT **Checksum**(var)
b = **Checksum**(var)
b = **ChecksumXOR**(var)

Remarks

Var	A string variable.
B	A numeric variable that is assigned with the checksum.

The checksum is computed by counting all the bytes of the string variable.
The checksumXOR is computed by Xor-ing all the bytes of the string variable.
Checksums are often used with serial communication.
The checksum is a byte checksum. The following VB code is equivalent :

```
Dim Check as Byte
Check = 0
For x = 1 To Len(s$)
  Check = check + ASC(mid$(s$,x,1))
Next

The following VB code is equivalent for ChecksumXOR
Dim Check as Byte
Check = 0
For x = 1 To Len(s$)
  Check = check XOR ASC(mid$(s$,x,1))
Next
```

See also

CRC8 809 , CRC16 813 , CRC32 817 , CRC16UNI 815, CRCMB 818

Example

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As String * 10                                       'dim
variable
S = "test"                                                 'assign
variable
Print Checksum(s)                                          'print value
(192)
End
```

7.14.2 CRC8

Action

Returns the CRC8 value of a variable or array.

Syntax

Var = **CRC8**(source , L)

Remarks

Var	The variable that is assigned with the CRC8 of variable source.
Source	The source variable or first element of the array to get the CRC8 of.
L	The number of bytes to check.

CRC8 is used in communication protocols to check if there are no transmission errors. The 1wire for example returns a CRC byte as the last byte from it's ID.

The code below shows a VB function of CRC8

```
Function Docrc8(s As String) As Byte
Dim j As Byte
Dim k As Byte
Dim crc8 As Byte
crc8 = 0
For m = 1 To Len(s)
  x = Asc(Mid(s, m, 1))
  For k = 0 To 7
   j = 1 And (x Xor crc8)
   crc8 = Fix(crc8 / 2) And &HFF
   x = Fix(x / 2) And &HFF
   If j <> 0 Then
   crc8 = crc8 Xor &H8C
   End If
```

```
  Next k
Next
Docrc8 = crc8
End Function
```

 When you want to use a different polynome, you can override the default by defining a constant named CRC8_POLY
Const CRC8_POLY = &HAA 'use a different value

 Please notice that the CRC8 function is the CRC8-MAXIM function. It is primarily intended for the 1WIRE routines. There exist a lot of different CRC8 variants. They differ in the start value, the polynom , if the result is XOR-ed and if the data is reflected or not. Reflection means that data is flipped. (See FLIP 1259)

CRC8 supports big strings in 2083.

See also

CHECKSUM 808 , CRC16 813, CRC16UNI 815 , CRC32 817 , TCPCHECKSUM 1574 , CRCMB 818

ASM

The following routine is called from mcs.lib : _CRC8
The routine must be called with Z pointing to the data and R24 must contain the number of bytes to check.
On return, R16 contains the CRC8 value.
The used registers are : R16-R19, R25.

```
;##### X = Crc8(ar(1) , 7)
Ldi R24,$07       ; number of bytes
Ldi R30,$64       ; address of ar(1)
Ldi R31,$00       ; load constant in register
Rcall _Crc8       ; call routine
Ldi R26,$60       ; address of X
St X,R16          ; store crc8
```

Example

```
$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim Ar(10) As Byte
Dim J As Byte
```

```
Ar(1) = 1
Ar(2) = 2
Ar(3) = 3

J = Crc8(ar(1) , 3)                                    'calculate
value which is 216
Print J
End
```

7.14.3 CRC8UNI

Action

Returns the CRC value of a variable or array.

Syntax

Var = **CRC8UNI**(source , L)

Remarks

Var	The variable that is assigned with the CRC8 of variable source.
Source	The source variable or first element of the array to get the CRC8 of.
L	The number of bytes to check.

CRC is used in communication protocols to check if there are no transmission errors. The CRC8 809 function in BASCOM is mainly intended to be used with 1WIRE.

The CRC8UNI uses the CCITT with polynome value 7.

 When you want to use a different polynome, you can override the default by defining a constant named CRC8_POLY
Const CRC8_POLY = &HAA 'use a different value

See also

CHECKSUM 808 , CRC16 813, CRC16UNI 815 , CRC32 817 , TCPCHECKSUM 1574 , CRCMB 818 , CRC8 809

Example

```
'-----------------------------------------------------------------------
--------
'name                    : crc8-16-32.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates CRC
'micro                   : Mega48
'suited for demo         : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
--------

$regfile = "m48def.dat"                                 ' specify
the used micro
```

```
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


Dim Ar(10) As Byte
Dim J As Byte
Dim W As Word
Dim L As Long
Dim S As String * 16


S = "123456789"

Ar(1) = 1
Ar(2) = 2
Ar(3) = 3


J = Crc8(ar(1) , 3)                                         'calculate
value which is 216
j = Crc8Uni(ar(1) , 3)                                      'calculate
unsing CCITT which is 72
W = Crc16(ar(1) , 3)                                        '24881
L = Crc32(ar(1) , 3)                                        '1438416925

'                   data , length, intial value , Poly, reflect input,
reflect output

Print Hex(Crc16Uni(S , 9 , 0 , &H1021 , 0 , 0))             'CRC-CCITT
(0x0000)   31C3
Print Hex(Crc16Uni(S , 9 , &HFFFF , &H1021 , 0 , 0))        'CRC-CCITT
(0xFFFF)   29B1
Print Hex(Crc16Uni(S , 9 , &H1D0F , &H1021 , 0 , 0))        'CRC-CCITT
(0x1D0F)   E5CC
Print Hex(Crc16Uni(S , 9 , 0 , &H8005 , 1 , 1))             'crc16
          BB3D
Print Hex(Crc16Uni(S , 9 , &HFFFF , &H8005 , 1 , 1))
'crc16-modbus         4B37

End
```

7.14.4 CRC16

Action
Returns the CRC16 value of a variable or array.

Syntax
Var = **CRC16**(source , L)

Remarks

Var	The variable that is assigned with the CRC16 of variable source. Should be a word or integer variable.
Source	The source variable or first element of the array to get the CRC16 value from. By default only normal RAM variables are supported. You can also use EEPROM memory when you add a constant to your project : **Const CRC16_EEPROM=1**
L	The number of bytes to check. This can be a numeric constant , byte or word variable. The maximum size to check is 65535.

CRC16 is used in communication protocols to check if there are no transmission errors.
The 1wire for example returns a CRC byte as the last byte from it's ID.
Use CRC8 for the 1wire routines.

There are a lot of different CRC16 routines. There is no real standard since the polynomial will vary from manufacture to manufacture.

The equivalent code in VB is shown below. There are multiple ways to implement it in VB. This is one of them.

VB CRC16 Sample
```
Private Sub Command1_Click()

Dim ar(10) As Byte
Dim b As Byte
Dim J As Integer

ar(1) = 1
ar(2) = 2
ar(3) = 3

b = Docrc8(ar(), 3)  ' call funciton
Print b
'calculate value which is 216

J = CRC16(ar(), 3)  ' call function
Print J

End Sub


Function Docrc8(ar() As Byte, bts As Byte) As Byte
Dim J As Byte
Dim k As Byte
```

```
Dim crc8 As Byte
crc8 = 0
For m = 1 To bts

  x = ar(m)
  For k = 0 To 7
   J = 1 And (x Xor crc8)
   crc8 = Fix(crc8 / 2) And &HFF
   x = Fix(x / 2) And &HFF
   If J <> 0 Then
     crc8 = crc8 Xor &H8C
   End If
  Next k
Next
Docrc8 = crc8
End Function

'*****************************************************************

Public Function CRC16(buf() As Byte, lbuf As Integer) As Integer
Dim CRC1 As Long
Dim b As Boolean
CRC1 = 0 ' init CRC
For i = 1 To lbuf ' for each byte
  CRC_MSB = CRC1 \ 256
  crc_LSB = CRC1 And 255
  CRC_MSB = CRC_MSB Xor buf(i)
  CRC1 = (CRC_MSB * 256) + crc_LSB

  For J = 0 To 7 Step 1 ' for each bit
    CRC1 = shl(CRC1, b)
    If b Then CRC1 = CRC1 Xor &H1021
Next J
Next i

CRC16 = CRC1
End Function

'Shift Left function
Function shl(n As Long, ByRef b As Boolean) As Long
  Dim L As Long
  L = n
  L = L * 2
  If (L > &HFFFF&) Then
    b = True
  Else
    b = False
  End If
  shl = L And &HFFFF&
End Function
```

See also

CHECKSUM 808 , CRC8 809, CRC16UNI 815 , CRC32 817 , TCPCHECKSUM 1574 , CRCMB 818 , CRC8UNI 811

ASM

The following routine is called from mcs.lib : _CRC16
The routine must be called with X pointing to the data. The soft stack –Y must contain the number of bytes to scan.
On return, R16 and R17 contain the CRC16 value.

The used registers are : R16-R19, R25.

```
;##### X = Crc16(ar(1) , 7)

Ldi R24,$07       ; number of bytes
St -y, R24
Ldi R26,$64       ; address of ar(1)
Ldi R27,$00       ; load constant in register
Rcall _Crc16      ; call routine
Ldi R26,$60       ; address of X
St X+,R16         ; store crc16 LSB
St X , R17        ; store CRC16 MSB
```

Example

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim Ar(10) As Byte
Dim J As Byte
Dim W As Word
Dim L As Long

Ar(1) = 1
Ar(2) = 2
Ar(3) = 3

J = Crc8(ar(1) , 3)                                        'calculate
value which is 216
W = Crc16(ar(1) , 3)                                       '24881
L = Crc32(ar(1) , 3)                                       '494976085
End
```

7.14.5 CRC16UNI

Action

Returns the CRC16 value of a variable or array.

Syntax

Var = **CRC16UNI**(source ,length , initial, polynomial,refin,refout)

Remarks

var	The variable that is assigned with the CRC16 of variable source. Should

	be a word or integer variable.
source	The source variable or first element of the array to get the CRC16 value from.
length	The number of bytes to check. The maximum value is 65535. (**&H**FFFF)
initial	The initial value of the CRC. This is usual 0 or **&H**FFFF.
polynomial	The polynomial value to use.
refin	Reflect the data input bits. Use 0 to disable this option. Use a non-zero value to enable this option.
refout	Reflect the data output. Use 0 to disable this option. Use a non-zero value to enable this option.

CRC16 is used in communication protocols to check if there are no transmission errors.
The 1wire for example returns a CRC byte as the last byte from it's ID.
Use CRC8 for the 1wire routines.

There are a lot of different CRC16 routines. There is no real standard since the polynomial will vary from manufacture to manufacture.

At http://www.ross.net/crc/download/crc_v3.txt you can find a great document about CRC calculation from Ross N. Williams. At the end you will find an example that is good for dealing with most CRC variations. The BASCOM CRC16UNI function is a conversion of this example.
There is a difference however : The CRC16UNI function does not XOR the output bytes. This because most CRC functions XOR with 0.

The example will show some of the most used combinations.

In version 2083 the function can handle more than 255 bytes. In previous versions the amount was limited to a maximum of 255.

See also

CHECKSUM 808 , CRC8 809, CRC16 813 , CRC32 817 , TCPCHECKSUM 1574 , CRCMB 818 , CRC8UNI 811

Example

```
'-----------------------------------------------------------------------------
'name                     : crc8-16-32.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates CRC
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------

$regfile = "m48def.dat"                                     ' specify the used micr
$crystal = 8000000                                          ' used crystal frequenc
$baud = 19200                                               ' use baud rate
$hwstack = 32                                               ' default use 32 for th
$swstack = 10                                               ' default use 10 for th
$framesize = 40                                             ' default use 40 for th


Dim Ar(10) As Byte
```

```
Dim J As Byte
Dim W As Word
Dim L As Long
Dim S As String * 16


S = "123456789"

Ar(1) = 1
Ar(2) = 2
Ar(3) = 3


J = Crc8(ar(1) , 3)                                          'calculate value which
W = Crc16(ar(1) , 3)                                         '24881
L = Crc32(ar(1) , 3)                                         '494976085

'                  data , length, intial value , Poly, reflect input, reflect outpu

Print Hex(crc16uni(s , 9 , 0 , &H1021 , 0 , 0))              'CRC-CCITT (0x0000)   3
Print Hex(crc16uni(s , 9 , &HFFFF , &H1021 , 0 , 0))         'CRC-CCITT (0xFFFF)   2
Print Hex(crc16uni(s , 9 , &H1D0F , &H1021 , 0 , 0))         'CRC-CCITT (0x1D0F)   E
Print Hex(crc16uni(s , 9 , 0 , &H8005 , 1 , 1))              'crc16                E
Print Hex(crc16uni(s , 9 , &HFFFF , &H8005 , 1 , 1))         'crc16-modbus         4


End
```

7.14.6 CRC32

Action

Returns the CRC32 value of a variable.

Syntax

Var = **CRC32**(source , L)

Remarks

Var	The LONG variable that is assigned with the CRC32 of variable source.
Source	The source variable or first element of the array to get the CRC 32 value from.
L	The number of bytes to check. This can be a word variable.

CRC32 is used in communication protocols to check if there are no transmission errors.

See also

CHECKSUM 808 , CRC8 809, CRC16 813 , CRC16UNI 815 , TCPCHECKSUM 1574 , CRCMB 818 , CRC8UNI 811

Example

```
$regfile = "m48def.dat"                                      ' specify
the used micro
$crystal = 8000000                                           ' used
crystal frequency
$baud = 19200                                                ' use baud
```

```
rate
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim Ar(10) As Byte
Dim J As Byte
Dim W As Word
Dim L As Long

Ar(1) = 1
Ar(2) = 2
Ar(3) = 3

J = Crc8(ar(1) , 3)                                          'calculate
value which is 216
W = Crc16(ar(1) , 3)                                         '24881
L = Crc32(ar(1) , 3)                                         '1438416925
End
```

7.14.7 CRCMB

Action

Returns the Modbus CRC value of a variable or array.

Syntax

Var = **CRCMB**(source , L)

Remarks

Var	The variable that is assigned with the modbus checksum of variable source. This should be a word variable.
Source	The source variable or first element of the array to get the checksum of.
L	The number of bytes to check.

CRC8 is used in communication protocols to check if there are no transmission errors. The Modbus checksum uses a different polynome.

Modbus.lbx or modbus.lib need to be included in your project using the $LIB directive

See also

CHECKSUM 808 , CRC16 813, CRC16UNI 815 , CRC32 817 , TCPCHECKSUM 1574 , CRC8 809 , CRC8UNI 811

Example

```
$regfile = "m48def.dat"                                      ' specify
```

```
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim Ar(10) As Byte
Dim W As Word

Ar(1) = 1
Ar(2) = 2
Ar(3) = 3

W = CrcMB(ar(1) , 3)                                        'calculate
value
Print W
End
```

7.15 Conversion

7.15.1 ASC

Action

Assigns a numeric variable with the ASCII value of the first character of a string.

Syntax

var = **ASC**(string [,index])

Remarks

Var	Target numeric variable that is assigned.
String	String variable or constant from which to retrieve the ASCII value.
Index	An optional index value. The index has a range from 1 to the length of the string. When no index is provided, the default value 1 will be used.

Note that only the first character of the string will be used.
When the string is empty, a zero will be returned.

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues.

Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only'

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Extended ASCII

As people gradually required computers to understand additional characters and non-printing characters the ASCII set became restrictive. As with most technology, it took a while to get a single standard for these extra characters and hence there are few varying 'extended' sets. The most popular is presented below.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	∑
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

See also

CHR 827

ASM

NONE

Example

```
'-----------------------------------------------------------------------
'name                : asc.bas
'copyright           : (c) 1995-2025, MCS Electronics
'purpose             : demonstrates ASC function
'micro               : Mega88
'suited for demo     : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------

$RegFile = "m88def.dat"                    ' specify the used micro
$crystal = 8000000                         ' used crystal frequency
```

```
$baud = 19200                                ' use baud rate
$hwstack = 32                               ' default use 32 for the hardware stack
$swstack = 10                               ' default use 10 for the SW stack
$framesize = 40                             ' default use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 , Clockpol = 0


Dim A As Byte , S As String * 10 , idx as Byte
Print "ASC demo"
S = "ABC"
A = Asc(s)
Print A                                     'will print 65

print "test with index"
a= asc(s,0) : print a                       'invalid range will return 0
a= asc(s,2) : print a
a= asc(s,100) : print a                     'invalid range will return 0


End
```

7.15.2 BCD

Action

Converts a variable stored in BCD format into a string.

Syntax

PRINT **BCD**(var)
LCD **BCD**(var)

Remarks

Var	Numeric variable to convert.

When you want to use an I2C clock device which stores its values in BCD format you can use this function to print the value correctly.
BCD() displays values with a leading zero.

The BCD() function is intended for the PRINT/LCD statements.
Use the MAKEBCD function to convert variables from decimal to BCD.
Use the MAKEDEC function to convert variables from BCD to decimal.

See also

MAKEDEC 1369 , MAKEBCD 1369

ASM

Calls: _BcdStr
Input: X hold address of variable
Output: R0 with number of bytes, frame with data.

Example

```
'-----------------------------------------------------------------------
---------
'name                         : bcd.bas
'copyright                    : (c) 1995-2025, MCS Electronics
'purpose                      : demonstration of split and combine BCD Bytes
```

```
'suited for demo          : yes
'commercial addon needed  : no
'use in simulator         : possible
'-----------------------------------------------------------------------
---------
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'=======================================================================
========
' Set up Variables
'=======================================================================
========
Dim A As Byte                                          'Setup A Variable
Dim B As Byte                                          'Setup B Variable
Dim C As Byte                                          'Setup C Variable

A = &H89
'=======================================================================
========
' Main
'=======================================================================
========
Main:
Print "Combined :   " ; Hex(a)                        'Print A

'-----------------------------------------------------------------------
--------
B = A And &B1111_0000                                  'Mask To Get Only
High Nibble Of Byte
Shift B , Right , 4                                    'Shift High
Nibble To Low Nibble Position , Store As B

C = A And &B0000_1111                                  'Mask To Get Only
Low Nibble Of Byte , Store As C

Print "Split :      " ; B ; " " ; C                   'Print B (High
Nibble) , C(low Nibble)

'-----------------------------------------------------------------------
--------
Shift B , Left , 4                                     'Shift Data From
Low Nibble Into High Nibble Position

A = B + C                                              'Add B (High
Nibble) And C(low Nibble) Together

Print "Re-Combined: " ; Hex(a); " " ; Bcd(a)          'Print A (re -
combined Byte)
End                                                    'End Program
```

7.15.3 BIN

Action
Convert a numeric variable into the binary string representation.

Syntax
Var = **Bin**(source)

Remarks

Var	The target string that will be assigned with the binary representation of the variable source.
Source	The numeric variable that will be converted.

The BIN() function can be used to display the state of a port.
When the variable source has the value &B10100011 the string named var will be assigned with "10100011".
It can be easily printed to the serial port.

See also
HEX 833 , STR 836 , VAL 839 , HEXVAL 832 , BINVAL 825

ASM
NONE

Example
```
$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim B As Byte
' assign value to B
B = 45

Dim S As String * 10
'convert to string
S = Bin(b)

'assign value to portb
Portb = 33
```

```
Print Bin(portb)

'of course it also works for other numerics
End
```

7.15.4 BINVAL

Action
Converts a string representation of a binary number into a number.

Syntax
var = **Binval**(s)

Remarks

Var	A numeric variable that is assigned with the value of s.
S	Variable of the string type. Should contain only 0 and 1 digits.

See also
STR 836 , HEXVAL 832 , HEX 833 , BIN 824 , VAL 839

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As String * 8
S = "11001100"

Dim B As Byte
' assign value to B
B = Binval(s)

Print B
End
```

7.15.5 BIN2GRAY

Action
Returns the Gray-code of a variable.

Syntax
var1 = **Bin2gray**(var2)

Remarks

var1	Variable that will be assigned with the Gray code.
var2	A variable that will be converted.

Gray code is used for rotary encoders. Bin2gray() works with byte , integer, word and long variables.
The data type of the variable that will be assigned determines if a byte, word or long conversion will be done.

See also
GRAY2BIN 831 , ENCODER 1254

ASM
Depending on the data type of the target variable the following routine will be called from mcs.lbx:
_grey2Bin for bytes , _grey2bin2 for integer/word and _grey2bin4 for longs.

Example
```
'-----------------------------------------------------------------------
-----------------
'name                     : graycode.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : show the Bin2Gray and Gray2Bin functions
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'Bin2Gray() converts a byte,integer,word or long into grey code.
'Gray2Bin() converts a gray code into a binary value

Dim B As Byte                                               ' could be
word,integer or long too

Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
  Print B ; Spc(10) ; Bin2gray(b)
```

```
Next

Print "GREY" ; Spc(8) ; "BIN"
For B = 0 To 15
  Print B ; Spc(10) ; Gray2bin(b)
Next

End
```

7.15.6 CHR

Action

Convert a numeric variable or a constant to a string with a length of 1 character. The character represents the ASCII value of the numeric value.

Syntax

PRINT **CHR**(var)
s = **CHR**(var)

Remarks

Var	Numeric variable or numeric constant.
S	A string variable.

When you want to print a character to the screen or the LCD display,
you must convert it with the CHR() function.

When you use PRINT numvar, the value will be printed.
When you use PRINT Chr(numvar), the ASCII character itself will be printed.
The Chr() function is handy in combination with the LCD custom characters where you can redefine characters 0-7 of the ASCII table.
Since strings are terminated with a null byte which is the same as Chr(0) , you can not embed a Chr(0) into a string.
When using Chr(0) with the LCD to display a customer character, use : LCD "tekst" ; chr(0) ; "more tekst"

See also

ASC 819

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : chr.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows how to use the CHR() and BCD()
function and
'                           HEX() function in combination with a PRINT
statement
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
```

```
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Dim K As Byte

K = 65
Print K ; Chr(k) ; K ; Chr(66) ; Bcd(k) ; Hex(k)
End
```

7.15.7 FORMAT

Action

Formats a numeric string.

Syntax

target = **FORMAT**(source, "mask")

Remarks

target	The string that is assigned with the formatted string.
source	The source string that holds the number.
mask	The mask for formatting the string. When spaces are in the mask, leading spaces will be added when the length of the mask is longer than the source string. " " '8 spaces when source is "123" it will be " 123". When a + is in the mask (after the spaces) a leading + will be assigned when the number does not start with the - (minus) sign. "+" with number "123" will be "+123". When zero's are provided in the mask, the string will be filled with leading zero's. " +00000" with 123 will become " +00123" An optional decimal point can be inserted too: "000.00" will format the number 123 to "001.23" Combinations can be made but the order must be : spaces, + , 0 an optional point and zero's. In version 2080 the mask may be a variable as well.

When you do not want to use the overhead of the single or double, you can use the LONG. You can scale the value by a factor for example 100.
Then use FORMAT to show the value.
For example :

```
Dim L as Long, X as Long , Res as Long
L = 1
```

```
X = 2
Res = L / X
' Now this would result in 0 because an integer or Long does not support floating p
' But when you scale L with a factor 100, you get :
L = 100
X = 2
Res = L / X   '50
```

Now Res will be 50. To show it the proper way we can use FORMAT. Format works with strings so the variables need to be converted to string first.

```
Dim S1 as string * 16 : s1 = Str(Res)
Print Format(s1,"000.00")
```

See also

FUSING 830 , STR 836

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : format.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : FORMAT
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim S As String * 10
Dim I As Integer

S = "12345"
S = Format(s , "+")
Print S                                                    ' +12345

S = "123"
S = Format(s , "00000")
Print S                                                    ' 00123

S = "12345"
S = Format(s , "000.00")
Print S                                                    ' 123.45

S = "12345"
S = Format(s , "+000.00")
Print S                                                    ' +123.45
End
```

7.15.8 FUSING

Action
FUSING returns a formatted string of a single value.

Syntax
target = **FUSING**(source, "mask")

Remarks

target	The string that is assigned with the formatted string.
source	The source variable of the type SINGLE that will be converted
mask	The mask for formatting the string. The mask is a string constant that always must start with #. After the decimal point you can provide the number of digits you want the string to have: #.### will give a result like 123.456. Rounding is used when you use the # sign. So 123.4567 will be converted into 123.457 When no rounding must be performed, you can use the & sign instead of the # sign. But only after the DP. #.&&& will result in 123.456 when the single has the value 123.4567

When the single is zero, 0.0 will be returned, no matter how the mask is set up.

See also
FORMAT 828 , STR 836 , CONFIG SINGLE 1059

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : fusing.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : FUSING
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim S As Single , Z As String * 10
```

```
'now assign a value to the single
S = 123.45678
'when using str() you can convert a numeric value into a string
Z = Str(s)
Print Z                                                    'prints
123.456779477

Z = Fusing(s , "#.##")

'now use some formatting with 2 digits behind the decimal point with
rounding
Print Fusing(s , "#.##")                                   'prints
123.46

'now use some formatting with 2 digits behind the decimal point without
rounding
Print Fusing(s , "#.&&")                                   'prints
123.45

'The mask must start with #.
'It must have at least one # or & after the point.
'You may not mix & and # after the point.
End
```

7.15.9 GRAY2BIN

Action

Returns the numeric value of a Gray code.

Syntax

var1 = **GRAY2BIN**(var2)

Remarks

var1	Variable that will be assigned with the binary value of the Grey code.
var2	A variable in Grey format that will be converted.

Gray code is used for rotary encoders. Gray2bin() works for byte, integer, word and long variables.

See also

BIN2GRAY 825

ASM

Depending on the data type of the target variable the following routine will be called from mcs.lbx:
_Bin2grey for bytes , _Bin2Grey2 for integer/word and _Bin2grey4 for longs.

Example

```
'-------------------------------------------------------------------
-----------------
'name                    : graycode.bas
'copyright               : (c) 1995-2025, MCS Electronics
```

```
'purpose                  : show the Bin2Gray and Gray2Bin functions
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                          ' specify
the used micro
$crystal = 4000000                                               ' used
crystal frequency
$baud = 19200                                                    ' use baud
rate
$hwstack = 32                                                    ' default
use 32 for the hardware stack
$swstack = 10                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space

'Bin2Gray() converts a byte,integer,word or long into grey code.
'Gray2Bin() converts a gray code into a binary value

Dim B As Byte                                                    ' could be
word,integer or long too

Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
  Print B ; Spc(10) ; Bin2gray(b)
Next

Print "GREY" ; Spc(8) ; "BIN"
For B = 0 To 15
  Print B ; Spc(10) ; Gray2bin(b)
Next
End
```

7.15.10 HEXVAL

Action

Convert string representing a hexadecimal number into a numeric variable.

Syntax

var = **HEXVAL**(x)

Remarks

Var	The numeric variable that must be assigned.
X	The hexadecimal string that must be converted.

In VB you can use the VAL() function to convert hexadecimal strings.
But since that would require an extra test for the leading &H signs that are required in VB, a separate function was designed.

The data may only contain hex decimal characters : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F, a,b,c,d,e,f. Other data will lead to conversion errors. If you need spaces to be filtered you can use the alternative library named hexval.lbx
Include it to your code with $LIB "hexval.lbx" and the conversion routine from this library will be used instead of the one from mcs.lbx. The alternative library will also

set the ERR flag if an illegal character is found.

See also
HEX 833 , VAL 839 , STR 836 , BIN 824 , BINVAL 825

Example
```
$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Dim L As Long

Dim S As String * 8
Do
  Input "Hex value " , S
  L = Hexval(s)
  Print L ; Spc(3) ; Hex(l)
Loop
```

7.15.11 HEX

Action
Returns a string representation of a hexadecimal number.

Syntax
var = **HEX**(x)

Remarks

var	A string variable.
X	A numeric variable of data type Byte, Integer, Word, Long, Single or Double.

See also
HEXVAL 832 , VAL 839 , STR 836 , BIN 824 , BINVAL 825

Example
```
$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
```

```
crystal frequency
$baud = 19200                                                ' use baud
rate
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim B As Byte , J As Integer , W As Word , L As Long
B = 1 : J = &HF001
W = &HF001
L = W

Print B ; Spc(3) ; Hex(b)
Print J ; Spc(3) ; Hex(j)
Print W ; Spc(3) ; Hex(w)
Print L ; Spc(3) ; Hex(l)
End
```

7.15.12 MANCHESTERDEC

Action

This functions decodes a Manchester encoded word into a byte.

Syntax

target = ManChesterDec(source)

Remarks

target	The byte variables that is assigned with the decoded Manchester value.
source	A Word variable containing the Manchester encoded value.

Manchester encoding (also known as phase encoding) is a line code in which the encoding of each data bit is either low then high, or high then low, for equal time. It is a self-clocking signal with no DC component. Because each input bit is represented as 01 or 10, the resulting data is twice the size of the input data.
Manchester encoding is used with RF and IR data transmission.

When there is an error in the decoding, register R25 will be set to 255.

See also

MANCHESTERENC 835

Example

```
'-----------------------PROJECT-----------------------------------
-------------
'name                 ManchesterCoding.BAS
```

```
'copyright           © 2018, MCS
'purpose             DEMO MANCHESTER ENCODING and DECODING
'micro               M1280
'-----------------------------------------------------------------
$regfile = "m1280def.dat"                                        '
specify the uC used
$crystal = 32000000                                              '
Oscillator frequency
$hwstack = 40                                                    '
hardware stack
$swstack = 40                                                    '
software stack
$framesize = 40                                                  '
frame space

Dim B As Byte , J As Byte , W As Word
For J = 0 To 255
   W = Manchesterenc(j)                                          '
encode into manchester code whith results into a WORD
   B = Manchesterdec(w)                                          '
decode it back
   If R25 <> 0 Then                                              '
when an error occurs, register r25 is 255
      Print "ERROR"
   End If
   Print J ; " " ; Hex(w) ; " " ; B
Next
```

7.15.13 MANCHESTERENC

Action

This functions encodes a byte into a Manchester encoded word.

Syntax

target = ManChesterEnc(source)

Remarks

target	A variable with a minimum data length of 2 such as a word.
source	A byte containing the data to convert.

Manchester encoding (also known as phase encoding) is a line code in which the encoding of each data bit is either low then high, or high then low, for equal time. It is a self-clocking signal with no DC component. Because each input bit is represented as 01 or 10, the resulting data is twice the size of the input data.
Manchester encoding is used with RF and IR data transmission.

See also

MANCHESTERDEC 834

Example

```
'-----------------------PROJECT-----------------------------------
-------------
'name                 ManchesterCoding.BAS
'copyright            © 2018, MCS
'purpose              DEMO MANCHESTER ENCODING and DECODING
'micro                M1280
'-----------------------------------------------------------------
$regfile = "m1280def.dat"                                        '
specify the uC used
$crystal = 32000000                                              '
Oscillator frequency
$hwstack = 40                                                    '
hardware stack
$swstack = 40                                                    '
software stack
$framesize = 40                                                  '
frame space

Dim B As Byte , J As Byte , W As Word
For J = 0 To 255
   W = Manchesterenc(j)                                          '
encode into manchester code whith results into a WORD
   B = Manchesterdec(w)                                          '
decode it back
   If R25 <> 0 Then                                              '
when an error occurs, register r25 is 255
      Print "ERROR"
   End If
   Print J ; " " ; Hex(w) ; " " ; B
Next
```

7.15.14 STR

Action

Returns a string representation of a number.

Syntax

var = **STR**(x [,digits])

Remarks

var	A string variable.
X	A numeric variable.
digits	An options parameter, only allowed for singles and doubles. It specifies how many digits after the comma/point are used.

	When using with a single, you need to use : CONFIG SINGLE=SCIENTIFIC 1059

 The string must be big enough to store the result. So if you have a string like this : Dim S as string * 4, and you use it on a single with the value 0.00000001 then there is not enough space in the string to hold the result. Strings that are assigned with Str() should be dimmed 16 characters long.

You do not need to convert a variable into a string before you print it.
When you use PRINT var, then you will get the same result as when you convert the numeric variable into a string, and print that string.
The PRINT routine will convert the numeric variable into a string before it gets printed to the serial port.

As the integer conversion routines can convert byte, integer, word and longs into a string it also means some code overhead when you do not use longs. You can include the alternative library named mcsbyte 1764.lbx then. This library can only print bytes. There is also a library for printing integers and words only. This library is named mcsbyteint 1764.
When you use these libs to print a long you will get an error message.

See also
VAL 839 , HEX 833 , HEXVAL 832 , MCSBYTE 1764 , BIN 824 , STR2DIGITS 837 , FUSING 830

Difference with VB
In VB STR() returns a string with a leading space. BASCOM does not return a leading space.

Example
```
Dim A As Byte , S As String * 10
A = 123
S = Str(a)
Print S                                                    ' 123
'when you use print a, you will get the same result.
'but a string can also be manipulated with the string routines.
End
```

7.15.15 STR2DIGITS

Action
This statement will convert a string into an array of binary numbers.

Syntax
STR2DIGITS s , ar(1)

Remarks

s	A string variable that holds a number. For example "12345"
ar(1)	The first element of a byte array that will be assigned with the binary representation of the digits.

	After the conversion, the first element will be assigned with the number of processed digits. The next element will become the most right digit of the string, the last element will become the first character of the string. In this example with string "12345" ar(1) = 5 ar(2) = 5 ar(3) = 4 ar(4) = 3 ar(5) = 2 ar(6) = 1 Your array need to be big enough to hold all digits and the digit counter.

You can convert a string into a number with VAL() and a number into a string with STR().
In some cases, it is required to have access to all the individual digits of a variable. While you can use a loop and MOD to get all digits, the STR2DIGITS will work for bytes, word, and longs.

Non numeric digits will not be converted properly. For example, in a string "-0" , the 0 which is ASCII 48, will be converted into a 0. The - is 45 and will result in 45-48=-3, and in byte form : 253.
The dot (.) will be converted into 254.

See also

STR 836, VAL 839

Example

```
'-----------------------------------------------------------------------
--------
'               ARDUINO-Duemilanove.BAS
'           Also tested with ARDUINO NANO V3.0
'              (c) 1995-2025, MCS Electronics
'  This is a sample file for the Mega328P based ARDUINO board
'  Select Programmer 'ARDUINO' , 57600 baud and the proper COM port
'-----------------------------------------------------------------------
--------
$regfile= "m328pdef.dat"    ' used micro
$crystal=16000000           ' used xtal
$baud=19200                 ' baud rate we want
config clockdiv=1           ' either use this or change the divider fuse
byte
'-----------------------------------------------------------------------
--------

dim w as word
dim s as string * 6, ar(6) as byte

config portb=output         ' make portb an output
do
  toggle portb              ' toggle level
  waitms 1000               ' wait 1 sec
  print "Duemilanove"       ' test serial com
```

```
  w=w+1 : s=str(w)          ' convert w to a string
  str2digits s,ar(1)        ' convert string into an array with binary
numbers
loop
```

7.15.16 VAL

Action
Converts a string representation of a number into a number.

Syntax
var = **VAL**(s)

Remarks

Var	A numeric variable that is assigned with the value of s.
S	Variable of the string type.

It depends on the variable type which conversion routine will be used. Single and Double conversion will take more code space.
When you use INPUT, internal the compiler also uses the VAL routines.
In order to safe code space, there are different conversion routines. For example BINVAL and HEXVAL are separate routines.
While they could be added to the compiler, it would mean a certain overhead as they might never be needed.
With strings as input or the INPUT statement, the string is dynamic and so all conversion routines would be needed.

The VAL() conversion routine does not check for illegal characters. If you use them you get a wrong result or 0.
If you want to check for illegal characters you can add a constant named _VALCHECK to your code with a value of 1.
This will include some code that will set the internal ERR variable to 0 or 1. If illegal characters are found, ERR will return 1.
Since VAL is used for the INPUT statement too, this will also work for the INPUT statement.

See also
STR 836 , HEXVAL 832 , HEX 833 , BIN 824 , BINVAL 825 , STR2DIGITS 837

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
```

```
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Dim A As Byte , S As String * 10
S = "123"
A = Val(s)                                                    'convert
string
Print A                                                       ' 123

S = "12345678"
Dim L As Long
L = Val(s)
Print L
End
```

Example2

```
$regfile = "m48def.dat"                                       ' specify
the used micro
$crystal = 8000000                                            ' used
crystal frequency
$baud = 19200                                                 ' use baud
rate
$hwstack = 32                                                 ' default
use 32 for the hardware stack
$swstack = 10                                                 ' default
use 10 for the SW stack
$framesize = 40                                               ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Const _VALCHECK=1                                              ' TEST VAL
()
Dim A As Byte , S As String * 10
S = "123"
A = Val(s)                                                    'convert
string
Print A  ; " ERR:" ; Err                                      ' 123

S = "1234a5678"
Dim L As Long
L = Val(s)
Print L ; " ERR:" ; Err
End
```

7.16 CAN

7.16.1 CANBAUD

Action

Sets the baud rate of the CAN bus.

Syntax

CANBAUD = value

Remarks
All devices on the CAN bus need to have the same baud rate. The value must be a constant. The baud rate depends on the used crystal. The compiler uses the $CRYSTAL value to calculate the CAN baud rate. Higher baud rates require a higher system clock.

See also
CONFIG CANBUS 889, _CONFIG CANMOB 893, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843, CANID 842, CANSELPAGE 844, CANGETINTS 841

Example
```
Canbaud = 125000                                    ' use 125 KB
```

7.16.2 CANGETINTS

Action
Reads the CAN interrupt registers and store into the _CAN_MOBINTS word variable.

Syntax
CANGETINTS

Remarks
This statement is intended to be used in the CAN Interrupt routine. It will read the CAN interrupt registers and stores it into a word variable.
Multiple Message Objects can cause an interrupt at the same time. This means that all message objects need to be checked for a possible interrupt.
In the example this is done with a For Next loop.

```
Cangetints                                          ' read all
the interrupts into variable _can_mobints

      For _can_int_idx = 0 To 14                    ' for all
message objects
        If _can_mobints._can_int_idx = 1 Then       ' if this
message caused an interrupt
          Canselpage _can_int_idx                   ' select
message object
```

The loop checks all bits and if a message object interrupt has been set, the message object will be selected with CANSELPAGE.

See also
CONFIG CANBUS 889, CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843, CANID 842, CANSELPAGE 844

Example

7.16.3 CANID

Action
Returns the ID from the received CAN frame.

Syntax
value = **CANID()**

Remarks
The CANID function can return a 11 bit or 29 bit ID. You need to assign it to a WORD or DWORD variable.
The CANID functions works at the current selected MOB and is typically used inside the CAN interrupt.

See also
CONFIG CANBUS 889, CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843, CANSELPAGE 844, CANGETINTS 841

Example
```
Dim  _canid As Dword
_canid = Canid()                                    ' read the identifier
```

7.16.4 CANCLEARALLMOBS

Action
Clear all Message Objects.

Syntax
CANCLEARALLMOBS

Remarks
Use CANCLEARALLMOBS after you reset the CAN controller to set all registers in the proper state. All registers belonging to the MOB will be clear (set to 0).

See also
CONFIG CANBUS 889, CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANSEND 845, CANRECEIVE 843, CANID 842, CANSELPAGE 844, CANGETINTS 841

Example

```
Canclearallmobs                          ' clear alle message
objects
```

7.16.5 CANCLEARMOB

Action

Clears a Message Object.

Syntax

CANCLEARMOB ObjectNr

Remarks

The ObjectNr is the number of the Message Object you want to clear. This is a number in the range 0-14.
A message object need to be cleared before it can be used. CONFIG CANMOB will clear the object by default.
You can also use CANCLEARALLMOBS to clear all message objects.

See also

CONFIG CANBUS 889 , CONFIG CANMOB 893 , CANBAUD 840, CANRESET 844, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843 , CANID 842, CANSELPAGE 844, CANGETINTS 841

Example

7.16.6 CANRECEIVE

Action

Receives data from a received CAN frame and stores it into a variable.

Syntax

numrec = **CANRECEIVE(**var [, bytes]**)**

Remarks

numrec	Number of bytes received.
var	The variable into which the received data is stored. This must be a numeric variable or array. Version 2076 supports strings as well.
bytes	This is an optional parameter and specifies the number of bytes to retrieve.

The compiler will use the data type of the variable to determine how many bytes need to be retrieved. So when you use a variable that was DIM 1228 ensioned as a long, an attempt will be made to read 4 bytes.

The CANRECEIVE function operates on the current selected Message Object which is selected with CANSELPAGE.
The CANRECEIVE function is intended to be used inside the CAN interrupt routine. After you have retrieved the data from the received CAN frame, the Message Object is free to be used again. You MUST configure it again in order to receive a new interrupt.

See also
CONFIG CANBUS 889, CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANID 842, CANSELPAGE 844, CANGETINTS 841

Example
```
Breceived = Canreceive(porta)                    ' read the data and
store in PORTA
Print #2 , "Got : " ; Breceived ; " bytes"       ' show what we received
Print #2 , Hex(porta)
Config Canmob = -1 , Bitlen = 11 , Msgobject = Receive , Msglen = 1 ,
Autoreply = Disabled , Clearmob = No
          ' reconfig with value -1 for the current MOB and do not set
ID and MASK
```

7.16.7 CANRESET

Action
Reset the CAN controller.

Syntax
CANRESET

Remarks
CANRESET will reset the CAN controller. It is also reset when the processor is reset.

See also
CONFIG CANBUS 889, CONFIG CANMOB 893, CANBAUD 840, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843, CANID 842, CANSELPAGE 844, CANGETINTS 841

Example
```
Canreset                    ' reset can controller
```

7.16.8 CANSELPAGE

Action
Selects the Message Object index or page.

Syntax
CANSELPAGE index

Remarks
All 15 message objects share the same registers. With CANSELPAGE you select the index of the MOB you want to access.
The index is a constant or variable in the range of 0-14.

You should save and restore the CANPAGE register when changing the index. This is shown in the CAN example 889.

See also
CONFIG CANBUS 889, CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843, CANID 842, CANGETINTS 841

Example

7.16.9 CANSEND

Action
Puts the Message Object into Transmit mode and send out data.

Syntax blocking function
status = **CANSEND(**object, var[,bytes]**)**

Syntax non blocking statement
CANSEND object, var[,bytes]

Remarks

status	The status of sending the frame. This should be 0 if there was no problem. If there is an error it will return 1 or higher. The return value is the CANSTMOB register content with the TX bit cleared.
object	The message object number in the range from 0-14. The MOB must have been configured into the DISABLED mode before CANSEND can be used.
var	A variable or array which content will be send. The data type of the variable will be used to determine the number of bytes to send.
bytes	This is an optional value. You can specify how many bytes must be transmitted.

The CANSEND function will disable the TX interrupt and then polls the CANSTMOB register for a change of flags. The TX flag is cleared so that a successful transmission returns a 0.
In case of ACK errors or other errors, a value other then 0 will be returned. Right after the status has changed, the TX and Error interrupt are enabled again and the

CAN interrupt routine is executed. You need to reconfigure the MOB in all cases otherwise you can not send new data.

The CANSEND statement will send the data and returns immediately.
The advantage is that your code can continue running other code. Before new data can be sent it must however have been processed. Check out the CAN-elektor-V2.bas example from the samples\CAN folder.

See also

CONFIG CANBUS 889, CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANRECEIVE 843, CANID 842, CANSELPAGE 844, CANGETINTS 841

Example

Have a look at CONFIG CANBUS 889 for a full example.
The code below only demonstrates that you MUST configure the MOB again in the interrupt routine.
The code below is taken from the sample you find under CONFIG CANBUS
The \examples\CAN folder contains 2 examples too.

```
Elseif Canstmob.6 = 1 Then                              'transmission ready
   Config Canmob = -1 , Bitlen = 11 , Msgobject = Disabled , Msglen = 1
, Clearmob = No
   ' reconfig with value -1 for the current MOB and do not set ID and
MASK
Elseif Canstmob.0 = 1 Then                              'ACK ERROR
    Config Canmob = -1 , Bitlen = 11 , Msgobject = Disabled , Msglen =
1 , Clearmob = No
   ' reconfig with value -1 for the current MOB and do not set ID and
MASK
```

7.17 CLEAR

Action

Clear serial input or output buffer

Syntax

CLEAR bufname

Remarks

Bufname	Serial buffer name to clear. SERIALIN, SERIALIN0 - COM1/UART0 input buffer SERIALIN1 - COM2/UART1 input buffer SERIALIN2 - COM3/UART2 input buffer SERIALIN3 - COM4/UART3 input buffer SERIALIN4 - COM5/UART4 input buffer SERIALIN5 - COM6/UART5 input buffer

	SERIALIN6 - COM7/UART6 input buffer SERIALIN7 - COM8/UART7 input buffer SERIALOUT,SERIALOUT0 - COM1/UART0 output buffer SERIALOUT1 - COM2/UART1 output buffer SERIALOUT2 - COM3/UART2 output buffer SERIALOUT3 - COM4/UART3 output buffer SERIALOUT4 - COM5/UART4 output buffer SERIALOUT5 - COM6/UART5 output buffer SERIALOUT6 - COM7/UART6 output buffer SERIALOUT7 - COM8/UART7 output buffer

When you use buffered serial input or buffered serial output, you might want to clear the buffer.
While you can make the head pointer equal to the tail pointer, an interrupt could be active which might result in an update of the buffer variables, resulting in an unexpected result.
The CLEAR statement will reset the head and tail pointers of the ring buffer, and it will set the buffer count variable to 0. The buffer count variable is new and introduced in 1.11.8.3. It counts how many bytes are in the buffer.
The internal buffercount variable is named _RS_BUFCOUNTxy , where X is **R** for **R**eceive, and **W** for **W**rite, and y is 0 for the first UART, and 1 for the second UART.

See also
CONFIG SERIALIN 1051, CONFIG SERIALOUT 1057

ASM
Calls _BUF_CLEAR from MCS.LIB

Example
```
CLEAR SERIALIN
```

7.18 CLOCKDIVISION

Action
Will set the system clock division available in some MEGA chips.

Syntax
CLOCKDIVISON = var

Remarks

Var	Variable or numeric constant that sets the clock division. Valid values are from 2-129. A value of 0 will disable the division.

On the MEGA 103 and 603 the system clock frequency can be divided so you can save power for instance. A value of 0 will disable the clock divider. The divider can divide

from 2 to 127. So the other valid values are from 2 - 127.

Some routines that rely on the system clock will not work proper anymore when you use the divider. WAITMS for example will take twice the time when you use a value of 2.

Most new processors support a limited number of division factors which can be selected using CONFIG CLOCKDIV 909.

See also
POWERSAVE 1399 , CONFIG CLOCKDIV 909

Example
```
$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Clockdivision = 2
```

7.19 CLOSE

Action
Closes an opened device.

Syntax
OPEN "device" for MODE As #channel
CLOSE #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits. The format for COM1 is : COM1:

	Some chips have 2 UARTS. You can use COM2: to open the second HW UART. Other chips might have 4 or 8 UARTS. The format for the software UART is: COMpin:speed,8,N,stop bits[, INVERTED] Where pin is the name of the PORT-pin. Speed must be specified and stop bits can be 1 or 2. An optional parameter ,INVERTED can be specified to use inverted RS-232. Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232.
MODE	You can use BINARY or RANDOM for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT.
Channel	The number of the channel to open. Must be a positive constant >0.

The statements that support the device are PRINT , INPUT and INPUTHEX , INKEY, WAITKEY.

Using CLOSE on a serial device is optional. Only a file as used with AVR-DOS requires a CLOSE.

The best place for the CLOSE statement is at the end of your program.

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

 For the AVR-DOS file system, you may place the CLOSE at any place in your program. This because the file system supports real file handles.
For the UART, SPI or other devices, you do not need to close the device. Only AVR-DOS needs a CLOSE so the file will be flushed.

See also

OPEN 1386 , PRINT 1501

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : open.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates software UART
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 10000000                                        ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
```

```
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space


Dim B As Byte

'Optional you can fine tune the calculated bit delay
'Why would you want to do that?
'Because chips that have an internal oscillator may not
'run at the speed specified. This depends on the voltage, temp etc.
'You can either change $CRYSTAL or you can use
'BAUD #1,9610

'In this example file we use the DT006 from www.simmstick.com
'This allows easy testing with the existing serial port
'The MAX232 is fitted for this example.
'Because we use the hardware UART pins we MAY NOT use the hardware UART
'The hardware UART is used when you use PRINT, INPUT or other related
statements
'We will use the software UART.
Waitms 100

'open channel for output
Open "comd.1:19200,8,n,1" For Output As #1
Print #1 , "serial output"


'Now open a pin for input
Open "comd.0:19200,8,n,1" For Input As #2
'since there is no relation between the input and output pin
'there is NO ECHO while keys are typed
Print #1 , "Number"
'get a number
Input #2 , B
'print the number
Print #1 , B

'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
   'store in byte
   B = Inkey(#2)
   'when the value > 0 we got something
   If B > 0 Then
      Print #1 , Chr(b)                                      'print the
character
   End If
Loop Until B = 27


Close #2
Close #1


'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B
```

```
'When you dont want to use a level inverter such as the MAX-232
'You can specify ,INVERTED :
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

7.20 COMPARE

Action

This function performs a byte compare on two variables.

Syntax

result **= COMPARE(** var1, var2, bytes**)**

Remarks

result	A word variable that is assigned with the result of the function. When the 2 variables are equal, the value will be 0. When the 2 variables differ, the index is returned of the position that differs.
var1 , var2	Any kind of variable like a long or string. Constants are not supported.
Bytes	The number of bytes to test. The maximum value must fit into a word. (65535).

See also

NONE

Example

```
'-----------------------------------------------------------------------
---
'name                     : compare.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates byte COMPARE function, written
by MWS
'micro                    : Mega88
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
---
' purpose: byte-wise compare
' arg Val1: first value to compare, type = don't care
' arg Val2: second value to compare, type = don't care
' arg BtComp: count of bytes to compare, can be a constant or a
variable
' range is 1 to 65535 bytes
' result: zero if all bytes within range of BtComp are matching
'         1 up to BtComp if there's a miss,
' zero is used for signaling a comlete match, so Config Base has no
```

```
effect
' 1 is always the first byte of the variable, whatever type of variable
it is
'-------------------------------------------------------------------------
---

$regfile = "m328pdef.dat"
$crystal = 16000000
$hwstack = 40
$swstack = 32
$framesize = 32

Const Testver = 2                                           ' edit for
different tests 0,1 or 2

Dim Mmpos As Word                                           ' dimension
word var to hold the result, i.e. mismatch position
Dim btt As Word                                             ' bytes to
test

#if Testver = 0
   Dim Val_a(8) As Byte                                     ' byte
array vs. byte array
   Dim Val_b(8) As Byte                                     ' arrays
are initialyzed 0
   Btt = 8
   Val_a(4) = 1                                             ' test it
#elseif Testver = 1
   Dim Val_a As Double                                      ' Double
vs. byte array
   Dim Val_b(8) As Byte
   Btt = 8
   Val_b(2) = 1                                             ' test it
#elseif Testver = 2                                         ' compare
strings
   Dim Val_a As String * 16
   Dim Val_b As String * 16
   Btt = 12
   Val_a = "Hello Bascom"
   Val_b = "Hello Bascon"                                   ' find the
mismatch
#endif

Mmpos = Compare(val_a , Val_b , Btt)

If Mmpos > 0 Then
   Print "We have a miss at pos: " ; Mmpos
Else
   Print "Match!"
End If
End
```

7.21 CONFIGURATION

7.21.1 CONFIG

The CONFIG statement is used to configure the various hardware devices.
Some CONFIG statements depend on the processor platform. Since some platforms have unique hardware not found on other platforms.

DIRECTIVE	RE-USABLE	NORMAL AVR	XMEGA	XTINY / MEGAX /AVRX	XMEGA ONLY	XTINY/ UPDI ONLY
CONFIG 1WIRE 857	NO	X	X	X		
CONFIG ACAX|ACBX 862	YES		X		X	
CONFIG ACI 861	YES	X				
CONFIG ACx 861 2083 NEW	YES			X		X
CONFIG ADC 864	NO	X				
CONFIG ADCA|ADCB 867	YES		X		X	
CONFIG ADC0|ADCx 880 2083 NEW	YES			X		X
CONFIG ATEMU 883	NO	X				
CONFIG BASE 886	NO	X	X	X		
CONFIG BCCARD 887	NO	X				
CONFIG CANBUS 889	YES	X				
CONFIG CANMOB 893	YES	X				
CONFIG CLOCK 895 2083 ENHANCED	NO	X	X	X		
CONFIG CLOCKDIV 909	YES	X				
CONFIG COM1 909 2083 ENHANCED	YES	X	X	X		
CONFIG COM2 911 also COM3 - COM8	YES	X	X	X		
CONFIG COMx 913	YES	X	X	X		
CONFIG DACA|DACB 918	YES		X		X	
CONFIG DACX 923 2083 NEW	YES			X		X
CONFIG DATE 925	NO	X	X	X		
CONFIG DCF77 927	NO	X	X			
CONFIG DEBOUNCE 935	NO	X	X	X		
CONFIG DMA 936	YES		X		X	
CONFIG DMACHx 937	YES		X		X	
CONFIG DMXSLAVE 949	NO	X	X	X		
CONFIG DP 951	NO	X	X	X		
CONFIG EDMA 944			X		x	
CONFIG EDMAx 945			X		x	
CONFIG EEPROM 952	NO		X	X	X	
CONFIG ERROR 953	NO	X	X	X		
CONFIG EVENT_SYSTEM 953 XMEGA	YES		X		X	
CONFIG EVENT_SYSTEM 956	YES			X		X

XTINY 2083 NEW						
CONFIG EXTENDED_PORT 958	NO	X				
CONFIG FT800 959	NO	X	X	X		
CONFIG FUSES 962	NO			X	X	X
CONFIG GRAPHLCD 964	NO	X	X	X		
CONFIG HITAG 970	NO	X				
CONFIG I2CBUS 974	YES	X	X	X		
CONFIG I2CDELAY 975	NO	X				
CONFIG I2CSLAVE 978	NO	X	X			
CONFIG INPUT 983	NO	X	X	X		
CONFIG INPUTBIN 984	NO	X	X	X		
CONFIG INTx 985	YES	X	X	X		
CONFIG INTVECTORSELECTION 987	YES	X	X	X		
CONFIG KBD 988	NO	X	X			
CONFIG KEYBOARD 988	NO	X	X	X		
CONFIG LCD 992	NO	X	X	X		
CONFIG LCDBUS 998	NO	X	X	X		
CONFIG LCDMODE 1001	NO	X	X	X		
CONFIG LCDPIN 1001	NO	X	X	X		
CONFIG OPAMP 1005 2085 NEW	YES			X		X
CONFIG OSC 1008 XMEGA	YES		X	X		
CONFIG OSC 1009 XTINY 2085 NEW	YES			X		
CONFIG PORT 1011	YES	X	X	X		
CONFIG PORT_MUX 1014 2084 NEW	YES			X		X
CONFIG POWERMODE 1017	YES	X	X	X		
CONFIG POWER_REDUCTION 1023	NO		X		X	
CONFIG PRIORITY 1026 XMEGA	YES		X		X	
CONFIG PRIORITY 1027 XTINY 2083 NEW	YES			X		X
CONFIG PRINT 1028	NO	X	X	X		
CONFIG PRINTBIN 1029	NO	X	X	X		
CONFIG PS2EMU 1030	NO	X				
CONFIG RAINBOW 1033	NO	X	X	X		
CONFIG RC5 1037	NO	X				
CONFIG RC5SEND 1042	NO			X		X
CONFIG RND 1044	NO	X	X	X		
CONFIG SERIALIN 1051	NO	X	X	X		
CONFIG SERIALIN1 1051	NO	X	X			
CONFIG SERIALIN2 1051	NO	X	X			
CONFIG SERIALIN3 1051	NO	X	X			
CONFIG SERIALOUT 1057	NO	X	X			
CONFIG SERIALOUT1 1057	NO	X	X			
CONFIG SERIALOUT2 1057	NO	X	X			
CONFIG SERIALOUT3 1057	NO	X	X			
CONFIG SERVOS 1071	NO	X	X			

CONFIG SHIFTIN 1060	NO	X	X	X		
CONFIG SINGLE 1059	YES	X	X	X		
CONFIG SDA 1046	NO	X	X	X		
CONFIG SCL 1049	NO	X	X	X		
CONFIG SPI 1061	NO	X				
CONFIG SPIx 1068	YES		X	X	X	
CONFIG STRCHECK 1075 2086 NEW	NO	X	X	X		
CONFIG SUBMODE 1079	NO	X	X	X		
CONFIG SYSCLOCK 1081 XMEGA	YES		X		X	
CONFIG SYSCLOCK 1082 XTINY 2083 NEW	YES			X		X
CONFIG TCXX 1094	YES		X	X	X	
CONFIG TCA0 1084	YES			X		X
CONFIG TCB0, TCB1 1087 2084 NEW	YES			X		X
CONFIG TCD0 1090 2084 NEW	YES			X		X
CONFIG TCPIP 1098	NO	X	X	X		
CONFIG TWI 1116	YES	X	X	X		
CONFIG TWISLAVE 1123	NO	X				
CONFIG TWIxSLAVE 1128	NO		X		x	
CONFIG TIMER0 1109	YES	X				
CONFIG TIMER1 1112	YES	X				
CONFIG TIMER2 and 3 1115	YES	X				
CONFIG USB 1132	NO	X				
CONFIG USI 1138	NO	X				
CONFIG VARPTRMODE 1142	YES	X	X	X		
CONFIG VPORT 1144	YES		X		X	
CONFIG VREF 1147 XTINY 2083 NEW	YES			X		X
CONFIG VREGPWR 1148 AVRX 2085 NEW	YES			X		X
CONFIG WATCHDOG 1149	YES	X	X	X		
CONFIG WAITSUART 1149	NO	X	X	X		
CONFIG X10 1156	NO	X				
CONFIG XPIN 1158	YES	X	X	X		
CONFIG XRAM 1161	YES	X	X			
CONFIG ZCDx 1167 2085 NEW	YES			X		X

Some CONFIG directives are intended to be used once. Others can be used multiple times. For example you can specify that a port must be set to input after you have specified that it is used as an input.

You cannot change the LCD pins during run time. In that case the last specification will be used or an error message will be displayed.

Some configuration commands are only available to the Xmega. An X in the 'Xmega Only' column indicates that the command can only be used for an Xmega processor. Some configuration commands are exclusive for the Xtiny processors. An X in the 'Xtiny Only' column indicates that the commands can only be used for an Xtiny 460 processor.

With XTYINY 460 we also mean MEGAX 490 and AVRX 503.

PRESERVE and OVERWRITE

In version 2084 a part of the hard coded option logic is moved to the DAT files.
Some CONFIG statements like CONFIG PORT_MUX have a new value : PRESERVE or OVERWRITE.
Other CONFIG statements might have an option named REGMODE. This REGMODE option has the same possible values : PRESERVE and OVERWRITE.

So what is this OVERWRITE or PRESERVE about?
When you configure the various options for a piece of hardware, the compiler will convert these options to the proper values and write this to the proper registers.
Only registers that are changed will be written too. But normally the whole register will be written too.
When a register sets just one option for example the clock speed, that is no problem. But when the register also has a few bits that set other hardware it might become a problem.
When you create the initial CONFIG statement there is no problem since all required bits will be set. But if you want to change a configuration later and you want to change just a single part of the configuration you could erase an option.

Lets add a simple sample. Assume there is a register named REGCTRLA and it has 8 bits. The 3 lower bits are used to set the pre-scaler. The upper 1 bit is used to enable some output pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
O	x	x	x	x	S	S	S

The x mean don't care. It does not mean what we write to it.
Now when you would : CONFIG TEST=DUMMY, PRESCALER=2, OUTPUT=ENABLED
Here you configure all the bits in the register. So it is not needed to preserve any bits, the compiler can simply write the proper value to the register.

When REGMODE is not used it is the same as OVERWRITE and the same as : CONFIG TEST=DUMMY, PRESCALER=2, OUTPUT=ENABLED, REGMODE=OVERWRITE
Here all the bits will be written. But also when you use : CONFIG TEST=DUMMY, PRESCALER=2
This has the implicit OVERWRITE. What happens however is that only the prescaler value is provided. Since there is no preservation for other bits, the compiler will write a zero for the P bit.

And that is why the PRESERVE option exist : it will load the register value, alter it, and write it back. This will use more code.

When you use the default OVERWRITE mode the compiler will try to re-use register values. This also means that the compiler output might vary depending on the settings !
For example when you set an option that need to write to 3 registers, and all registers values are different, 3 registers are loaded with those values and written to the hardware registers.
But when the options you select result in a similar values, it is not needed to load the same value multiple times, and a different register will be used.
So the resulting binary might become bigger/smaller depending on the options.

7.21.2 CONFIG 1WIRE

Action
Configure the pin to use for 1WIRE statements and override the compiler setting.

Syntax
CONFIG 1WIRE = pin [, extended=0|1]

Remarks

Pin	The port pin to use such as PORTB.0
extended	An optional constant value of 0 or 1. This is an optional parameter

The CONFIG 1WIRE statement overrides the compiler setting. It is the preferred that you use it. This way the setting is stored in your source code.
You can configure only one pin for the 1WIRE statements because the idea is that you can attach multiple 1WIRE devices to the 1WIRE bus.

You can however use multiple pins and thus multiple busses. All 1wire commands and functions need the port and pin in that case. A CONFIG 1WIRE statement is not need in that case either.

The 1wire commands and function will automatically set the DDR and PORT register bits to the proper state. You do not need to bring the pins into the right state yourself.

It is important that you use a pull up resistor of 4K7 ohm on the 1wire pin(for 5V VCC). The pull up resistor of the AVR is not sufficient.

Also notice that some 1wire chips also need +5V. 1 wire is just marketing since you need GND anyway. The least is 2 wires and typical you need 3 wires.

Extended
The extended option is only required when you use multiple busses/pins and if these pins mix normal and extended addresses.
Let's clear that up. When the 1wire code was written in 1995 all the port addresses were normal I/O addresses. These are addresses that fit in the I/O space (address < &H60). To save code, register R31 was cleared in the library and the port register was passed in R30.
When Atmel introduced the extended I/O registers with address >&HFF, it was possible to set R31 to a fixed value when the user port was an extended I/O address. But when you want to mix the addresses, there is no other way then to pass the word address of the I/O register to the library code.
And that is exactly what EXTENDED=1 will do. It will use more code. This support was written for a customer that already made his PCB's. We do advise to use the same port when you use multiple pins.

ATMEGA128 PORTF
The ATMEGA128 PORTF is split up. Normally, the DDR, PIN and PORT registers are in the same order.
For example : PORTB = &H18 , DDRB = &H17 and PINB = &H16
But PORTF in the MEGA128 is different : PINF = &H00 , PORTF = &H62 , DDRF = &H61
You need a special library named **M128-1wire-PortF.lib** 1766 for this processor and port. This library is fixed to portF

See also

1WRESET 718, 1WREAD 720, 1WWRITE 729, 1WIRECOUNT 716, 1WRESET 718, 1WSEARCHFIRST 723, 1WSEARCHNEXT 725

Example

```
'-----------------------------------------------------------------------
---------
'name                     : 1wire.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates 1wreset, 1wwrite and 1wread()
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"
$crystal = 8000000

$hwstack = 32                                                 ' default
use 32 for the hardware stack
$swstack = 10                                                 'default use
10 for the SW stack
$framesize = 40                                               'default use
40 for the frame space


Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"


Config 1wire = Portb.0                                        'use this
pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte


Do
  Wait 1
  1wreset                                                     'reset the
device
  Print Err                                                   'print error
1 if error
  1wwrite &H33                                                'read ROM
command
  For I = 1 To 8
    Ar(i) = 1wread()                                          'place into
array
  Next

'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)                                             'read 8
bytes
```

```
  For I = 1 To 8
     Print Hex(ar(i));                                          'print
output
  Next
  Print                                                         'linefeed
Loop


'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!

'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
  Ar(i) = 0                                                     'clear array
to see that it works
Next

1wreset Pinb , 2                                                'use this
port and  pin for the second device
1wwrite &H33 , 1 , Pinb , 2                                     'note that
now the number of bytes must be specified!
'1wwrite Ar(1) , 5,pinb,2

'reading is also different
Ar(1) = 1wread(8 , Pinb , 2)                                    'read 8
bytes from portB on pin 2

For I = 1 To 8
  Print Hex(ar(i));
Next

'you could create a loop with a variable for the bit number !
For I = 0 To 3                                                  'for pin 0-3
  1wreset Pinb , I
  1wwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8 , Pinb , I)
  For A = 1 To 8
    Print Hex(ar(a));
  Next
  Print
Next
End
```

Xmega Example

```
'-----------------------------------------------------------------------
---------
'name                    : XM128-1wire.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates 1wreset, 1wwrite and 1wread()
'micro                   : Xm128A1
'suited for demo         : no
'commercial addon needed  : no
' pull-up of 4K7 required to VCC from Portb.0
' DS2401 serial button connected to Portb.0
'-----------------------------------------------------------------------
---------
$regfile = "xm128a1def.dat"
$crystal = 32000000

$lib "xmega.lib" : $external _xmegafix_clear : $external
_xmegafix_rol_r1014
```

```
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 32                                                'default use
10 for the SW stack
$framesize = 32                                              'default use
40 for the frame space

'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

'configure UART
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8


'configure 1wire pin
Config 1wire = Portb.0                                       'use this
pin

Dim Ar(8) As Byte , A As Byte , I As Byte

Print "start"

A = 1wirecount()
Print A ; " devices found"

'get first
Ar(1) = 1wsearchfirst()

For I = 1 To 8                                               'print the
number
  Print Hex(ar(i));
Next
Print

Do
   'Now search for other devices
   Ar(1) = 1wsearchnext()                                    ' get next
device
   For I = 1 To 8
    Print Hex(ar(i));
   Next
   Print
Loop Until Err = 1

Waitms 2000


Do
  1wreset                                                    'reset the
device
  Print Err                                                  'print error
1 if error

  1wwrite &H33                                               'read ROM
command
'  Ar(1) = 1wread(8)    you can use this instead of the code below

  For I = 1 To 8
    Ar(i) = 1wread()                                         'place into
array
```

```
  Next

  For I = 1 To 8
     Print Hex(ar(i));                                    'print
output
  Next
  Print                                                   'linefeed
  Waitms 1000
Loop


End
```

7.21.3 CONFIG ACI

Action
Configures the Analog Comparator.

Syntax
CONFIG ACI = ON|OFF, COMPARE = ON|OFF, TRIGGER=TOGGLE|RISING|FALLING

Remarks

ACI	Can be switched on or off
COMPARE	Can be on or off. When switched ON, the TIMER1 in capture mode will trigger on ACI too.
TRIGGER	Specifies which comparator events trigger the analog comparator interrupts.

See also
NONE

Example
NONE

7.21.4 CONFIG ACX

Action
Configures the Analog Comparator of the Xtiny.

Syntax
CONFIG ACX = state, RUNMODE=runmode, OUTPUT=otp ,TRIGGER=trigger, LOW_POWER=lowpow, HYSMODE=hys , MUX_INVERT=muxinv , MUXMIN=muxmin , MUXPOS=muxpos

Remarks

ACIX	The name of the Analog comparator : AC0,AC1, AC2.

	Some XTINY processors have multiple comparators.
State	ON or OFF. Select ON to turn the comparator on. By default it is OFF.
Runmode	Possible values : ENABLED : In Standby sleep mode, the peripheral continues operation DISABLED : In Standby sleep mode, the peripheral is halted
Trigger	Specifies which comparator event triggers the analog comparator interrupts. This options are : RISING, FALLING or BOTH.
Hysmode	To prevent quick toggling, a hysteresis is built in. You can chose the mode : - OFF - 10 (10 mV) - 25 (25 mV) - 50 (50 mV)
lowpow	ENABLED or DISABLED. When you enable this mode the current through the comparator is reduced. It reduces power consumption but increase the reaction time of the comparator.
Muxinv	ENABLED or DISABLED (default) When enabled the output of the AC is inverted. This effectively inverts the input to all the peripherals connected to the signal, and also affects the internal status signals.
Output	ENABLED or DISABLED (default). When the output is enabled, the output of the comparator is routed to the output pin of the port.
muxmin	Negative input MUX selection. Possible values : - AINN0 : negative pin 0 - AINN1 : Negative pin 1 - VREF : voltage reference - DAC : DAC output
muxpos	Positive input MUX selection. Possible values : - AINP0 : positive PIN 0 (default) - AINP1 : positive pin 1

See also
NONE

Example

7.21.5 CONFIG ACAX|ACBX

Action
Configures the Analog Comparator of the Xmega.

Syntax
CONFIG ACXX = state, TRIGGER=trigger, HISPEED=speed, HYSMODE=hys , MUXPLUS=mp , MUXMIN=mm , OUTPUT=otp , SCALE=scale , WINDOW=w , WINTMODE = wint

Remarks

ACXX	The name of the Analog comparator : ACA0,ACA1, ACB0 or ACB1

	Some XMEGA chips might not have (all) comparators.
State	ON or OFF. Select ON to turn the comparator on. By default it is off.
HiSpeed	When ENABLED, the comparator hi speed mode is activated. Default mode is DISABLED.
Trigger	Specifies which comparator event triggers the analog comparator interrupts. This options are : RISING, FALLING or BOTH / TOGGLE.
Hysmode	To prevent quick toggling, a hysteresis is built in. You can chose the mode : - OFF - SMALL - LARGE
MuxPlus	This option controls which pin is connected to the positive input of the comparator. Possible values : 0-7, DAC. When you chose 7, DAC will also be used. So 7 and DAC are equivalent.
MuxMin	This option controls which pin is connected to the negative input of the comparator. Possible values : 0-7, DAC, BANDGAP, SCALER. 0 - connects pin 0 1 - connects pin 1 2 - connects pin 3 ! 3 - connects pin 5 4 - connects pin 7 5 - connects the DAC output (same as DAC option) 6 - connects the BANDGAP voltage (same as BANDGAP option) 7 - connects the SCALER output (same as SCALE option)
Output	Enabled or Disabled (default). When the output is enabled, the output of the comparator is routed to pin 7 of the port. For ACA1 the output is routed to the AC1OUT pin if the Xmega supports this.
Scale	The input voltage of the negative mux pin can be scaled. The scale value must be in range from 0-63. The scale output voltage is calculated as : (vcc * (scale+1)) / 64 Thus a value of 63 would give VCC. And 32 would give vcc/2
Windows	Enabled or Disabled (default). When enabled, the two comparators of the port (ACA0 + ACA1) or (ACB0 + ACB1) form a window discriminator so you can control if a voltage is in the range of the lower and upper comparator.
WintMode	The status register contains the window state. (bit 6 and 7). You can also fire an interrupt at one of the states: ABOVE : interrupt on signal above window INSIDE : interrupt on signal inside window BELOW : interrupt on signal below window OUTSIDE : interrupt on signal outside window

A window is used in battery voltage meters. you could set the lower voltage to 12 V. And the upper voltage to 14 V.
If the voltage is inside this window : >=12V and <=14V then the battery is OK.
If the voltage is below the battery need to be charged.
If the voltage is above the window the battery if fully charged. The mentioned values are just an example.

See also
NONE

Example

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-AC.bas
'  This sample demonstrates the Analog Comparator
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 64
$framesize = 64
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014

'First Enable The Osc Of Your Choice , make sure to enable 32 KHz clock
or use an external 32 KHz clock
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

'setup comparator pin 0 and pin 1 are the input of portA. Pin 7 is an
output in this sample
Config Aca0 = On , Hysmode = Small , Muxplus = 0 , Muxmin = 1 , Output =
Enabled



Do
  Print Bin(aca_status)
  Print Aca_status.4                                        ' output ac0
  Waitms 1000
Loop
```

7.21.6 CONFIG ADC

Action

Configures the A/D converter.

Syntax

CONFIG ADC = single, PRESCALER = AUTO, REFERENCE = opt

Remarks

ADC	Running mode. May be SINGLE or FREE. This is the converter mode and has nothing to do with single ended or differential input mode.
PRESCALER	A numeric constant for the clock divider. Use AUTO to let the compiler generate the best value depending on the XTAL

REFERENCE	The options depend on the used micro. Some chips like the M163 have additional reference options. In the definition files you will find : ADC_REFMODEL = x This specifies which reference options are available. The possible values are listed in the table below.

Chip	Modes	ADC_REFMODEL
2233,4433,4434,8535, m103,m603, m128103	OFF AVCC	0
m165, m169, m325, m3250, m645, m6450, m329,m3290, m649, m6490,m48,m88,m168	OFF AVCC INTERNAL or INTERNAL_1.1	1
tiny15,tiny26	AVCC OFF INTERNAL INTERNALEXTCAP	2
tiny13	AVCC INTERNAL	3
tiny24,tiny44,tiny84	AVCC EXTERNAL or OFF INTERNAL or INTERNAL_1.1	4
m164,m324,m644,m640, m1280, m1281,m2561, m2560	AREF or OFF AVCC INTERNAL1.1 INTERNAL_2.56	5
tiny261,tiny461,tiny861, tiny25,tiny45,tiny85	AVCC EXTERNAL or OFF INTERNAL_1.1 INTERNAL_2.56_NOCAP INTERNAL_2.56_EXTCAP	7
CAN128, PWM2_3, USB1287, m128, m16, m163, m32, m323, m64	AREF or OFF AVCC INTERNAL or INTERNAL_2.56	8
	You may also use VALUE=value	

When you use VALUE=value, you may specify any value. The disadvantage is that when you port your code from one chip to another it will not work.
While the AREF, AVCC, etc. are all converted to the right settings, the value can not be converted.

The AD converter is started automatic when you use the CONFIG ADC command. You can use STOP 1544 ADC and START 1538 ADC to disable and enable the power of the AD converter.

The GETADC 1265() function is intended to be used with the SINGLE running mode. This means that each time you call GETADC(), a conversion is started. If you use the free running mode, you need to retrieve the value from the AD converter yourself. For example by reading the internal ADC word variable.

See also

GETADC 1265 , CONFIG ADCx 867

Example

```
'-----------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
or M163 micro
'micro                    : Mega163
'suited for demo          : yes
'commercial addon needed  : no
'use in simulator         : possible
' Getadc() will also work for other AVR chips that have an ADC converter
'-----------------------------------------------------------------------
---------
$regfile = "m163def.dat"                                    ' we use the
M163
$crystal = 4000000

$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               'default use
10 for the SW stack
$framesize = 40                                             'default use
40 for the frame space


'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc  ' NOT required since it will start automatic

'With STOP ADC, you can remove the power from the chip
'Stop Adc

Dim W As Word , Channel As Byte

Channel = 0
'now read A/D value from channel 0
Do
  W = Getadc(channel)
  Print "Channel " ; Channel ; " value " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
Loop
End

'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
'OFF      : AREF, internal reference turned off
'AVCC     : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin
```

```
'Using the additional param on chip that do not have the internal
reference will have no effect.
```

7.21.7 CONFIG ADCA|ADCB

Action

Configures the A/D converter of the Xmega.

See also ATXMEGA 425 for base info on ATXMEGA.

Syntax

CONFIG ADCA | ADCB = mode, CONVMODE=sign, RESOLUTION=res, DMA=dma, REFERENCE=ref,EVENT_MODE=evt, EVENT_CHANNEL=evtchan, PRESCALER=pre, BANDGAP=gap, TEMPREF=tref, SWEEP=sweep, CH0_GAIN=gain, CH0_INP= inp, MUX0=mux, CH1_GAIN=gain, CH1_INP= inp, MUX1=mux , CH2_GAIN=gain, CH2_INP= inp, MUX2=mux, CH3_GAIN=gain, CH3_INP= inp, MUX3=mux

Remarks

mode	Running mode. May be SINGLE or FREE.
sign	The conversion mode. This can be **SIGNED** or **UNSIGNED**. When choosing SIGNED you should assign the result to an integer. When choosing UNSIGNED you should assign the result to a word. The default is UNSIGNED. When the ADC uses differential input, **SIGNED** mode must be used, when using single ended input both signed or **UNSIGNED** mode can be used. Note: • Conversion mode is configured for the whole ADC, not individually for each channel, which means that the ADC must be put in the signed mode even if only one of the channels uses differential inputs. • Negative values are not negative inputs on the IO pins, but higher voltage level on the negative input in respect to the positive input. Even though the resulting value can be negative. For example +1.4 V on negative Input and +0.3 V on positive input is OK. • Do not apply Voltages below GND or above VCC !!
res	The resolution of the conversion. Valid values are : - **8BIT** - **12BIT**. This is the default - **LEFT12BIT**. This will result in a left aligned 21 bit value.
dma	If you want to use the DMA channel, you can select which DMA channels must be used: - **OFF** (no DMA) - **CH01** (channel 0 + 1) - **CH012** (channel 0 + 1 + 2) - **CH0123** (channel 0 + 1 + 2 + 3)
ref	Selects the reference to use. Valid options : - **INT1V**. For internal 1V reference - **INTVCC**. For internal voltage divided by 1.6 - **AREFA**. External reference from AREF pin on PORT A. - **AREFB**. External reference from AREF pin on PORT B.
gap	Enables the bangap reference. Use **ENABLED** or **DISABLED**.

	Setting this bit enables the bandgap to prepare for ADC measurement. Note that if any other functions are using the bandgap already, this bit does not need to be set. This could be when the internal 1V reference is used in ADC or DAC, or if the Brown-out Detector is enabled.
tref	Enables the temperature reference. Use **ENABLED** or **DISABLED**. Setting this bit enables the temperature reference to prepare for ADC measurement
sweep	Selects which channels are included in a sweep when a channel sweep is triggered by the event system or in the free running mode. Valid options : - **CH0** : channel 0 included - **CH01** : channel 0 and 1 included - **CH012** : channel 0-2 included - **CH0123** : all channels are included
evtchan	Event channel selection. This selects which channel should trigger which ADC channel. Valid options: - **CH0123**. Event channel 0, 1, 2, 3 as selected inputs - **CH1234**. Event channel 1, 2, 3, 4 as selected inputs - **CH2345**. Event channel 2,3, 4, 5 as selected inputs - **CH3456**. Event channel 3, 4, 5, 6 as selected inputs - **CH4567**. Event channel 4, 5, 6, 7 as selected inputs - **CH456**. Event channel 4, 5, 6 as selected inputs - **CH67**. Event channel 6 and 7 as selected inputs - **CH7**. Event channel 7 as selected input
evt	Event channel mode selection. This selects how many of the selected event channel are in use. Valid options: - **NONE**. Event system is not used - **CH0**. Event channel with the lowest number, defined by evtchan triggers conversion on channel 0 - **CH01**. Event channel with the two lowest numbers, defined by evtchan trigger conversion on channel 0 and 1 respectively - **CH012**. Event channel with the three lowest numbers, defined by evtchan trigger conversion on channel 0, 1 and 2 respectively - **CH0123**. Event channel defined by evtchan trigger conversion on channel 0, 1, 2 and 3 respectively - **SWEEP**. One sweep of all active ADC channels defined by SWEEP on incoming event channel with the lowest number, defined by evtchan - **SYNCSWEEP**. One sweep of all active ADC channels defined by SWEEP on incoming event channel with the lowest number, defined by evtchan. In addition, the conversion will be synchronized on event to ensure a very accurate timing for the conversion.
pre	Prescaler value. The prescaler divides the system clock and applies it to the A/D converter. Valid prescaler values : - 4, 8, 16, 32, 64, 128, 256 and 512
gain	Each of the 4 channels can have a different gain. Valid values are : 1,2,4,8,16,32 and 64
inp	Each of the 4 channels can have a different mode. The 4 modes are : - **INTERNAL**. For example for temperature measurement - **SINGLE_ENDED**. For measuring positive voltages - **DIFF**. For differential input without gain which allows to measure

	negative voltages. - **DIFFWGAIN**. Same as DIFF but with gain.
mux	Selects the MUX to use with the channel. This must be a numeric constant. The value depends on the mode. See details below under *How to select the MUX to use with the channel*. At run time you can change the ADC**x**_CH**y**_MUXCTRL register. Where x is A or B, and y is the channel 0-3.

XMEGA chips are grouped into different families. For example the features of an A-family device differ from a B-family or D-family device.
An example for a A-family device is ATXMEGA128**A**1.

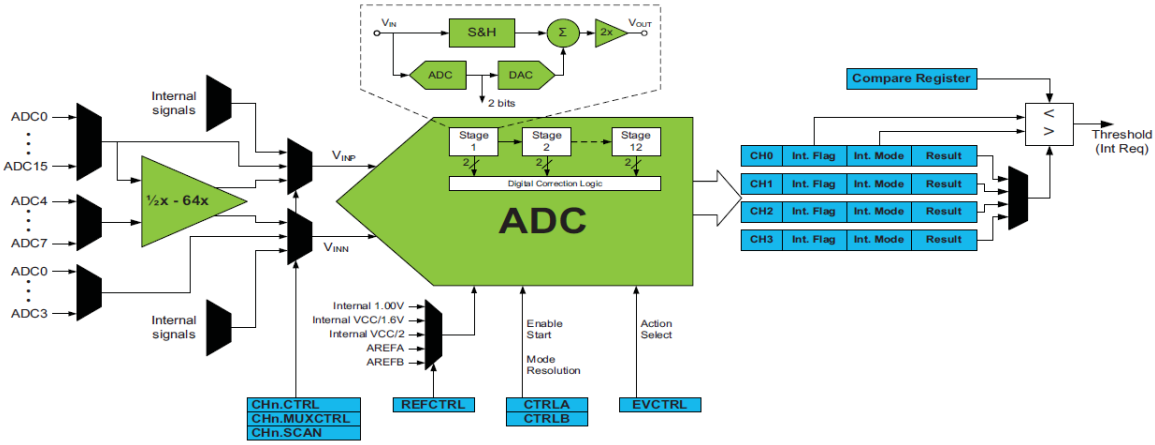
The following table show the differences of the different XMEGA families:

	AVR XMEGA A	AVR XMEGA B	AVR XMEGA D
ADCA	Yes	Yes	Yes
ADCB	Yes	Yes	- -
Channel 0	Yes	Yes	Yes
Channel 1	Yes	- -	- -
Channel 2	Yes	- -	- -
Channel 3	Yes	- -	- -
Architecture	Pipelined	Cyclic	Cyclic
Max ADC frequency	2MHz	1.4Mhz	1.4MHz
Single propagation ADC cycles number (12 bits)	7	7	7
Single propagation ADC cycles number (8 bits)	5	5	5
Max sample per second (12 bits)	2Msps	200Ksps	200Ksps
ADC result to DMA	Yes	Yes	- -
SWEEP mode (channel sweep)	Yes	- -	- -
Number of Internal inputs	4	3	3
Internal inputs	Temp, Vcc/10, Bandgap, DAC	Temp, Vcc/10, Bandgap	Temp, Vcc/10, Bandgap

x 0.5 Gain	- -	Yes	- -
Voltage reference = INTVCC/2	- -	Yes	- -

The XMEGA A-family ADC conversion block has a 12-stage pipelined architecture capable of sampling several signals almost parallel. There are four input selection multiplexers with individual configurations. The separate configuration settings for the four multiplexers can be
viewed as virtual channels, with one set of result registers each, all sharing the same ADC conversion block.

ADC overview of XMEGA AU (Xmega with USB):



So with the pipelined structure, four basic elements (Virtual Channels) can be used at the same time.
Each signal propagates through the 12-stage pipelined ADC Block (12-stage for 12-Bit), where one bit is converted at each stage.
The propagation time for one single 12-Bit signal conversion through the pipeline is 7 ADC clock cycles for 12-bit conversions. If Gain
is used the propagation time increases by one cycle.

When free running mode is configured an ADC channel will continuously sample and do new conversions.

12-Bit = [MSB , Bit 10 , Bit 9 , Bit 8, Bit 7 , Bit 6, Bit 5, Bit 4, Bit 3, Bit 2, Bit 1, LSB]

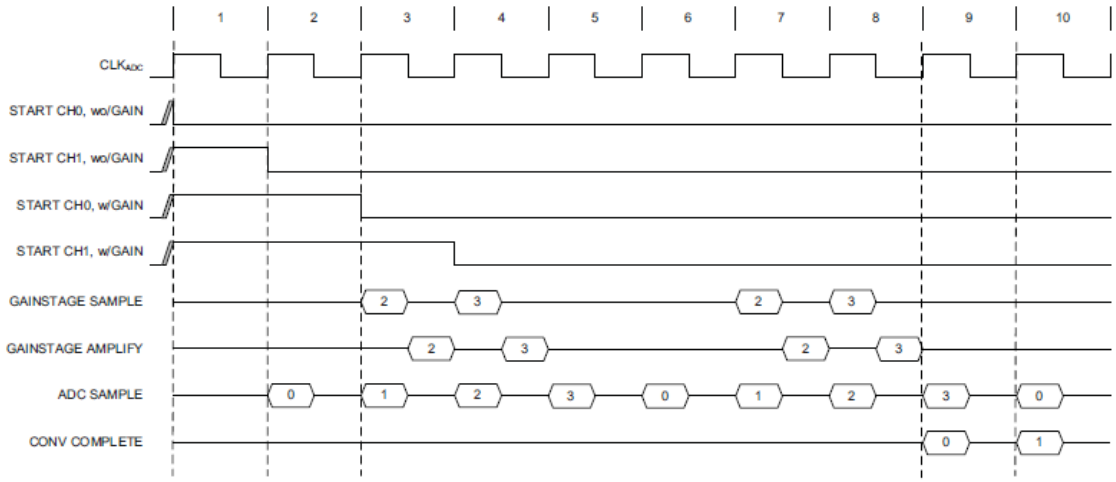
If 4 Virtual ADC Channels are used the pipelined architecture will work as following:

ADC Clock Cycle 1: Start Ch0 without gain
ADC Clock Cycle 2: Channel 0 **MSB** (Bit11)
ADC Clock Cycle 3: Channel 0 Bit9, Channel 1 MSB
ADC Clock Cycle 4: Channel 0 Bit7, Channel 1 Bit9, Channel 2 MSB
ADC Clock Cycle 5: Channel 0 Bit5, Channel 1 Bit7, Channel 2 Bit9, Channel 3 MSB
ADC Clock Cycle 6: Channel 0 Bit3, Channel 2 Bit5, Channel 2 Bit7, Channel 3 Bit9
ADC Clock Cycle 7: Channel 0 Bit1, Channel 2 Bit3, Channel 2 Bit5, Channel 3 Bit7

ADC Clock Cycle 8: Channel 0 **LSB**
ADC Clock Cycle 9: Channel 0 conversion complete .....
ADC Clock Cycle 10 Channel 1 conversion complete ....
....
.....

The even elements (0, 2, 4 …) of 12-stage pipelined ADC Block will be enabled during the high level of the ADC
clock, and the odd elements (1, 3 , 5 …) of 12-stage pipelined ADC Block will be enabled during the low level of the
ADC clock.

After four ADC clock cycles all 4 ADC channels have done the first sample bit (the MSB).



For further details see Atmel Application Notes and data sheets.

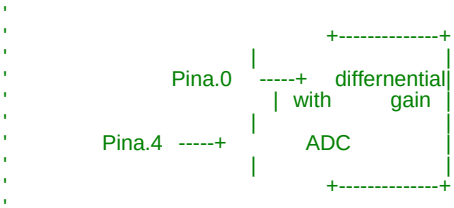
If real simultaneous conversions are needed on different channels then you need to use 2 ADC's. For example Channel 0 of ADCA and Channel 0 of ADCB an A-family device can be measured absolute simultaneously.

Selectable voltage input types:

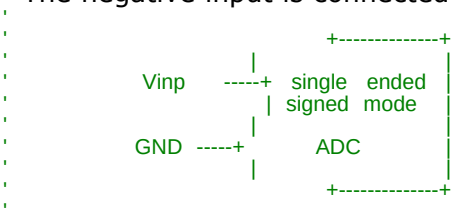
- Differential measurement without gain
  The ADC must be in signed mode when differential input is used
  Pin 0...Pin 7 can be selected as positive input
  Pin **0**...Pin **3** can be sleected as negative input

```
                          +--------------+
                          |              |
Pina.0  -----+    differential|
             | without  gain |
Pina.1  -----+       ADC      |
                          |              |
                          +--------------+
```

- Differential measurement with gain
  The gain is selectable to 1/2x, 1x, 2x, 4x, 8x, 16x, 32x and 64x gain
  The ADC must be in signed mode when differential input is used
  Pin 0...Pin 7 can be selected as positive input
  Pin **4**...Pin **7** can be sleected as negative input



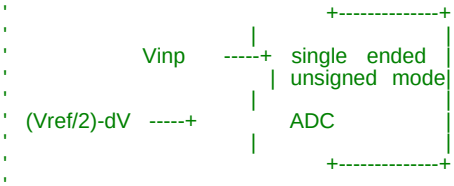
• Single ended input (signed mode)
The ADC is differential, so for single ended measurements the negative input is connected to a fixed internal value.
The negative input is connected to internal ground (GND) in signed mode.



• Single ended input (unsigned mode)
In unsigned mode the negative input is connected to half of the voltage reference (Vref) voltage minus a fixed device specific negative offset
The approximate value corresponding to ground is around 200. This value corresponds to the digital result of ΔV (0.05 * 4096).
This value also depend on the selected voltage reference so you should measure the real value by first selecting the voltage reference.
(Δ V = Vref * 0.05)

How to measure the offset ?
Connect the ADC input pin (Vinp) to GND and measure the offset.
This is also called offset calibration. This value can be stored for example in EEPROM and is therefore available for all other measurements.
See also example below.

This offset calibration value is then subtracted to each ADC output
The offset enables the ADC to measure for example zero crossing in unsigned mode.



• Internal input
The ADC is differential, so for single ended measurements the negative input is connected to a fixed internal value

How to select the MUX to use with the channel

Mux0 = &B0_0000_000

Bit 0...2 of MUX0 = MUX selection on negative ADC input (For internal or single-ended measurements, these bits are not in use.)
Bit 3...6 of MUX0 = MUX selection on Positive ADC input

Input mode = INTERNAL:

MUX POSITIVE INPUT	Group Configuration	Description
0000	TEMP	Temperature Reference
0001	Bandgap	Bandgap voltage
0010	SCALEDVCC	1/10 scaled Vcc
0011	DAC	DAC output

For example:

```
W = Getadc( adcb , 0 , &B0_0011_000)                    'Measure DAC
```

Another example:

```
Ch0_gain = 1 , Ch0_inp = INTERNAL , Mux0 = &B0_0011_000 'configure MUX0 to measure internal DAC
```

Input mode = SINGLE_ENDED, DIFF or DIFFWGAIN:

MUX POSITIVE INPUT	Group Configuration	Description
0000	Pin0	ADC0
0001	Pin1	
0010	Pin2	
0011	Pin3	
0100	Pin4	
0101	Pin5	
0110	Pin6	
0111	Pin7	
1000	Pin8	
1001	Pin9	
1010	Pin10	
1011	Pin11	
1100	Pin12	
1101	Pin13	
1110	Pin14	
1111	Pin15	ADC15

Input mode = DIFF:

MUX NEGATIVE INPUT	Group Configuration	Description
000	Pin0	ADC0
001	Pin1	
010	Pin2	
011	Pin3	ADC3
100	reserved	reserved
101	GND	
110	reserved	reserved
111	INTGND	inernal GND

Input mode = DIFFWGAIN:

MUX NEGATIVE INPUT	Group Configuration	Description
000	Pin4	ADC0
001	Pin5	
010	Pin6	
011	Pin7	ADC3
100	INTGND	internal GND
101	reserved	reserved
110	reserved	reserved
111	GND	GND

Example:
```
Ch1_gain = 1 , Ch1_inp = Diffwgain , Mux1 = &B0_0001_001
```

Positive Input = PIN1
Negative Input = PIN5

Calculation of ADC Value:

G = Gain
TOP with 12-bit resolution:

- TOP value of a signed result is 2047 and the results will be in the range -2048 to +2047 (0xF800 - 0x07FF). This is 11-bit plus sign bit (+ or -).
- TOP value of of an unsigned result is 4095 and the results will be in the range 0 to +4095 (0x0 - 0x0FFF). This is 12-bit.

For single ended and internal measurements GAIN is always 1 and Vinp is internal Ground.

In signed mode, negative and positive results are generated:
Vinp and Vinn = the positive and negative inputs to the ADC

ADC Resolution = ((Vinp - Vinn)/Vref) * G * (TOP + 1)

Example for signed differential input (with gain):

TOP = 2047
Vinp = +0.3V
Vinn = +1.4V
Vref = Vcc/1.6 = 3.3V/1.6 = 2.0625
G = 1

ADC Resolution = ((Vinp - Vinn)/Vref) * G * (TOP + 1)
ADC Resolution = ((0.3 - 1.4)/2.0625) * 1 * (2047 + 1)
ADC Resolution = - 1092

Example for unsigned single ended:
TOP = 4095
Vinp = +1.0V
Vref = 3.323Volt/1.6 = 2.076875
Δ V = Vref * 0.05 = 2.0625 * 0.05 = 103.1mV
G = 1

ADC Resolution = ((Vinp - (-Δ V))/Vref) * G * (TOP + 1)
ADC Resolution = ((1.0 + 0.103125)/2.076875) * 1 * (4095 + 1)
ADC Resolution = 2175

The offset needs to be subtracted to get the right value.
See also example below where the real ADC Resolution was output over terminal with the ATXMEGA256A3BU (Measure Offset in Single Ended Unsigned Mode).

ADC Compare function

Another feature of XMEGA ADC is a 12-bit compare function. The ADC compare register can hold a 12-bit value that represents a threshold voltage. Each ADC Channel can be configured to automatically compare its result with this compare value to give an interrupt or event only when the result is above or below the threshold. All four ADC Channels share the same compare register but you can decide which ADC channel is working in compare mode.

For ADC A you need to set register **ADCA_CMP** and configure the interrupt.

The used interrupt for this feature is the ADC conversion complete interrupt of the according channel which will (when configured in compare mode) only fire when the compare condition is met.

To configure the interrupt for example for ADC A Channel 0 the register **ADCA_CH0_INTCTRL** need to be set to:
- Compare Result Below Threshold
- Compare Result Above Threshold

instead of a conversion complete interrupt.

ADC Calibration:

The production signature row offers several bytes for ADC calibration. The ADC is calibrated during production testing, and the calibration value must be loaded from the signature row into the ADC registers (CAL registers).
Register **ADCA_CALL** = Low Byte of calibration value
Register **ADCA_CALH** = High Byte of calibration value
The calibration corrects the capacitor mismatch of the switched capacitor technology. This ADC calibration value copy should be done in a setup routine before using the ADC.
See also READSIG (reads a byte from the signature area in the XMEGA)

ADC Clock Frequency

The ADC clock need to be set within the recommended speed limits for the ADC module to guarantee correct operation.
For example for a ATXMEGA A4U device the minimum is 100Khz and the maximum is 2MHz (for internal signals like internal temp the max. value is 125KHz). The ADC clock is derived from a prescaled version of the XMEGA peripheral clock which is set with the Prescaler value paramter.

Don't confuse ADC Clock frequency with ADC conversion speed. So even if you set the ADC Clock frequency to 2MHz you can sample at a rate of for example 20KHz ! Because the maximum ADC Clock Frequency is 1/4 of the peripheral clock of an ATXMEGA you can not sample at a rate higher than one fourth of the system clock speed.


Take care on the source impedance of the analog signal source. If the source impedance is too high, the internal sampling
capacitor will not be charged to the correct level and the result will not be accurate. In Atmel application Note AVR1300 you find details regarding sample rate vs. source impedance of analog signal source.

Additional Best Practise
Some additional best practise to use ADC with XMEGA:

- Switch off unused peripheral parts with CONFIG POWER_REDUCTION 1023 to eliminate noise.
- Put the XMEGA in the "Idle" sleep mode directly after starting the ADC conversion to reduce noise from the CPU
- Use the lowest gain possible to avoid amplifying external noise
- Apply offset and gain calibration to the measurement

External Voltage Reference (REFA and REFB)
The internal reference voltages like INT1V is derived from the bandgap voltage. Parameter like gain error of bandgap voltage can be found in the device data sheet.

An external voltage reference can be more accurate compared to the internal voltage reference but is depending on the external circuit. The max. voltage for external ref on REFA pin (with ADC A this is PINA.0) is Vrefmax = Vcc - 0.6V so with Vcc=3.3V this is 2.7V. And external Vref must be at least 1V.

 The external reference pin AREFA or AREFB is shared with the DAC module !

See also Atmel Application Note AVR1012: XMEGA A Schematic Checklist

For example a reference diode (like LM336-2.5V) can be used or a shunt voltage reference like LM4040 as external reference.

For Maximum Performance use Event System and DMA Controller combined with ADC
See config DMA 936, config DMAchx 937, config Event_System 953

See also
GETADC 1265 , CONFIG ADC 864, ATXMEGA 425

Example for Single Conversion:
```
'-----------------------------------------------------------------------
---------
'setup the ADC-A converter
Config Adca = Single , Convmode = Unsigned , Resolution = 12bit , Dma =
Off , Reference = Int1v , Event_mode = None , Prescaler = 32 , Ch0_gain
= 1 , Ch0_inp = Single_ended , Mux0 = 0       'you can setup other
channels as well

W = Getadc(adca , 0)
```

Example for Free Running Mode:
```
'Configure ADC of Port A in FREE running mode
Config Adca = Free , Convmode = Signed , Resolution = 12bit , Dma = Off
, _
 Reference = Intvcc , Event_mode = None , Prescaler = 256 , Sweep =
Ch01 , _
```

```
 Ch0_gain = 1 , Ch0_inp = Diffwgain , Mux0 = &B00000000 , _
 Ch1_gain = 1 , Ch1_inp = Diffwgain , Mux1 = &B00001001

 ' With MuxX you can set the 4 MUX-Register
 ' ADCA_CH0_MUXCTRL   (for Channel 0)
 ' ADCA_CH1_MUXCTRL   (for Channel 1)
 ' ADCA_CH2_MUXCTRL   (for Channel 2)
 ' ADCA_CH3_MUXCTRL   (for Channel 3)

 ' Mux0 = &B00000000 means in Signed Mode:
 ' MUXPOS Bits = 000 --> Pin 0 is positive Input for Channel 0
 ' MUXNEG Bits = 00  --> Pin 4 is negative Input for Channel 0   (Pin 4
because of Differential with gain)

 ' Mux1 = &B00001001 means in Signed Mode:
 ' MUXPOS Bits = 001 --> Pin 1 is positive Input for Channel 1
 ' MUXNEG Bits = 01  --> Pin 5 is negative Input for Channel 1   (Pin 5
because of Differential with gain)
```

Measure Offset in Single Ended Unsigned Mode:

With this example we want to measure the offset in single ended unsigned mode and also the output of the internal 1.0 Voltage reference to DAC B PINB.2. Also the signature row with calibration byte is in the example.

1. With the used ATXMEGA256A3BU the voltage on DAC B was measured with an DMM and the value was: **1.014V**
2. After changing the gain calibration register of DAC B Ch0 to DACB_GAINCAL = 160 then the DAC B Ch0 analog output value was the expected **1.000V**
3. The offset in single ended unsigned mode is **208**
4. Now we connect the DAC B output (Pinb.2) to ADC B input (Pinb.0): the ADC resolution is **2180**
5. Vref = 3.323Volt/1.6 = 2.076875 (Vcc was also double checked by a DMM)
6. 2.076875/4095 = 507.1733822 µV
7. 2180* 507.1733822 µV = **1.1056379 V**
8. So here we see the difference of the DAC output **1.000V** to the measured value in single ended unsigned mode of **1.1056379 V** is **0.10564 V**
9. When we subtract now the offset from the measured result (2180 - 208 = 1972) we are getting closer to the DAC B output
10. 1972 * 507.1733822 µV = **1.0001V**

```
'(
Single ended input (unsigned mode)
In unsigned mode the negative input is connected to half of the voltage
reference (Vref) voltage minus a fixed device specific negative offset
The approximate value corresponding to ground is around 200. This value
corresponds to the digital result of ?V (0.05 * 4096). This value also
depend on the selected voltage reference so you should measure the real
value by first selecting the voltage reference.
   (?V = Vref * 0.05)

How to measure the offset ?
Connect the ADC input pin (Vinp) to GND and measure the offset.
This is also called offset calibration. This value can be stored for
example in EEPROM and is therefore available for all other measurements.

This offset calibration value is then subtracted to each ADC output
The offset enables the ADC to measure for example zero crossing in
unsigned mode.
')

$regfile = "XM256A3BUDEF.DAT"
```

```
$crystal = 32000000                                          '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80

Config Osc = Enabled , 32mhzosc = Enabled                    '32MHz
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Portr.0 = Output
Led0 Alias Portr.0                                           'LED 0
Config Portr.1 = Output
Led1 Alias Portr.1                                           'LED 1

Config Com5 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Open "COM5:" For Binary As #1

Dim B As Byte
dim j as byte

'First print the complete signature row
For J = 0 To 37
    b = Readsig(j) : Print #1, j ;" = " ; b
Next

'Read calibration bytes from Signature row
'ADCB
B = Readsig(24)                                              'ADCB
Calibration Byte 0
ADCB_CALL = b                                                 'write the
value to the register
Print #1 , "DCB Calibration Byte 0 = " ; B
B = Readsig(25)                                              'ADCB
Calibration Byte 1
ADCB_CALH = b
Print #1 , "DCB Calibration Byte 1 = " ; B
'DACB
B = Readsig(32)                                              'DACB
Calibration Byte 0 (DACBOFFCAL)
DACB_CH0OFFSETCAL = b                                         'write to
the DACB offset register
Print #1 , "DACB Calibration Byte 0 = " ; B
B = Readsig(33)                                              'DACB
Calibration Byte 1 (DACBGAINCAL)
DACB_GAINCAL = 160
Print #1 , "DACB Calibration Byte 1 = " ; B

'Configure the DAC output to output
Config Dacb = Enabled , Io0 = Enabled , Channel = Single , Reference =
Int1v , Interval = 64 , Refresh = 64

Dim W As Word
'-------------------------------------------------------------------------
---------
'setup the ADC-B converter (there is no DAC A on ATXMEGA256A3BU)
Config Adcb = Single , Convmode = Unsigned , Resolution = 12bit , Dma =
Off , Reference = Intvcc , Event_mode = None , Prescaler = 32 , _
 Ch0_gain = 1 , Ch0_inp = Single_ended , Mux0 = &B00000000  'you can
setup other channels as well

Dacb0 = 4095                                                 '1 V output
on portb.2
Do
 Wait 1
```

```
 'Connect PINB.0 with GND to measure the offset in unsigned mode
 W = Getadc(adcb , 0)                                        'Measure
PINA.0
 Print #1 , "W = " ; W
Loop

End                                                          'end program
```

Internal measure the DACB output with ADC B:

For this example you do not need a connection from DACB output to ADC B.
We use the internal DACB output and measure it with ADCB so the DACB must be configured to output also internal and the ADC B must be configured to measure from internal DAC.

Don't forget to subtract the offset from the measured value as we use unsigned mode.

```
$regfile = "XM256A3BUDEF.DAT"
$crystal = 32000000                                          '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80

Config Osc = Enabled , 32mhzosc = Enabled                    '32MHz
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Portr.0 = Output
Led0 Alias Portr.0                                           'LED 0
Config Portr.1 = Output
Led1 Alias Portr.1                                           'LED 1

Config Com5 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Open "COM5:" For Binary As #1

Dim B As Byte
dim j as byte

'First print the complete signature row
For J = 0 To 37
    b = Readsig(j) : Print #1, j ;" = " ; b
Next

'Read calibration bytes from Signature row
'ADCB
B = Readsig(24)                                              'ADCB
Calibration Byte 0
ADCB_CALL = b                                                'write the
value to the register
Print #1 , "DCB Calibration Byte 0 = " ; B
B = Readsig(25)                                              'ADCB
Calibration Byte 1
ADCB_CALH = b
Print #1 , "DCB Calibration Byte 1 = " ; B
'DACB
B = Readsig(32)                                              'DACB
Calibration Byte 0 (DACBOFFCAL)
DACB_CH0OFFSETCAL = b                                        'write to
the DACB offset register
Print #1 , "DACB Calibration Byte 0 = " ; B
```

```
B = Readsig(33)                                             'DACB
Calibration Byte 1 (DACBGAINCAL)
DACB_GAINCAL = b
Print #1 , "DACB Calibration Byte 1 = " ; B

'Configure the DAC output to output
Config Dacb = Enabled , Io0 = Enabled , Channel = Single ,
INTERNAL_OUTPUT = enabled, Reference = Int1v , Interval = 64 , Refresh =
64

Dim W As Word
'-----------------------------------------------------------------------
---------
'setup the ADC-B converter (there is no DAC A on ATXMEGA256A3BU)
'For internal Measurements use Unsigned mode, 12 bit, Internal 1.00 V
Reference
Config Adcb = Single , Convmode = Unsigned , Resolution = 12bit , Dma =
Off , Reference = Intvcc , Event_mode = None , Prescaler = 512 , _
 Ch0_gain = 1 , Ch0_inp = INTERNAL , Mux0 = &B0_0011_000  'configure
MUX0 to measure internal DAC

Dacb0 = 4095                                                '1 V

Do
 Wait 1
 W = Getadc(adcb , 0 , &B0_0011_000)                        'Measure DAC
 Print #1 , "W = " ; W
Loop

End                                                         'end program
```

7.21.8 CONFIG ADC0-ADCX

Action

Configures the A/D converter of the Xtiny

Syntax

CONFIG ADC0 | ADCx = mode, RUNMODE=runmode, RESOLUTION=res, ADC=adc, SAMPLE_ACCU=samp_acc, SAMPLE_CAP=samp_cap, SAMPLE_DELAY=samp_dly, SAMPLE_LEN=samp_len,
REFERENCE=ref,PRESCALER=pre, INIT_DELAY=init_dly,ASDV=asdv, WINDOW_COMP=win_cmp, MUX=mux

Remarks

mode	AD converter mode. - SINGLE (default mode for a single conversion) - FREE. In FREE mode a new conversion cycle is started immediately after a previous conversion has completed.
runmode	Possible values: ENABLED : In Standby sleep mode, the peripheral continues operation DISABLED : In Standby sleep mode, the peripheral is halted
res	The resolution of the conversion. Valid values are : - **8BIT** - **10BIT**. This is the default
adc	ENABLED or DISABLED. By default the AD converter is DISABLED.

samp_acc	This value selects how many consecutive ADC sampling results are accumulated automatically. Possible values : - 0 : (accumulation disabled, default value) - 2, 4,8,16,32,64 : number of accumulated samples.
samp_cap	Sample capacitance selection. Possible values : - BELOW_1V : Recommended for reference voltage values below 1V. - ABOVE_1V : Reduced size of sampling capacitance. Recommended for higher reference voltages.
samp_dly	Sampling Delay Selection : Numeric constant between 0 and 15. These bits define the delay between consecutive ADC samples. The programmable Sampling Delay allows modifying the sampling frequency during hardware accumulation, to suppress periodic noise sources that may otherwise disturb the sampling. The SAMPDLY field can be also modified automatically from sampling cycle to another, by setting the ASDV bit. The delay is expressed as CLK_ADC cycles and is given directly by the bitfield setting. The sampling cap is kept open during the delay.
samp_len	Sample Length. Numeric constant between 0 and 31. These bits extend the ADC sampling length in number of CLK_ADC cycles. By default the sampling time is two CLK_ADC cycles. Increasing the sampling length allows sampling sources with higher impedance. The total conversion time increased with the selected sampling length.
ref	Voltage Reference selection. Possible values : - INTERNAL : internal reference. See CONFIG VREF - VDD : VDD
prescale	Prescaler selection. This is the division from the peripheral clock to the ADC clock. Possible values : 2,4,8,16,32,64,128,256
init_dly	Initialization delay. These bits defines the initialization/startup delay before the first sample when enabling the ADC or changing to internal reference voltage. Setting this delay will ensure that the reference, muxes, etc are ready before starting the first conversion. The initialization delay will also take place when waking up from deep sleep to do a measurement. The delay is expressed as a number of CLK_ADC cycles. Possible values : - 0 : Delay 0 CLK_CYCLES (no delay) - 16 : Delay 16 CLK_CYCLES - 32 : Delay 32 CLK_CYCLES - 64 : Delay 64 CLK_CYCLES - 128 : Delay 128 CLK_CYCLES - 256 : Delay 256 CLK_CYCLES
asdv	Automatic Sampling Delay Variation. ENABLED or DISABLED. Selecting ENABLED, enables automatic sampling delay variation between ADC conversions. The purpose of varying sampling instant is to randomize the sampling instant and thus avoid standing frequency components in frequency spectrum. The value of the SAMPDLY bits is automatically incremented by one after each sample. When the Automatic Sampling Delay Variation is enabled and the SAMPDLY value reaches &HF, it wraps around to 0.
win_cmp	Window Comparator Mode. This field enables and defines when the interrupt flag is set in Window

	Comparator mode. RESULT is the 16-bit accumulator result. WINLT and WINHT are 16-bit lower threshold value and 16-bit higher threshold value, respectively. Possible values : - NONE : No windows comparison (default) - BELOW : result < WINLT - ABOVE : result > WINHT - INSIDE : WINLT < result < WINHT - OUTSIDE : result < WINLT or result > WINHT
mux	Mux position. This bit field selects which single-ended analog input is connected to the ADC. If these bits are changed during a conversion, the change will not take effect until this conversion is complete. Possible values : - GND : 0V, GND - TEMPSENSE : Temperature sensor - INTREF : Internal reference (from VREF) - DAC0 : DAC0 output 0-11 : ADC input pin 0-11

The MUX value is an optional initial value.This value writes to the ADC0_MUXPOS register. The GETADC() function will also write to this register.

See Also

CONFIG VREF 1147 , GETADC 1265

Example

```
'-----------------------------------------------------------------
----------------
'name                     : adc.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates ADC and DAC. Notice that
DAC is not available on all processors
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24

'set the system clock and prescaler
Config Sysclock = 20mhz , Prescale = 1

'configure the USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1
```

```
'configre the internal reference to be 1v1 for both the ADC and
the DAC
Config Vref = Dummy , Adc0 = 1v1 , Dac0 = 1v1

'configure the ADC0 to read the DAC
Config Adc0 = Single , Resolution = 10bit , Adc = Enabled ,
Reference = Internal , Prescaler = 32 , Sample_len = 1 ,
Sample_cap = Above_1v , Init_delay = 32 , Mux = Dac0

'configure the DAC. We do not output the signal on a port pin
otherwise out_enable would be required too
Config Dac0 = Enabled

'dimension a variable
Dim W As Word

Print "Test ADC"

'set the DAC to halve the output which would be halve of 1.1V
which is 0.55V
Dac0_data = 127

Do
  'when getadc() does not have parameters, it will use the
current mux setting
  'other options are : getadc(channel)  and getadc(adc0 | adc1 ,
channel)
   W = Getadc() : Print "W:" ; W
   'output should be 512
   Waitms 1000
Loop

End
```

7.21.9 CONFIG ATEMU

Action

Configures the PS/2 keyboard data and clock pins.

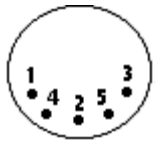
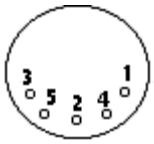
Syntax

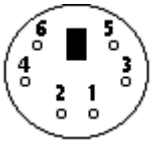
CONFIG ATEMU = int , DATA = data, CLOCK=clock [,INIT=VALUE]

Remarks

Int	The interrupt used such as INT0 or INT1.
DATA	The pin that is connected to the DATA line. This must be the same pin as the used interrupt.
CLOCK	The pin that is connected to the CLOCK line.
INIT	An optional value that will identify the keyboard. By default or when

	omitted this is &HAB83. The code that identifies a keyboard. Some mother boards/BIOS seems to require the reverse &H83AB. By making it an option you can pass any possible value. The MSB is passed first, the LSB last.

Male	Female	5-pin DIN (AT/ XT):
 (Plug)	 (Socket)	1 - Clock 2 - Data 3 - Not Implemented 4 - Ground 5 - +5v

Male	Female	6-pin Mini-DIN (PS/2):
 (Plug)	 (Socket)	1 - Data 2 - Not Implemented 3 - Ground 4 - +5v 5 - Clock 6 - Not Implemented

Old PC's are equipped with a 5-pin DIN female connector. Newer PC's have a 6-pin mini DIN female connector.
The male sockets must be used for the connection with the micro.

Besides the DATA and CLOCK you need to connect from the PC to the micro, you need to connect ground. You can use the +5V from the PC to power your microprocessor.

The config statement will setup an ISR that is triggered when the INT pin goes low. This routine you can find in the library.
The ISR will retrieve a byte from the PC and will send the proper commands back to the PC.

The SENDSCANKBD statement allows you to send keyboard commands.

Note that unlike the mouse emulator, the keyboard emulator is also recognized after your PC has booted.

 The PS2 Keyboard and mouse emulator needs an additional commercial addon library.

See also

SENDSCANKBD 1443

Example

```
'-------------------------------------------------------------------------
-----------------
'name                    : ps2_kbdemul.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : PS2 AT Keyboard emulator
'micro                   : 90S2313
'suited for demo         : no, ADD ONE NEEDED
'commercial addon needed : yes
'-------------------------------------------------------------------------
-----------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

$lib "mcsbyteint.lbx"                                       ' use
optional lib since we use only bytes

'configure PS2 AT pins
Enable Interrupts                                           ' you need
to turn on interrupts yourself since an INT is used
Config Atemu = Int1 , Data = Pind.3 , Clock = Pinb.0
'                  ^------------------------ used interrupt
'                               ^----------- pin connected to DATA
'                                         ^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin


Waitms 500                                                  ' optional
delay

'rcall _AT_KBD_INIT
Print "Press t for test, and set focus to the editor window"
Dim Key2 As Byte , Key As Byte
Do
    Key2 = Waitkey()                                        ' get key
from terminal
    Select Case Key2
      Case "t" :
      Waitms 1500
      Sendscankbd Mark                                      ' send a
scan code
      Case Else
    End Select
Loop
Print Hex(key)

Mark:                                                       ' send mark
Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 , &H2D ,
```

```
 &H42 , &HF0 , &H42
'     ^ send 12 bytes
'           m                   a                   r
 k
```

7.21.10 CONFIG BASE

Action
This option specifies the lower boundary of all arrays.

Syntax
CONFIG BASE= value

Remarks
By default the first element of an array starts at 1. With CONFIG BASE=0 you can override this default so that all arrays start at 0.
In some cases it is simpler that elements start at 0.
A constant named _BASE reflects the setting. You can not change the BASE at run time.

 When you change this setting in existing code, you need to alter your code. For example when you used this code:
Dim a(10) as byte : a(10) = 10
And you set CONFIG BASE=0, it will mean that element 10 is invalid.

While in QB an additional element is created, this is not a good idea in bascom because it will require more space.

See also
DIM 1228

Example
```
CONFIG BASE=0
Dim ar(10) as byte , j as byte
For j=0 to 9 'array uses element 0-9
   ar(j)=j
Next
```

Example
```
CONFIG BASE=1
Dim ar(10) as byte , j as byte
For j=1 to 10  ' arrays uses element 1-10
  ar(j)=j
Next
```

7.21.11 CONFIG BCCARD

Action
Initializes the pins that are connected to the BasicCard.

Syntax
CONFIG BCCARD = port , IO=pin, RESET=pin

Remarks

Port	The PORT of the micro that is connected to the BasicCard. This can be PORTB or PORTD and will depend on the used micro.
IO	The pin number that is connected to the IO of the BasicCard. Must be in the range from 0-7
RESET	The pin number that is connected to the RESET of the BasicCard. Must be in the range from 0-7

The variables SW1, SW2 and _BC_PCB are automatically dimensioned by the CONFIG BCCARD statement.

 This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

See Also
BCRESET 1844 , BCDEF 1837 , BCCALL 1838

Example
```
'-----------------------------------------------------------------------
-------
'                          BCCARD.BAS
' This AN shows how to use the BasicCard from Zeitcontrol
'                          www.basiccard.com
'-----------------------------------------------------------------------
-------
'connections:
' C1 = +5V
' C2 = PORTD.4 - RESET
' C3 = PIN 4   - CLOCK
' C5 = GND
' C7 = PORTD.5 - I/O

'  /--------------------------------\
'  |                                |
'  |     C1  C5                     |
'  |     C2  C6                     |
'  |     C3  C7                     |
'  |     C4  C8                     |
'  |                                |
'  \--------------------------------/
'
'


'----------- configure the pins we use ------------
Config Bccard = PORTD , Io = 5 , Reset = 4
```

```
'                                    ^ PORTD.4
'                          ^------------ PORTD.5
'                ^---------------------- PORT D

'Load the sample calc.bas into the basiccard


' Now define the procedure in BASCOM
' We pass a string and also receive a string
Bcdef Calc(string)

'We need to dim the following variables
'SW1 and SW2 are returned by the BasicCard
'BC_PCB must be set to 0 before you start a session

'Our program uses a string to pass the data so DIM it
Dim S As String * 15
'Baudrate might be changed
$baud = 9600
' Crystal used must be 3579545 since it is connected to the Card too
$crystal = 3579545

'Perform an ATR
Bcreset

'Now we call the procedure in the BasicCard
'bccall funcname(nad,cla,ins,p1,p2,PRM as TYPE,PRM as TYPE)
S = "1+1+3"                                                 ' we want to
calculate the result of this expression

Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
'                                   ^--- variable to pass that holds the
expression
'                               ^------- P2
'                           ^----------- P1
'                       ^--------------- INS
'                   ^------------------- CLA
'           ^--------------------------- NAD
'For info about NAD, CLA, INS, P1 and P2 see your BasicCard manual
'if an error occurs ERR is set
' The BCCALL returns also the variables SW1 and SW2
Print "Result of calc : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb)  ' for test you can see that it toggles between 0
and 40
Print "Error : " ; Err

'You can call this or another function again in this session

S = "2+2"
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Result of calc : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb)  ' for test you can see that it toggles between 0
and 40
Print "Error : " ; Err

'perform another ATR
Bcreset
Input "expression " , S
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Answer : " ; S
```

```
'----and now perform an ATR as a function
Dim Buf(25) As Byte , I As Byte
Buf(1) = Bcreset()
For I = 1 To 25
  Print I ; "  " ; Hex(buf(i))
Next
'typical returns :
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81  T=1 indication
'TD2 = 31  TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20  IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45  BWT blocl waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
'         B  a  s  i  c  C  a  r  d     Z  C   1  2  3

'and another test
'define the procedure in the BasicCard program
Bcdef Paramtest(byte , Word , Long )

'dim some variables
Dim B As Byte , W As Word , L As Long

'assign the variables
B = 1 : W = &H1234 : L = &H12345678

Bccall Paramtest(0 , &HF6 , 1 , 0 , 0 , B , W , L)
Print Hex(sw1) ; Spc(3) ; Hex(sw2)
'and see that the variables are changed by the BasicCard !
Print B ; Spc(3) ; Hex(w) ; "  " ; Hex(l)

'try the echotest command
Bcdef Echotest(byte)
Bccall Echotest(0 , &HC0 , &H14 , 1 , 0 , B)
Print B
End                                                    'end program
```

7.21.12 CONFIG CANBUSMODE

Action

Configures the CAN bus mode.

Syntax

CONFIG CANBUSMODE =mode

Remarks

mode	The CAN bus can be set to 3 different modes. - ENABLED : TxCAN and RxCAN are enabled. - STANDBY : TxCAN is recessive and the receiver is disabled. The registers and mobs can be accessed. - LISTENING : This mode is transparant for the CAN channel. It enables a hardware loop[back from the internal TxCAN to the RxCAN. It provides a recessive level on the TxCAN output pin. It does NOT disable the RxCAN pin.

The CAN commands are intended for the AVR processor AT90CANXXX series.
You need to terminate the bus with 120 ohm at both ends.

Your code always need a number of statements. The best solution is to use the can-elektor.bas sample to get started.

CANRESET
Will reset the CAN controller. Use this only once.

CANCLEARALLMOBS
Will clear all message objects. This is best to be done right after the CANRESET.

CANBAUD
All devices on the bus need to have the same baud rate. Set the BAUD right after you have cleared all objects.

CONFIG CANBUSMODE
Now you chose the mode the bus will work in. This is ENABLED in most cases.

CONFIG CANMOB
Here you define the properties of each Message Object. This need to be done only once. But after the message object has been used, you need to configure it again so the new MOB can be used again.

CANGIE , ON CAN_IT
Since the interrupt TX, RX and ERR interrupts are used you need to assign a value of &B10111000 to CANGIE.
You also need to assign an interrupt routine to the CANIT interrupt.

In the main code you can send data using CANSEND.

The interrupt routine.
The **CANPAGE** register is saved into the **_CAN_PAGE** variable. This is required since the interrupt may not change the **CANPAGE** register.
Then **CANGETINTS** is used to retreive all message object interrupt flags. The value is stored in **_CAN_MOBINTS.**

Since multiple Message Objects can cause an interrupt we check all message objects with a For.. Next loop to test all bits. If the bit is set, the Message Object is selected with **CANSELPAGE**.
Then the **CANSTMOB** register is tested for a number of bits/flags.
If bit 5 is set, it means that a frame was received. For the demo the ID is read with **CANID**.
The **CANRECEIVE** function reads the data from the frame into a variable. In the example the variable is a PORT which will change value depending on the receive data byte.
After this the **CONFIG CANMOB** is used with a value of -1 to indicate that the operation must be done on the current selected MOB.
The object is put back into receive mode.

If bit 6 is set it means that data was transmitted with success. Again, we use **CONFIG CANMOB** so the object can be used again. For transmitting we put the object into DISABLED mode.

And lastly we test bit 0, the **MOB** error bit. It if was set it means there was an error when data was sent using **CANSEND**. We must use **CONFIG CANMOB** so the **MOB** can be used again.

We must clear the **CANSIT1** and **CANSIT2** flag registers before we exit the interrupt routine. We also need to reset the interrupt flags in **CANGIT**. This is done by writing the same value back to **CANGIT**. A one will clear the flag if it was set.
Last we restore the **CANPAGE** register by writing **_CAN_PAGE** back to it.

While the interrupt routine shows some PRINT statements, it is not a good idea to print inside the/a interrupt routine. You should keep the delay as short as possible otherwise you might not be able to process all CAN frames.

As you can see in the sample, the MOB's are configured at the start AND once they are used so they can be re-used.
In the example all lines are important except for the PRINT lines.

See also

CONFIG CANMOB 893, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843, CANID 842, CANSELPAGE 844, CANGETINTS 841

Example

```
'-----------------------------------------------------------------------
-
'                       CAN-Elektor.bas
'    bascom-avr demo for Auto-CANtroller board
'-----------------------------------------------------------------------
-
$regfile = "m32can.dat"                                     ' processor
we use

$crystal = 12000000                                         ' Crystal 12
MHz
$hwstack = 64
$swstack = 32
$framesize = 40

'$prog &HFF , &HCF , &HD9 , &HFF                             '
generated. Take care that the chip supports all fuse bytes.
Config Porta = Output                                       ' LED
Config Portc = Input                                        ' DIP switch
Portc = 255                                                 ' activate
pull up

Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Open "COM2:" For Binary As #2

Dim _canpage As Byte , _canid As Dword , _can_int_idx As Byte ,
_can_mobints As Word
Dim Breceived As Byte , Bok As Byte , Bdil As Byte

On Can_int Can_int                                          ' define the
CAN interrupt
Enable Interrupts                                           ' enable
interrupts

Canreset                                                    ' reset can
controller
Canclearallmobs                                             ' clear alle
message objects
```

```
Canbaud = 125000                                            ' use 125 KB

Config Canbusmode = Enabled                                 '
enabled,standby,listening
Config Canmob = 0 , Bitlen = 11 , Idtag = &H0120 , Idmask = &H0120 ,
Msgobject = Receive , Msglen = 1 , Autoreply = Disabled       'first mob
is used for receiving data
Config Canmob = 1 , Bitlen = 11 , Idtag = &H0120 , Msgobject = Disabled
, Msglen = 1        ' this mob is used for sending data

Cangie = &B10111000                                         ' CAN
GENERAL INTERRUPT and TX and RX and ERR
Print #2 , "Start"

Do
  If Pinc <> Bdil Then                                      ' if the
switch changed
     Bdil = Pinc                                            ' save the
value
     Bok = Cansend(1 , Pinc)                                ' send one
byte using MOB 1
     Print #2 , "OK:" ; Bok                                 ' should be
0 if it was send OK
  End If
Loop

'********************** CAN CONTROLLER INTERRUPT ROUTINE
**********************
'multiple objects can generate an interrupt
Can_int:
_canpage = Canpage                                          ' save can
page because the main program can access the page too
  Cangetints                                                ' read all
the interrupts into variable _can_mobints

  For _can_int_idx = 0 To 14                                ' for all
message objects
     If _can_mobints._can_int_idx = 1 Then                  ' if this
message caused an interrupt

        Canselpage _can_int_idx                             ' select
message object

        If Canstmob.5 = 1 Then                              ' we
received a frame
           _canid = Canid()                                 ' read the
identifier
           Print #2 , Hex(_canid)

           Breceived = Canreceive(porta)                    ' read the
data and store in PORTA
           Print #2 , "Got : " ; Breceived ; " bytes"       ' show what
we received
           Print #2 , Hex(porta)
           Config Canmob = -1 , Bitlen = 11 , Msgobject = Receive ,
Msglen = 1 , Autoreply = Disabled , Clearmob = No
           ' reconfig with value -1 for the current MOB and do not set
ID and MASK
        Elseif Canstmob.6 = 1 Then
'transmission ready
           Config Canmob = -1 , Bitlen = 11 , Msgobject = Disabled ,
Msglen = 1 , Clearmob = No
           ' reconfig with value -1 for the current MOB and do not set
```

```
ID and MASK
        Elseif Canstmob.0 = 1 Then                          'ack error
when sending data                          'transmission ready
           Print #2 , "ERROR:" ; Hex(canstmob)
           Config Canmob = -1 , Bitlen = 11 , Msgobject = Disabled ,
Msglen = 1 , Clearmob = No
        End If
   End If
 Next
 Cangit = Cangit                ' clear interrupt flags
 Canpage = _canpage                                          ' restore
page
Return
```

7.21.13 CONFIG CANMOB

Action

Configures one of the 15 **CAN M**essage **OB**jects.

Syntax

CONFIG CANMOB=mob,BITLEN=bitlen,IDTAG=tag,IDMASK=mask,
MSGOBJECT=mode,MSGLEN=msglen,AUTOREPLY=reply , CLEARMOB=clrmob

Remarks

mob	The mob(message object) is a number or variable with a range from 0-14. Number 15 is reserved by Atmel. There are 15 message objects you can use but only one set of registers. The CANPAGE register is used to select the proper MOB. This is all handled by the compiler. Internally, the mob you pass will set the CANPAGE register. When you use a value of -1 , the configuration is done on the current selected MOB (or CANPAGE). A reconfigure does not need to set the IDTAG and IDMASK again. While you can use a constant or variable, you can not use a variable with a value of -1 to reconfigure the mob. A reconfigure requires a constant of -1.
bitlen	The CAN controller supports CAN messages with 11 bit ID's and with 29 bit ID's. And ID is an identifier. The lowest ID has the highest priority. Using 11 bit ID's has the advantage that it takes less time and as a result, you could send more messages. Just like with traffic, the bus capacity is limited. The baud rate and the message length all play a role. Valid values are 11 and 29. You can use a constant or variable. Using variables will increase code.
idtag	The IDTAG is the identifier you assign to the message object. When the MOB is used for transmitting, the IDTAG is used for the CAN ID. When the MOB is used for receiving, the IDTAG is used as a filter. Each time a message is sent or received, an interrupt is generated. This will interrupt the main process. For efficient usage, you need to set the IDTAG to filter only the ID's of interest. The IDMASK can be used together with the IDTAG to create a range. You can use a constant or variable to define the IDTAG. Using a variable

	will increase code.
mask	The IDMASK is only used when the MOB is used in receiving mode. It must be used together with IDTAG to create a range where the MOB will respond to. The following examples are for CAN rev A with 11 bit ID's. Example 1: you only want to filter ID &H0317. In this case you set the IDTAG to &H317. The IDMASK need to be set to &HFFFF in this case. A '1' for a bit in IDMASK means that the corresponding '1' in IDTAG is checked. When set a bit in IDMASK to '0' it means the corresponding bit in IDTAG can have any value. **Full filtering**: to accept only ID = 0x317 in part A. - ID MSK = 111 1111 1111 b - ID TAG = 011 0001 0111 b Example 2: you want to filter ID &H310-&H317. You can set the IDTAG to &H310 and the IDMASK to &HFFF8. The last 3 bits are set to 0 this way which means that &H310 is valid, but so is &H311, &H312, etc. **Partial filtering**: to accept ID from 0x310 up to 0x317 in part A. - ID MSK = 111 1111 1000 b - ID TAG = 011 0001 0xxx b Example 3: you want to filter from &H0000 to &H7FF. This means you need to respond to all messages. The IDMASK need to be set to 0. It will not matter to which value you set IDTAG since all 11 bits of IDMASK are set to 0. **No filtering**: to accept all ID from 0x000 up to 0x7FF in part A. - ID MSK = 000 0000 0000 b - ID TAG = xxx xxxx xxxx b You can use a constant or variable to define the IDMASK. Using a variable will increase code.
mode	The mode in which the MOB will be used. - DISABLED (0). The MOB is free to be used. - TRANSMIT (1). The MOB data will be transmitted. - RECEIVE (2). The MOB will wait for a message that matches the ID and MASK. - RECEIVE_BUFFERED (3). This mode can be used to receive multiple frames. The CANSEND function will use the TRANSMIT mode. You should chose the DISABLED mode when configuring the MOB for transmission. Instead of the mentioned parameter names, you can also use a variable to set the mode. This variable must have a value between 0 and 3.
msglen	This is the message length of the message in bytes. In receive mode you set it to the number of bytes you expect. The CANRECEIVE function will return the number of bytes read. When the MOB is used for transmitting, it will define the length of the data. The length can also be 0 to send frames without data. The msglen can be a constant or variable. The maximum number of bytes that can be sent or received is 8.
reply	This option can set ENABLED or DISABLED. If you use a variable, a 0 will disable auto reply, a 1 will enable auto reply.

	Auto reply can be used to reply to a remote frame. A remote frame is a frame without data. Since a remote frame has no data, you can reuse the MOB to send data as a reply to a remote frame.
clrmob	By default all registers of a MOB are cleared when you configure the MOB. When you reconfigure the MOB, or want to respond to an auto reply, you do not want to clear the MOB. In such a case you can use CLEARMOB=NO to prevent clearing of the registers.

While CONFIG CANMOB can dynamically set up the MOB (using variables instead of constants), it will increase code. So use a constant if possible.

See also

CONFIG CANBUSMODE 889, CANBAUD 840, CANRESET 844, CANCLEARMOB 843, CANCLEARALLMOBS 842, CANSEND 845, CANRECEIVE 843, CANID 842, CANSELPAGE 844, CANGETINTS 841

Example

```
Config Canmob = 0 , Bitlen = 11 , Idtag = &H0120 , Idmask = &H0120 ,
Msgobject = Receive , Msglen = 1 , Autoreply = Disabled       'first mob
is used for receiving data
Config Canmob = 1 , Bitlen = 11 , Idtag = &H0120 , Msgobject = Disabled
, Msglen = 1  ' this mob is used for sending data

Config Canmob = -1 , Bitlen = 11 , Msgobject = Disabled , Msglen = 1 ,
Clearmob = No ' reconfig with value -1 for the current MOB and do not
set ID and MASK
```

7.21.14 CONFIG CLOCK

Action

Configures the timer to be used for the **Time**$ and **Date**$ variables.

Syntax

CONFIG CLOCK = SOFT | USER [, GOSUB = SECTIC]

Syntax Xmega

CONFIG CLOCK = SOFT | USER [, GOSUB = SECTIC] [,RTC=rtc] [,RTC32=rtc32] [, HIGHESR=highesr]

Syntax Xtiny

CONFIG CLOCK = SOFT | USER [, GOSUB = SECTIC] [,RTC=rtc] , RUNMODE=runmode

Remarks

Soft	Use SOFT for using the software based clock routines. You need to add an ENABLE INTERRUPTS statement to your code

	since the SOFT mode uses the timer in interrupt mode. The timer interrupt is enabled automatic but the global interrupt you need to enable yourself. While the compiler could enable the global interrupt automatic, you would not have control anymore when it is enabled when using multiple interrupts. In general you enable global interrupts after all interrupts are setup. For the SOFT mode you need to connect a special low frequency crystal with a value of 32768 Hz to the ASYNC TIMER oscillator pins. Use USER to write/use your own code in combination with an I2C clock chip for example.
Sectic	This option allows to jump to a user routine with the label sectic. Since the interrupt occurs every second you may handle various tasks in the sectic label. It is important that you use the name SECTIC and that you return with a RETURN statement from this label. The usage of the optional SECTIC routine will use 30 bytes of the hardware stack. This option only works with the SOFT clock mode. It does not work in USER mode. [, GOSUB = SECTIC] is only for SOFT mode.
RTC **XMEGA**	This option is only available for processors with an RTC (XMEGA). This option sets the RTC clock source. Valid parameters are : 1KHZ_INT32KHZ_ULP 1 kHz from internal 32 kHz ULP 1KHZ_32KHZ_CRYSTOSC 1 kHz from 32 kHz Crystal Oscillator on TOSC 1KHZ_INT32KHZ_RCOSC 1 kHz from internal 32 kHz RC Oscillator 32KHZ_32KHZ_CRYSTOSC 32 kHz from 32 kHz Crystal Oscillator on TOSC The 1KHz clocks will load the PER register with 1000-1 and the 32 KHz clock will load PER with a value of 32768-1. The overflow mode is used and you can use the compare overflow if required. Do not forget to enable the 32 KHz oscillator and the interrupts as shown in the Xmega example.
RTC **XTINY**	This option is only available for processors with an RTC (XTINY). This option sets the RTC clock source. Valid parameters are : 32KHZ_32KHZ_INTOSC : 32 KHz from OSCULP32K 1KHZ_INT32KHZ_ULP : 1 KHz from OSCULP32K 32KHZ_32KHZ_CRYSTOSC : 32 KHz from XOSC32K EXT_OSC_TOSC1 : External clock from TOSC1 pin. When configuring the RTC to use either XOSC32K or the external clock on TOSC1, XOSC32K needs to be enabled and the Source Select bit (SEL) and Run Standby bit (RUNSTDBY) in the XOSC32K Control A register of the Clock Controller (CLKCTRL.XOSC32KCTRLA) must be configured accordingly.
RTC32	This option is available for few XMEGA chips. You can use it instead

	of the RTC. In fact when a processor has an RTC32, it does not have an RTC. You can not use both RTC and RTC32 together. RTC32 only accepts one value : 1KHZ_32KHZ_CRYSTOSC This also means that you must use/connect an external 32 KHz crystal. When you use the RTC32, the battery back register VBAT_CTRL is initialized and setup.
HIGHESR	This option is available for few XMEGA chips which have RTC32 hardware. This option will set HIGH ESR mode when a value of '1' is selected. By default this option is 0/off. HIGH ESR consumes more power.
runmode	This only applies to the XTINY. Possible values : ENABLED : In Standby sleep mode, the peripheral continues operation DISABLED : In Standby sleep mode, the peripheral is halted

When you use the CONFIG CLOCK (in soft or user mode) directive the compiler will **DIM the following BYTE variables automatic** :
_sec
_min
_hour
_day
_month
_year

 The DATETIME library will also be included by the compiler. For this reason it is important that you use CONFIG CLOCK when you use any of the date time functions.

The variables **Time**$ and **Date**$ will also be dimensioned. These are special variables since they are treated different. See TIME$ 1208 and DATE$ 1191.
Following a way to set **Time**$ and **Date**$:

```
Date$ = "11/11/00"
Time$ = "02:20:00"
```

You can change the date format by using: `Config Date = Mdy , Separator = "/"`

```
' ANSI-Format
```

See CONFIG DATE 925

The **_sec, _min** and other internal variables can be changed by the user too.
But of course changing their values will change the **Time**$ and **Date**$ variables.

The compiler also creates an ISR that gets updated once a second. This works for AVR chips which can be asynchronously clocked from the TOSC1/2 pins.
TOSC1 = **T**imer **Osc**illator Pin 1
TOSC2 = **T**imer **Osc**illator Pin 2

For example the Timer/Counter 2 of an ATMEGA16 can be used as a Real Time Counter (RTC). The Timer/Counter 2 will then be asynchronously clocked from the TOSC Pin's. The Timer/Counter 2 can NOT be used for other tasks when configured in asynchronous mode.

 Notice that you need to connect a 32768 Hz crystal in order to use the timer in async mode, the mode that is used for the clock timer in SOFT mode. You also need to enable interrupts because of the interrupt service routine.

When you choose the **USER** option, only the internal variables are created (like _sec , _min , _hour....).
With the USER option you need to write the clock code yourself (so the USER need to update for example the System Second or Secofday).
This means the one second clock must be generated by a "USER" source like a Timer which use the internal clock or an XTAL depending on the Xtal configuration.
There are so called "AVR Timer Calculator" online available where you input the clock frequency from xtal, which Timer you use (8 or 16 Bit) and the period you want to achieve (like 1 second or 1000ms) than it will give you number which you need to configure the timer.
You also configure the interrupt of the timer and then the program will jump to the timer interrupt routine where you can set the new system second.

```
Config  Clock = User                                    'Use  USER  to  write/use  your
own  code
```

You also need to include the following labels with config clock = user:

```
Getdatetime:
 'called when date or time is read
Return

Setdate:
'called when date$ is set
Return

Settime:
'scanned when time$  is set
Return
```

Example for config clock = user in Bascom-Simulator

Following example use **$sim** so it can be used in Bascom-Simulator. It uses config clock in user mode.
The second tick is generated by Timer1 and the time updated in the Timer interrupt service routine.

You can run this example direct in Bascom Simulator and you need to CLICK ON RUN BUTTON (in the simulator) go to **Interrupts Tab** and hit the **OVF1 BUTTON** to simulate an Timer interrupt.

Then you will see how the program jump to the interrupt service routine and updates the time !!

The Simulator output give you following:

```
01.09.09
00:00:01
00:00:02
00:00:03
00:00:04
00:00:05
00:00:06
00:00:07
```

That's it !

```
$regfile = "m16def.dat"
$crystal = 12000000
$hwstack = 80
$swstack = 80
$framesize = 80
$baud = 19200
$sim                                                    'ONLY FOR
SIMULATOR MODE !!!!

Dim second_tick As Long

Config Clock = User                                     'Use USER to
write/use your own code
Config Date = Dmy , Separator = .
'Day.Month.Year
Config Timer1 = Timer , Prescale = 256
On Timer1 Timer_irq
Const Timer_preload = 18661
'Timervorgabe für Sekunden Takt

Enable Timer1
Enable Interrupts

Date$ = "01.09.09"
Time$ = "00:00:00"

Print Date$

Do
      !NOP
Loop

End                     'end program

Timer_irq:                                              'Timer1 IRQ
(once per second)
  Incr Second_tick
  Time$ = Time(second_tick)
  Timer1 = Timer_preload

  Print Time$                                           'only for
Bascom-Simulator
Return

Settime:
Return

Getdatetime:
Return

Setdate:
Return
```

Using a DS1307 with config clock

See the **datetime_test1.bas** example from the SAMPLES\DATETIME folder that shows how you can use a DS1307 clock chip for the date and time generation.
See also example below !

Using config clock with ATXMEGA

With ATXMEGA there are devices with 16-Bit RTC like ATXMEGA128A1 and 32-Bit RTC

like ATXMEGA256A3B or ATXMEGA256A3BU.

ATXMEGA with 16-Bit RTC:
- Can be used with one of the two internal RC oscillator options or external 32.768kHz crystal oscillator
- The internal 32 kHz Ultra Low Power (ULP) is a very low power clock source, and it is not designed for high accuracy.
- If you want to use the internal 32Khz RC oscillator you need to enable it with config osc

```
Config Osc = Disabled , 32mhzosc = Enabled , 32khzosc = Enabled
```

ATXMEGA with 32-Bit RTC (for example ATXMEGA256A3B or ATXMEGA256A3BU):
- An external 32.768kHz crystal oscillator must be used as the clock source
- The 32-Bit RTC is combined with a Battery Backup System

Numeric Values to calculate with Date and Time:

- SecOfDay: (Type LONG) Seconds elapsed since Midnight. 00:00:00 start with 0 to 85399 at 23:59:59.
- SysSec: (Type LONG) Seconds elapsed since begin of century (at 2000-01-01!). 00:00:00 at 2000-01-01 start with 0 to 2147483647 (overflow of LONG-Type) at 2068-01-19 03:14:07
- DayOfYear: (Type WORD) Days elapsed since first January of the current year.
- First January start with 0 to 364 (365 in a leap year)
- SysDay: (Type WORD) Days elapsed since begin of century (at 2000-01-01!). 2000-01-01 starts with 0 to 36524 at 2099-12-31
- DayOfWeek: (Type Byte) Days elapsed since Monday of current week. Monday start with 0 to Sunday = 6

With the numeric type calculations with Time and date are possible. Type 1 (discrete Bytes) and 2 (Strings) can be converted to an according numeric value. Than Seconds (at SecOfDay and SysSec) or Days (at DayOfYear, SysDay), can be added or subtracted. The Result can be converted back.

See also

TIME$ 1208 , DATE$ 1191 , CONFIG DATE 925, Memory usage 267, Date and Time Routines 1835

ASM

The following ASM routines are called from datetime.lib
_soft_clock. This is the ISR that gets called once per second.

Example 1

```
'-----------------------------------------------------------------------
------------------
'name                    : megaclock.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : shows the new TIME$ and DATE$ reserved
variables
'micro                   : Mega103
```

```
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'With the 8535 and timer2 or the Mega103 and TIMER0 you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code

'This example is written for the STK300 with M103
Enable Interrupts

'[configure LCD]
$lcd = &HC000                                               'address for
E and RS
$lcdrs = &H8000                                             'address for
only E
Config Lcd = 20 * 4                                         'nice
display from bg micro
Config Lcdbus = 4                                           'we run it
in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus                                        'tell about
the bus mode

'[now init the clock]
Config Date = Mdy , Separator = /                           ' ANSI-
Format

Config Clock = Soft                                         'this is how
simple it is
'The above statement will bind in an ISR so you can not use the TIMER
anymore!
'For the M103 in this case it means that TIMER0 can not be used by the
user anymore

'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/00"

'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used

Time$ = "02:20:00"

'-----------------------------------------------------

'clear the LCD display
Cls

Do
```

```
  Home                                                    'cursor home
  Lcd Date$ ; "  " ; Time$                               'show the
date and time
Loop

'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
_day = 1
'For the _year variable only the year is stored, not the century
End
```

Xmega Sample

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-RTC.bas
'  This sample demonstrates the Xmega128A1 RTC
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 64
$framesize = 64

Config Portb = Output

'First Enable The Osc Of Your Choice , make sure to enable 32 KHz clock
or use an external 32 KHz clock
Config Osc = Enabled , 32mhzosc = Enabled , 32khzosc = Enabled
' For the CLOCK we use the RTC so make sure the 32 KHZ osc is enabled!!!

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

Open "COM1:" For Binary As #1

Config Clock = Soft , Rtc = 1khz_int32khz_ulp               ' we select
the internal 1 KHz clock from the 32KHz internal oscillator
'the following clocks can be used to clock the RTC
' 1KHZ_INT32KHZ_ULP      1 kHz from internal 32 kHz ULP
' 1KHZ_32KHZ_CRYSTOSC    1 kHz from 32 kHz Crystal Oscillator on TOSC
' 1KHZ_INT32KHZ_RCOSC    1 kHz from internal 32 kHz RC Oscillator
' 32KHZ_32KHZ_CRYSTOSC   32 kHz from 32 kHz Crystal Oscillator on TOSC


Config Priority = Static , Vector = Application , Lo = Enabled       '
the RTC uses LO priority interrupts so these must be enabled !!!
Enable Interrupts                                           ' as usual
interrupts must be enabled

Do
  Print Time$                                               ' print the
time
  Waitms 1000
Loop

'TO USE THE SECTIC in the sample you must use GOSUB=SECTIC in CONFIG
CLOCK !!!

Sectic:
```

```
  Toggle Portb                                                    'optional
toggle some leds when using the gosub=sectic option
Return
```

Example 2

```
$regfile = "m128def.dat"
$hwstack = 80
$swstack = 80
$framesize = 160
$crystal = 8000000
$baud = 19200

Enable Interrupts

'[now init the clock]
Config Date = Mdy , Separator = /                                 ' ANSI-Format

Config Clock = Soft                                               'this is how simple it is
'The above statement will bind in an ISR so you can not use the TIMER anymore!

'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/05"

'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used

Time$ = "23:59:50"
Do
    Waitms 500
    Print Date$ ; Spc(3) ; Time$
Loop
```

Example 3 (using DS1307 with Config clock)

```
 '-----------------------------------------------------------------------------
'                                DateTime_test.bas
'      This sample show how to use the Date-Time routines from the DateTime.Lib
'      written by Josef Franz Vögel
'-----------------------------------------------------------------------------

$regfile = "m328pdef.dat"
$crystal = 12e6                                                   '16MHz
$hwstack = 80
$swstack = 80
$framesize = 160


Const Clockmode = 1
'use i2c for the clock

#if Clockmode = 1
  Config Clock = Soft                                             ' we use build in clock
  Disable Interrupts
#else
  Config Clock = User                                             ' we use I2C for the clock
  'configure the scl and sda pins (using software I2C routines)
  Config Sda = Portd.6
  Config Scl = Portd.5
  I2cinit

  'address of ds1307
  Const Ds1307w = &HD0                                            ' Addresses of Ds1307 clock
  Const Ds1307r = &HD1
#endif


'configure the date format
Config Date = Ymd , Separator = -                                 ' ANSI-Format
'This sample does not have the clock started so interrupts are not enabled
'    Enable Interrupts

'dim the used variables
Dim Lvar1 As Long
Dim Mday As Byte
```

```
Dim Bweekday As Byte , Strweekday As String * 10
Dim Strdate As String * 8
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Bday As Byte , Bmonth As Byte , Byear As Byte
Dim Lsecofday As Long
Dim Wsysday As Word
Dim Lsyssec As Long
Dim Wdayofyear As Word



' ================== DayOfWeek ============================================
' Example 1 with internal RTC-Clock

_day = 4 : _month = 11 : _year = 2 ' Load RTC-Clock for example -
testing
Bweekday = Dayofweek( )
Strweekday = Lookupstr( bweekday , Weekdays)
Print "Weekday-Number of " ; Date$ ; " is " ; Bweekday ; " = " ; Strweekday


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 26 : Bmonth = 11 : Byear = 2
Bweekday = Dayofweek( bday)
Strweekday = Lookupstr( bweekday , Weekdays)
Strdate = Date( bday)
Print "Weekday-Number of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear ; " is " ;
Bweekday ; " (" ; Date( bday) ; ") = " ; Strweekday


' Example 3 with System Day
Wsysday = 2000 ' that is 2005-06-23
Bweekday = Dayofweek( wsysday)
Strweekday = Lookupstr( bweekday , Weekdays)
Print "Weekday-Number of System Day " ; Wsysday ; " (" ; Date( wsysday) ; ") is " ;
Bweekday ; " = " ; Strweekday


' Example 4 with System Second
Lsyssec = 123456789 ' that is 2003-11-29 at
21:33:09
Bweekday = Dayofweek( lsyssec)
Strweekday = Lookupstr( bweekday , Weekdays)
Print "Weekday-Number of System Second " ; Lsyssec ; " (" ; Date( lsyssec) ; ") is " ;
Bweekday ; " = " ; Strweekday



' Example 5 with Date-String
Strdate = "04-11-02" ' we have configured Date in
ANSI
Bweekday = Dayofweek( strdate)
Strweekday = Lookupstr( bweekday , Weekdays)
Print "Weekday-Number of " ; Strdate ; " is " ; Bweekday ; " = " ; Strweekday



' ================ Second of Day ==========================================
' Example 1 with internal RTC-Clock
_sec = 12 : _min = 30 : _hour = 18 ' Load RTC-Clock for example -
testing

Lsecofday = Secofday( )
Print "Second of Day of " ; Time$ ; " is " ; Lsecofday


' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
Bsec = 20 : Bmin = 1 : Bhour = 7
Lsecofday = Secofday( bsec)
Print "Second of Day of Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(
bsec) ; ") is " ; Lsecofday


' Example 3 with System Second
Lsyssec = 1234456789
Lsecofday = Secofday( lsyssec)
Print "Second of Day of System Second " ; Lsyssec ; " ( " ; Time( lsyssec) ; ") is " ;
Lsecofday


' Example 4 with Time - String
```

```
Strtime = "04:58:37"
Lsecofday = Secofday(strtime)
Print "Second of Day of " ; Strtime ; " is " ; Lsecofday


' ================== System Second ========================================

' Example 1 with internal RTC-Clock
                              ' Load RTC-Clock for example - testing
_sec = 17 : _min = 35 : _hour = 8 : _day = 16 : _month = 4 : _year = 3

Lsyssec = Syssec( )
Print "System Second of " ; Time$ ; " at " ; Date$ ; " is " ; Lsyssec


' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day / Month / Year)
Bsec = 20 : Bmin = 1 : Bhour = 7 : Bday = 22 : Bmonth = 12 : Byear = 1
Lsyssec = Syssec(bsec)
Strtime = Time(bsec)
Strdate = Date(bday)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ; Lsyssec


' Example 3 with System Day

Wsysday = 2000
Lsyssec = Syssec(wsysday)
Print "System Second of System Day " ; Wsysday ; " (" ; Date(wsysday) ; " 00:00:00) is " ;
Lsyssec


' Example 4 with Time and Date String
Strtime = "10:23:50"
Strdate = "02-11-29"                                        ' ANSI-Date
Lsyssec = Syssec(strtime , Strdate)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ; Lsyssec          ' 91880630



' ==================== Day Of Year ========================================
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2                         ' Load RTC-Clock for example -
testing
Wdayofyear = Dayofyear( )
Print "Day Of Year of " ; Date$ ; " is " ; Wdayofyear


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wdayofyear = Dayofyear(bday)
Print "Day Of Year of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear ; " (" ; Date(
bday) ; ") is " ; Wdayofyear


' Example 3 with Date - String
Strdate = "04-10-29"                                        ' we have configured ANSI
Format
Wdayofyear = Dayofyear(strdate)
Print "Day Of Year of " ; Strdate ; " is " ; Wdayofyear


' Example 4 with System Second

Lsyssec = 123456789
Wdayofyear = Dayofyear(lsyssec)
Print "Day Of Year of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ; ") is " ;
Wdayofyear


' Example 5 with System Day
Wsysday = 3000
Wdayofyear = Dayofyear(wsysday)
Print "Day Of Year of System Day " ; Wsysday ; " (" ; Date(wsysday) ; ") is " ; Wdayofyear



' ================== System Day ========================================
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2                         ' Load RTC-Clock for example -
testing
Wsysday = Sysday( )
```

```
Print "System Day of " ; Date$ ; " is " ; Wsysday


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wsysday = Sysday(bday)
Print "System Day of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear ; " (" ; Date(
bday) ; ") is " ; Wsysday


' Example 3 with Date - String
Strdate = "04-10-29"
Wsysday = Sysday(strdate)
Print "System Day of " ; Strdate ; " is " ; Wsysday

' Example 4 with System Second
Lsyssec = 123456789
Wsysday = Sysday(lsyssec)
Print "System Day of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ; ") is " ; Wsysday


' ================== Time ================================================
' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour) to Time - String
Bsec = 20 : Bmin = 1 : Bhour = 7
Strtime = Time(bsec)
Print "Time values: Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; " converted to
string " ; Strtime


' Example 2: Converting System Second to Time - String
Lsyssec = 123456789
Strtime = Time(lsyssec)
Print "Time of Systemsecond " ; Lsyssec ; " is " ; Strtime


' Example 3: Converting Second of Day to Time - String
  Lsecofday = 12345
Strtime = Time(lsecofday)
Print "Time of Second of Day " ; Lsecofday ; " is " ; Strtime


' Example 4: Converting System Second to defined Clock - Bytes (Second / Minute / Hour)

Lsyssec = 123456789
Bsec = Time(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour="
 ; Bhour ; " (" ; Time(lsyssec) ; " ) "


' Example 5: Converting Second of Day to defined Clock - Bytes (Second / Minute / Hour)
Lsecofday = 12345
Bsec = Time(lsecofday)
Print "Second of Day " ; Lsecofday ; " converted to Sec=" ; Bsec ; " Min=" ; Bmin ; "
Hour=" ; Bhour ; " (" ; Time(lsecofday) ; " ) "

' Example 6: Converting Time-string to defined Clock - Bytes (Second / Minute / Hour)
Strtime = "07:33:12"
Bsec = Time(strtime)
Print "Time " ; Strtime ; " converted to Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour


' ============================ Date ========================================

' Example 1: Converting defined Clock - Bytes (Day / Month / Year) to Date - String
Bday = 29 : Bmonth = 4 : Byear = 12
Strdate = Date(bday)
Print "Dat values: Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear ; " converted to
string " ; Strdate


' Example 2: Converting from System Day to Date - String
Wsysday = 1234
Strdate = Date(wsysday)
Print "System Day " ; Wsysday ; " is " ; Strdate


' Example 3: Converting from System Second to Date String
Lsyssec = 123456789
Strdate = Date(lsyssec)
Print "System Second " ; Lsyssec ; " is " ; Strdate


' Example 4: Converting SystemDay to defined Clock - Bytes (Day / Month / Year)
```

```
Wsysday = 2000
Bday = Date(wsysday)
Print "System Day " ; Wsysday ; " converted to Day=" ; Bday ; " Month=" ; Bmonth ; "
Year=" ; Byear ; " (" ; Date(wsysday) ; ")"


' Example 5: Converting Date - String to defined Clock - Bytes (Day / Month / Year)
Strdate = "04-08-31"
Bday = Date(strdate)
Print "Date " ; Strdate ; " converted to Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear


' Example 6: Converting System Second to defined Clock - Bytes (Day / Month / Year)
Lsyssec = 123456789
Bday = Date(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Day=" ; Bday ; " Month=" ; Bmonth ; "
Year=" ; Byear ; " (" ; Date(lsyssec) ; ")"


' ================ Second of Day elapsed

Lsecofday = Secofday( )
_hour = _hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1

Lsyssec = Syssec( )
_day = _day + 1
Lvar1 = Syssecelapsed(lsyssec)
Print Lvar1




Looptest:

' Initialising for testing
_day = 1
_month = 1
_year = 1
_sec = 12
_min = 13
_hour = 14


Do
   If _year > 50 Then
      Exit Do
   End If

   _sec = _sec + 7
  If _sec > 59 Then
     Incr _min
     _sec = _sec - 60
  End If

  _min = _min + 2
  If _min > 59 Then
     Incr _hour
     _min = _min - 60
  End If

  _hour = _hour + 1
  If _hour > 23 Then
     Incr _day
     _hour = _hour - 24
  End If

  _day = _day + 1


  If _day > 28 Then
     Select Case _month
        Case 1
           Mday = 31
        Case 2
           Mday = _year And &H03
           If Mday = 0 Then
              Mday = 29
           Else
```

```
                Mday = 28
            End If
        Case 3
            Mday = 31
        Case 4
            Mday = 30
        Case 5
            Mday = 31
        Case 6
            Mday = 30
        Case 7
            Mday = 31
        Case 8
            Mday = 31
        Case 9
            Mday = 30
        Case 10
            Mday = 31
        Case 11
            Mday = 30
        Case 12
            Mday = 31
    End Select
    If _day > Mday Then
        _day = _day - Mday
        Incr _month
        If _month > 12 Then
            _month = 1
            Incr _year
        End If
    End If
  End If
  If _year > 99 Then
    Exit Do
  End If

Lsecofday = Secofday( )
Lsyssec = Syssec( )
Bweekday = Dayofweek( )
Wdayofyear = Dayofyear( )
Wsysday = Sysday( )


Print Time$ ; " " ; Date$ ; " " ;  Lsecofday ; " " ;  Lsyssec ; " " ;  Bweekday ; " " ;
Wdayofyear ; " " ; Wsysday


Loop
End


'only when we use I2C for the clock we need to set the clock date time
#if Clockmode = 0
'called from datetime.lib
Dim Weekday As Byte
Getdatetime:
  I2cstart                                                ' Generate start code
  I2cwbyte Ds1307w                                        ' send address
  I2cwbyte 0                                              ' start address in 1307

  I2cstart                                                ' Generate start code
  I2cwbyte Ds1307r                                        ' send address
  I2crbyte _sec , Ack
  I2crbyte _min , Ack                                     ' MINUTES
  I2crbyte _hour , Ack                                    ' Hours
  I2crbyte Weekday , Ack                                  ' Day of Week
  I2crbyte _day , Ack                                     ' Day of Month
  I2crbyte _month , Ack                                   ' Month of Year
  I2crbyte _year , Nack                                   ' Year
  I2cstop
  _sec = Makedec(_sec) : _min = Makedec(_min) : _hour = Makedec(_hour)
  _day = Makedec(_day) : _month = Makedec(_month) : _year = Makedec(_year)
Return

Setdate:
  _day = Makebcd(_day) : _month = Makebcd(_month) : _year = Makebcd(_year)
  I2cstart                                                ' Generate start code
  I2cwbyte Ds1307w                                        ' send address
  I2cwbyte 4                                              ' starting address in 1307
  I2cwbyte _day                                           ' Send Data to SECONDS
  I2cwbyte _month                                         ' MINUTES
  I2cwbyte _year                                          ' Hours
  I2cstop
Return

Settime:
```

```
   _sec = Makebcd(_sec) : _min = Makebcd(_min) : _hour = Makebcd(_hour)
   I2cstart                                                ' Generate start code
   I2cwbyte Ds1307w                                        ' send address
   I2cwbyte 0                                              ' starting address in 1307
   I2cwbyte _sec                                           ' Send Data to SECONDS
   I2cwbyte _min                                           ' MINUTES
   I2cwbyte _hour                                          ' Hours
   I2cstop
Return

#endif


Weekdays:
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" , "Saturday" , "Sunday"
```

7.21.15 CONFIG CLOCKDIV

Action

Sets the clock divisor.

Syntax

CONFIG CLOCKDIV = constant

Remarks

constant	The clock division factor to use. Possible values are 1 , 2 , 4 , 8 ,16 , 32 ,64 , 128 and 256.

The options to set the clock divisor is available in most new chips. Under normal conditions the clock divisor is one. Thus an oscillator value of 8 MHz will result in a system clock of 8 MHz. With a clock divisor of 8, you would get a system clock of 1 MHz.
Low speeds can be used to generate an accurate system frequency and for low power consumption.
Some chips have a 8 or 16 division enabled by default by a fuse bit.
You can then reprogram the fuse bit or you can set the divisor from code.

When you set the clock divisor take care that you adjust the $CRYSTAL directive also. $CRYSTAL specifies the clock frequency of the system. So with 8 MHz clock and divisor of 8 you would specify $CRYSTAL = 1000000.

Some older chips use a different method for clock division. These chips do not support CONFIG CLOCK but they might support CLOCKDIVSION 847.

See also

$CRYSTAL 625 , CLOCKDIVISION 847

Example

```
CONFIG CLOCKDIV = 8 'we divide 8 MHz crystal clock by 8 resulting in 1
MHz speed
```

7.21.16 CONFIG COM1

Action

Configures the UART of AVR chips that have an extended UART like the M8.

Syntax

CONFIG COM1 = baud , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1|2,databits=4|6|7|8|9,clockpol=0|1

Remarks

baud	Baud rate to use. Use 'dummy' to leave the baud rate at the $baud value.
synchrone	0 for asynchrone operation (default) and 1 for synchrone operation.
Parity	None, disabled, even or odd
Stopbits	The number of stop bits : 1 or 2
Databits	The number of data bits : 4,5,7,8 or 9.
Clockpol	Clock polarity. 0 or 1.

 Note that not all AVR chips have the extended UART. Most AVR chips have a UART with fixed communication parameters. These are : No parity, 1 stop bit, 8 data bits.

Normally you set the BAUD rate with $BAUD or at run time with BAUD. You may also set the baud rate when you open the COM channel. It is intended for the Mega2560 that has 4 UARTS and it is simpler to specify the baud rate when you open the channel. It may also be used with the first and second UART but it will generate additional code since using the first UART will always result in generating BAUD rate init code.

See Also

CONFIG COM2 909 , CONFIG COMx 913

Example

```
'-----------------------------------------------------------------------
------------------
'name                       :
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : test for M128 support in M128 mode
'micro                      : Mega128
'suited for demo            : yes
'commercial addon needed    : no
'-----------------------------------------------------------------------
------------------

$regfile = "m128def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$baud1 = 19200
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space
```

```
'By default the M128 has the M103 compatibility fuse set. Set the fuse
to M128
'It also runs on a 1 MHz internal oscillator by default
'Set the internal osc to 4 MHz for this example DCBA=1100

'use the m128def.dat file when you wanto to use the M128 in M128 mode
'The M128 mode will use memory from $60-$9F for the extended registers

'Since some ports are located in extended registers it means that some
statements
'will not work on these ports. Especially statements that will set or
reset a bit
'in a register. You can set any bit yourself with the PORTF.1=1
statement for example
'But the I2C routines use ASM instructions to set the bit of a port.
These ASM instructions may
'only be used on port registers. PORTF and PORTG will not work with I2C.


'The M128 has an extended UART.
'when CONFIG COMx is not used, the default N,8,1 will be used
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'try the second hardware UART
Open "com2:" For Binary As #1

'try to access an extended register
Config Portf = Output
'Config Portf = Input

Print "Hello"
Dim B As Byte
Do
   Input "test serial port 0" , B
   Print B
   Print #1 , "test serial port 2"
Loop

Close #1
End
```

7.21.17 CONFIG COM2

Action

Configures the UART of AVR chips that have a second extended UART like the M128.

Syntax

CONFIG COM2 = baud , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1|2,databits=4|6|7|8|9,clockpol=0|1

Remarks

baud	Baud rate to use. Use 'dummy' to leave the baud rate at the $baud1 value.
synchrone	0 for asynchrone operation (default) and 1 for synchrone operation.

Parity	None, disabled, even or odd
Stopbits	The number of stopbits : 1 or 2
Databits	The number of databits : 4,5,7,8 or 9.
Clockpol	Clock polarity. 0 or 1.

Normally you set the BAUD rate with $BAUD or at run time with BAUD. You may also set the baud rate when you open the COM channel. It is intended for the Mega2560 that has 4 UARTS and it is simpler to specify the baud rate when you open the channel. It may also be used with the first and second UART but it will generate additional code since using the first or second UART will always result in generating BAUD rate init code.

 Note that not all AVR chips have the extended UART. Most AVR chips have a UART with fixed communication parameters. They are : No parity, 1 stopbit, 8 data bits.

See Also

CONFIG COM1 909 , CONFIG COMx 913

Example

```
'-------------------------------------------------------------------------
------------------
'name                       :
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : test for M128 support in M128 mode
'micro                      : Mega128
'suited for demo            : yes
'commercial addon needed    : no
'-------------------------------------------------------------------------
------------------

$regfile = "m128def.dat"                    ' specify the used micro
$crystal = 4000000                          ' used crystal frequency
$baud = 19200                               ' use baud rate
$baud1 = 19200                              ' default use 32 for the
hardware stack
$swstack = 10                               ' default use 10 for the SW
stack
$framesize = 40                             ' default use 40 for the frame
space


'By default the M128 has the M103 compatibility fuse set. Set the fuse
to M128
'It also runs on a 1 MHz internal oscillator by default
'Set the internal osc to 4 MHz for this example DCBA=1100

'use the m128def.dat file when you wanto to use the M128 in M128 mode
'The M128 mode will use memory from $60-$9F for the extended registers

'Since some ports are located in extended registers it means that some
statements
'will not work on these ports. Especially statements that will set or
reset a bit
'in a register. You can set any bit yourself with the PORTF.1=1
statement for example
'But the I2C routines use ASM instructions to set the bit of a port.
These ASM instructions may
```

```
'only be used on port registers. PORTF and PORTG will not work with I2C.


'The M128 has an extended UART.
'when CONFIG COMx is not used, the default N,8,1 will be used
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'try the second hardware UART
Open "com2:" For Binary As #1

'try to access an extended register
Config Portf = Output
'Config Portf = Input

Print "Hello"
Dim B As Byte
Do
   Input "test serial port 0" , B
   Print B
   Print #1 , "test serial port 2"
Loop

Close #1
End
```

7.21.18 CONFIG COMx

Action

Configures the UART of AVR chips that have an extended UART like the M2560.

Syntax

CONFIG COMx = baud , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1|2,databits=4|6|7|8|9,clockpol=0|1

Syntax Xmega

CONFIG COMx = baud , Mode=mode, Parity=parity, Stopbits=stopbits, Databits=databits

Syntax Xtiny/MegaX

CONFIG COMx = baud , Mode=mode, Parity=parity, Stopbits=stopbits, Databits=databits , Baud_Offset=baud_ofs , TX_RX_XC_XD_PIN=tx, TX=tx,RX=rx

Syntax AVRX

CONFIG COMx = baud , Mode=mode, Parity=parity, Stopbits=stopbits, Databits=databits , TX_RX_XC_XD_PIN=tx, TX=tx,RX=rx
There is no baud offset in the AVRX series

Remarks normal AVR

COMx	The COM port to configure. Value in range from 1-4

baud	Baud rate to use.
synchrone	0 for asynchrone operation (default) and 1 for synchrone operation.
Parity	None, disabled, even or odd
Stopbits	The number of stop bits : 1 or 2
Databits	The number of data bits : 4,5,7,8 or 9.
Clockpol	Clock polarity. 0 or 1.

 Note that not all AVR chips have the extended UART. Most AVR chips have a UART with fixed communication parameters. These are : No parity, 1 stopbit, 8 data bits.
The Mega2560 does support 4 UART's.

Remarks Xmega

COMx	The COM port to configure. Value in range from 1-8
baud	Baud rate to use. If the baud rate can be generated accurately depends on the system clock.
mode	The USART mode, this can be : - ASYNCHRONEOUS or 0 (default) for asynchronous operation. - SYNCHRONEOUS or 1 , for synchronous operation. - IRDA or IRCOM for IRDA operation - SPI or MSPI for operation as SPI controller *** The mode must be provided since the baud calculation depends on the selected mode ***
Parity	None, disabled, even or odd
Stopbits	The number of stop bits : 1 or 2
Databits	The number of data bits : 5,6,7,8 or 9.

In the Xmega the registers have a fixed offset. This allows to use dynamic UARTS : you can change settings at run time by using a variable. This will use some more code when using just one UART but will save code when using multiple UARTS because you need only one copy of the code.

In the Xmega you MUST use CONFIG COM before you can use the UART. The CONFIG commands makes a call to _INIT_XMEGA_UART where the various parameters are passed to setup the UART. You also need to specify the baud rate. Do not use $BAUD.

The CLOCKPOL for the SPI mode has been removed, it will be added to a configuration command for the SPI.

The CONFIG COM will set the TX pin to output mode. This are the following pins :

UART	TX pin	RX pin	BAUD 1489
COM1 - UART_C0	PORTC.3	PORTC.2	BAUD
COM2 - UART_C1	PORTC.7	PORTC.6	BAUD1
COM3 - UART_D0	PORTD.3	PORTD.2	BAUD2
COM4 - UART_D1	PORTD.7	PORTD.6	BAUD3
COM5 - UART_E0	PORTE.3	PORTE.2	BAUD4
COM6 - UART_E1	PORTE.7	PORTE.6	BAUD5
COM7 - UART_F0	PORTF.3	PORTF.2	BAUD6

COM8 - UART_F1	PORTF.7	PORTF.6	BAUD7

In IRDA mode, depending on the module you use, it might be necessary to invert the logic level of the TX pin with CONFIG XPIN. For example when COM1 is used for the IRDA module, you would use : **CONFIG XPIN=**PORTC.3, INVERTIO=ENABLED

Remarks XTINY/MEGAX/AVRX

COMx	The COM port to configure. Value in range from 1-6
baud	Baud rate to use. If the baud rate can be generated accurately depends on the system clock.
mode	The USART mode, this can be : - ASYNCHRONEOUS : (default) for asynchronous operation. - SYNCHRONEOUS : for synchronous operation. - IRCOM : for IRDA operation - SPI : for operation as SPI controller *** The mode must be provided since the baud calculation depends on the selected mode ***
Parity	None, disabled, even or odd
Stopbits	The number of stop bits : 1 or 2
Databits	The number of data bits : 5,6,7,8 or 9.
Baud_offset	The Xtiny/MegaX has an internal oscillator that runs at 16 or 20 MHz. The center frequency can be off depending on temperature and voltage. The Xtiny has 4 calibrated offset values for the oscillator which can be used to correct the BAUD. The options are : - NONE : default, BAUD is calculated but no offset is used. All of the other values will read the signature row and will call code from xtiny.lib to compensate the BAUD value. - OSC16_3V3 : OSC runs at 16 MHz and at 3V3 - OSC16_5V : OSC runs at 16 MHz and at 5V - OSC20_3V3 : OSC runs at 20 MHz and at 3V3 - OSC20_5V : OSC runs at 20 MHz and at 5V It is up to the user to apply the selected voltage. It is up to the user to use the specified oscillator value. (change fuse bits). By default the 20 MHz internal osc is selected. So this setting does not change oscillator values, it only tells the compiler which sigrow must be loaded and used.
TX_RX_XC_XD_PIN This option was named TXPIN before	This options selects the pins used for the UART. The XTINY/MEGAX/AVRX has a port multiplexer. This multi plexer allows to chose which pins are used for hardware connected to the port pins. So this allows to chose one or more alternative pin locations for hardware such as USART,SPI, TWI and TIMER output. You can use CONFIG PORT_MUX when you use alternative pin positions. But CONFIG PORT_MUX 1014 does not set port direction. For the USART it is required to set the TX pin into output mode. That is why this option exists : you can chose the

	alternative pin location and the compiler will set the port pin into output mode and will set the proper port multiplexer register bit. You can chose between the default location starting with **DEF_** and the alternative location starting with **ALT**x_ Some processors also allow to disconnect the pins totally and they have a NONE option. It is important to understand that selecting an alternative pin will switch all the pins of that hardware device. For the USART this means you will switch both TX, RX, XCK and XDIR pin. You can not just change only the TX or RX. You can however change the pins dynamically at run time. Your hardware circuit should support this of course. Please note that you only should use this option when you use the alternative pin location since using this option create more code since the multiplexer is configured. The parameter value for TX_RX_XC_XD_PIN lists all associated pins in the following order : TX, RX, XCK, XDIR. For example : **Def**_pb2_pb3_pb1_pb0 which means that PortB.2 is connected to TX and RX is connected to PortB.3. ALT1_PA1_PA2_PA3_PA4 means that the alternative pin is used which is PA1 for TX in this case.
TX	By default both the transmitter and receiver are enabled. But there are cases where you only want to use the receiver. In such a case you can DISABLE the TX pin. TX=DISABLED. The default is enabled and there is no need to specify this. Possible options : ENABLED and DISABLED
RX	By default both the transmitter and receiver are enabled. But there are cases where you only want to use the transmiiter. In such a case you can DISABLE the RX pin. RX=DISABLED. The default is enabled and there is no need to specify this. Possible options : ENABLED and DISABLED

It is preferred to use CONFIG COM instead of using $BAUD.

 It is important that you specify all parameters of CONFIG COM. Do not omit one. The only optional parameter is TX_RX_XC_XD_PIN for the alternative USART pins, and the TX and RX options to disable pins.

See Also

CONFIG COM1 909 , CONFIG COM2 909

Example

```
'-----------------------------------------------------------------------
'name                         :
'copyright                    : (c) 1995-2025, MCS Electronics
```

```
'purpose                  : test for M2560 support
'micro                    : Mega2560
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------------

$regfile = "m2560def.dat"                                    ' specify the used micr
$crystal = 8000000                                           ' used crystal frequenc
$hwstack = 40                                                ' default use 32 for th
$swstack = 40                                                ' default use 10 for th
$framesize = 40                                              ' default use 40 for th

'The M128 has an extended UART.
'when CO'NFIG COMx is not used, the default N,8,1 will be used
Config Com1 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Config Com3 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Config Com4 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,


'Open all UARTS
Open "com2:" For Binary As #1
Open "Com3:" For Binary As #2
Open "Com4:" For Binary As #3


Print "Hello"                                               'first uart
Dim B As Byte
Dim Tel As Word

Do
  Incr Tel
  Print Tel ; " test serial port 1"
  Print #1 , Tel ; " test serial port 2"
  Print #2 , Tel ; " test serial port 3"
  Print #3 , Tel ; " test serial port 4"

  B = Inkey(#3)
  If B <> 0 Then
     Print #3 , B ; " from port 4"
  End If
  Waitms 500
Loop

Close #1
Close #2
Close #3
End
```

Xtiny Example

```
'-------------------------------------------------------------------
----------------
'name                     : serial.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates USART
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-------------------------------------------------------------------
```

```
----------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24

'set the system clock and prescaler
Config Sysclock = 16_20MHZ , Prescale = 1

'configure the USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'dimension a variable
Dim B As Byte

Config PORTC.1 = Output
Print "Test USART"

Do
   Print "Hello" ; Spc(3) ; B
   Waitms 1000
   Incr B
   Toggle PORTC.1
Loop


End
```

7.21.19 CONFIG DACA|DACB

Action

This statement configures the DACA or DACB in the Xmega.

Syntax

CONFIG DACx=dac, **IO0=**IO0, **IO1=**IO1, **INTERNAL_OUTPUT =**INTOTP, **CHANNEL=**channel, **TRIGGER_CH0=**trig0, **TRIGGER_CH1=**trig1, **REFERENCE=** ref, **LEFT_ADJUSTED=**adjusted, **EVENT_CHANNEL=**event, **INTERVAL=**interval, **REFRESH=**refresh

Remarks

DACX	Chose either DACA or DACB. DACA is connected to PORTA. DACB is connected to PORTB.
dac	ENABLED or DISABLED. Chose ENABLED to enable the DAC.
IO0	ENABLED or DISABLED. Chose ENABLED to enable output 0. Each DAC has 2 outputs. When multiple outputs are used, the DAC is using S&H.
IO1	ENABLED or DISABLED. Chose ENABLED to enable output 1.

Intotp	ENABLED or DISABLED. Chose ENABLED to enable the internal output.
Channel	SINGLE or DUAL. If both outputs are used, you need to enable the second output with IO1.
Trig0	ENABLED or DISABLED. Chose ENABLED to enable the trigger of channel 0.
Trig1	ENABLED or DISABLED. Chose ENABLED to enable the trigger of channel 1.
Ref	The DAC needs a stable voltage reference. You can chose one of the following: - INT1V. This will select the internal 1V reference - AVCC. This will use AVCC as reference. - AREFA. This will use AREFA as reference. - AREFB. This will use AREFB as reference. The output of the DAC can never be higher then the voltage reference. When you chose INT1V, the output is from 0-1V in 4096 steps.
Adjusted	ENABLED or DISABLED. By default the DAC output is right adjusted (this means the first 8 Bit are in the Low Byte and the following 4 Bit in the High Byte of the 16-bit Register). You can left alight the result.
Event	The event channel to use for the event system.
Interval	The minimum interval between 2 conversions. This is a value of : 1,2,4,8,16,32,64 or 128. The default in the register is 64. A value of 64 will give an interval of 64 clock cycles. The value is set in clock cycles and the time in µ Second depend on the CLKper (Peripheral Clock) setting. The minimum in SINGLE Channel mode is 1µS (1M conversions per seconds). The minimum in DUAL Channel mode (S/H mode) should no be below 1.5µS (666K conversions per second). In DUAL Channel mode the 50% increase of peripheral clock cycles is AUTOMATICALLY added by the XMEGA chip.
Refresh	The DAC channel refresh timing. This is the interval refresh time in DUAL channel mode. Possible values: OFF 16, 32, 128, 256, 512, 1014, 2048, 4096, 8192, 16384, 32768, 65536. A value of 16 means an interval of 16 clock cycles. The default loaded is 64. Note: Higher refresh rates causes higher power consumption. Manual conversions or Events between the refresh intervals do NOT affect the refresh intervals. This means the channels will be refreshed at a constant timing even when the data register are for example updated in between.

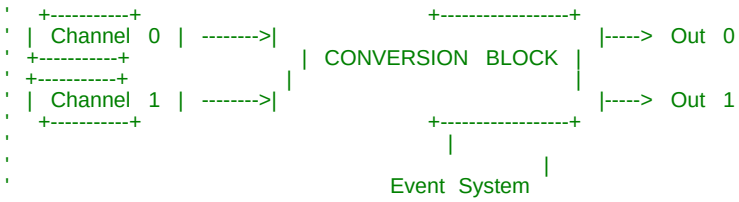
The DAC data register is available in the DACA0, DACA1 and DACB0 and DACB1 variables.

The DAC module can output conversion rates up to 1 M conversions per second with a resolution of 12 bits.

A DAC conversion can be triggered by:
- writing to the DAC data register (DACA0, DACA1 and DACB0 and DACB1)
- an Event over Event System (when configured to trigger from Event system the DAC data register can be updated several times without triggering an conversion. In case of an Event the latest value in the DAC data register will be used for conversion)

Trigger mode can be different between DAC Channels. For example DAC Channel 0 can be setup to work with Events while Channel 1 can be configured to start conversion when DAC data register is updated.

How to handle the two Data Channels with one conversion Block:



The fact that there are two data channels but one conversion block it needs to be configured by CHANNEL.
- If Channel is SINGLE: Channel 0 is used in continuous-drive output mode and Channel 0 is then always connected to conversion block.
- If Channel is DUAL: Both channels work in Sample and Hold (S/H) mode. The Sample and Hold keep the DAC output values during a conversion of the other channel. To refresh the output value in DUAL channel mode the refresh timing can be set.

What can you drive with the XMEGA DAC outputs ?
- The ouputs can drive loads of 1KOhm or capacitive loads of 100pF

It is possible to use the XMEGA DMA Controller to output data on DAC Channels.
See CONFIG DMACHx 937, CONFIG DMA 936
See also Example Nr 2 below.

Calibration of DAC:

To Calibrate to DAC you can use the values from the signature row or you can change manual the **Dacb_ch0offsetcal** and **Dacb_gaincal** register.
For example for using signature row for DACB Ch0 this is:

```
'DACB
B = Readsig(32)                                             'DACB
Calibration Byte 0 (DACBOFFCAL)
Dacb_ch0offsetcal = B                                       'write to
the DACB offset register
Print #1 , "DACB Calibration Byte 0 = " ; B
B = Readsig(33)                                             'DACB
Calibration Byte 1 (DACBGAINCAL)
Dacb_gaincal = B
Print #1 , "DACB Calibration Byte 1 = " ; B
```

See also Atmel Application Note AVR1301 for further details.

See also

START 1538 , STOP 1544, CONFIG EVENT_SYSTEM 953

Example Nr 1:

(For another example see also the example **xm128a1.bas** from the samples\chips folder)

```
$regfile = "xm256a3bdef.dat"
$crystal = 32000000                                         '32MHz
$hwstack = 64
$swstack = 40
$framesize = 40

Config Osc = Disabled , 32mhzosc = Enabled                  '32MHz

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1


Config Com7 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8       'Portf.2 and Portf.3 is COM7
Open "COM7:" For Binary As #1

Dim Var As Byte

Config Portf.0 = Output
Led1 Alias Portf.0

Config Portf.1 = Output
Led2 Alias Portf.1



Config Dacb = Enabled , Io0 = Enabled , Channel = Single , Reference =
Int1v , Interval = 64 , Refresh = 64
Dacb0 = 4095                                                '1 V output
on portb.2

'Start Dacb                                                  ' to enable
it
'Stop Dacb                                                   ' to
disable it

Do

 Incr Var
 Waitms 500
 Dacb0 = 4095                                               '1 V output
on portb.2
 Set Led1
 Reset Led2

 Waitms 500
 Reset Led1
 Dacb0 = 0                                                  '0 V output
on portb.2
 Set Led2

 Print #1 , "Tick " ; Var

Loop
```

```
End                                                    'end program
```

Example Nr 2 (Ouput an Array of data from SRAM to DAC B over DMA):

(This example is generating an sawtooth wave on DAC B Channel 0 = Portb.2 on ATXMEGA256A3B)

```
' Ouput an Array of data from SRAM to DAC B over DMA

' Timing: Timer/Counter TC0 feed the Event Channel 0
' Event Channel 0 feed the DAC B Channel 0

' Array Channel_0(1) is a word array filled with values

' DMA Channel 0 start at Channel_0(1) and increment until 8192 Byte (= 2*4096). After the DMA transaction the source address will be reloaded
' The destination address is the data register of DAC B Channel 0 and is incrementd once (to update the Low Byte and High Byte of the 12-Bit output value)

'Frequency of output signal = 32MHz/32 = 1MHz --> 1MHz/4096 (Sample_Count) = appx. 244Hz

$regfile = "xm256a3bdef.dat"
$crystal = 32000000                                    '32MHz
$hwstack = 64
$swstack = 40
$framesize = 40

Config Osc = Disabled , 32mhzosc = Enabled             '32MHz
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Priority = Static , Vector = Application , Lo = Enabled

Config Com7 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8
    'Portf.2 and Portf.3 is COM7
Open "COM7:" For Binary As #1

Print #1 ,
Print #1 , "Start DAC B Channel 0 over DMA Example"


Dim Var As Byte

Config Portf.0 = Output
Led1 Alias Portf.0

Config Portf.1 = Output
Led2 Alias Portf.1


Const Sample_count = 4096                              'Number of 12-Bit Samples (Measurement Values)
Dim Channel_0(sample_count) As Word                    'Array
Dim Dma_ready As Bit
Dim Dma_channel_0_error As Bit


Enable_dmach0 Alias Dma_ch0_ctrla.7                    'Enable DMA Channel 0

Dim I As Word

For I = 1 To 4096                                      'From 0V .....3.3Volt (with Reference = avcc)
    Channel_0(i) = I                                   'Generate a Sawtooth wave
Next


Config Tcc0 = Normal , Prescale = 1                    'Setup Timer/Counter TC0 in nomal mode , Prescale = 1 --> no prescaler
Tcc0_per = 31                                          '31 --> 32MHz/32 = 1MHz

Config Event_system = Dummy , Mux0 = Tcc0_ovf          'TCC 0 overflow --> Event Channel 0


' The xm256a3bd only have one DAC (DAC B)
Config Dacb = Enabled , Io0 = Enabled , Channel = Single , Trigger_ch0 = Enabled , Event_channel = 0 , Reference = Avcc , Interval = 4 , Refresh = 16
```

```
' DAC B Channel 0 is triggered by Event Channel 0


' DMA Interrupt
On Dma_ch0 Dma_ch0_int                                    'Interrupt will be enabled
with Tci = XX in Config DMAX
Config Dma = Enabled , Doublebuf = Disabled , Cpm = Rr    ' enable DMA, Double Buffer
disabled

' DMA Channel 0 is used here
Config Dmach0 = Enabled , Burstlen = 2 , Chanrpt = Enabled , Tci = Lo , Eil = Lo ,
Singleshot = Enabled , _
  Sar = Transaction , Sam = Inc , Dar = Burst , Dam = Inc , Trigger = &H25 , Btc = 8192 ,
Repeat = 0 , Sadr = Varptr(channel_0(1)) , Dadr = Varptr(dacb_ch0data1)

' Trigger = &H25 (DAC B Base Level Trigger) + Channel 0 = &H00 --> &H25
' Burstlen is 2 byte because the DAC output value is a 12-Bit value you need to transfer 2
byte
' Source address (the array) is incremented until all bytes transfered (8192 byte)
' Destination address (DAC B Channel 0) is incremented once to transfer the low byte and
high byte of the 12-bit value
' BTC = 8192 BYTE (needed to transfer the 4096 word)
' Reapeat = 0 --> repeat forever


Enable Interrupts


'Frequency of output signal = 32MHz/32 = 1MHz --> 1MHz/4096 (Sample_Count) = appx. 244Hz

Do


Loop

End                                                       'end program

'-----------------------[Interrupt Service Routines]-----------------------------

 ' Dma_ch0_int is for DMA Channel ERROR Interrupt A N D for TRANSACTION COMPLETE Interrupt
 ' Which Interrupt fired must be checked in Interrupt Service Routine
   Dma_ch0_int:

    If Dma_intflags.0 = 1 Then                            'Channel 0 Transaction
Interrupt Flag
       Set Dma_intflags.0                                 'Clear the Channel 0
Transaction Complete flag
       Set Dma_ready
     End If

    If Dma_intflags.4 = 1 Then                            'Channel 0 ERROR Flag
       Set Dma_intflags.4                                 'Clear the flag
       Set Dma_channel_0_error                            'Channel 0 Error
    End If

 Return
```

7.21.20 CONFIG DACX

Action

This statement configures the DAC0 or DACX in the Xtiny/Megax/AVRX.

Syntax

CONFIG DACx=dac, **IO0=**IO0, **RUNMODE=**runmode, **OUT_ENABLE =**out

Remarks

DACX	Chose either DAC0 or DACX.
dac	ENABLED or DISABLED. Chose ENABLED to enable the DAC.
runmode	Possible values : ENABLED : In Standby sleep mode, the peripheral continues operation

	DISABLED : In Standby sleep mode, the peripheral is halted
Out	ENABLED or DISABLED. Chose ENABLED to enable the output. This will also set PORTA.6 to output mode. Notice that only DAC0 has the ability to drive an output pin. The output pin varies per processor. The compiler will use the proper pin.

The DAC data register is available in the byte variable DAC0_DATA.
When present, the register names for DAC1 and DAC2 are : DAC1_DATA and DAC2_DATA.
In version 2086 the WRITEDAC statement exist to write to the registers.

See also

WRITEDAC 1610

Example

```
'-----------------------------------------------------------------
----------------
'name                     : dac.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates DAC
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24

'set the system clock and prescaler
Config Sysclock = 20mhz , Prescale = 1

'configure the USART
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'configure the internal reference to be 1v1 for both the ADC and
the DAC
Config Vref = Dummy , Adc0 = 1v1 , Dac0 = 1v1

'configure the DAC. We also drive the PA6 output pin
Config Dac0 = Enabled , Out_enable = Enabled

Print "Test DAC0"
```

```
Do
   Dac0_data = Dac0_data + 10
  ' or use the WRITEDAC statement :  WRITEDAC value
   Print "DAC0:" ; Dac0_data
   Waitms 100
Loop

End
```

7.21.21 CONFIG DATE

Action

Configure the Format of the Date String for Input to and Output from BASCOM – Date functions

Syntax

CONFIG DATE = DMY , Separator = char

Remarks

DMY	The Day, month and year order. Use DMY, MDY or YMD.
Char	The character used to separate the day, month and year. Old syntax : / , - or . (dot). Preferred new syntax : MINUS, SLASH or DOT. Example: Config Date = DMY, SEPARATOR=MINUS

The following table shows the common formats of date and the associated statements.

Country	Format	Statement
American	mm/dd/yy	Config Date = MDY, Separator = SLASH
ANSI	yy.mm.dd	Config Date = YMD, Separator = DOT
Britisch/French	dd/mm/yy	Config Date = DMY, Separator = SLASH
German	dd.mm.yy	Config Date = DMY, Separator = DOT
Italian	dd-mm-yy	Config Date = DMY, Separator = MINUS
Japan/Taiwan	yy/mm/dd	Config Date = YMD, Separator = SLASH
USA	mm-dd-yy	Config Date = MDY, Separator = MINUS

When you live in Holland you would use :
CONFIG DATE = DMY, separator = MINUS
This would print 24-04-02 for 24 November 2002.

When you line in the US, you would use :
CONFIG DATE = MDY , separator = SLASH
This would print 04/24/02 for 24 November 2002.

See also

CONFIG CLOCK 895 , DATE TIME functions 1835 , DayOfWeek 1181 , DayOfYear 1190 , SecOfDay 1203 , SecElapsed 1202 , SysDay 1206 , SysSec 1204 , SysSecElapsed 1206 , Time 1209 , Date 1193

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : megaclock.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows the new TIME$ and DATE$ reserved
variables
'micro                    : Mega103
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'With the 8535 and timer2 or the Mega103 and TIMER0 you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code

'This example is written for the STK300 with M103
Enable Interrupts

'[configure LCD]
$lcd = &HC000                                               'address for
E and RS
$lcdrs = &H8000                                             'address for
only E
Config Lcd = 20 * 4                                         'nice
display from bg micro
Config Lcdbus = 4                                           'we run it
in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus                                        'tell about
the bus mode

'[now init the clock]
Config Date = Mdy , Separator = SLASH                           ' ANSI-
Format

Config Clock = Soft                                         'this is how
simple it is
'The above statement will bind in an ISR so you can not use the TIMER
anymore!
'For the M103 in this case it means that TIMER0 can not be used by the
user anymore

'assign the date to the reserved date$
```

```
'The format is MM/DD/YY
Date$ = "11/11/00"

'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used

Time$ = "02:20:00"

'-----------------------------------------------------

'clear the LCD display
Cls

Do
  Home                                                    'cursor home
  Lcd Date$ ; "  " ; Time$                                'show the
date and time
Loop

'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
_day = 1
'For the _year variable only the year is stored, not the century
End
```

7.21.22 CONFIG DCF77

Action

Instruct the compiler to use DCF-77 radio signal to get atom clock precision time

Syntax

CONFIG DCF77 = pin , timer = timer **[** INVERTED=inv, CHECK=check, UPDATE=upd, UPDATETIME=updtime , TIMER1SEC=tmr1sec, SWITCHPOWER=swpwr, POWERPIN=pin, POWERLEVEL = pwrlvl , SECONDTICKS=sectick ,DEBUG=dbg , GOSUB = Sectic , PULSE=pulse **]**

Remarks

PIN	The input pin that is connected to the DCF-77 signal. This can be any micro processor pin that can be used as an input.
TIMER	The timer that is used to generate the compare interrupts, needed to determine the level of the DCF signal. Supported timers are : TIMER1. For Xmega : TCC0,TCC1,TCE0,TCE1,TCD0,TCD1,TCF0,TCF1 Xmega needs the MED priority set with CONFIG PRIORITY 1026 because the MED priority is used for the timer interrupt. For Xtiny platform : TCA0, TCA1, TCAx
INVERTED	This value is **0** by default. When you specify 1, the compiler will assume you use an inverted DCF signal. Most DCF-77 receivers have a normal output and an inverted output.
CHECK	Check is 1 by default. The possible values are : **0** - The DCF-77 parity bits are checked. No other checks are performed.

	Use it when you have exceptional signal strength **1** - The received minutes are compared with the previous received minutes. And the difference must be 1. **2** - All received values(minutes, hours, etc.) are compared with their previous received values. Only the minutes must differ with 1, the other values must be exactly the same. This value uses more internal ram but it gives the best check. Use this when you have bad signal reception.
UPDATE	Upd determines how often the internal date/time variables are updated with the DCF received values. The default value is **0**. There are 3 possible values : **0** - Continuous update. The date and time variables are updated every time the correct values have been received **1** - Hourly update. The date and time variables are updated once an hour. **2**- Daily update. The date and time variables are updated once a day. The UPDATE value also determines the maximum value of the UPDATETIME option.
UPDATETIME	This value depends on the used UPDATE parameter. When UPDATE is 1, the value must be in the range from 0-59. Start every hour at this minute with the new update. When UPDATE is 2, the value must be in the range from 0-23. Start every day at this hour with the new update. The default is **0**.
TIMER1SEC	16 bit timers with the right crystal value can generate a precise interrupt that fires every second. This can be used to synchronize only once a day or hour with the DCF values. The remaining time, the 1-sec interrupt will update the soft clock. By default this value is 0.
SWITCHPOWER	This option can be used to turn on/off the DCF-77 module with the control of a port pin. The default is **0**. When you specify a value of **1** , the DCF receiver will be switched off to save power, as soon as the clock is synchronized.
POWERPIN	The name of a pin like pinB.2 that will be used to turn on/off the DCF module.
POWERLEVEL	This option controls the level of the output pin that will result in a power ON for the module. 0 - When a logic 0 is applied to the power pin, the module is ON. 1 - When a logic 1 is applied to the power pin, the module is ON. Use a transistor to power the module. Do not power it from a port PIN directly. When you do power from a pin, make sure you sink the current. Ie : connect VCC to module, and GND of the module to ground. A logic 0 will then turn on the module.
SECONDTICKS	The number of times that the DCF signal state is read. This is the number of times per second that the interrupt is executed. This value is calculated by the compiler. The highest possible timer pre scale value is used and the lowest possible number of times that the interrupt is executed. This gives least impact on your main application. You can override the value by defining your own value. For example when you want to run some own code in the interrupt and need it to execute more often.
DEBUG	Optional value to fill 2 variables with debug info. DEBUG is on when a value of 1 is specified. By default, DEBUG is off. This has nothing to do with other DEBUG options of the compiler, it is only for the

	DCF77 code! When 1 is specified the compiler will create 2 internal variable named : bDCF_Pause and bDCF_Impuls. These values contain the DCF pulse length of the pause and the impulse. In the sample these values are printed.
GOSUB	The Sectic option will call a label in the main program every second. You have to insert this label yourself. You must also end it with a RETURN. The option is the same as used with CONFIG CLOCK 895
PULSE	This is an optional parameter that sets the pulse time in mS. The default is 150. When you have hardware that requires a shorter or longer pulse you can try a slightly higher or lower value. At all times you should use a value between 100 and 200 where 150 would be the optimum value.

The DCF decoding routines use a status byte. This byte can be examined as in the example.
The bits have the following meaning.

Bit	Explanation
0	The last reading of the DCF pin.
1	This bit is reserved.
2	This Bit is set, if after a complete time-stamp at second 58 the time-stamp is checked and it is OK. If after a minute mark (2 sec pause) this bit is set, the time from the DCF-Part is copied to the Clock-Part and this bit reset too. Every second mark also resets this bit. So time is only set, if after second 58 a minute mark follows. Normally this bit is only at value 1 from Second 58 to second 60/00.
3	This Bit indicates, that the DCF-Part should be stopped, if time is set. (at the option of updating once per hour or day).
4	This Bit indicated that the DCF-Part is stopped.
5	This bit indicates, that the CLOCK is configured the way, that during DCF-Clock is stopped, there is only one ISR-Call in one second.
6	This Bit determines the level of the DCF input-pin at the pulse (100/200 mSec part).
7	This bit indicates, that the DCF-Part has set the time of the Clock-part.

See Also

DCF77TIMEZONE 1210


You can read the Status-Bit 7 (DCF_Status.7), to check whether the internal clock was synchronized by the DCF-Part. You can also reset this Bit with RESET 1428 DCF_Status.7. The DCF-Part will set this bit again, if a valid time-stamp is received. You can read all other bits, but don't change them.

The DCF-77 signal is broadcasted by the German Time and Frequency department. The following information is copied from : http://www.ptb.de/en/org/4/44/_index.htm

The main task of the department time and frequency is the realization and dissemination of the base unit time (second) and the dissemination of the legal time in the Federal Republic of Germany.

The second is defined as the duration of 9 192 631 770 periods of the radiation

corresponding to the transition between the two hyper fine levels of the ground state of the cesium-133 atom.

For the realization and dissemination of the unit of time, the department develops and operates cesium atomic clocks as primary standards of time and frequency. In the past decades, these, as the worldwide most accurate atomic clocks, have contributed to the international atomic time scale (TAI) and represent the basis for the legal time in Germany. Dissemination of the legal time to the various users in industry, society, and research is performed via satellite, via a low frequency transmitter DCF77 and via an internet- and telephone service.

The department participates in the tests for the future European satellite navigation system „Galileo".

Presently the primary clocks realizing the time unit are augmented by Cs clocks with laser cooled atoms („Cs-fountain clocks") whose accuracy presently exceeds the clocks with thermal beams by a factor of 10 (frequency uncertainty of 1 . 10-15).

Future atomic clocks will most likely be based on atomic transitions in the optical range of single stored ions. Such standards are presently being developed along with the means to relate their optical frequencies without errors to radio-frequencies or 1 second pulsed.

As one may expect transitions in nuclei of atoms to be better shielded from environmental perturbations than electron-shell transitions which have been used so far as atomic clock references, the department attempts to use an optical transition in the nucleus of 229Th for a future generation of atomic clocks.

The work of the department is complemented by research in nonlinear optics (Solitons) and precision time transfer techniques, funded in the frame of several European projects and by national funding by Deutsche Forschungsgemeinschaft particularly in the frame of Sonderforschungsbereich 407 jointly with Hannover University.

The following information is copied from wikipedia : http://en.wikipedia.org/wiki/DCF77

The signal can be received in this area:

DCF77 is a long wave time signal and standard-frequency radio station. Its primary and backup transmitter are located in Mainflingen, about 25 km south-east of Frankfurt, Germany. It is operated by T-Systems Media Broadcast, a subsidiary of

Deutsche Telekom AG, on behalf of the Physikalisch-Technische Bundesanstalt, Germany's national physics laboratory. DCF77 has been in service as a standard-frequency station since 1959; date and time information was added in 1973.

The 77.5 kHz carrier signal is generated from local atomic clocks that are linked with the German master clocks in Braunschweig. With a relatively-high power of 50 kW, the station can be received in large parts of Europe, as far as 2000 km from Frankfurt. Its signal carries an amplitude-modulated, pulse-width coded 1 bit/s data signal. The same data signal is also phase modulated onto the carrier using a 511-bit long pseudo random sequence (direct-sequence spread spectrum modulation). The transmitted data repeats each minute
Map showing the range of the DCF77 signal.
Map showing the range of the DCF77 signal.

* the current date and time;
* a leap second warning bit;
* a summer time bit;
* a primary/backup transmitter identification bit;
* several parity bits.

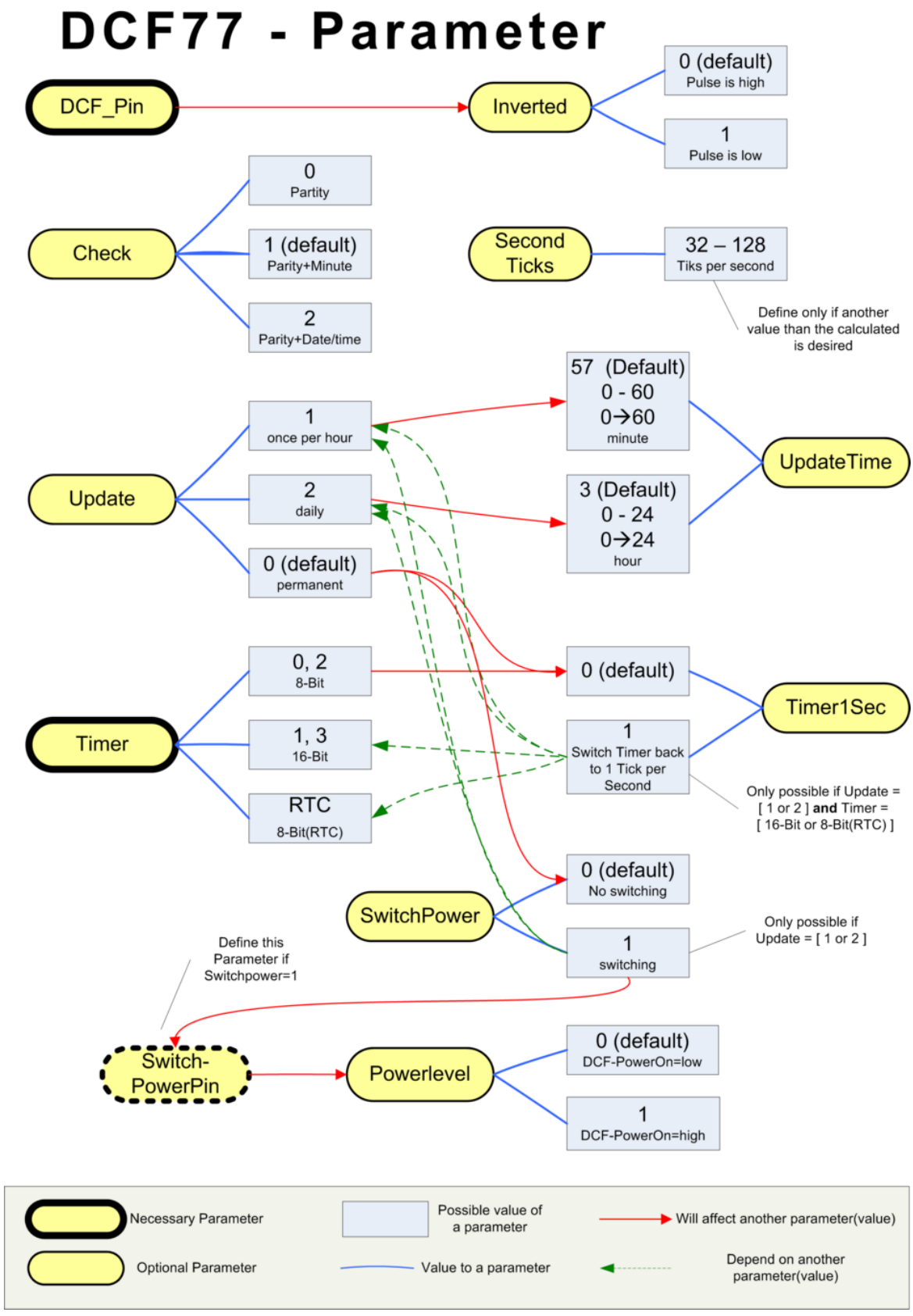
Since 2003, 14 previously unused bits of the time code have been used for civil defence emergency signals. This is still an experimental service, aimed to replace one day the German network of civil defense sirens.

The call sign stands for D=Deutschland (Germany), C=long wave signal, F=Frankfurt, 77=frequency: 77.5 kHz. It is transmitted three times per hour in morse code.

Radio clocks have been very popular in Europe since the late 1980s and most of them use the DCF77 signal to set their time automatically.

For further reference see wikipedia, a great on line information resource.

The DCF library parameters state diagram looks as following:



If the SECTIC option is used, the Sectic Interrupt routine should not need more time, than to the next timer interrupt. If you use a timer for dcf (and softclock) usually with 40 tics per second, the Sectic routine should take only less than 25msec.
If the Sectic routines needs more than this limit, you will lose accuracy of the

softclock time (especially during the time, where the clock is not synchronized by DCF) and also measurement of the length of the DCF-pulses.
If the SECTIC routine needs more time than the short DCF-pulse (100ms, with some instability in DCF-receiver may be 80ms) you will lose synchronization with the DCF-signal.
It is the principle of the DCF-routine, that the timer-interrupt measures the DCF-Pulse length and if you need more time in the interrupt routine as the duration from one timer interrupt to the next, you will get a problem.
Thus keep the SECTIC routine as short as possible and set a flag in the SECTIC routine, which is checked in a loop of the main-program.

See also
CONFIG DATE 925

ASM
_DCF77 from DCF77.LBX is included by the compiler when you use the CONFIG statement.

Example
```
$regfile = "M88def.dat"
$crystal = 8000000

$hwstack = 128
$swstack = 128
$framesize = 128

$baud = 19200

'Config Dcf77 = Pind.2 , Debug = 1 , Inverted = 0 , Check = 2 , Update =
0 , Updatetime = 30 , Switchpower = 0 , Secondticks = 50 , Timer1sec = 1
, Powerlevel = 1 , Timer = 1
Config Dcf77 = Pind.2 , Timer = 1 , Timer1sec = 1 , Debug = 1

Enable Interrupts
Config Date = Dmy , Separator = .

Dim I As Integer
Dim Sec_old As Byte , Dcfsec_old As Byte

Sec_old = 99 : Dcfsec_old = 99                                    ':
DCF_Debug_Timer = 0

' Testroutine für die DCF77 Clock
Print "Test DCF77 Version 1.00"
Do
   For I = 1 To 78
      Waitms 10
      If Sec_old <> _sec Then
         Exit For
      End If
      If Dcfsec_old <> Dcf_sec Then
         Exit For
      End If
   Next
   Waitms 220
   Sec_old = _sec
   Dcfsec_old = Dcf_sec
```

```
   Print Time$ ; " " ; Date$ ; " " ; Time(dcf_sec) ; " " ; Date(dcf_day)
 ; " " ; Bin(dcf_status) ; " " ; Bin(dcf_bits) ; " " ; Bdcf_impuls ; " "
 ; Bdcf_pause
Loop
End
```

Example Xtiny

```
'-----------------------------------------------------------
'              (c) 1995-2025 MCS Electronics
'    DCF 77 demo to demonstrate the DCF77 library from Josef
Vögel
'-----------------------------------------------------------
$regfile = "avrx64da64.dat"
$crystal = 24000000

$hwstack = 128
$swstack = 128
$framesize = 128

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz

'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'set up the COM por/USART connected to PA0 and PA1
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1


Config Dcf77 = PINB.2 , Timer = Tca0 , Debug = 1 , Check = 1 ,
Gosub = Sectic

Enable Interrupts
Config Date = Dmy , Separator =DOT


Dim I As Integer
Dim Sec_old As Byte , Dcfsec_old As Byte

Sec_old = 99 : Dcfsec_old = 99

' Testroutine für die DCF77 Clock
Print "Test DCF77 Version 1.02"

Print "Configuration"


Do
   For I = 1 To 78
```

```
      Waitms 10
      If Sec_old <> _sec Then
         Exit For
      End If
      If Dcfsec_old <> Dcf_sec Then
         Exit For
      End If
   Next
   Waitms 220
   Sec_old = _sec
   Dcfsec_old = Dcf_sec
   Print Time$ ; " " ; Date$ ; " " ; Time(dcf_sec) ; " " ; Date(
dcf_day) ; " " ; Bin(dcf_status) ; " " ; Bin(dcf_parity) ; " " ;
Bin(dcf_bits) ; " " ; Bdcf_impuls ; " " ; Bdcf_pause       '; " "
; db1 ; " " ; db2
   If Dcf_sec > 45 Then
      Reset Dcf_status.7
   End If
   Print "Timezone : " ; Dcf77timezone()
Loop


'optional, is called every second by the library
Sectic:
  !nop
Return


End
```

7.21.23 CONFIG DEBOUNCE

Action

Configures the delay time for the DEBOUNCE statement.

Syntax

CONFIG DEBOUNCE = time

Remarks

Time	A numeric constant which specifies the delay time in mS. The maximum delay is 65535.

When debounce time is not configured, 25 mS will be used as a default.

See also

DEBOUNCE 1212

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : deboun.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates DEBOUNCE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Config Debounce = 30                                       'when the
config statement is not used a default of 25mS will be used


  'Debounce Pind.0 , 1 , Pr 'try this for branching when high(1)
  Debounce Pind.0 , 0 , Pr , Sub
  Debounce Pind.0 , 0 , Pr , Sub
  '                    ^----- label to branch to
  '               ^---------- Branch when P1.0 goes low(0)
  '         ^---------------- Examine P1.0

  'When Pind.0 goes low jump to subroutine Pr
  'Pind.0 must go high again before it jumps again
  'to the label Pr when Pind.0 is low

  Debounce Pind.0 , 1 , Pr                                 'no branch
  Debounce Pind.0 , 1 , Pr                                 'will result
in a return without gosub
End

Pr:
  Print "PIND.0 was/is low"
Return
```

7.21.24 CONFIG DMA

Action

Configures the direct memory access (DMA) module of the XMEGA.

Syntax

CONFIG DMA=enabled|disabled, DOUBLEBUF=db, CPM=cpm

Remarks

DMA	By default the DMA is disabled. Use ENABLED to enable the module.
db	**DOUBLE BUFFER** This options will set the double buffer mode. By default is is DISABLED. To allow for continuous transfer, two channels can be interlinked so that the second takes over the transfer when the first is finished and vice versa. This is called double buffering. When a transmission is completed for the first channel, the second channel is enabled. When a request is detected on the second channel, the transfer starts and when this is completed the first channel is enabled again Modes : - DISABLED : No double buffer enabled - CH01 : Double buffer enabled on channel0/1 - CH23 : Double buffer enabled on channel2/3 - CH01CH23 : Double buffer enabled on channel0/1 and channel2/3
cpm	**Channel Priority Mode** If several channels request data transfer at the same time a priority scheme is available to determine which channel is allowed to transfer data. Application software can decide whether one or more channels should have a fixed priority or if a round robin scheme should be used. A round robin scheme means that the channel that last transferred data will have the lowest priority Modes : RR : Round Robin CH0RR123 : Channel0 > Round Robin (Channel 1, 2 and 3) CH01RR23 : Channel0 > Channel1 > Round Robin (Channel 2 and 3) CH0123 : Channel0 > Channel1 > Channel2 > Channel3

You also need to set the individual DMA channels using CONFIG DMACHx.

See also
CONFIG DMACHx 937 , START DMACHx 1538 , CONFIG EDMA 944 , CONFIG EDMAx 945

Example
See CONFIG DMACHx 937

7.21.25 CONFIG DMACHx

Action
Configures the direct memory access (DMA) channel of the XMEGA.

Syntax
CONFIG DMACHx=enabled|disabled,BURSTLEN=bl, CHANRPT=chrpt, CTR=ctr, SINGLESHOT=ss, TCI=tci, EIL=eil,SAR=sar, SAM=sam,DAR=dar,DAM=dam, TRIGGER,trig, BTC=btc, REPEAT=rpt,SADR=sadr, DADR=dadr

Remarks
In order to understand the various options better, we first have a better look at DMA.

Normally, when you want to transfer data, the processor need to execute a number of operations.
The BASCOM MEMCOPY for example will use processor instructions like LD (load data) and ST(store data) in a loop.
If you want to clear 32KB of memory you need at least 32 K instructions. This will consume time, and all this time the processor can not handle other tasks.
In a PC, you do not want to use the processor to be busy when you load a file from disk. The DMA controller will handle this. It can move blocks of memory between devices.

You can also send for example an array in SRAM to an USART over DMA so the processor will not be busy handling the transfer from the Array to the USART. See also the example below.

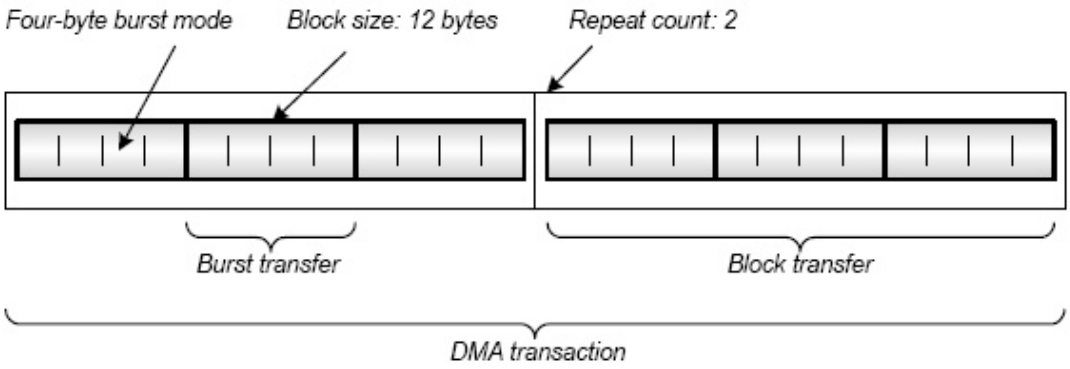
There is also an example to receive bytes over USART to SRAM in the Bascom-AVR/ Samples folders.

Before **CONFIG DMACHx** can be used you need to use **Config** Dma (CONFIG DMA 936)

DMA Transaction
A complete DMA read and write operation between memories and/or peripherals is called a DMA transaction.
A transaction is done in data blocks and the size of the transaction (number of bytes to transfer) is selectable from software and controlled by the block size and repeat counter
settings. Each block transfer is divided into smaller bursts

Block Transfer and Repeat
The size of the block transfer is set by the Block Transfer Count Register, and can be anything from 1 byte to 64 KBytes.
A repeat counter can be enabled to set a number of repeated block transfers before a transaction is complete. The repeat is from 1 to 255 and unlimited repeat count can be achieved by
setting the repeat count to zero.

Burst Transfer
As the AVR CPU and DMA controller use the same data buses a block transfer is divided into smaller burst transfers. The burst transfer is selectable to 1, 2, 4, or 8 bytes.
This means that, if the DMA acquires a data bus and a transfer request is pending it will occupy the bus until all bytes in the burst transfer is transferred.
A bus arbiter controls when the DMA controller and the AVR CPU can use the bus. The CPU always has priority, so as long as the CPU request access to the bus, any pending burst transfer
must wait. The CPU requests bus access when it executes an instruction that write or read data to SRAM, I/O memory, EEPROM and the External Bus Interface



DMACHx	There are 4 DMA channels numbered 0-3. By default these DMA channels are disabled. Use ENABLED to enable the channel.
bl	**BURSTLEN** Each DMA channel has an internal transfer buffer that is used for 2, 4 and 8 byte burst transfers. When a transfer is triggered, a DMA channel will wait until the transfer buffer contains two bytes before the transfer starts. For 4 or 8 byte transfer, any remaining bytes is transferred as soon as they are ready for a DMA channel. The buffer is used to reduce the time the DMA controller occupy the bus. Options : - 1 : 1 byte burst mode - 2 : 2 byte burst mode - 4 : 4 byte burst mode - 8 : 8 byte burst mode
chanrpt	**Channel Repeat** Setting this bit enables the repeat mode. In repeat mode, this bit is cleared by hardware in the beginning of the last block transfer. The REPCNT register should be configured before setting the REPEAT bit. When using the CONFIG command, the compiler will handle this. Options : Enabled : enabled repeat mode Disabled : disabled repeat mode
ctr	**DMA Channel Transfer Request** Setting this bit requests a data transfer on the DMA Channel. This bit is automatically cleared at the beginning of the data transfer Options : Enabled : request transfer
ss	**DMA Channel Single Shot Data transfer** Setting this bit enables the single shot mode. The channel will then do a burst transfer of BL bytes on the transfer trigger. This bit can not be changed if the channel is busy. Options : Enabled : enable SS mode.
tci	**DMA Channel Transaction Complete Interrupt Level** The interrupt can be turned OFF, or be given a priority LO, MED or HI
eil	**DMA Channel Error Interrupt Level** The interrupt can be turned OFF, or be given a priority LO, MED or HI
sar	**Source Address Reload** The channel source address can be reloaded the following way:

	NONE : No reload performed. BLOCK : DMA source address register is reloaded with initial value at end of each block transfer. BURST : DMA source address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA source address register is reloaded with initial value at end of each transaction.
sam	**Source Address Mode** The source address can be altered the following way : FIXED : The source address remains the same. INC : The source address is incremented by one. DEC : The source address is decremented by one. If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte.
dar	**Channel Destination Address Reload** The channel destiny address can be reloaded the following way: NONE : No reload performed. BLOCK : DMA destiny address register is reloaded with initial value at end of each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction.
dam	**Destiny Address Mode** The destiny address can be altered the following way : FIXED : The destiny address remains the same. INC : The destiny address is incremented by one. DEC : The destiny address is decremented by one. If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the destiny address is increased after each byte. In case of a byte array it would start with array(1) and the next byte would be array(2) which will be transferred and so on.
trigger	**Trigger Source Select** The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can manual start a DATA TRANSFER with START DMACHx statement. You can find the hardware trigger values in the datasheet. For example, EVENTSYS channel 0 would be 1. And EVENSTYS channel 1 would be 1. In case of for example an USART you need to add the base value and add an offset. Example: Base value for USARTC0 is &H4B Offset for (RXC) Receive complete is &H00 Offset for (DRE) Data Register Empty is &H01 So when you want to use the DRE the trigger is &H4B + &H01 = &H4C
btc	**Block Transfer Count** The BTC represents the 16-bit value TRFCNT. Which also means the max

	value is 64Kbyte. TRFCNT defines the number of bytes in a block transfer. The value of TRFCNT is decremented after each byte read by the DMA channel. When TRFCNT reaches zero, the register is reloaded with the last value written to it. When repeat is 1, this is the total amount of bytes to send in the DMA transaction.
repeat	**Repeat Counter Register** REPCNTcounts how many times a block transfer is performed. For each block transfer this register will be decremented. Unlimited repeat is activated by setting this register to 0.

sadr	**Source Address**
	This is the address of the DMA source. For example, the address of a variable. Or the address of a register. Use VARPTR 1604() to find the address of a variable.
	For example if the source address is an array:
	`sadr = varptr(ar(1))`
	For example if the source address is an hardware address like from an USART:
	`sadr = Varptr(usarte0_data)`
	
	or ADC A Channel 0:
	`Sadr = Varptr(adca_ch0_res)`

dadr	**Destination Address**
	The destiny address.
	This can be also for example an array in SRAM:
	`dadr = varptr(dest(1))`
	This can be also for example a hardware recourse like USART:
	`Dadr = Varptr(usarte0_data)`
	
	or for example for DAC B Channel 0:
	`Dadr = Varptr(dacb_ch0datal)`

After you have configured the DMA channel, you can start the transfer with the START DMACHx statement.
This will write the TRFREQ bit in the CTRLA register.

Setting the TRFREQ Bit (DMA Channel Transfer Request) requests a DATA TRANSFER on the DMA channel.
Setting this bit requests a data transfer on the DMA Channel. This bit is automatically cleared at
the beginning of the data transfer.

To enable the DMA Channel you need to set the Dma_chX_ctrla.7 bit.
For example for DMA Channel 0 this is **Set** Dma_ch0_ctrla.7
Setting this bit enables the DMA channel. This bit is automatically cleared when the transaction
is completed.

See also

CONFIG DMA 936, START DMACHx 1538, ATXMEGA 425, CONFIG EDMA 944, CONFIG EDMAx 945

Example (copy SRAM Array to another SRAM Array

over DMA):

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                   xm128A1-DMA.bas
'  This sample demonstrates DMA with an Xmega128A1
'-----------------------------------------------------------------
$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled 'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 38400 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

dim ar(100) as byte, dest(100) as byte,j as byte ,w as word

for j=1 to 100
  ar(j)=j ' create an array and assign a value
next

print "DMA DEMO"
config dma= enabled, doublebuf=disabled,cpm = RR ' enable DMA

'you can configure 4 DMA channels
config dmach0=enabled ,burstlen=8,chanrpt=enabled, tci=off,eil=off, sar=
none,sam=inc,dar=none,dam=inc ,trigger=0,btc=100 ,repeat =1,sadr=varptr(
ar(1)),dadr=varptr(dest(1))

start dmach0 ' this will do a manual/software DMA transfer, when
trigger<>0 you can use a hardware event as a trigger source

for j=1 to 50
  print j;"-";ar(j);"-";dest(j)  ' print the values
next
end
```

Example (send an array to USART over DMA):

```
'Terminal Output of following example:
'(

----- Array to USART over DMA -----

Hello Bascom
Hello  XMEGA

')

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
```

```
Config Priority = Static , Vector = Application , Lo = Enabled

' DMA Interrupt
On Dma_ch0 Dma_ch0_int
'Interrupt will be enabled with  Tci = XX in Config DMAX

Config Com5 = 38400 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Open "COM5:" For Binary As #5

Dim My_array(15) As Byte
Dim My_string As String * 14 At My_array(1) Overlay
Dim Dma_ready As Bit
Dim Dma_channel_0_error As Bit

Enable_dmach0 Alias Dma_ch0_ctrla.7                          'Enable DMA
Channel 0

Print #5 ,
Print #5 , "----- Array to USART over DMA -----"
Print #5 ,
Config Dma = Enabled , Doublebuf = Disabled , Cpm = Rr       ' enable DMA
'configure  DMA channel
Config Dmach0 = Disabled , Burstlen = 1 , Chanrpt = Disabled , Tci = Lo
, Eil = Off , Singleshot = Enabled , Sar = Transaction , _
 Sam = Inc , Dar = None , Dam = Fixed , Trigger = &H8C , Btc = 14 ,
Repeat = 0 , Sadr = Varptr(my_array(1)) , Dadr = Varptr(usarte0_data)
' BURSTLEN = 1
' Tci = Lo , Eil = Off --> enable TRANSACTION COMPLETE Interrupt
' Singleshot = Enabled --> Setting this bit enables the single shot
mode.
' The channel will then do a burst transfer of BL bytes on the transfer
trigger.
' SAR (Source Address Reload) = After each transaction
' SAM  = inc --> source address is increased after each byte
' DAR =  NONE --> No reload performed
' DAM (Destiny Address Mode) --> Fixed --> The address remains the same
' Trigger = &H8C --> Base Value of USARTE0 = &H8B + Offset for DRE (Data
Register Empty)= 1 --> &H8C
' BTC = 14 -->  Block Transfer Count is 14 Byte
' We start with Dmach0 = Disabled --> will be enabled when we need it

' Start dmach0 --> will set the TRFREQ Bit (DMA Channel Transfer
Request).
' Setting this bit requests a DATA TRANSFER on the DMA channel.

' We use here    Enable_dmach0 Alias Dma_ch0_ctrla.7 This bit is
automatically cleared when the DMA TRANSACTION is completed

Enable Interrupts
My_string = "Hello Bascom" + Chr(13) + Chr(10)               ' Hello
Bascom + Carriage Return + Line Feed

Set Enable_dmach0                                            ' Enable the
DMA Channel 0 (This bit is automatically cleard when transaction is
completed)

Bitwait Dma_ready , Set                                      ' Wait until
first DMA transaction is ready (DMA TRANSACTION COMPLETE Interrupt)
Reset Dma_ready

My_string = "Hello  XMEGA" + Chr(13) + Chr(10)

Set Enable_dmach0                                            ' Enable the
```

```
DMA Channel 0 (This bit is automatically cleard when transaction is
completed)

End

'----------[Interrupt Service
Routines]-------------------------------------

' Dma_ch0_int is for DMA Channel ERROR Interrupt A N D for TRANSACTION
COMPLETE Interrupt
' Which Interrupt fired must be checked in Interrupt Service Routine

Dma_ch0_int:                                                    ' DMA
Transaction complete

  If Dma_intflags.0 = 1 Then                                    ' Channel 0
Transaction Interrupt Flag
     Set Dma_intflags.0                                         ' Clear the
Channel 0 Transaction Complete flag
     Set Dma_ready
  End If

'(
  If Dma_intflags.4 = 1 Then                                    ' Channel 0
ERROR Flag
    Set Dma_intflags.4                                          ' Clear the
flag
    Set Dma_channel_0_error                                     ' Channel 0
Error
  End If
')

Return
```

7.21.26 CONFIG EDMA

Action

Configures the enhanced direct memory access (DMA) module of the XMEGA.

Syntax

CONFIG EDMA=enabled|disabled, DOUBLEBUF=db, CPM=cpm , CHM=chm

Remarks

DMA	By default the DMA is disabled. Use ENABLED to enable the module.
db	**DOUBLE BUFFER** This options will set the double buffer mode. By default is is DISABLED. To allow for continuous transfer, two channels can be interlinked so that the second takes over the transfer when the first is finished and vice versa. This is called double buffering. When a transmission is completed for the first channel, the second channel is enabled. When a request is detected on the second channel, the transfer starts and when this is completed the first channel is enabled again Modes : - DISABLED : No double buffer enabled - CH01 : Double buffer enabled on channel0/1 - CH23 : Double buffer enabled on channel2/3

	- CH01CH23 : Double buffer enabled on channel0/1 and channel2/3
cpm	**Channel Priority Mode** If several channels request data transfer at the same time a priority scheme is available to determine which channel is allowed to transfer data. Application software can decide whether one or more channels should have a fixed priority or if a round robin scheme should be used. A round robin scheme means that the channel that last transferred data will have the lowest priority Modes : RR : Round Robin CH0RR123 : Channel0 > Round Robin (Channel 1, 2 and 3) CH01RR23 : Channel0 > Channel1 > Round Robin (Channel 2 and 3) CH0123 : Channel0 > Channel1 > Channel2 > Channel3
chm	**Channel Mode** The channel mode selects the mode. Possible options for channel mode are : PER0123 : 4 peripheral channels 0,1,2,3 STD0 : 1 standard channel, 2 peripheral channels 2,3 STD2 : 2peripheral channels 0,1, 1 standard channel 2 STD02 : 2 standard channels 0,2

You also need to set the individual EDMA channels using CONFIG EDMACHx.

See also
CONFIG DMACHx 937 , START DMACHx 1538 , CONFIG DMA 936 , CONFIG EDMAx 945

Example
See CONFIG DMACHx 937

7.21.27 CONFIG EDMAx

Action
Configures the enhanced direct memory access (DMA) channel of the XMEGA.

Syntax
CONFIG EDMACHx=enabled|disabled,BURSTLEN=bl, CHANRPT=chrpt, CTR=ctr, SINGLESHOT=ss, TCI=tci, EIL=eil,SAR=sar, SAM=sam,DAR=dar,DAM=dam, TRIGGER,trig, BTC=btc,SADR=sadr, DADR=dadr

Remarks
In order to understand the various options better, we first have a quick look at DMA. Please consult the help topic CONFIG DMAx 937 and the atmel documentation for the EDMA.
Normally, when you want to transfer data, the processor need to execute a number of operations.
The BASCOM MEMCOPY for example will use processor instructions like LD (load data) and ST(store data) in a loop.
If you want to clear 32KB of memory you need at least 32 K instructions. This will

consume time, and all this time the processor can not handle other tasks.
In a PC, you do not want to use the processor to be busy when you load a file from disk. The EDMA controller will handle this. It can move blocks of memory between devices while the processor performs other tasks.

You can also send for example an array in SRAM to an USART over EDMA so the processor will not be busy handling the transfer from the Array to the USART.

There is also an example to receive bytes over USART to SRAM in the Bascom-AVR/ Samples folders.

Before **CONFIG EDMACHx** can be used you need to use **Config** EDMA (CONFIG DMA 944)

DMACHx	There are 4 DMA channels numbered 0-3. By default these DMA channels are disabled. Use ENABLED to enable the channel.
bl	**BURSTLEN** Each DMA channel has an internal transfer buffer that is either 1 or 2 byte long. The buffer is used to reduce the time the DMA controller occupy the bus. Options : - 1 : 1 byte burst mode - 2 : 2 byte burst mode
chanrpt	**Channel Repeat** Setting this bit enables the repeat mode. In repeat mode, this bit is cleared by hardware in the beginning of the last block transfer. Options : Enabled : enabled repeat mode Disabled : disabled repeat mode
ctr	**DMA Channel Transfer Request** Setting this bit requests a data transfer on the DMA Channel. This bit is automatically cleared at the beginning of the data transfer Options : Enabled : request transfer
ss	**DMA Channel Single Shot Data transfer** Setting this bit enables the single shot mode. The channel will then do a burst transfer of BL bytes on the transfer trigger. This bit can not be changed if the channel is busy. Options : Enabled : enable SS mode.
tci	**DMA Channel Transaction Complete Interrupt Level** The interrupt can be turned OFF, or be given a priority LO, MED or HI
eil	**DMA Channel Error Interrupt Level** The interrupt can be turned OFF, or be given a priority LO, MED or HI
sar	**Source Address Reload** The channel source address can be reloaded the following way: NONE : No reload performed. BLOCK : DMA source address register is reloaded with initial value at end of each block transfer. BURST : DMA source address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA source address register is reloaded with initial

	value at end of each transaction.
sam	**Source Address Mode** The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte.
dar	**Channel Destination Address Reload** The channel destiny address can be reloaded the following way: NONE : No reload performed. BLOCK : DMA destiny address register is reloaded with initial value at end of each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction.
dam	**Destiny Address Mode** The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. In case of an byte array it would start with array(1) and the next byte would be array(2) which will be transferred and so on.
trigger	**Trigger Source Select** The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can manual start a DATA TRANSFER with START DMACHx statement. You can find the hardware trigger values in the datasheet. For example, EVENTSYS channel 0 would be 1. And EVENSTYS channel 1 would be 1. In case of for example an USART you need to add the base value and add an offset. Example: Base value for USARTC0 is &H4B Offset for (RXC) Receive complete is &H00 Offset for (DRE) Data Register Empty is &H01 So when you want to use the DRE the trigger is &H4B + &H01 = &H4C
btc	**Block Transfer Count** The BTC represents the 16-bit value TRFCNT. Which also means the max value is 64Kbyte. TRFCNT defines the number of bytes in a block transfer. The value of TRFCNT is decremented after each byte read by the DMA channel. When TRFCNT reaches zero, the register is reloaded with the last value written to it. When repeat is 1, this is the total amount of bytes to send in the DMA transaction.
sadr	**Source Address** This is the address of the DMA source. For example, the address of a variable. Or the address of a register. Use VARPTR 1604() to find the address of a variable.

	For example if the source address is an array:
	`sadr = varptr(ar(1))`
	For example if the source address is an hardware address like from an USART:
	`sadr = Varptr(usarte0_data)`
	
	or ADC A Channel 0:
	`Sadr = Varptr(adca_ch0_res)`

dadr	**Destination Address**
	The destiny address.
	This can be also for example an array in SRAM:
	`dadr = varptr(dest(1))`
	This can be also for example a hardware recourse like USART:
	`Dadr = Varptr(usarte0_data)`
	
	or for example for DAC B Channel 0:
	`Dadr = Varptr(dacb_ch0datal)`

After you have configured the DMA channel, you can start the transfer with the START EDMACHx statement.
This will write the TRFREQ bit in the CTRLA register.

Setting the TRFREQ Bit (DMA Channel Transfer Request) requests a DATA TRANSFER on the EDMA channel.
Setting this bit requests a data transfer on the DMA Channel. This bit is automatically cleared at
the beginning of the data transfer.

See also

CONFIG DMA 936 , START DMACHx 1538, ATXMEGA 425 , CONFIG EDMA 944

Example

```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                    xm128A1-DMA.bas
'  This sample demonstrates DMA with an Xmega32E5
'-----------------------------------------------------------------
$regfile = "xm32e5def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 38400 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

Dim Ar(100) As Byte , Dest(100) As Byte , J As Byte , W As Word
```

```
For J = 1 To 100
  Ar(j) = J                                                 ' create an
array and assign a value
Next

Print "DMA DEMO"
Config Edma = Enabled , Doublebuf = Disabled , Cpm = Rr     ' enable
DMA

'you can configure 4 DMA channels
Config Edmach0 = Enabled , Burstlen = 1 , Chanrpt = Enabled , Tci = Off
 , Eil = Off , Sar = None , Sam = Inc , Dar = None , Dam = Inc ,
Trigger = 0 , Btc = 100 , Sadr = Varptr(ar(1)) , Dadr = Varptr(dest(1))

Start Edmach0                                               ' this will
do a manual/software DMA transfer, when trigger<>0 you can use a
hardware event as a trigger source

For J = 1 To 50
  Print J ; "-" ; Ar(j) ; "-" ; Dest(j)                     ' print the
values
Next
End
```

7.21.28 CONFIG DMXSLAVE

Action

Configures the DMX-512 slave.

Syntax

CONFIG DMXSLAVE = com, Channels=nchannels, DmxStart = nstart, Store=nstore

Remarks

com	The UART you want to use for the communication with the DMX-512 bus. This depends on the micro processor. In most cases this is COM1.
Channels	A numeric constant that defines the maximum number of channels you can receive. When you like to process all DMX data, you need to use 512 since 512 is the maximum. When you make a simple device a number of 8 would be sufficient.
DmxStart	The slave starting address. This is 1 by default. You will receive data starting at address 'Start'.
Store	The number of bytes you will receive and store.

You must chose the crystal/oscillator speed in a way that 250000 baud will give no errors. Typical 4, 8 and 16 MHz will work fine.
When you want to be sure, check the compiler report. It should have 0% error.

Since the DMX slave is running in interrupt mode on the background, you must ENABLE interrupts.
The serial interrupts used, is enabled by the CONFIG DMXSLAVE command.

So how does this work? When you configure the DMXSLAVE, it will receive data in interrupt mode. It will store the data into a byte arrays named _DMX_RECEIVED
The first byte stored into this array is the value for address 'DMXSTART' : the address you defined with DMXSTART.
The number of bytes stored in the array depends on the 'STORE' setting.

Example : Config Dmxslave = Com1 , Channels = 16 , DmxStart = 3 , Store = 1
This will setup an array _DMX_RECEIVED that can hold 16 bytes. So the maximum value for STORE would be 16 too. In the example our address is 3, and we store only address 3.
We can dynamic change the DMXSTART address and the number of bytes to get !
For this purpose you can change the automatic generated internal variables _DMX_ADDRESS and _DMX_CHANNELS_TOGET
_DMX_ADDRESS defines the starting address. And _DMX_CHANNELS_TOGET defines the number of bytes to store after the address matches.

All platforms are supported.

See also
NONE

Example
```
'-----------------------------------------------------------------
'                          dmx-receive.bas
'                   (c) 1995-2025 MCS Electronics
' this sample demonstates receiving a DMX datastream in the background
'-----------------------------------------------------------------
'we use a chip with 2 UARTS so we can print some data
$regfile = "m162def.dat"
'you need to use a crystal that can generate a good 250 KHz baud
'For example 8 Mhz, 16 or 20 Mhz
$crystal = 8000000
'define the stack
$hwstack = 40
$swstack = 32
$framesize = 32


'these are the pins we use. COM1/UART1 is used for the DMX data
'          TX    RX
' COM1   PD.1   PD.0       DMX
' COM2   PB.3   PB.2       RS-232


Config Dmxslave = Com1 , Channels = 16 , DMXstart = 3 , Store = 1
'this will set up the code. an array named _dmx_channels will contain
the data
'the channels will define the size. So when you want to receive data for
8 channels, you set it to 8.
'the maximum size is 512 for retrieving all data
'START defines the starting address. By default it is 1. Thus the array
will be filled starting at address 3 in the example
'STORE defines how many bytes you want to store
'By default, 1 channel is read. But you can alter the variable
_dmx_channelels_toget to specify how many bytes you want to receive
'So essential you need to chose how many bytes you like to receive. Most
slaves only need 1 - 3 bytes. It would be a waste of space to define
more channels then,
'Then you set the slave address with the variable : _dmx_address , which
```

```
is also set by the optional [START]
'And finally you chose how many bytes you want to receive that start at
the specified address. You do this by setting the _dmx_channels_toget
variable.
'Example :
'   Config Dmxslave = Com1 , Channels = 16 , Start = 300 , Store = 4
'   this would store the bytes from address 300 - 303. the maximum would
be 315 since channels is set to 16
'   Config Dmxslave = Com1 , Channels = 8 , Start = 1 , Store = 8
'   this would store the bytes from address 1 - 8. the maximum would be
8 since channels is set to 8

Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Open "COM2:" For Binary As #1
Print #1 , "MCS  DMX-512 test"

'since DMX data is received in an ISR routine, you must enable the
global interrupts
Enable Interrupts


Dim J As Byte
Do
    If Inkey(#1) = 32 Then                                  ' when you
press the space bar
      For J = 1 To _dmx_channels                            ' show the
data we received
         Print #1 , _dmx_received(j) ; " " ;
      Next
      Print #1,
   Elseif Inkey(#1) = 27 Then                               'you ca
dynamic change the start address and the channels
     Input #1 , "start " , _dmx_address
     Input #1 , "channels " , _dmx_channels_toget

   End If
Loop

'typical you would read a DIP switch and use the value as the address


End
```

7.21.29 CONFIG DP

Action

This option sets the character used for the decimal point for singles and fusing.

Syntax

CONFIG DP= "dp"

Remarks

The decimal point is a dot (.) by default. The STR() and FUSING functions convert a single into a string. The fraction is separated by a dot. In a number of counties the comma is used as a separator.

Old Syntax:
Valid options are : CONFIG DP = "." and CONFIG DP = ","

New preferred Syntax:
Valid options are : CONFIG DP = DOT and CONFIG DP = COMMA

This options only sets the character for str() and fusing for singles. In your code you still need to code with a dot : var = 1234.333

See also
NONE

Example
```
CONFIG DP = ","
Dim s as single
S = 1234.56
print s
```

7.21.30 CONFIG EEPROM

Action
Setup memory mode for EEPROM in XMEGA.

Syntax
CONFIG EEPROM=mode

Remarks

mode	MAPPED, or QUICK.
	In Xmega, the EEPROM can be mapped into memory so it can be used with pointer operations such as LD,ST,LDS and STS.
	When EEPROM is mapped, EEPROM memory will start at &H1000. The advantage of mapping the EEPROM is that reading the EEPROM becomes much more simpler.
	
	When you use the BASCOM EEPROM routines, you must include this statement before you use the EEPROM.
	
	To maintain compatibility with code and other AVR chips you can still use address 0 for the EEPROM. The library will add an offset of &H1000 to the address.
	
	When you use the QUICK mode, you also use mapped mode but for read operations, the library read routine will not be used but instead the address is internally increased with &H1000 and a normal pointer operation is used.
	This allows code like :
	If SomeEEPROMvar = 10000 Then
	End If

See also
Memory usage 267

Example
```
Config Eeprom = Mapped
```

7.21.31 CONFIG ERROR

Action
Instructs the compiler to ignore one or more errors.

Syntax
CONFIG ERROR=ignore, err=ignore [err1=ignore]

Remarks
In some situations you might want to ignore an error. For example if a new version adds a certain check that was not available in a previous version you will get errors. If you ignore the error, the code will compile without errors. This will not work in any situation. Some errors can not be ignored. You should never use this option for a finished product.

See also
NONE

Example
```
Config Error = Ignore , 369 = Ignore

Lbl:

Dim Lbl As Word  ' this would generate an error 369 without the
ignore !!!
```

7.21.32 CONFIG EVENT_SYSTEM

Action
This statement configures the Xmega event routing.

Syntax
CONFIG EVENT_SYSTEM = dummy, MUX**x**=MUX, QD**x**=QD, QDI**x**=QDI, QDIRM**x**=QDIRM,DIGFLT**x**=DIGFLT
The letter X is used to indicate that a value between 0 and 7 can be used. So there is MUX0, MUX1, MUX2,MUX3 etc.

Remarks
The Event System is a set of features for inter peripheral communication. It enables the possibility

for a change of state in one peripheral to automatically trigger actions in other peripherals.
The change of state in a peripheral that will trigger actions in other peripherals is configurable in
software. It is a simple, but powerful system as it allows for autonomous control of peripherals
without any use of interrupt, CPU or DMA resources.

There are 8 multiplexers and 8 control registers. Register 0, 2 and 4 can be used for quadrature decoding.

MUX	There are 8 multiplexers, named MUX0-MUX7. The MUX is used to select an event source.There are many sources for events : NONE : disabled, default RTC_OVF : Real Timer overflow RTC_CMP : Real Timer compare match ACA_CH0 : analog comparator ACA, channel 0 ACA_CH1 : analog comparator ACA, channel 1 ACA_WIN : analog comparator ACA, window ACB_CH0 : analog comparator ACB, channel 0 ACB_CH1 : analog comparator ACB, channel 1 ACB_WIN : analog comparator ACB, window ADCA_CH0- ADCA_CH3 : ADCA channel 0-3 ADCB_CH0- ADCB_CH3 : ADCB channel 0-3 PORTA.0 - PORTA.7 : PORT A pin 0-7 PORTB.0 - PORTB.7 : PORT B pin 0-7 PORTC.0 - PORTC.7 : PORT C pin 0-7 PORTD.0 - PORTD.7 : PORT D pin 0-7 PORTE.0 - PORTE.7 : PORT E pin 0-7 PORTF.0 - PORTF.7 : PORT F pin 0-7 PRESCALER1, PRESCALER2, PRESCALER4, PRESCALER8, PRESCALER16, PRESCALER32, PRESCALER64,PRESCALER128,PRESCALER256, PRESCALER512,PRESCALER1024,PRESCALER2048,PRESCALER4096, PRESCALER8192,PRESCALER16384 : The clock divided by 1,2,4,8,16,32,64,128,256 etc. TCC0_OVF : Timer TC0 overflow TCC0_ERR : Timer TC0 error TCC0_CCA : Timer TC0 capture or compare match A TCC0_CCB : Timer TC0 capture or compare match B TCC0_CCC : Timer TC0 capture or compare match C TCC0_CCD : Timer TC0 capture or compare match D TCC1_OVF : Timer TC1 overflow TCC1_ERR : Timer TC1 error TCC1_CCA : Timer TC1 capture or compare match A TCC1_CCB : Timer TC1 capture or compare match B TCC1_CCC : Timer TC1 capture or compare match C TCC1_CCD : Timer TC1 capture or compare match D **Dito for TCD0, TCD1, TCE0, TCE1, TCF0 and TCF1**
QD	Enables or disables the quadrature decoder. Will only work on QD0,QD2 and QD4.
QDI	Enables or disables the quadrature decode index. Will only work on QDI0, QDI2 and QDI4.
QDIRM	Quadrature decode index recognition mode. This is a numeric constant

	between 0 and 3. Each value represents the 2 possible bit values for the two input signals. Will only work on QDIRM0, QDIRM2 and QDIRM4.
DIGFLT	Defines the length of digital filtering used. Events will be passed through to the event channel only when the event source has been active and sampled with the same level for a number of peripheral clock for the number of cycles as defined by DIGFLT. The number of samples is in the range from 1-8. The default is 1 sample.

See also
ATXMEGA 425

Example 1:
```
' Select PortC.0 as INPUT to event channel 0
' Digflt0 = 8 --> Enable Digital Filtering for Event Channel 0.
' The Event must be active for 8 samples in order to be passed to the
Event system
' Event Channel 1 INPUT = Timer/Counter C0 Overflow
' Event Channel 2 INPUT = Analog Input Port A Channel 0
' Event Channel 3 INPUT = Real Timer overflow
Config Event_system = Dummy , _
Mux0 = Portc.0 , Digflt0 = 8 , _
Mux1 = Tcc0_ovf , _
Mux2 = Adca_ch0 , _
Mux3 = Rtc_ovf
```

Example 2:
```
'Event Channel 7 is input for the Timer/Counter TcD1 overflow
Config Event_system = Dummy , Mux7 = Tcd1_ovf
```

Example 3:
```
' Using the Counter/Timer to count events like a falling edge on Pine.5

$regfile = "xm256a3bdef.dat"
$crystal = 32000000                                         '32MHz
$hwstack = 64
$swstack = 40
$framesize = 40

Config Osc = Disabled , 32mhzosc = Enabled                  '32MHz

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1


Config Com7 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8        'Portf.2 and Portf.3 is COM7
Open "COM7:" For Binary As #1

'Config Interrupts
Config Priority = Static , Vector = Application , Lo = Enabled , Med =
Enabled       'Enable Lo Level Interrupts

Dim Timer_overflow As Bit
```

```
Print #1 , "---Event Counting with Timer C0 over Event Channel 0 from
PINE.5----"

Config Porte.5 = Input
Config Xpin = Porte.5 , Outpull = Pullup , Sense = Falling  'enable
Pullup and reaction on falling edge
Config Event_system = Dummy , Mux0 = Porte.5 , Digflt0 = 8
'Eventchannel 0 =  PINE.5, enable digital filtering
Config Tcc0 = Normal , Prescale = E0 , Event_source = E0 , Event_action
= Capture' Normal = no waveform generation, Event Source = Event Channel
0

On Tcc0_ovf Timerd0_int
Enable Tcc0_ovf , Lo                                          'Enable
overflow interrupt in LOW Priority
Tcc0_per = 5                                                  'Interrupt
when Count > 5
Enable Interrupts

'#################MAINLOOP#################################################
########
Do

 Wait 1
 Print #1 , "TCC0_CNT = " ; Tcc0_cnt                          'Actual
Count

 If Timer_overflow = 1 Then
    Reset Timer_overflow
    Print #1 , "TCC0_OVERVLOW"                                'Print it
when Overflow Interrupt is fired
 End If

Loop
'#################MAINLOOP#################################################
########


End

Timerd0_int:
 Set Timer_overflow
Return
```

7.21.33 CONFIG EVENT_SYSTEM XTINY

Action

This statement configures the Xtiny event routing.

Syntax

CONFIG EVENT_SYSTEM = dummy, ASYNCCHx=asyncX,SYNCCHy=syncY, ASzz=asyncUserZZ,Sn=syncUserN
CONFIG EVENT_SYSTEM = dummy, ASYNCCH0=asyncX,SYNCCH00=syncY, AS00=asyncUserZZ,S0=syncUserN

Remarks

The Event System (EVSYS) enables direct peripheral-to-peripheral signaling. It allows a change in one peripheral (the Event Generator) to trigger actions in other peripherals (the Event Users) through Event
channels, without using the CPU. It is designed to provide short and predictable

response times between peripherals, allowing for autonomous peripheral control and interaction, and also for synchronized timing
of actions in several peripheral modules. It is thus a powerful tool for reducing the complexity, size, and execution time of the software.
A change of the Event Generator's state is referred to as an Event, and usually corresponds to one of the peripheral's interrupt conditions. Events can be directly forwarded to other peripherals using the
dedicated Event routing network. The routing of each channel is configured in software, including event generation and use.
Only one trigger from an Event generator peripheral can be routed on each channel, but multiple channels can use the same generator source. Multiple peripherals can use events from the same channel.
A channel path can be either asynchronous or synchronous to the main clock. The mode must be selected based on the requirements of the application.
The Event System can directly connect analog and digital converters, analog comparators, I/O port pins, the real-time counter, timer/counters, and the configurable custom logic peripheral. Events can also be generated from software and the peripheral clock.

The are 4 asynchronous event channels (ASYNCCHx) and 2 synchronous event channels(SYNCCHy).

ASYNCCHx	Async channel x where x is in the range from 0 to 3. Each channel has different generators of events. For channels 0-3 : CCL_LUT0 CCL_LUT1 AC0_OUT TCD0_CMPBCLR TCD0_CMPASET TCD0_CMPBSET TCD0_PROGEV RTC_OVF RTC_CMP Channel 0 also has the following sources: PORTA.0 - PORTA.7 UPDI Channel 1 also had the following sources : PORTB.0 - PORTB.7 Channel 2 also had the following sources : PORTC.0 - PORTB.5 Channel 4 also had the following sources : PIT_DIV8192 - PIT_DIV64
SYNCCHy	Synchroneous channely where y is in the range from 0-1. Each channel has different generators of events. For channel 0-1 : TCB0 TCA0_OVF_LUNF TCA0_HUNF TCA0_CMP0 TCA0_CMP1 TCA0_CMP2

	Channel 0 also has the following sources: PORTC.0 - PORTC.5 PORTA.0- PORTA.7 Channel 1 also has the following sources: PORTB.0 - PORTB.7
ASzz	Asynchronous user channel input selection. There are 11 channels in the range from 0-10. Each channel selects different input which is fixed for each channel. 00 : TCB0 01 : ADC0 02 : CCL_LUT0EV0 03 : CCL_LUT1EV0 04 : CCL_LUT0EV1 05 : CCL_LUT1EV1 06 : TCD0_EV0 07 : TCD0_EV1 08 : EVOUT0 09 : EVOUT1 10 : EVOUT2 The values are fixed : - OFF - SYNCCH0 - SYNCCH1 - ASYNCCH0 - ASYNCCH1 - ASYNCCH2 - ASYNCCH3
Sn	Synchronous user channel input selection. There are 2 channels in the range from 0-1. Each channel selects different input which is fixed for each channel. 00 : TCA0 01 : USART0 The values are fixed : - OFF - SYNCCH0 - SYNCCH1

See also

7.21.34 CONFIG EXTENDED_PORT

Action

Configures compiler to generate warning or error when transforming extended port register.

Syntax

CONFIG EXTENDED_PORT = WARNING|ERROR

Remarks
A lot of AVR chips have so called extended registers. When the AVR was designed the designers did not set aside enough space for the hardware registers. A number of instructions work only with the lower 32 addresses, and a number only work on registers with an address till &H3F.
SRAM memory was moved up and the space after &H5F was used for registers. These are extended registers.
For these chips, the SRAM starts at &H100 or higher.

Because INP, OUT, SBI, SBI, SBIC, SBIS, etc. will not work on these extended registers, the compiler changes this automatic when needed. When INP or OUT is used, this is not a problem. LDS or STS can be used with the same register.
But an instruction like SBIC that will test a pin , needs a temporarily register. Register R23 is used for this.

When you write your own ASM you might want to get a warning or an error. For this purpose you can use CONFIG EXTENDED_PORT.

When you use WARNING there will be a warning in the report file. When you use ERROR, you will get an error and your code will not compile.

See also
NONE

7.21.35 CONFIG FT800

Action
This compiler option is required to setup the FT8xx SPI interface.

Syntax
CONFIG FT8xx = spi [, FTSAVE=ftsave , FTDEBUG=ftdebug] , FTCS=ftcs [, FTPD = ftpd] [,FTCHIP=800|801] [,PLATFORM=platform] [,LCD_SCREEN-lcdscr] [, LCD_ROTATE=lcdrotate] [,LCD_CALIBRATION=calib]

Remarks

spi	The SPI interface used for the FT800/FT801/FT810 processor. This may be : - SPI, for normal AVR processors - SPI*, SPIC, SPID, SPIE, SPIF, for XMEGA processors. * When you use SPI on an XMEGA processor, the compiler will use SPIC, the default SPI.
ftsave	This is an optional parameter with a default of 0. The possible values are 0 and 1. With this option enabled, the parameters passed to the various FT800 routines are limited in range. For example when a parameter is expected in the range from 0-31 it would not matter if you pass 32. But limiting the range increases code. It is best to make sure yourself that you pass the proper values.
ftdebug	This is an optional parameter with a default of 0. The possible values are 0 and 1. With this option enabled, the SPI communication can be

	monitored. A label named **_FTDBG** is called in your code. So when using this option you need to insert this label into your code. You also need to DIM a byte named **ftdebug**. This byte will be filled with the parameter sent to the SPI.
	Make sure you put a RETURN after the label and save registers you use:
	
	_FTDBG
	  Pushall
	  print hex(ftdebug)
	  Popall
	RETURN

FTCS	The name of the SPI port pin connected to the CS pin of the FT800. This would be SS in most cases. This pin is set to output and to logic level 1.
FTPD	The name of the port pin connected to the PD pin of the FT800. This is an optional pin, it depends on your hardware. Gameduino2 does not require it. EVE demo boards do require this pin.
FTCHIP	The kind of FT chip. FT800 is the default and when used, FTCHIP does not need to be specified. FT801 is similar to the FT800 but has a capacitive touch screen and gestures support. Possible values : - 800 : FT800 (default) - 801 : FT801 - 810, 811,812, 813 : FT81x
PLATFORM	The used hardware platform. Default is EVE from FTDI. Possible values : - EVE (default) - GAMEDUINO2 : popular alternative FTDI hardware
LCD_SCREEN	The kind of LCD screen. Possible values : 480272 : 480x272 pixels LCD (default) 320240 : 320x240 pixels LCD 800480 : 800x480 pixels LCD 800600 : 800x600 pixels LCD As you might have noticed, the value is the same as the screen size without the x.
LCD_ROTATE	The LCD can be used in normal horizontal mode or upside down 180 degrees. Possible values : - 0 : horizontal (default) - 1 : 180 degrees rotated
LCD_CALIBRATION	The LCD requires calibration. This need to be done once but you can force calibration using LCD_CALIBRATION parameter. Possible values : - 0 : No calibration - 1 : Force Calibration (default)

The CONFIG FT800 statement will inform the compiler to use the FT800.LIB. It will also create an ALIAS for the CS and PD pins you specify.
The FT800 is controlled by the SPI interface. This means that you need to configure the SPI the usual way.

 Since FT800 was the first graphic processor, you will find FT800 mentioned in the help. But as of version 2080, there is also support for the new FT810 chip.

BASCOM FT8xx support is implemented in the following way:

- a low level communication library FT800.LIB
- ASM include macros which are located in the FT800.LIB. Unlike sub routines, the code is included and not called.
- BASCOM high level commands such as CMD32, RD8(), RD16(), etc.
- FT80x include files : an include file with the declarations (FT81x.INC) and an include file with the actual code (FT81x_FUNCTIONS.INC).

You may modify the code from the include files. The code will reveal some new options. It is important to understand these new options.
- Passing values using BYREG 1221
- Passing values using BYSTACK 1221
- CMDFTSTACK 1651

In version 2079, FT801 support is included. A number of constants are removed from the include file and are now a parameter of CONFIG FT800. These constants are : FT_PlatForm , FT_LCDscreen , FT_LcdCal , FT_RotateDisplay and FT_CHIP. These constants are remarked for reference.

Here are some sample configurations

AdamShield:
Config FT800 = Spi , Ftsave = 0 , Ftdebug = 0 , Ftcs = Portd.4 , Ftpd = Portd.3, ftChip=800, LcdScreen=480272, PlatForm=Eve

GAMEDUINO2:
Config FT800=spi , ftsave=0, ftdebug=0 , ftcs=portb.0, ftChip=800, LcdScreen=480272, PlatForm=Gameduino2

VM801P - FTDI:
Config FT800 = Spi , ftsave = 0, ftdebug = 0 , Ftcs = Portb.1 , Ftpd = Portd.4, ftChip=**801**, LcdScreen=480272, PlatForm=Eve, Lcd_Rotate=1, Lcd_Calibration=0

See also
FT800 1619 , CMDFTSTACK 1651 , CMD32 1641 , RD8 1694, RD16 1694, RD32 1695 , WR32 1706

Partial Example
```
' FT800 Gauges Application demonstrating interactive Gauges using Lines & Custom Font
' FT800 platform.
' Original code from http://www.ftdichip.com/Support/SoftwareExamples/EVE/FT_App_Gauges.zip
' Requires Bascom 2.0.7.8 or greater

$Regfile = "M328pdef.dat"
$Crystal = 8000000
$Baud = 19200
$HwStack = 90
$SwStack = 90
$FrameSize = 300
$NOTYPECHECK

Config ft800=spi , ftsave=0, ftdebug=0 , ftcs=portb.2, ftpd=portb.1

Config Base = 0
Config Submode = New
Config Spi = Hard , Interrupt = Off , Data_Order = Msb , Master = Yes , Polarity = Low , Phase = 0 , Clockrate = 4, Noss = 1
SPSR = 1  ' Makes SPI run at 8Mhz instead of 4Mhz


' Swaps Scales
```

```
' 1 = Resitive - Random
' 0 = Random   - Resistive
Const Resistive = 0

#If Resistive = 0
   Const First = 1
   Const Second = 0
#Else
   Const First = 0
   Const Second = 1
#EndIf

$Include "FT80x.inc"
$Include "FT80x_Functions.inc"

Declare Sub cs(Byval i As Byte)
Declare Function da (Byval i As Long) As Word
Declare Sub Polar(byval R As Long , Byval Th As Word)
Declare Sub Polarxy(byval R As Long , Byval Th As Word , Byref X As Long , Byref Y As Long)
Declare Sub IntroFTDI
Declare Sub Gauges
Declare Function Read_Keys() As Byte

' General Program Variables and Declarations
Dim temp_tag  As Byte
Dim ox As Long


Spiinit

if FT800_Init()=1 then end   ' Initialise the FT800

Gauges

Do

Loop
```

Remark

In the samples, **Noss = 1** is used for CONFIG SPI. This means that the SS pin must be set by the user. In case of CONFIG FT800, the compiler always set the specified pin for FTCS to output and to logic 1. When possible you should use NOSS=0 and use the dedicated SPI SS pin. But for multiple SPI devices on the bus that is not possible since you will have multiple CS pins, and in these cases you should use NOSS=1, so you can control the SS logic level.

7.21.36 CONFIG FUSES

Action

This configuration option sets the value of Xtiny,MegaX and AVRX fuses and lock bits.

Syntax

CONFIG FUSES=ON|OFF, LOCK=ON|OFF,FUSEx=f,urowx=u

Remarks

The MCS UPDI programmer can insert the current fuse values into the code. You can also create the values yourself.

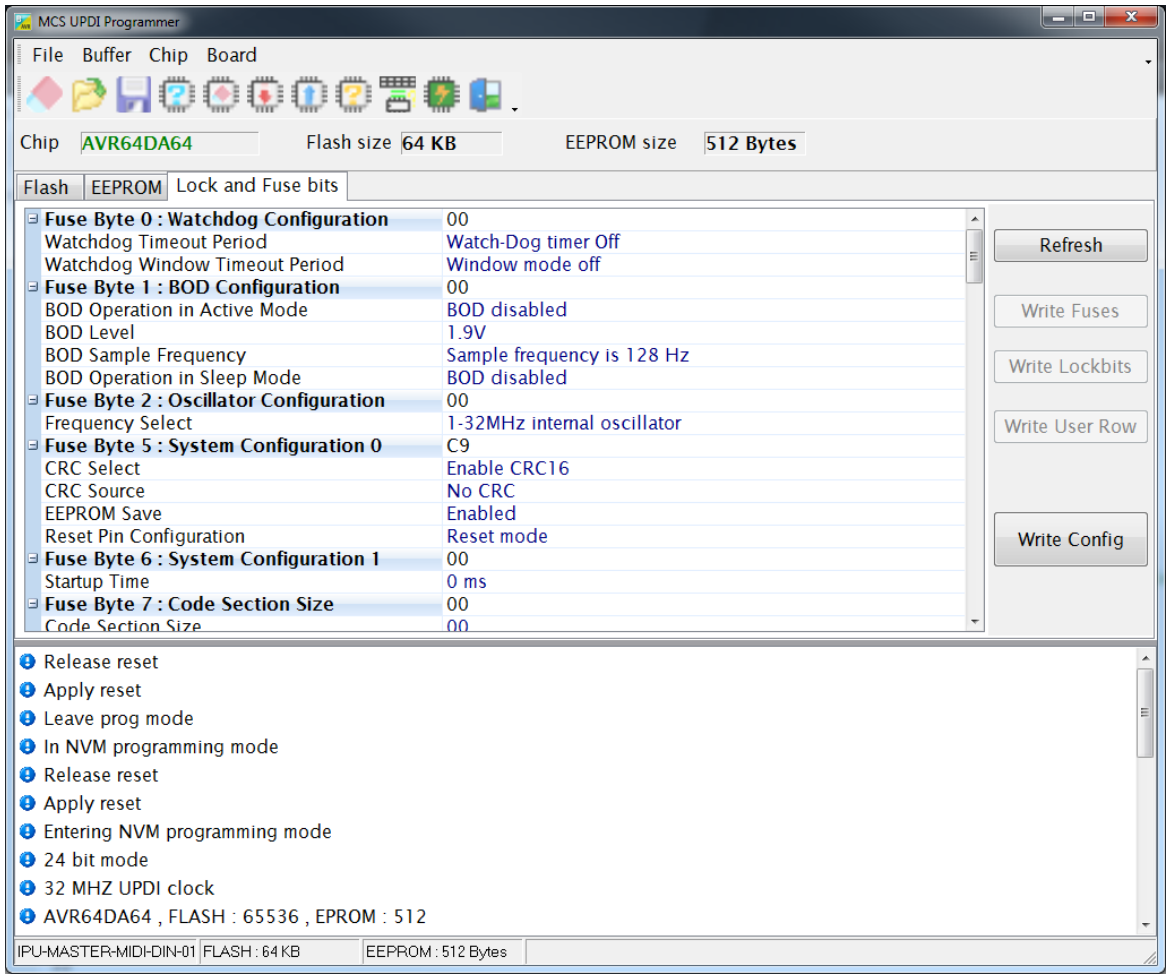
FUSES	ON or OFF. ON - When programming the specified fuses will be programmed.

	OFF- When programming there will be no automatic programming When the CONFIG FUSES is inserted the value will also be set to OFF. So you must manually change it to ON.
LOCK	ON or OFF. ON - When ON and FUSES=ON, the lock bit will be programmed when the processors is programmed. OFF- When programming the LOCK bits will be ignored. Thus the processor will not be locked. When CONFIG FUSES is inserted the value will be set to OFF. So you need to manually set it to ON when you want the processor to be locked. Please take in mind that when the processor is locked you can not program it any longer. You need to use the UNLOCK option.
FUSEx	The x is in the range from 0 to 7 depending on the processor. Some processors might have even more fuses.
f	The value the fuse is set too. This is a numeric constant.
UROWx	The x is in the range from 0 to 31 or less/more depending on the processor. Userrow fuses can be set by the user. They can be read in your code by their register.

The Xtiny/MegaX/AVRX platform has a lot of fuses.
The advised method of getting the proper CONFIG FUSES :

- set the fuses using the programmer Lock & Fuses TAB.
- do **not** set the lock byte.
- when satisfied, set the cursor at the proper place in your code
- click the WRITE CONFIG button.

For example :



This will create this line of code : Config Fuses=Off,Lock=OFF,Fuse0=&H00, Fuse1=&H00,Fuse2=&H00,Fuse5=&HC9,Fuse6=&H00,Fuse7=&H00,Fuse8=&H00

As you can see, a fuse will only be listed when the value differs from the value 255 (&HFF)

See also

$PROG 689

Example

```
Config  Fuses=Off,Lock=OFF,Fuse0=&H00,Fuse1=&H00,Fuse2=&H00,Fuse5=&HC9,
Fuse6=&H00,Fuse7=&H00,Fuse8=&H00
```

7.21.37 CONFIG GRAPHLCD

Action

Configures the Graphical LCD display.

Syntax

Config GRAPHLCD = type , DATAPORT = port, CONTROLPORT=port , CE = pin , CD = pin , WR = pin, RD=pin, RESET= pin, FS=pin, MODE = mode

Remarks

Type	This must be 240X64, 128X128, 128X64 , 160X48 , 240X128, 192X64 , SED180X32 or 192X64SED. For SED displays use 128X64sed or 120X64SED or SED180X32 For 132x132 color displays, use COLOR For EADOG128x64 use 128X64EADOGM For SSD1325 96x64 use 96X64SSD1325. See SSD1325lib 1785. For custom libs : CUSTOM. The following options are optional for custom LCD: - cols= num of cols in pixels - rows= num of rows in pixels - kind= any number to specify the lcd - lcdname="somename" , an optional name to identify the LCD
Dataport	The name of the port that is used to put the data on the LCD data pins db0-db7. PORTA for example.
Controlport	This is the name of the port that is used to control the LCD control pins. PORTC for example
Ce	The pin number that is used to enable the chip on the LCD.
Cd	The pin number that is used to control the CD pin of the display.
WR	The pin number that is used to control the /WR pin of the display.
RD	The pin number that is used to control the /RD pin of the display.
FS	The pin number that is used to control the FS pin of the display. Not needed for SED based displays.
RESET	The pin number that is used to control the RESET pin of the display.
MODE	The number of columns for use as text display. Use 8 for X-pixels / 8 = 30 columns for a 240 pixel screen. When you specify 6, 240 / 6 = 40 columns can be used.
	EADOG128M pins for SPI mode. This display only can write data. As a result, a number of graphical commands are not supported.
CS1	Chip select for EADOG128x64
A0	A0 line for EADOG128x64. This is the line that controls data/command
SI	This is the serial input pin for the EADOG128x64.
SCLK	This is the clock pin for the EADOG128x64.
	ST7565R parallel data mode A 128x64 graphical display which supports all graphic commands
dataport	The data port connected to the display. For example portJ
CS1	the chip enabled line
A0	the chip data/command mode pin
RST	the reset pin of the chip
WR	The /WR line of the chip
RD	The /RD line of the chip
C86	This pin selects the transfer mode.
PM	Some displays have this PM pin which sets the parallel mode
example	Config Graphlcd = 128 * 64eadogm ,dataport=portj, Cs1 = Porth.0 , A0 = Porth.2 , rst= Porth.1 , wr = Porth.3 , Rd = Porth.4,c86=porth.6

The first graphical LCD chip supported was T6963C. There are also drivers for other

LCD's such as SED and KS0108. The most popular LCD's will be supported with a custom driver.

The following connections were used for the T6963C:

```
PORTA.0 to PORTA.7 to DB0-DB7 of the LCD
PORTC.5 to FS, font select of LCD
PORTC.2 to CE, chip enable of LCD
PORTC.3 to CD, code/data select of LCD
PORTC.0 to WR of LCD, write
PORTC.1 to RD of LCD, read
PORTC.4 to RESET of LCD, reset LCD
```

The LCD used from www.conrad.de needs a negative voltage for the contrast.

Two 9V batteries were used with a pot meter.
Some displays have a Vout that can be used for the contrast(Vo)

The T6963C displays have both a graphical area and a text area. They can be used together. The routines use the XOR mode to display both text and graphics layered over each other.

The statements that can be used with the graphical LCD are :

CLS 1322, will clear the graphic display and the text display
CLS GRAPH will clear only the graphic part of the display
CLS TEXT will only clear the text part of the display

LOCATE 1347 row,column : Will place the cursor at the specified row and column
The row may vary from 1 to 16 and the column from 1 to 40. This depends on the size and mode of the display.

CURSOR 1325 ON/OFF BLINK/NOBLINK can be used the same way as for text displays.

LCD 1335 : can be handled the same way as for text displays.

SHOWPIC 1355 X, Y , Label : Show image where X and Y are the column and row and Label is the label where the picture info is placed.

PSET 1348 X, Y , color : Will set or reset a pixel. X can range from 0-239 and Y from 9-63. When color is 0 the pixel will turned off. When it is 1 the pixel will be set on.

$BGF 609 "file.bgf" : inserts a BGF file at the current location

LINE 1344 (x0,y0) – (x1,y1) , color : Will draw a line from the coordinate x0,y0 to x1,y1. Color must be 0 to clear the line and 255 for a black line.

BOX 1317 (x0,y0)-(x1,y1), color : Will draw a box from x0,y0 to x1,y1. Color must be 0 to clear the box and 255 for a black line.

BOXFILL 1319 (x0,y0)-(x1,y1), color : Will draw a filled box from x0,y0 to x1,y1. Color must be 0 or 255.

The Graphic routines are located in the glib.lib or glib.lbx files.
You can hard wire the FS and RESET and change the code from the glib.lib file so these pins can be used for other tasks.

COLOR LCD

Color displays were always relatively expensive. The mobile phone market changed that. And Display3000.com , sorted out how to connect these small nice colorful displays.
You can buy brand new Color displays from Display3000. MCS Electronics offers the same displays.
There are two different chip sets used. One chipset is from EPSON and the other from Philips. For this reason there are two different libraries. When you select the wrong one it will not work, but you will not damage anything.
LCD-EPSON.LBX need to be used with the EPSON chipset.
LCD-PCF8833.LBX need to be used with the Philihps chipset.

Config Graphlcd = Color , Controlport = Portc , Cs = 1 , Rs = 0 , Scl = 3 , Sda = 2

Controlport	The port that is used to control the pins. PORTA, PORTB, etc.
CS	The chip select pin of the display screen. Specify the pin number. 1 will mean PORTC.1
RS	The RESET pin of the display
SCL	The clock pin of the display
SDA	The data pin of the display

As the color display does not have a built in font, you need to generate the fonts yourself.
You can use the Fonteditor 238 for this task.

A number of statements accept a color parameter. See the samples below in **bold**.

LINE	Line(0 , 0) -(130 , 130) , **Blue**
LCDAT	Lcdat 100 , 0 , "12345678" , **Blue , Yellow**
CIRCLE	Circle(30 , 30) , 10 , **Blue**
PSET	32 , 110 , **Black**
BOX	Box(10 , 30) -(60 , 100) , **Red**

See also

SHOWPIC 1355 , PSET 1348 , $BGF 609 , LINE 1344 , LCD 657 , BOX 1317 , BOXFILL 1319

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : t6963_240_128.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : T6963C graphic display support demo 240 *
128
'micro                    : Mega8535
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8535.dat"                                      ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
```

```
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

'-----------------------------------------------------------------------
'                     (c) 1995-2025 MCS Electronics
'                  T6963C graphic display support demo 240 * 128
'-----------------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin                   connected to
' 1        GND             GND
'2        GND             GND
'3        +5V             +5V
'4        -9V             -9V potmeter
'5        /WR             PORTC.0
'6        /RD             PORTC.1
'7        /CE             PORTC.2
'8        C/D             PORTC.3
'9        NC              not conneted
'10       RESET           PORTC.4
'11-18    D0-D7            PA
'19       FS              PORTC.5
'20       NC              not connected

'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT   to clear only the text display
' CLS GRAPH  to clear only the graphical part

Cursor Off

Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30


Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "12345678901234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"

Wait 2
```

```
Cls Text


'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255                                  ' diagonal
line
Line(0 , 127) -(239 , 0) , 255                                  ' diagonal
line
Line(0 , 0) -(240 , 0) , 255                                    ' horizontal
upper line
Line(0 , 127) -(239 , 127) , 255                                'horizontal
lower line
Line(0 , 0) -(0 , 127) , 255                                    ' vertical
left line
Line(239 , 0) -(239 , 127) , 255                                ' vertical
right line


Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
   Pset X , 20 , 255                                            ' set the
pixel
Next

For X = 0 To 140
   Pset X , 127 , 255                                           ' set the
pixel
Next

Wait 2

'circle time
'circle(X,Y), radius, color
'X,y is the middle of the circle,color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
  Circle(20 , 20) , X , 255                                     ' show
circle
  Wait 1
  Circle(20 , 20) , X , 0                                       'remove
circle
  Wait 1
Next

Wait 2

For X = 1 To 10
  Circle(20 , 20) , X , 255                                     ' show
circle
  Waitms 200
Next
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Test:
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje                                        ' show 2
since we have a big display
```

```
Wait 2
Cls Text                                                    ' clear the
text
End


'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

7.21.38 CONFIG HITAG

Action

Configures the timer and HITAG variables.

Syntax

CONFIG HITAG = prescale, TYPE=tp, DOUT = dout, DIN=din , CLOCK=clock, INT=int
CONFIG HITAG = prescale, TYPE=tp, DEMOD= demod, INT=@int

Remarks

syntax for HTRC110

prescale	The pre scaler value that is used by TIMER0. A value of 8 and 256 will work at 8 MHz.
tp	The kind of RFID chip you use. Use HTRC110.
DOUT	The pin that is connected to the DOUT pin of the HTRC110. This pin is used in input mode since DOUT is an output. A pin that support the pin-change interrupt or the PCINT should be selected.
DIN	The pin that is connected to the DIN pin of the HTRC110. This pin is used in output mode. You can chose any pin that can be used in output mode.
CLOCK	The pin that is connected tot the CLOCK pin of the HTRC110. This pin is used in output mode. You can chose any pin that can be used in output mode.
INT	The interrupt used. Note that you need to precede the interrupt with an @ sign. For example for INT1 you provide : @INT1

syntax for EM4095

prescale	The pre scaler value that is used by TIMER0. A value of 8 and 256 will work at 8 MHz.
tp	The kind of RFID chip you use. Use EM4095.
demod	The pin that is connected to the DEMOD pin of the EM4095. This pin is used in input mode. A pin that support the pin-change interrupt or the PCINT should be selected.
INT	The interrupt used. Note that you need to precede the interrupt with an @ sign. For example for INT1 you provide : @INT1

The CONFIG HITAG command will generate a number of internal used variables and constants.

Constants : _TAG_MIN_SHORT, _TAG_MAX_SHORT , _TAG_MIN_LONG and _TAG_MAX_LONG.
See the description of READHITAG to see how they are calculated. The actual value will depend on the prescaler value you use.

Variables for HTRC110 :
```
_htr_statemachine , a byte that is used to maintain a state machine.
_htcbit , a byte that will hold the received bit.
_htcbitcount , a byte to store the number of received bits.
_htcmpulse , a byte that stores the pulse
_htr_pulse_state , a byte that is used to maintain the pulse state machine.
_htc_retries, a byte that is used for the number of retries.
_tagdelta , a byte that will held the delta time between 2 edges.
_tagtime , a byte with the actual timer0 value when an edge is detected.
_taglasttime , a byte with the previous edge time, needed to calculate the
 delta time.
_tagparbit , a byte that will held the parity.
_tagdata , a byte where the bits are stored before they are loaded into the serial
 number array.
_tagid , a word that points to the serial number array
```

The HTRC110.LBX contains a number of other constants that are used to control the HTRC chip.
The _init_Tag routine is called automatically.

 The clock output of the Mega88 is used to drive the HTRC110. Since the clock output of the internal oscillator is 8 MHz, the HTRC110 is also configured to work at 8 MHz.
The .equ for Tag_set_config_page3 = &H40 + 48 + Fsel0 in the LBX. You can set it to 12 and 16 MHz too but you can not drive it from the clock output then.
The datasheet specifies the following for FSEL1 and FSEL0

FSEL1	FSEL0	Frequency
0	0	4 MHz
0	1	8 MHz
1	0	12 MHz
1	1	16 MHz

So when you want to use a different frequency you can edit the equ in the lbx.
.equ Tag_set_config_page3 = &H40 + 48 + Fsel0 ' 8 Mhz
For 16 Mhz it would become :
.equ Tag_set_config_page3 = &H40 + 48 + Fsel0 + Fsel1 ' 8 Mhz

When you want to send a custom command you can call the internal routine :
_Send_htrc110_cmdR25
Just load R25 with the proper value before you do :
R25=8 'readphase command
!call _Send_htrc110_cmdR25

Variables for EM4095 :
```
_tagflag , a byte that stores the return flag that will be loaded with 1 when a
 valid tag is detected
_tag_insync ,a byte that is used to store the state of the bit stream.
_tag_bitcount , a byte that stores the total bits when not in sync yet
_tag_tbit , a  byte that stores the total received bits
_tag_par , a byte that stores the parity
_tag_timeout ,a byte that is loaded with the time that will be tried to
detect an RFID chip
_taglasttime , a byte that stores the last time a valid edge was detected
```

_tagid , a word that points to the serial number array

See also

READHITAG 1417

Example HTRC110

```
'-----------------------------------------------------------------------
'                     (c) 1995-2025  , MCS Electronics
' sample : readhitag.bas
' demonstrates usage of the READHITAG() function
'-----------------------------------------------------------------------

$regfile = "m88def.dat"                                     ' specify chip
$crystal = 8000000                                          ' used speed
$baud = 19200                                               ' baud rate
'Notice that the CLOCK OUTPUT of the micro is connected to the clock input of
 the HTRC110
'PORTB.0 of the Mega88 can optional output the clock. You need to set the
 fusebit for this option
'This way all parts use the Mega88 internal oscillator

'The code is based on Philips(NXP) datasheets and code. We have signed an
 NDA to get the 8051 code
'You can find more info on Philips website if you want their code
Print "HTC110 demo"

Config Hitag = 64 , Type = Htrc110 , Dout = Pind.2 , Din = Pind.3 , Clock = Pind.4
'                ^ use timer0 and select prescale value 64
'                     ^ we used htrc110 chip
'                                    ^-- dout of HTRC110 is connected to PIND.2
'                                        which will be set to input mode
'                                              ^ DIN of HTRC100 is connected
'                                             to PIND.3 which will be set to ou
'                                                                   ^clock of
'                     HTRC110 is connected to PIND.4 which is set to output mode
'
'the config statement will generate a number of constants and
 internal variables used by the code
'the htrc110.lbx library is called

Dim Tags(5) As Byte                                         'each tag has 5 byte se
Dim J As Byte                                               ' a loop counter

'you need to use a pin that can detect a pin level change
'most INT pins have this option
'OR , you can use the PCINT interrupt that is available on some chips

'In case you want PCINT option
' Pcmsk2 = &B0000_0100        'set the mask to ONLY use the pin connected to DOUT
' On Pcint2 Checkints         'label to be called
' Enable Pcint2               'enable this interrupt

'In case you want to use INT option
On Int0 Checkints                   ' PIND.2 is INT0
Config Int0 = Change                'you must configure the pin to work in pin chang

Enable Interrupts                   ' enable global interrupts

Do
 If Readhitag(tags(1)) = 1 Then     'check if there is a new tag ID
```

```
     For J = 1 To 5                 'print the 5 bytes
         Print Hex(tags(j)) ; ",";
     Next
  Else                              'there was nothing
    Print "Nothing"
  End If
  Waitms 500                        'some delay
Loop


'this routine is called by the interrupt routine
Checkints:
 Call _checkhitag                   'you must call this label
 'you can do other things here but keep time to a minimum
Return
```

Example EM4095

```
'-------------------------------------------------------------------------------
'                  (c) 1995-2025 MCS Electronics
'  This sample will read a HITAG chip based on the EM4095 chip
'  Consult EM4102 and EM4095 datasheets for more info
'-------------------------------------------------------------------------------
'  The EM4095 was implemented after an idea of Gerhard Günzel
'  Gerhard provided the hardware and did research at the coil and capacitors.
'  The EM4095 is much simpler to use than the HTRC110. It need less pins.
'  A reference design with all parts is available from MCS
'-------------------------------------------------------------------------------
$regfile = "M88def.dat"
$baud = 19200
$crystal = 8000000
$hwstack = 40
$swstack = 40
$framesize = 40


'Make SHD and MOD low


Dim Tags(5) As Byte         'make sure the array is at least 5 bytes
Dim J As Byte

Config Hitag = 64 , Type = Em4095 , Demod = Pind.3 , Int = @int1

Print "Test EM4095"


'you could use the PCINT option too, but you must mask all pins out
so it will only respond to our pin
' Pcmsk2 = &B0000_0100
' On Pcint2 Checkints
' Enable Pcint2
On Int1 Checkints Nosave        'we use the INT1 pin all regs are saved in the lib
Config Int1 = Change            'we have to config so that on each pin change the r
Enable Interrupts               'as last we have to enable all interrupts



Do
   Print "Check..."

   If Readhitag(tags(1)) = 1 Then   'this will enable INT1
      For J = 1 To 5
         Print Hex(tags(j)) ; ",";
```

```
      Next
      Print
  Else
     Print "Nothing"
  End If
  Waitms 500
Loop


Checkints:
 Call _checkhitag        'in case you have used a PCINT, you could have other code
Return
```

7.21.39 CONFIG I2CBUS

Action

This configuration statement defines the SCL and SDA pins of an I2C multibus.

Syntax

CONFIG I2CBUS= bus , SCL=scl , SDA=sda

Remarks

bus	A numeric value in the range from 0 to 15.
scl	The SCL pin used for the specified bus.
sda	The SDA pin used for the specified bus.

While XMEGA supports multiple TWI busses, the normal AVR only supports on TWI or no I2C bus. The CONFIG I2CBUS is a software solution to use multiple I2C busses.
An internal variable is created named I2CBUS. This is a BYTE variable.
You need to assign this variable a value before you use the usual I2C statements.
When you want to use a different bus, you just assign the variable a new bus index value.
Have a look at the sample. It creates 4 busses. Since I2CINIT is required, a loop is used to call the I2CINIT statement for all busses.
And another loop is used to send data to all 4 busses.

 Both SCL and SDA pins must be on the same PORT. Also, the PIN, DDR and PORT register addresses of the processor must be in ascending order and need to exist.

For example the M1284P portA group :
PORTA = $02
DDRA = $01
PINA = $00

This is ok to use. But some processors have no DDR register because a port can only be used in output or input mode. Such a port can not be used.

An example of a bad port is PORTF in the M128. As you can see there is a gap in the address between PINF and DDRF and this will make it fail.
PORTF = $62

```
DDRF  = $61
PINF  = $00
```

ASM
The I2C routines are located in the i2c_multibus.lib.

See also
CONFIG SCL 1049 , CONFIG SDA 1046, Using the I2C protocol 297 , I2CINIT 1300

Example
```
'-----------------------------------------------------------------------
'name                  : I2C-multibus.bas
'copyright             : (c) 1995-2025, MCS Electronics
'purpose               : demonstrates I2C multibus library
'micro                 : Mega88
'suited for demo       : no, lib not included in demo
'commercial addon needed : no
'-----------------------------------------------------------------------
$regfile="m88def.dat"
$crystal=8000000
$hwstack=32
$swstack=24
$framesize=24

config i2cbus=0,scl=portc.0,sda= portc.1 'each bus requires a configuration of the SCL and SDA pins
config i2cbus=1,scl=portc.2,sda= portc.3 'this sample creates 4 busses
config i2cbus=2,scl=portd.2,sda= portd.3
config i2cbus=3,scl=portd.4,sda= portd.5

Dim j as Byte

For j=0 to 3                        'the first bus is 0 !!!
  i2cbus=j                          'select the BUS
  i2cinit                           'init the pins and state
Next

do
  for j=0 to 3
    i2cbus=j                        'select the bus
    I2CSend &H40, &B01010101        'send some data
  next
  waitms 100
loop

end
```

7.21.40 CONFIG I2CDELAY

Action
Compiler directive that overrides the internal I2C delay routine.
(Only for Software I2C Routines)

Syntax
CONFIG I2CDELAY = value

Remarks

value	A numeric value in the range from 1 to 255. A higher value means a slower I2C clock.

	You may use a value of 0 too but it will result in a value of 256.

For the I2C routines the clock rate is calculated depending on the used crystal. In order to make it work for all I2C devices the slow mode is used. When you have faster I2C devices you can specify a low value.

By default a value of 5 is used. This will give a 200 kHZ clock.
When you specify 10, 10 uS will be used resulting in a 100 KHz clock.

When you use a very low crystal frequency, it is not possible to work with high clock frequencies.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.
For chips that have hardware TWI, you can use the MasterTWI lib.

See also

CONFIG SCL 1049 , CONFIG SDA 1046, Using the I2C protocol 297

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : i2c.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: I2CSEND and I2CRECEIVE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'We use here the Software I2C Routines
Config Scl = Portb.4
Config Sda = Portb.5
I2cinit

Config I2cdelay = 10                                       '100KHz

Declare Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
Declare Sub Read_eeprom(byval Adres As Byte , Value As Byte)

Const Addressw = 174                                       'slave write
address
Const Addressr = 175                                       'slave read
```

```
address

Dim B1 As Byte , Adres As Byte , Value As Byte                'dim byte

Call Write_eeprom(1 , 3)                                      'write value
of three to address 1 of EEPROM
Call Read_eeprom(1 , Value) : Print Value                     'read it
back
Call Read_eeprom(5 , Value) : Print Value                     'again for
address 5


'-------- now write to a PCF8474 I/O expander -------
I2csend &H40 , 255                                            'all outputs
high
I2creceive &H40 , B1                                          'retrieve
input
Print "Received data " ; B1                                   'print it
End

Rem Note That The Slaveaddress Is Adjusted Automaticly With I2csend &
I2creceive
Rem This Means You Can Specify The Baseaddress Of The Chip.

'sample of writing a byte to EEPROM AT2404
Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
    I2cstart                                                  'start
condition
    I2cwbyte Addressw                                         'slave
address
    I2cwbyte Adres                                            'asdress of
EEPROM
    I2cwbyte Value                                            'value to
write
    I2cstop                                                   'stop
condition
    Waitms 10                                                 'wait for 10
milliseconds
End Sub

'sample of reading a byte from EEPROM AT2404
Sub Read_eeprom(byval Adres As Byte , Value As Byte)
   I2cstart                                                   'generate
start
   I2cwbyte Addressw                                          'slave
adsress
   I2cwbyte Adres                                             'address of
EEPROM
   I2cstart                                                   'repeated
start
   I2cwbyte Addressr                                          'slave
address (read)
   I2crbyte Value , Nack                                      'read byte
   I2cstop                                                    'generate
stop
End Sub

' when you want to control a chip with a larger memory like the 24c64 it
requires an additional byte
' to be sent (consult the datasheet):
' Wires from the I2C address that are not connected will default to 0 in
most cases!

'   I2cstart                                                  'start
```

```
condition
'   I2cwbyte &B1010_0000                                    'slave
address
'   I2cwbyte H                                              'high
address
'   I2cwbyte L                                              'low address
'   I2cwbyte Value                                          'value to
write
'   I2cstop                                                 'stop
condition
'   Waitms 10
```

7.21.41 CONFIG I2CSLAVE

The I2C-Slave library is intended to create I2C slave chips. This is an add-on library that is not included in Bascom-AVR by default. It is a commercial add on library. It is available from MCS Electronics
The I2C Slave add on can turn some chips into a I2C slave device. You can start your own chip plant this way.

Most new AVR chips have a so called TWI/I2C interface. As a customer of the I2C slave lib, you can get both libs.
The **i2cslave.lib** works in interrupt mode and is the best way as it adds less overhead and also less system resources.

With this add-on library you get both libraries:

- **i2cslave.lib** and **i2cslave.lbx** : This library is used for AVR's which have no hardware TWI/I2C interface like for example ATTINY2313 or ATTINY13. In this case TIMER0 and INT0 is used for SDA and SCL (Timer0 Pin = SCL, INT0 Pin = SDA). Only AVR' with TIMER0 and INT0 on the same port can use this library like for example ATTINY2313 or ATTINY13. The i2cslave.lib file contains the ASM source. The i2cslave.lbx file contains the compiled ASM source. See **CONFIG I2CSLAVE** below.

- **i2c_TWI-slave.LBX** : This library can be used when an AVR have an TWI/I2C hardware interface like for example ATMEGA8, ATMEGA644P or ATMEGA128. In this case the hardware SDA and SCL pin's of the AVR will be used (with ATMEGA8: SCL is PORTC.5 and SDA is PORTC.4). This library will be used when USERACK = OFF. When USERACK =ON then **i2c_TWI-slave-acknack.LBX** will be used. See also Config TWISLAVE 1123

Action
Configures the I2C slave mode for ATTINY and ATMEGA devices.

Before you begin
Copy the library files into the BASCOM-AVR\LIB directory.

Syntax
CONFIG I2CSLAVE = address , INT = interrupt , TIMER = tmr
(This function is part of the I2C-Slave library. This is an add-on library that is not included in Bascom-AVR by default. It is a commercial add on library. It is available from MCS Electronics)

Remarks

Address	The slave address you want to assign to the I2C slave chip. This is an

	address that must be even like &H60. So &H61 cannot be used. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases.
Interrupt	The interrupt that must be used. This is INT0 by default.
Tmr	The timer that must be used. This is TIMER0 by default.

The library was written for TIMER0 and INT0.

While the interrupt can be specified, you need to change the library code when you use a non-default interrupt. For example when you like to use INT1 instead of the default INT0.

The same applies to the TIMER. You need to change the library when you like to use another timer.

You can not use these interrupts yourself. It also means that the SCL and SDA pins are fixed.

CONFIG I2CSLAVE will enable the global interrupts.

Timer0 and INT0 Pin's of Various AVR's

The I2C slave routines use the TIMER0 and INT0.

The following table lists the pins for the various chips

Chip	SCL	SDA
AT90S1200	PORTD.4	PORTD.2
AT90S2313	PORTD.4	PORTD.2
AT90S2323	PORTB.2	PORTB.1
AT90S2333	PORTD.4	PORTD.2
AT90S2343	PORTB.2	PORTB.1
AT90S4433	PORTD.4	PORTD.2
ATTINY22	PORTB.2	PORTB.1
ATTINY13	PORTB.2	PORTB.1
ATTINY2313	PORTD.4	PORTD.2
ATMEGA1280	PORTD.7	PORTD.0
ATMEGA128CAN	PORTD.7	PORTD.0
ATMEGA168	PORTD.4	PORTD.2
ATMEGA2560	PORTD.7	PORTD.0
ATMEGA2561	PORTD.7	PORTD.0
ATMEGA48	PORTD.4	PORTD.2
ATMEGA88	PORTD.4	PORTD.2
ATMEGA8	PORTD.4	PORTD.2

After you have configured the slave address, you can insert your code.

A do-loop would be best:

```
Do
   ' your code here
Loop
```

After your main program you need to insert two labels with a return:

When the master needs to read a byte, the following label is always called.
You must put the data you want to send to the master in variable _a1 which is register R16

```
I2c_master_needs_data:
'when your code is short, you need to put in a waitms statement
'Take in mind that during this routine, a wait state is active and the
master will wait
'After the return, the waitstate is ended
Config Portb = Input ' make it an input

_a1 = Pinb ' Get input from portB and assign it
Return
```

When the master writes a byte, the following label is always called.
It is your task to retrieve variable _A1 and do something with it
_A1 is register R16 that could be destroyed/altered by BASIC statements
For that reason it is important that you first save this variable.

```
I2c_master_has_data:
   'when your code is short, you need to put in a waitms statement
   'Take in mind that during this routine, a wait state is active and the
master will wait
   'After the return, the waitstate is ended

   Bfake = _a1                    ' this is not needed but it shows how
you can store _A1 in a byte
   'after you have stored the received data into bFake, you can alter R16
   Config Portb = Output       ' make it an output since it could be an
input
   Portb = _a1                     'assign _A1 (R16)
Return
```

See Also

CONFIG TWI 1116 , CONFIG TWISLAVE 1123, I2C TWI Slave 1831

Debugging Hint's

If you encounter a problem first check:

- Do you use the correct Pin's for SDA and SCL ?
- Pull-up Resistor from SDA and SCL to Vcc ?
- Try to reduce clockrate from I2C Master
- Try to use **waitms XX** between the **I2CWBYTE** in the I2C Master AVR
- Try to reduce code in the interrupt routine

Example

```
'-----------------------------------------------------------------------
-----------------
'name                    : i2c_pcf8574.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : shows how you could use the I2C slave
```

```
library to create a PCF8574
'micro                    : AT90S2313
'suited for demo          : NO, ADDON NEEDED
'commercial addon needed  : yes
'-------------------------------------------------------------------------
-----------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 3684000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'This program shows how you could use the I2C slave library to create a
PCF8574
'The PCF8574 is an IO extender chip that has 8 pins.
'The pins can be set to a logic level by writing the address followed by
a value
'In order to read from the pins you need to make them '1' first

'This program uses a AT90S2313, PORTB is used as the PCF8574 PORT
'The slave library needs INT0 and TIMER0 in order to work.
'SCL is PORTD.4 (T0)
'SDA is PORTD.2 (INT0)
'Use 10K pull up resistors for both SCL and SDA

'The Slave library will only work for chips that have T0 and INT0
connected to the same PORT.
'These chips are : 2313,2323, 2333,2343,4433,tiny22, tiny12,tiny15, M8
'The other chips have build in hardware I2C(slave) support.

'specify the slave address. This is &H40 for the PCF8574
'You always need to specify the address used for write. In this case
&H40 ,

'The config i2cslave command will enable the global interrupt enable
flag !
Config I2cslave = &B01000000                                ' same as
&H40
'Config I2cslave = &H40 , Int = Int0 , Timer = Timer0
'A byte named _i2c_slave_address_received is generated by the compiler.
'This byte will hold the received address.

'A byte named _i2c_slave_address is generated by the compiler.
'This byte must be assigned with the slave address of your choice

'the following constants will be created that are used by the slave
library:

' _i2c_pinmask = &H14
' _i2c_slave_port = Portd
' _i2c_slave_pin = Pind
' _i2c_slave_ddr = Ddrd
' _i2c_slave_scl = 4
' _i2c_slave_sda = 2

'These values are adjusted automatic depending on the selected chip.
'You do not need to worry about it, only provided as additional info
```

```
'by default the PCF8574 port is set to input
Config Portb = Input
Portb = 255                                                 'all pins
high by default

'DIM a byte that is not needed but shows how you can store/write the I2C
DATA
Dim Bfake As Byte


'empty loop
Do
   ' you could put your other program code here
   'In any case, do not use END since it will disable interrupts

Loop


'here you can write your other program code
'But do not forget, do not use END. Use STOP when needed

'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!
'             The following labels are called from the slave library
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!

'When the master wants to read a byte, the following label is allways
called
'You must put the data you want to send to the master in variable _a1
which is register R16
I2c_master_needs_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended
  Config Portb = Input                                      ' make it an
input
  _a1 = Pinb                                                ' Get input
from portB and assign it
Return


'When the master writes a byte, the following label is always called
'It is your task to retrieve variable _A1 and do something with it
'_A1 is register R16 that could be destroyed/altered by BASIC statements
'For that reason it is important that you first save this variable

I2c_master_has_data:
  'when your code is short, you need to put in a waitms statement
  'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended

  Bfake = _a1                                               ' this is
not needed but it shows how you can store _A1 in a byte
  'after you have stored the received data into bFake, you can alter R16
  Config Portb = Output                                     ' make it an
output since it could be an input
  Portb = _a1                                               'assign _A1
(R16)
Return
```

```
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!

'You could simply extend this sample so it will use 3 pins of PORT D for
the address selection
'For example portD.1 , portd.2 and portD.3 could be used for the address
selection
'Then after the CONFIG I2CSLAVE = &H40 statement, you can put code like:
'Dim switches as Byte   ' dim byte
'switches = PIND        ' get dip switch value
'switches = switches and &H1110 ' we only need the lower nibble without
the LS bit
'_i2c_slave_address = &H40 + switches ' set the proper address
```

7.21.42 CONFIG INPUT

Action

Instruct the compiler to modify serial input line terminator behaviour

Syntax

CONFIG INPUT1 = term , ECHO=echo

Syntax Xmega

CONFIG INPUT1|INPUT2|INPUT3|INPUT4|INPUT5|INPUT6|INPUT7|INPUT8 = term , ECHO=echo

Remarks

INPUT	Use INPUT or INPUT1 for COM1, INPUT2 for COM2, INPUT3 for COM3, etc.
Term	A parameter with one of the following values : CR - Carriage Return (default) LF - Line Feed CRLF - Carriage Return followed by a Line Feed LFCR - Line Feed followed by a Carriage Return
Echo	A parameter with one of the following values : CR - Carriage Return LF - Line Feed CRLF - Carriage Return followed by a Line Feed (default) LFCR - Line Feed followed by a Carriage Return

The 'term' parameter specifies which character(s) are expected to terminate the INPUT 1493 statement with serial communication. It has no impact on the DOS file system INPUT.
In most cases, when you press <ENTER> , a carriage return(ASCII 13) will be sent. In some cases, a line feed (LF) will also be sent after the CR. It depends on the terminal emulator or serial communication OCX control you use.

The 'echo' parameter specifies which character(s) are send back to the terminal emulator after the INPUT terminator is received. By default CR and LF is sent. But you can specify which characters are sent. This can be different characters then the 'term'

characters. So when you send in your VB application a string, and end it with a CR, you can send back a LF only when you want.

 When NOECHO is used, **NO** characters are sent back even while configured with CONFIG INPUT

For the XMega you can specify for each UART how it should handle input and echo.

For the first UART you may use INPUT0, INPUT1 or just INPUT. For the second UART you must use INPUT2, for UART3 -> INPUT3, etc.

See also
INPUT 1493

ASM
NONE

Example
```
Config Input1 = CR , Echo = CRLF
Dim S as String * 20
Input "Hello ",s
```

7.21.43 CONFIG INPUTBIN

Action
Configure INPUTBIN behavior

Syntax
CONFIG INPUTBIN = extended

Remarks

extended	This mode is the only mode. It allows to receive packets greater than 255 bytes. The maximum packet size is 64 KB. Because support for big packets requires more code, it is made optional. You can not change between normal and extended mode dynamically. If you chose to use extended mode, this will be used for all your PRINTBIN code.

See also
CONFIG PRINT 1028 , PRINTBIN 1504 , INPUTBIN 1497 , CONFIG PRINTBIN 1029

Example
```
$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 8000000                                          ' used
```

```
crystal frequency
$baud = 19200                                                ' use baud
rate
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Config Inputbin = Extended
Dim A(1000)
Inputbin A(1) ; 1000
```

7.21.44 CONFIG INTx

Action

Configures the way the interrupts 0,1 and 4-7 will be triggered.

Syntax

CONFIG INTx = state
Where X can be 0,1 and 4 to 7 in the MEGA chips.

Remarks

state	LOW LEVEL to generate an interrupt while the pin is held low. Holding the pin low will generate an interrupt over and over again. FALLING to generate an interrupt on the falling edge. RISING to generate an interrupt on the rising edge. CHANGE to generate an interrupt on the change of the edge. Not all microprocessors support CHANGE.

The MEGA103 has also INT0-INT3. These are always low level triggered so there is no need /possibility for configuration.
The number of interrupt pins depend on the used chip. Most chips only have int0 and int1.

XMEGA

For the XMEGA you need to use CONFIG XPIN 1158.

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : spi-softslave.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows how to implement a SPI SLAVE with
software
'micro                    : AT90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
```

```
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'Some atmel chips like the 2313 do not have a SPI port.
'The BASCOM SPI routines are all master mode routines
'This example show how to create a slave using the 2313
'ISP slave code


'define the constants used by the SPI slave
Const _softslavespi_port = Portd                            ' we used
portD
Const _softslavespi_pin = Pind                              'we use the
PIND register for reading
Const _softslavespi_ddr = Ddrd                              ' data
direction of port D

Const _softslavespi_clock = 5                               'pD.5 is
used for the CLOCK
Const _softslavespi_miso = 3                                'pD.3 is
MISO
Const _softslavespi_mosi = 4                                'pd.4 is
MOSI
Const _softslavespi_ss = 2                                  ' pd.2 is SS
'while you may choose all pins you must use the INT0 pin for the SS
'for the 2313 this is pin 2

'PD.3(7),  MISO  must be output
'PD.4(8),  MOSI
'Pd.5(9) , Clock
'PD.2(6),  SS /INT0

'define the spi slave lib
$lib "spislave.lbx"
'sepcify wich routine to use
$external _spisoftslave

'we use the int0 interrupt to detect that our slave is addressed
On Int0 Isr_sspi Nosave
'we enable the int0 interrupt
Enable Int0
'we configure the INT0 interrupt to trigger when a falling edge is
detected
Config Int0 = Falling
'finally we enabled interrupts
Enable Interrupts

'
Dim _ssspdr As Byte                                         ' this is
out SPI SLAVE SPDR register
Dim _ssspif As Bit                                          ' SPI
interrupt revceive bit
```

```
Dim Bsend As Byte , I As Byte , B As Byte                    ' some other
demo variables

_ssspdr = 0                                                  ' we send a
0 the first time the master sends data
Do
   If _ssspif = 1 Then
   Print "received: " ; _ssspdr
   Reset _ssspif
   _ssspdr = _ssspdr + 1                                     ' we send
this the next time
   End If
Loop
```

7.21.45 CONFIG INTVECTORSELECTION

Action

Sets or resets the IVSEL bit to chose the vector table address.

Syntax

CONFIG INTVECTORSELECTION = enabled|disabled
CONFIG INTVECTORSELECTION = boot|normal

Remarks

Some processors with a boot loader have a special register and switch that enables the user to chose the interrupt vector table address.
By default the address is &H0000. When running a boot loader application which requires interrupts, you can use $BOOTVECTOR to create an interrupt vector table (IVR).
The processor must be forced to load the vector addresses from the boot vector address instead of the default 0000. This is where you use CONFIG INTVECTORSELECTION = enabled.

Instead of 'enabled' you can also use 'boot'. And instead of 'disabled' you may also use 'normal'.
Enabled and disabled describe the status of the IVSEL bit while boot and normal are more clear about the address.

Do not forget to reset the IVSEL bit using CONFIG INTVECTORSELECTION = disabled in your normal application. We advise to use a watchdog time out to reset the processor after the boot loader has finished. This will reset all registers to their defaults and this will disable the IVSEL bit too.

See Also

$LOADER 667 , $BOOTVECTOR 613

Example

See $LOADER 667

7.21.46 CONFIG KBD

Action
Configure the GETKBD() function and tell which port to use.

Syntax
CONFIG KBD = PORTx , DEBOUNCE = value [, DELAY = value] [,COLS=cols]

Remarks

PORTx	The name of the PORT to use such as PORTB or PORTD.
DEBOUNCE	By default the debounce value is 20. A higher value might be needed. The maximum is 255.
Delay	An optional parameter that will cause Getkbd() to wait the specified amount of time after the key is detected. This parameter might be added when you call GetKbd() repeatedly in a loop. Because of noise and static electricity, wrong values can be returned. A delay of say 100 mS, can eliminate this problem.
COLS	This value is 4 by default. Some chips do not have port pin 7 and for these cases you can use COLS=3, or COLS=2. This does assume that columns are connected to the high port nibble.

The GETKBD() function can be used to read the pressed key from a matrix keypad attached to a port of the uP.
You can define the port with the CONFIG KBD statement.

In addition to the default behavior you can configure the keyboard to have 6 rows instead of 4 rows.

CONFIG KBD = PORTx , DEBOUNCE = value , rows=6, row5=pinD.6, row6=pind.7

This would specify that row5 is connected to pind.6 and row7 to pind.7
Note that you can only use rows=6. Other values will not work.

See also
GETKBD 1275

Example
```
'-----------------------------------------------------------------------
-----------------
'name                     : getkbd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : GETKBD
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
```

```
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

'specify which port must be used
'all 8 pins of the port are used
Config Kbd = Portb

'dimension a variable that receives the value of the pressed key
Dim B As Byte

'loop for ever
Do
  B = Getkbd()
  'look in the help file on how to connect the matrix keyboard
  'when you simulate the getkbd() it is important that you press/click
the keyboard button
  ' before running the getkbd() line !!!
  Print B
  'when no key is pressed 16 will be returned
  'use the Lookup() function to translate the value to another one
' this because the returned value does not match the number on the
keyboad
Loop
End
```

7.21.47 CONFIG KEYBOARD

Action

Configure the GETATKBD() function and tell which port pins to use.

Syntax

CONFIG KEYBOARD = PINX.y , DATA = PINX.y , KEYDATA = table

Remarks

KEYBOARD	The PIN that serves as the CLOCK input.
DATA	The PIN that serves as the DATA input.
KEYDATA	The label where the key translation can be found. The AT keyboard returns scan codes instead of normal ASCII codes. So a translation table s needed to convert the keys. BASCOM allows the use of shifted keys too. Special keys like function keys are not supported.

The AT keyboard can be connected with only 4 wires: clock,data, gnd and vcc.
Some info is displayed below. This is copied from an Atmel data sheet.

The INT0 or INT1 shown can be in fact any pin that can serve as an INPUT pin.

The application note from Atmel works in interrupt mode. For BASCOM we rewrote the code so that no interrupt is needed/used.

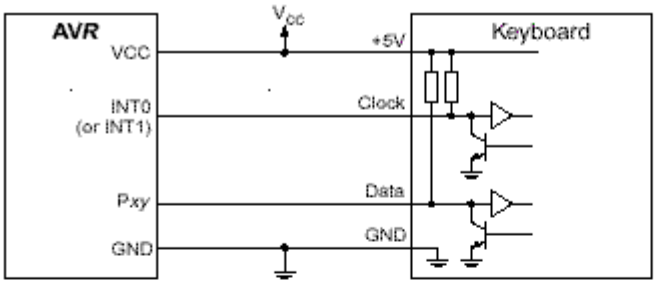


Table 1. AT Keyboard Connector Pin Assignments

AT Computer		
Signals	**DIN41524, Female at Computer, 5-pin DIN 180°**	**6-pin Mini DIN PS2 Style Female at Computer**
Clock	1	5
Data	2	1
nc	3	2,6
GND	4	3
+5V	5	4
Shield	Shell	Shell

See also

GETATKBD 1270

Example

```
'-----------------------------------------------------------------------
-----------------
'name                       : getatkbd.bas
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : PC AT-KEYBOARD Sample
'micro                      : Mega48
'suited for demo            : yes
'commercial addon needed    : no
'-----------------------------------------------------------------------
-----------------

$regfile = "8535def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'For this example :
'connect PC AT keyboard clock to PIND.2 on the 8535
'connect PC AT keyboard data to PIND.4 on the 8535
```

```
'The GetATKBD() function does not use an interrupt.
'But it waits until a key was pressed!

'configure the pins to use for the clock and data
'can be any pin that can serve as an input
'Keydata is the label of the key translation table
Config Keyboard = Pind.2 , Data = Pind.4 , Keydata = Keydata

'Dim some used variables
Dim S As String * 12
Dim B As Byte

'In this example we use SERIAL(COM) INPUT redirection
$serialinput = Kbdinput

'Show the program is running
Print "hello"

Do
  'The following code is remarked but show how to use the GetATKBD()
function
  ' B = Getatkbd()     'get a byte and store it into byte variable
  'When no real key is pressed the result is 0
  'So test if the result was > 0
  ' If B > 0 Then
  '    Print B ; Chr(b)
  ' End If

  'The purpose of this sample was how to use a PC AT keyboard
  'The input that normally comes from the serial port is redirected to
the
  'external keyboard so you use it to type
  Input "Name " , S
  'and show the result
  Print S
  'now wait for the F1 key , we defined the number 200 for F1 in the
table
  Do
    B = Getatkbd()
  Loop Until B <> 0
  Print B
Loop
End

'Since we do a redirection we call the routine from the redirection
routine
'
Kbdinput:
'we come here when input is required from the COM port
'So we pass the key into R24 with the GetATkbd function
' We need some ASM code to save the registers used by the function
$asm
push r16            ; save used register
push r25
push r26
push r27

Kbdinput1:
rCall _getatkbd     ; call the function
tst r24             ; check for zero
breq Kbdinput1      ; yes so try again
pop r27             ; we got a valid key so restore registers
pop r26
```

```
pop r25
pop r16
$end Asm
'just return
Return

'The tricky part is that you MUST include a normal call to the routine
'otherwise you get an error
'This is no clean solution and will be changed
B = Getatkbd()

'This is the key translation table

Keydata:
'normal keys lower case
Data 0 , 0 , 0 , 0 , 0 , 200 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0
Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 ,
50 , 0
Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116 ,
114 , 53 , 0
Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 , 117
, 55 , 56 , 0
Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 ,
112 , 43 , 0
Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0
Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 ,
0 , 0

'shifted keys UPPER case
Data 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
Data 0 , 0 , 0 , 0 , 0 , 81 , 33 , 0 , 0 , 0 , 90 , 83 , 65 , 87 , 34 ,
0
Data 0 , 67 , 88 , 68 , 69 , 0 , 35 , 0 , 0 , 32 , 86 , 70 , 84 , 82 ,
37 , 0
Data 0 , 78 , 66 , 72 , 71 , 89 , 38 , 0 , 0 , 76 , 77 , 74 , 85 , 47 ,
40 , 0
Data 0 , 59 , 75 , 73 , 79 , 61 , 41 , 0 , 0 , 58 , 95 , 76 , 48 , 80 ,
63 , 0
Data 0 , 0 , 0 , 0 , 0 , 96 , 0 , 0 , 0 , 0 , 13 , 94 , 0 , 42 , 0 , 0
Data 0 , 62 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 ,
0 , 0
```

7.21.48 CONFIG LCD

Action

Configure the LCD display and override the compiler setting.

Syntax

CONFIG LCD = LCDtype , CHIPSET=KS077 | Dogm163v5 | DOG163V3 | DOG162V5 | DOG162V3 | ST7032 [,CONTRAST=value] [,BEFORE=0|1] [,AFTER=0|1]

BEFORE and AFTER. with a parameter value of 1 a sub will be called _lcdBefore and _lcdAfter

Remarks

LCDtype	The type of LCD display used. This can be :

	40x4,16x1, 16x2, 16x4, 16x4, 20x2, 20x4, 16x1a or 20x4A. Default 16x2 is assumed.
Chipset KS077	Most text based LCD displays use the same chip from Hitachi. But some use the KS077 which is highly compatible but needs an additional function register to be set. This parameter will cause that this register is set when you initialize the display.
CHIPSET DOGM	The DOGM chip set uses a special function register that need to be set. The 16 x 2 LCD displays need DOG162V3 for 3V operation or DOG162V5 for 5V operation. The 16 x 3 LCD displays need DOG163V3 for 3V operation or Dogm163v5 for 5V operation
CHIPSET ST7032	This chip is used on I2C lcd's. It requires library Lcd_RX1602A5. See example 3 below.
CONTRAST	The optional contrast parameter is only supported by the EADOG displays. By default a value from the manufacture is used. But you might want to override this value with a custom setting. The default values are : - DOGM162V5 : &H74 - DOGM162V3 : &H78 - DOGM163V5 : &H7C - DOGM163V3 : &H70
BEFORE	This is an optional parameter. A value of 1 will result in a call to a routine named _LCDBEFORE, each time LCD value|"text" is used. This allows you as a user to turn off interrupts or perform other tasks.
AFTER	This is an optional parameter. A value of 1 will result in a call to a routine named _LCDAFTER, each time LCD value|"text" is ended. This allows you as a user to turn on interrupts or perform other tasks.

When you have a 16x2 display, you don't have to use this statement.
The 16x1a is special. It is used for 2x8 displays that have the address of line 2, starting at location &H8.
The 20xA is also special. It uses the addresses &H00, &H20, &H40 and &H60 for the 4 lines. It will also set a special function register.

The CONFIG LCD can only be used once. You can not dynamic(at run time) change the pins.
When you want to initialize the LCD during run time, you can use the INITLCD 1334 statement.

The BEFORE and AFTER parameters can be used to call some user code just before data is shown on the LCD, and when finished.
For example, you could toggle a LED on/off. Or set some background light. Or disable interrupts before showing data, and enable interrupts afterwards.
You must use DECLARE SUB to declare the called labels. Or you may use normal labels and exit with RETURN.

In version 2084 a constant is created named _TEXTLCDKIND which contains a value based on the selected LCD.
The values are :

LCD	Value
16x1	161
16x2	162
16x3	163
16x4	164

20x2	202
24x2	242
40x4	404
20x4A	1204
40x2	402
20x4	204
16x1A	1610
20x4VFD	2204

See Also

CONFIG LCDPIN 1001 , CONFIG LCDBUS 998 , INITLCD 1334

Example1

```
'-----------------------------------------------------------------------
-----------------
'name                     : lcd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
SHIFTCURSOR, HOME
'                           CURSOR, DISPLAY
'micro                    : Mega8515
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m8515.dat"                                      ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation


'note : tested in PIN mode with 4-bit

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
```

```
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector

Rem with the config lcdpin statement you can override the compiler
settings


Dim A As Byte
Config Lcd = 16x2                                          'configure lcd
screen


'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                          'clear the
LCD display
Lcd "Hello world."                                           'display
this at the top line
Wait 1
Lowerline                                                    'select the
lower line
Wait 1
Lcd "Shift this."                                            'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                            'shift the
text to the right
   Wait 1                                                    'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                             'shift the
text to the left
   Wait 1                                                    'wait a
moment
Next

Locate 2 , 1                                                 'set cursor
position
Lcd "*"                                                      'display
this
Wait 1                                                       'wait a
moment

Shiftcursor Right                                            'shift the
cursor
Lcd "@"                                                      'display
this
Wait 1                                                       'wait a
moment

Home Upper                                                   'select line
1 and return home
Lcd "Replaced."                                              'replace the
text
Wait 1                                                       'wait a
```

```
moment

Cursor Off Noblink                                          'hide cursor
Wait 1                                                      'wait a
moment
Cursor On Blink                                             'show cursor
Wait 1                                                      'wait a
moment
Display Off                                                 'turn
display off
Wait 1                                                      'wait a
moment
Display On                                                  'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                  'goto home
on line three
Home Fourth
Home F                                                      'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                         'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                         'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                  'value into
ACC
!rCall _write_lcd                                           'put it on
LCD
End
```

Example2

```
'-----------------------------------------------------------
'                        EADOG-M163.bas
'          Demonstration for EADOG 163 display
'                        (c) 1995-2025, MCS Electronics
'-----------------------------------------------------------
'


$regfile = "M8515.dat"
$crystal = 4000000
'I used the following settings
'Config Lcdpin = Pin , Db4 = Portb.2 , Db5 = Portb.3 , Db6 = Portb.4 , Db7 = Portb.5 , E =
Portb.1 , Rs = Portb.0

'CONNECT vin TO 5 VOLT
Config Lcd = 16x3 , Chipset = Dogm163v5                     '16*3 type LCD display
```

```
'other options for chipset are DOG163V3 for 3Volt operation


'Config Lcd = 16 * 3 , Chipset = Dogm163v3 , Contrast = &H72              '16*3 type LCD display
'The CONTRAST can be specified when the default value is not what you need


'The EADOG-M162 is also supported :
'Chipset params for the DOGM162 : DOG162V5, DOG162V3

Cls                                                                      'Dit maakt het scherm leeg
Locate 1 , 1 : Lcd "Hello      World"
Locate 2 , 1 : Lcd "line      2"
Locate 3 , 1 : Lcd "line      3"
End
```

Example3

```
'-----------------------------------------------------------------------------
'name                : LCD-RX1602A5.bas
'copyright           : (c) 1995-2025, MCS Electronics
'purpose             : demonstrates I2C LCD library
'micro               : Mega88
'suited for demo     : yes
'commercial addon needed  : no
'The used library was sponsored by Lab microelectronic GmbH
'-----------------------------------------------------------------------------

$regfile = "m88def.dat"
$crystal = 8000000
$hwstack = 32
$swstack = 32
$framesize = 64

const vmode = 3                             ' 3V mode

$lib "Lcd_RX1602A5.lbx"
$lib "i2c_twi.lbx"                          ' use hardware twi or remark for software I2C

Config Twi = 100000                         ' 100kHz
config lcd = 16x2 , chipset = st7032

config SCL=PORTC.5
config SDA=PORTC.4


I2cinit

lcd_reset alias portc.2                     ' pin used for LCD RESET
lcd_light alias portd.7                     ' pin used for back light

Config lcd_reset = Output                   ' Display Reset
Config lcd_light = Output                   ' Display Licht


lcd_light = 1                               ' activate background LED
Lcd_reset = 0                               ' RESET mode
waitms 100
Lcd_reset = 1                               ' normal mode

initlcd                                     ' init LCD
lcdcontrast 30                              'a value between 30 and 40 works best at 3V

Do
  Cls
  Locate 1 , 1 : Lcd "test"
  Waitms 100                                '
Loop


End
```

Example 4

```
declare sub _lcdbefore()
```

```
declare sub _lcdafter()
config PORTB.0=OUTPUT
config LCD=16x2, before=1,after=1
CLS
LCD "test"
End

sub _lcdbefore()
  set portb.0
end sub

sub _lcdafter()
   reset portb.0
end sub
```

7.21.49 CONFIG LCDBUS

Action

Configures the LCD data bus and overrides the compiler setting.

Syntax

CONFIG LCDBUS = constant

Remarks

Constant	4 for 4-bit operation, 8 for 8-bit mode (default)

Use this statement together with the $LCD = address statement.

When you use the LCD display in the bus mode the default is to connect all the data lines. With the 4-bit mode, you only have to connect data lines d7-d4.

See also

CONFIG LCD 992

Example

```
'-----------------------------------------------------------
'                 (c) 1995-2025 MCS Electronics
'-----------------------------------------------------------
'  file: LCD.BAS
'  demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
'        CURSOR, DISPLAY
'-----------------------------------------------------------

'note : tested in bus mode with 4-bit on the STK200
'LCD   -   STK200
'------------------
'D4         D4
```

```
'D5          D5
'D6          D6
'D7          D7
'WR          WR
'E           E
'RS          RS
'+5V         +5V
'GND         GND
'V0          V0
'   D0-D3 are not connected since 4 bit bus mode is used!


'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Rem with the config lcdpin statement you can override the compiler
settings

$regfile = "8515def.dat"
$lcd = &HC000
$lcdrs = &H8000
Config Lcdbus = 4

Dim A As Byte
Config Lcd = 16x2                                          'configure lcd
screen
'other options are 16 * 2 , 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                         'clear the
LCD display
Lcd "Hello world."                                          'display
this at the top line
Wait 1
Lowerline                                                   'select the
lower line
Wait 1
Lcd "Shift this."                                           'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                           'shift the
text to the right
   Wait 1                                                   'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                            'shift the
text to the left
   Wait 1                                                   'wait a
moment
Next

Locate 2 , 1                                                'set cursor
position
Lcd "*"                                                     'display
this
Wait 1                                                      'wait a
```

```
moment

Shiftcursor Right                                          'shift the
cursor
Lcd "@"                                                    'display
this
Wait 1                                                     'wait a
moment

Home Upper                                                 'select line
1 and return home
Lcd "Replaced."                                            'replace the
text
Wait 1                                                     'wait a
moment

Cursor Off Noblink                                         'hide cursor
Wait 1                                                     'wait a
moment
Cursor On Blink                                            'show cursor
Wait 1                                                     'wait a
moment
Display Off                                                'turn
display off
Wait 1                                                     'wait a
moment
Display On                                                 'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                 'goto home
on line three
Home Fourth
Home F                                                     'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                        'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                        'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                 'value into
ACC
!rCall _write_lcd                                          'put it on
LCD
```

7.21.50 CONFIG LCDMODE

Action
Configures the LCD operation mode and overrides the compiler setting.

Syntax
CONFIG LCDMODE = type

Remarks

Type	**PORT** Will drive the LCD in 4-bit port mode and is the default. In PORT mode you can choose different PIN's from different PORT's to connect to the upper 4 data lines of the LCD display. The RS and E can also be connected to a user selectable pin. This is very flexible since you can use pins that are not used by your design and makes the board layout simple. On the other hand, more software is necessary to drive the pins. **BUS** will drive the LCD in bus mode and in this mode is meant when you have external RAM and so have an address and data bus on your system. The RS and E line of the LCD display can be connected to an address decoder. Simply writing to an external memory location select the LCD and the data is sent to the LCD display. This means the data-lines of the LCD display are fixed to the data-bus lines. Use $LCD 657 = address and $LCDRS 662 = address, to specify the addresses that will enable the E and RS lines.

See also
CONFIG LCD 992 , $LCD 657 , $LCDRS 662

Example
```
Config LCDMODE = PORT 'the report will show the settings
Config LCDBUS = 4  '4 bit mode
LCD "hello"
```

7.21.51 CONFIG LCDPIN

Action
Override the LCD-PIN select options.

Syntax
CONFIG LCDPIN = PIN , DB4= PN,DB5=PN, DB6=PN, DB7=PN, E=PN, RS=PN [WR=PIN] [BUSY=PIN] [MODE=mode]
CONFIG LCDPIN = PIN , PORT=PORTx, E=PN, RS=PN

Remarks

PN	The name of the PORT pin such as PORTB.2 for example.
PORTX	When you want to use the LCD in 8 bit data, pin mode, you must specify

	the PORT to use.
PIN	A port pin that is connected to the busy pin. The busy pin is only supported by the 20x4VFD display.
MODE	A mode for the 20x4VFD display. Options : 0 : 4 bit parallel upper nibble first 1 : 4 bit parallel lower nibble first

You can override the PIN selection from the Compiler Settings with this statement, so a second configuration lets you not choose more pins for a second LCD display.

The config command is preferred over the option settings since the code makes clear which pins are used. The CONFIG statement overrides the Options setting.

The PIN and MODE are only for the 20x4VFD display. See also LCDAUTODIM 1338

The WR pin is optional. When you select the WR pin, an alternative library will be used. This library uses the WR pin and reads the BUSY signal from the LCD.
The library lcd4busy_anypin will be used, which is based on Luciano's LUC_lcd4busy library.
Notice that since 2040 version, the compiler will generate LCD port pin info which you can use for your own libs.

By default the WR pin is optional and the WR signal of the LCD should be connected to ground. This saves the pin for other purposes. When you have enough pins, you better use the WR-pin.
If you do not connect the WR pin to ground but to a pin, and you do not specify the WR pin, but you set the logic level to 0 in your code, you have to use an INITLCD command after you have set the WR pin to 0.

See also

CONFIG LCD 992 , CONFIG LCDMODE 1001 , CONFIG LCDBUS 998

Example

```
'-----------------------------------------------------------------------
------------------
'name                    : lcd.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
SHIFTCURSOR, HOME
'                          CURSOR, DISPLAY
'micro                   : Mega8515
'suited for demo         : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8515.dat"                                      ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
```

```
use 40 for the frame space


$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation


'note : tested in PIN mode with 4-bit

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector

Rem with the config lcdpin statement you can override the compiler
settings


Dim A As Byte
Config Lcd = 16x2                                          'configure lcd
screen


'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                         'clear the
LCD display
Lcd "Hello world."                                          'display
this at the top line
Wait 1
Lowerline                                                   'select the
lower line
Wait 1
Lcd "Shift this."                                           'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                           'shift the
text to the right
   Wait 1                                                   'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                            'shift the
text to the left
   Wait 1                                                   'wait a
moment
Next
```

```
Locate 2 , 1                                                    'set cursor
position
Lcd "*"                                                         'display
this
Wait 1                                                          'wait a
moment

Shiftcursor Right                                               'shift the
cursor
Lcd "@"                                                         'display
this
Wait 1                                                          'wait a
moment

Home Upper                                                      'select line
1 and return home
Lcd "Replaced."                                                 'replace the
text
Wait 1                                                          'wait a
moment

Cursor Off Noblink                                              'hide cursor
Wait 1                                                          'wait a
moment
Cursor On Blink                                                 'show cursor
Wait 1                                                          'wait a
moment
Display Off                                                     'turn
display off
Wait 1                                                          'wait a
moment
Display On                                                      'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                      'goto home
on line three
Home Fourth
Home F                                                          'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                             'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                             'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                      'value into
```

```
ACC
!rCall _write_lcd                                          'put it on
LCD
End
```

7.21.52 CONFIG MODBUS

Action
This directive sets the MAKEMODBUS data mode.

Syntax
CONFIG MODBUS = DEFAULT | VAR

Remarks
When not configured, or when DEFAULT is chosen, the number of bytes passed in MakeModBus, is determined by the data type of the variable.

When configured to VAR, the content of the variable is used to pass the number of data bytes. The maximum value is 255.

See also
MAKEMODBUS 1499

Example
```
Print #1 , Makemodbus(2 , 1 , 8 , X);                ' slave 2, function
1, address 8 , send X byes where X is loaded with the number of bytes
```

7.21.53 CONFIG OPAMP

Action
This configuration statement configures the OPAMP (OPerational AMPlifier).

Syntax
CONFIG OPAMP=ENABLED|DISABLED , OPTIONn=VALUEn
The options and values depend on the processor. Below is a list of options and values. There can be multiple OPAMP'S. The **X** indicates the number in the range from 0-2.

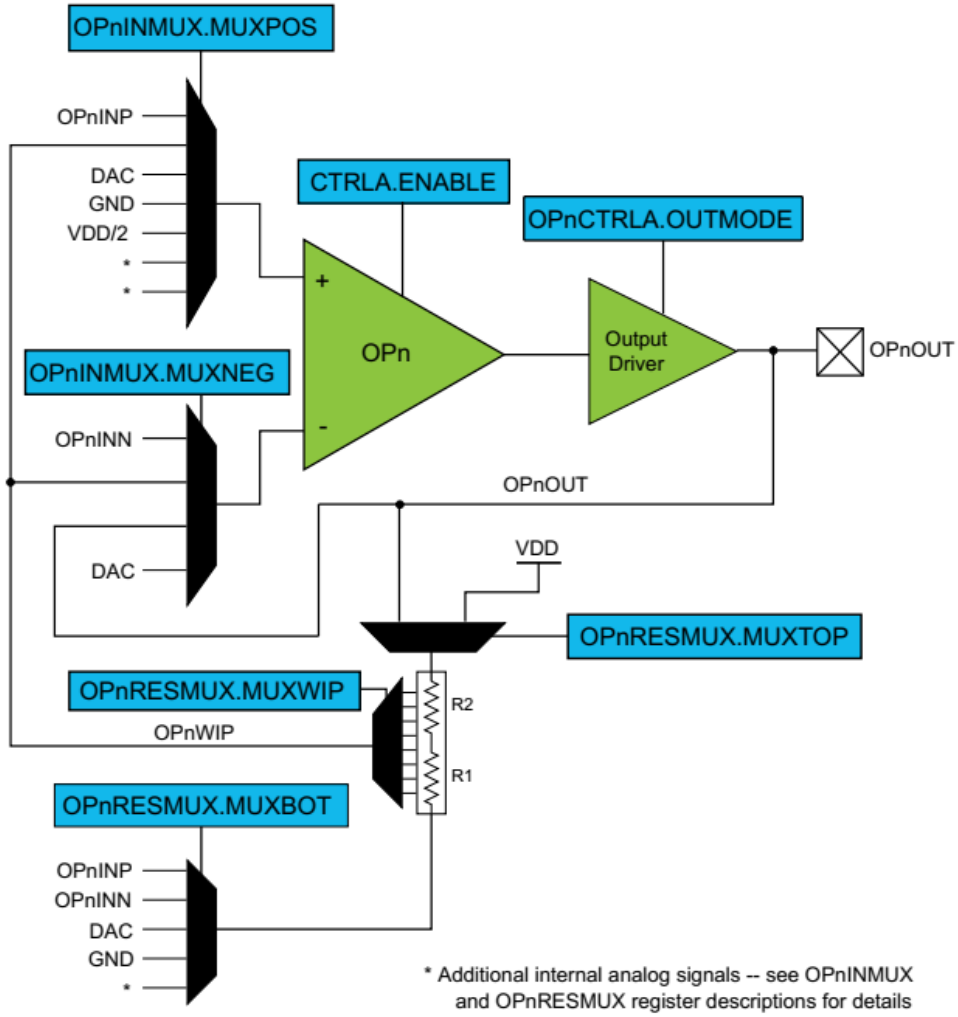
Remarks

OPTION	VALUE
OPAMP	- ENABLED to enable the OPAMP. - DISABLED to disable the OPAMP.
TIME_BASE	Controls the maximum value of a counter that counts CLK_PER cycles to achieve a time interval equal to or larger than 1 µs. It should be written with one less than the number of CLK_PER cycles that are equal to or larger than 1 µs. This is used for the internal timing of the warm up and settling times.
INP_RANGE	Selects the op amp input voltage range. - RAIL_TO_RAIL : The op amp input voltage range is rail-to-rail

	- REDUCED : The op amp input voltage range and power consumption are reduced
OPAMPx_RUNMODE	Run in standby mode. - DISABLED : the OPx is disabled when in standby sleep mode and its output driver is disabled. - ENABLED : the OPx will continue operating as configured in standby sleep mode.
OPAMP0_ALWAYS_ON	Controls whether the OPAMP is always on or not. - DISABLED : the OPx is not always on but can be enabled by the ENABLE EVENTx and disabled by the DISABLE EVENTx - ENABLED : the OPx is always on
OPAMP0_EVENTS	Controls event reception and generation. - DISABLED : No events are enabled for OPx - ENABLED : All events are enabled for OPx
OPAMP0_OUTMODE	Selects the output mode for the output driver - OFF : the output driver for OPx is disabled but this can overridden by the DRIVEx event - NORMAL : the output driver for OPx is enabled in NORMAL mode
OPAMP0_MUXWIP	Multiplexer for wiper. Selects the resistor ladder wiper (potentiometer) position - R1_15R_R2_1R : R1=15R , R2=1R - R1_14R_R2_2R : R1=14R , R2=2R - R1_12R_R2_4R : R1=12R , R2=4R - R1_8R_R2_8R : R1=8R , R2=8R - R1_6R_R2_10R : R1=6R , R2=10R - R1_4R_R2_12R : R1=4R , R2=12R - R1_2R_R2_14R : R1=2R , R2=14R - R1_1R_R2_15R : R1=1R , R2=15R
OPAMP0_MUXBOT	Selects the analog signal connected to the bottom resistor in the resistor ladder - OFF : multiplexer off - INP : positive input pin for OPx - INN : negative input pin for OPx - DAC : DAC output (DAC and DAC output buffer must be enabled) - LINKOUT : OPx-1 output. When selecting LINKOUT for OP0 MUXBOT is connected to the output of OP2 - GND : ground
OPAMP0_MUXTOP	Selects the analog signal connected to the top resistor in the resistor ladder - OFF : multiplexer off - OUT : OPx output - VDD : VDD
OPAMP0_MUXNEG	Selects which analog signal is connected to the inverting(-) input of OPx - INN : negative input pin for OPx - WIP : wiper from OPx resistor ladder - OUT : OPx output (unity gain) - DAC : DAC output (DAC and DAC output buffer must be enabled)
OPAMP0_MUXPOS	Selects which analog signal is connected to the non inverting(+) input of OPx - INP : positive input pin for OPx - WIP : wiper from OPx resistor ladder - DAC : DAC output (DAC and DAC output buffer must be enabled) - GND : ground - VDDDIV2 : VDD / 2 - LINKOUT : OPx-1 output. Only available for OP1 and OP2

	- LINKWIP : wiper from OP0 resistor ladder. Setting only available for OP2
OPAMP0_SETTLE	Specifies the number of microseconds allowed for the opamp to settle. This value together with the value in TIME_BASE is used by an internal timer to determine when to generate the READYx event and set the SETTLED flag in the OPx_STATUS register
OPAMP0_CAL	This value is a calibration value that adjusts the input offset voltage of the op amp. &H00 provides the most negative value of offset adjustment &H80 provides no offset adjustment, and &HFF provides the most positive value of offset adjustment
REGMODE	- OVERWITE : the entire register is updated. - PRESERVE : the register bits are preserved. See also the AVRX 503 topic.

The OPAMP has settings common for all OP's and each individual OPAMP (named OPx) has settings.
The following image from the datasheet shows a block diagram.



See also
NONE

Example

7.21.54 CONFIG OSC XMEGA

Action
Select and enable the oscillators available to the Xmega

See also ATXMEGA 425

Syntax Xmega
CONFIG OSC=ENABLED|DISABLED , **PLLOSC=**ENABLED|DISABLED, **EXTOSC=** ENABLED|DISABLED, **32KHZOSC=**ENABLED|DISABLED, **32MHZOSC=**ENABLED| DISABLED, **RANGE=**range, **32KHZPOWERMODE=**powermode, **XOSC_SEL__STARTUP=**xosc_sel_startup , **PLLSOURCE=**pll , **PLLDIV2=**plldiv , **PLLMUL=**pllmul , **32MHZCALIB=** 32mhzcalib , **2MHZCALIB=** 2mhzcalib , **2MHZDFL=** 2MHZDFL , **32MHZDFL=** 32MHZDFL

Remarks

OSC	Use ENABLED to enable the internal 2 MHZ oscillator. This oscillator is enabled by default. Use DISABLED to disable the internal oscillator.
PLLOSC	Use ENABLED to enable the PLL oscillator. The oscillator is disabled by default.
EXTOSC	Use ENABLED to enable the external oscillator. The external oscillator is disabled by default.
32KHZOSC	Use ENABLED to enable the internal 32 KHz oscillator. This oscillator is disabled by default.
32MHZOSC	Use ENABLED to enable the internal 32 MHz oscillator. This oscillator is disabled by default.
RANGE	Specify the range of the external oscillator. - 400KHZ_2MHZ - 2MHZ_9MHZ - 9MHZ_12MHZ - 12MHZ_16MHZ This option is only needed when using the external oscillator.
32KHZPOWERMODE	Select the power mode of the 32 KHz interal oscillator. This can be NORMAL or LOW_POWER. The default is NORMAL
XOSC_SEL_STARTUP	The type and startup type of the crystal or resonator can be specified. Use a value of : - EXTCLK (6 CLK) , will select external clock - 32KHZ (for 16 CLK) , will select 32.768 TOSC - XTAL_256CLK (for 256 CLK), will select 0.4-16 MHz XTAL - XTAL_1KCLK (for 1K CLK) , will select 0.4-16 MHz XTAL - XTAL_16CLK (for 16K CLK) , will select 0.4-16 MHz XTAL
PLLSOURCE	This option let you select the oscillator source of the PLL oscillator. Valid options are : - RC2MHZ , the internal 2 MHz oscillator (default) - RC32MHZ , the internal 32 MHz oscillator - EXTCLOCK , an external clock signal or oscillator
PLLDIV2	This option let you select the PLL two divider. Valid options are ENABLED and DISABLED

PLLMUL	This option let you specify the PLL multiplication factor. The numeric value must be in the range from 1-31. A value of 0 disables the multiplication.
32MHZCALIB	This option allow you to specify the calibration source for the 32MHZ oscillator. The possible options are : - RC32K , selects the 32.768 KHZ internal oscillator - XOSC32, selects the 32.768 KHz crystal oscillator on TOSC - USBSOF , selects USB start of frame
2MHZCALIB	This option allow you to specify the calibration source for the internal 2MHZ oscillator. The possible options are : - 32KHZINT , selects the 32.768 KHZ internal oscillator. (default) - 32KHZ_EXT_TOSC, selects the 32.768 KHz crystal oscillator on TOSC
32MHZDFL	This option will enable or disable the DFLL and auto calibration of the 32 MHZ oscillator. Possible values : - ENABLED - DISABLED
2MHZDFL	This option will enable or disable the DFLL and auto calibration of the 2 MHZ oscillator. Possible values : - ENABLED - DISABLED

You can also use automatic calibration. This will calibrate the 32 MHz oscillator using the 32 KHz oscillator.
The required code :

Config Osc = (enabled or disabled), 32mhzosc = Enabled , **32khzosc = enabled**
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
OSC_DFLLCTRL.0 = 1 'enable
DFLLRC32M_CTRL.0 = 1 'enable

See also
CONFIG SYSCLOCK 1081

Example
```
Config Osc = Enabled , 32mhzosc = Enabled  ' enable 2 MHz and 32 MHz
interal oscillators
```

PLL Example
```
  'Clock:  32 MHz    External 4 MHz Xtal, PLL x 8
Config osc = enabled , EXTOSC = enabled , pllosc = enabled , _
 range = 2MHZ_9MHZ , startup = XTAL_16KCLK , pllsource = extclock ,
pllmul = 8
Config Sysclock = Pll , Prescalea = 1 , Prescalebc = 1_1
```

7.21.55 CONFIG OSC XTINY

Action
Select and enables the oscillators available to the Xtiny/MegaX and AVRX

See also AVRX 503

Syntax

CONFIG OSC=ENABLED|DISABLED , OPTIONn=VALUEn
The options and values depend on the processor. Below is a list of options and values.

Remarks

OPTION	VALUE
OSC	ENABLED. The internal HF oscillator is always enabled. There is no option to disable it. So this is a kind of dummy variable.
RUNMODE	DISABLED or ENABLED. With RUNMODE enabled the oscillator will be forced to be always on. Otherwise it is only on when required.
AUTOTUNE	DISABLED or ENABLED. When enabled the HF oscillator can be tuned with the 32 KHz crystal oscillator. There is a CLKCTRL_OSCHFTUNE register that can be modified to tune.
FREQUENCY	This selects the frequency of the HF oscillator. Options are : [1MHZ,2MHZ,3MHZ,4MHZ,8MHZ,12MHZ,16MHZ,20MHZ,24MHZ]. Notice that the $CRYSTAL directive should match the setting.
PLL_RUNMODE	DISABLED or ENABLED. With RUNMODE enabled the oscillator will be forced to be always on. Otherwise it is only on when required.
PLL_SOURCE	OSCHF or XOSCHF. THis is the clock source for the PLL. Which is either the internal HF OSC or the external HF OSC.
PLL_MUL	DISABLED, 2 or 3. This is the PLL multiplication factor. With DISABLED, the PLL is disabled.
OSC32_RUNMODE	DISABLED or ENABLED. With RUNMODE enabled the oscillator will be forced to be always on. Otherwise it is only on when required.
XOSC32_RUNMODE	DISBLED or ENABLED. With RUNMODE enabled the oscillator will be forced to be always on. Otherwise it is only on when required.
XOSC32	DISBLED or ENABLED. This option allows to enable the EXTERNAL 32 KHZ oscillator.
XOSC32_SEL_STARTUP	XTAL_1KCLK , XTAL_16KCLK, XTAL_32KCLK or XTAL_64KCLK. These options set the 32 OSC crystal start up time in cycles.
XOSC32_EXT_SRC	EXT_XTAL or EXT_CLOCK_TOSC1. The source for the oscillaror. Either a crystal or an external clock signal on pin 1.
XOSC32_LPMODE	DISABLED or ENABLED. This option sets the Low Power mode.
XOSCHF	DISABLED,ENABLED]
XOSCHF_RUNMODE	DISABLED or ENABLED. With RUNMODE enabled the oscillator will be forced to be always on. Otherwise it is only on when required.
XOSCHF_SEL_STARTUP	XTAL_256CLK,XTAL_1KCLK or XTAL_4KCLK. The external HF oscillator crystal start up time.
XOSCHF_EXT_SRC	EXT_XTAL or EXT_CLOCK_XTALHF1. This options selects the source for the external HF oscillator clock source. Either a crystal or an external clock signal on the XTALHF1 pin
XOSCHF_RANGE	MAX_8MHZ,MAX_16MHZ,MAX_24MHZ or MAX_32MHZ. The maximum frequency supported for the external crystal. The larger the range selected

	the higher the current consumption by the oscillator.

As you can see there are a number of oscillators available.
The internal HF oscillator. An internal 32 KHZ oscillator. An external 32 KHz xtal can be connected for an external 32 KHz oscillator, and a HF crystal can be connected to a High Frequency external crystal.

See also
CONFIG SYSCLOCK 1082

Example
NONE

7.21.56 CONFIG PORT

Action
Sets the port or a port pin to the right data direction.

Syntax
CONFIG PORTx = state
CONFIG PINx = state
CONFIG PORTx.y = state
CONFIG PINx.y = state

Remarks

state	A numeric constant that can be INPUT or OUTPUT. INPUT will set the data direction register to input for port X. OUTPUT will set the data direction to output for port X. You can also use a number for state. **&B**00001111, will set the upper nibble to input and the lower nibble to output. You can either set a single port pin or a whole port to input or output. When you set a single pin , you can use INPUT, OUTPUT, 0 or 1. When you set a complete port, you can use INPUT, OUTPUT or a numeric constant that fits into a byte.
x	A valid port letter such as A,B,C etc. Example : CONFIG PORT**B** = INPUT Example : CONFIG PINB=OUTPUT
y	A valid pin number in the range of 0-7. Example : CONFIG PINB.**0**=OUTPUT Example : CONFIG PORTB.1=INPUT

The best way to set the data direction for more than 1 pin, is to use the CONFIG PORT, statement and not multiple lines with CONFIG PIN statements.

You may not use variables for the port letters and pin numbers. If you need to dynamically set a pin direction, you can use this form : SET PORTB.somepin , where

somepin may be a constant or a variable.
If the the port itself is also dynamic, then you could use OUT with the proper address.

PORT and PIN can equally be used. PIN can be used to indicate that you set a single pin. And PORT can be used to indicate that you set the complete PORT. But they both do the same.
There could be a reason to use PIN or PORT : when using an ALIAS like in this example:

Switch ALIAS PINB.0
LED ALIAS PORTB.1
CONFIG SWITCH=INPUT
CONFIG LED=OUTPUT
If SWITCH=0 THEN ' this works only on the PIN register

 When you want to read the status of an input pin you must use the PIN register.
When you want to set the output level of an output pin you must use the PORT register.
So you never write to a PIN register. Exceptions are for processors that have special ports that can toggle when you write to the PIN register.
The compiler will handle that automatic when you use the TOGGLE statement.
The example below show how to read a pin configured to act as an input pin and how to change a pin configured as output pin.

See Also

AVR Internal hardware ports 248 , SET 1438, RESET 1428, TOGGLE 1595

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : port.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: PortB and PortD
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte , Count As Byte

'configure PORT D for input mode
Config Portd = Input
```

```
'reading the PORT, will read the latch, that is the value
'you have written to the PORT.
'This is not the same as reading the logical values on the pins!
'When you want to know the logical state of the attached hardware,
'you MUST use the PIN register.
A = Pind

'a port or SFR can be treated as a byte
A = A And Portd

Print A                                                    'print it

Bitwait Pind.7 , Reset                                     'wait until
bit is low


'We will use port B for output
Config Portb = Output

'assign value
Portb = 10                                                 'set port B
to 10
Portb = Portb And 2

Set Portb.0                                                'set bit 0
of port B to 1

Incr Portb

'Now a light show on the STK200
Count = 0
Do
  Incr Count
  Portb = 1
  For A = 1 To 8
    Rotate Portb , Left                                    'rotate bits
left
    Wait 1
  Next
  'the following 2 lines do the same as the previous loop
  'but there is no delay
'  Portb = 1
'  Rotate Portb , Left , 8
Loop Until Count = 10
Print "Ready"

'Again, note that the AVR port pins have a data direction register
'when you want to use a pin as an input it must be set low first
'you can do this by writing zeros to the DDRx:
'DDRB =&B11110000  'this will set portb1.0,portb.1,portb.2 and portb.3
to use as inputs.

'So : when you want to use a pin as an input set it low first in the
DDRx!
'     and read with PINx
'     and when you want to use the pin as output, write a 1 first
'     and write the value to PORTx
End
```

7.21.57 CONFIG PORT_MUX

Action

This configuration option allows you to configure the PORTMUX. The PORTMUX allows to chose alternative pin locations.

Syntax

CONFIG PORT_MUX = val0 , opt1=val1,opt2=val2, optx=valx

Remarks

val0	There are 2 possible settings : - OVERWITE : the entire register is updated. - PRESERVE : the register bits are preserved. See a detailed explanation below.
opt1, opt2, optx	These are the various options which will depend on the processor. Possible options are : - EVOUT0 : event output enable - EVOUTx : event output x enable - LUTx : alternative pin location CCL LUTx - USARTx : alternative pin location USARTx - SPI0 : alternative pin location SPI0 - TWI0 : alternative pin location TWI0 - TCA0x : alternative pin location wave output - TCB0x : alternative pin location wave output
valx	The option value. It is either ENABLED or DISABLED The default register value is DISABLED.

You can use the CTRL+SPACE key combination to get a list of options and values. This only works when you specified the definition file with $REGFILE. And when there are no errors in your code.

The PORTMUX is a convenient piece of hardware. It allows you to swap pin locations of hardware that share pins. As the pins are limited most pins share hardware functions.
For example for the TINY816 portA.1 : Besides being a normal port pin it is also MOSI, AIN1 and LUT0-IN1.
Now the PB2 and PB3 pins are used for TX/RX and TOSC1 and TOSC2. This means that you can not use the external oscillator AND the UART TX/RX pins. You need to chose.
But since the TX/RX pins have the option to be swapped with an alternative pin location, you can now use both !
So you would swap the USART0 pins from PB3(RX),PB2(TX),PB1(XCK) to PA1,PA2, PA3.
These PA1,PA2 and PA3 location are normally intended for the SPI and if you need that, you can also swap the SPI to PC0,PC1,PC2 and PC3.

Notice that all the device pins are swapped that belong to a device.
The following table from the data sheet make things more clear ;

5.1 Multiplexed Signals

Table 5-1. PORT Function Multiplexing

VQFN 20-Pin	SOIC 20-Pin	Pin Name (1,2)	Other/Special	ADC0	PTC(4)	AC0	DAC0	USART0	SPI0	TWI0	TCA0	TCB0	TCD0	CCL
19	16	PA0	RESET/ UPDI	AIN0										LUT0-IN0
20	17	PA1		AIN1				TxD(3)	MOSI	SDA(3)				LUT0-IN1
1	18	PA2	EVOUT0	AIN2				RxD(3)	MISO	SCL(3)				LUT0-IN2
2	19	PA3	EXTCLK	AIN3				XCK(3)	SCK		WO3			
3	20	GND												
4	1	VDD												
5	2	PA4		AIN4	X0/Y0			XDIR(3)	SS		WO4		WOA	LUT0-OUT
6	3	PA5		AIN5	X1/Y1	OUT					WO5	WO	WOB	
7	4	PA6		AIN6	X2/Y2	AINN0	OUT							
8	5	PA7		AIN7	X3/Y3	AINP0								LUT1-OUT
9	6	PB5	CLKOUT	AIN8		AINP1					WO2(3)			
10	7	PB4		AIN9		AINN1					WO1(3)			LUT0-OUT(3)
11	8	PB3	TOSC1					RxD			WO0(3)			
12	9	PB2	TOSC2, EVOUT1					TxD			WO2			
13	10	PB1		AIN10	X4/Y4			XCK		SDA	WO1			
14	11	PB0		AIN11	X5/Y5			XDIR		SCL	WO0			
15	12	PC0							SCK(3)			WO(3)	WOC	
16	13	PC1							MISO(3)				WOD	LUT1-OUT(3)
17	14	PC2	EVOUT2						MOSI(3)					
18	15	PC3							SS(3)		WO3(3)			LUT1-IN0

The compiler will set the proper registers based on your configuration.
There are 2 important settings : OVERWRITE and PRESERVE.
CONFIG PORT_MUX=PRESERVE will preserve the other settings in case they are not all configured.
Imagine a register with 4 bits and your setting only changes one bit. The compiler will read the data, change the bit and write it back.
When you change all 4 bits, the compiler will just write the new value since there is no need to preserve the old value.

When you use CONFIG PORT_MUX=OVERWRITE, the compiler will not preserve the old values, it will just write the new value. Since all registers are default 0 this is not a problem in many cases. But it could be when you dynamic change the settings.
It is important that you specify all settings on one line or use the line continuation character. This will give the best code.
When 1 register is updated, lds/sts is used while when multiple registers are updated, a pointer is used.
So we would recommend to use OVERWRITE for the initial setup. Normally there is no need to change the configuration at run time. But when you do need to change it, use the PRESERVE mode.

Other CONFIG statements might also support the OVERWRITE/PRESERVE switch. You will find this when the REGMODE option is present among the options.

When the port multiplexer is configured it will not change the port direction settings. You need to do so yourself when that is required.
For example when you use the default settings for the USART/COM, the TX is set to output mode.
When you change the UART pins with the multiplexer you need to set the new TX pin to output mode.

There is also a simpler way to just set the alternative pins for the USART. The CONFIG COMx have an option : TXPIN=xxx
Where xxx is either the default (DEF_PA0) or ALT1_PA4 or NONE. The setting values depend on the used processor.
DEF_ means that this is the default value. So you do not need to specify it. In fact when you use the default value you should not specify it since it will create more code

because the port_mux is automatically set and the port mux registers are preserved.
ALT1_ means that this is the first alternative value. So PORTA4 would be used instead of PORTA.0
NONE means that none of the pins are connected.

Since the TX pin is set to output mode, the preferred way to set the USART alternative pin is using CONFIG COM.
The PORT_MUX will be updated automatically. In this scenario you should however use the PRESERVE mode since otherwise you might erase the USART alternative TX setting !

See also
NONE

Example
```
'-----------------------------------------------------------------
----------------
'name                     : portmux.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates PORT_MUX
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24

'set the system clock and prescaler
Config Sysclock = 16_20mhz , Prescale = 1

'configure the USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'dimension a variable
Dim B As Byte

Print "Test USART"

For B = 1 To 10
   Print "Hello" ; Spc(3) ; B
   Waitms 1000
Next
```

```
'now use the port mux to switch the USART pins
Config Port_mux = Overwrite , Usart0 = Alt1_pa1pa4 , Evout0 =
Enabled
'we need to set the new TX pin to output outselfs. This is pin
PA1 for the tiny816
Config Porta.1 = Output
Do
   Print "ALT TX" ; Spc(3) ; B
   Waitms 1000
   Incr B
Loop

End
```

7.21.58 CONFIG POWERMODE

Action

Put the micro processor in one of the supported power reserving modes.
Config Powermode is for ATTINY, ATMEGA and ATXMEGA devices.

Syntax

CONFIG POWERMODE = mode

Example

```
Config Powermode = Powerdown
```

or

```
CONFIG POWERMODE = IDLE
```

Remarks

The mode depends on the micro processor.
Some valid options for ATTINY and ATMEGA are :
- IDLE
- POWERDOWN
- STANDBY
- ADCNOISE
- POWERSAVE

Valid option for ATXMEGA are:
- Idle
- PowerDown
- PowerSave
- Standby
- ExStandby

The modes and their exact behaviour is different on all processors. The following description from the data sheet is for the Mega88P.

Keep in mind that you can only achieve the low current consumption of ATTINY and ATMEGA in PowerDown mode when you also consider the "MINIMIZING POWER

CONSUMPTION"
section in the data sheet like:

```
' 1. Disable/Switch off ADC
' 2. Disable/Switch off Analog Comparator
' 3. Disable Brown-out Detection when not needed
' 4. Disable internal voltage reference
' 5. Disable Watchdog Timer when not needed
' 6. Disable the digital input buffer
' 7. Enable Pull-up or pull-down an all unused pins
```

In case of ATXMEGA see also CONFIG POWER_REDUCTION 1023 to reduce the power consuption in all modes.

 If you measure the current consumption not between the LDO and AVR don't forget to use Low Quiescent Current LDO for example MCP1700, AS1375 or TPS78233 to really get close to the current consumption in the data sheet.

 You can also minimize power consumption by keeping the clock frequency as low as possible if sleep modes are not used.

Wake up from Sleep Modes

In the AVR data sheets you find under the sleep modes the **wake up sources** for sleep modes.

For example for an ATTINY25/45/85. The only wake up Sources from PowerDown are:
- INT0 and Pin Change (For INT0, only level interrupt)
- USI Start Condition
- Watchdog Interrupt

The wake up sources for an ATXMEGA32A4U from powerdown are:
- USB Resume
- Asynchronous Port Interrupts
- TWI Address Match Interrupts

Asynchronous pin-change sensing with ATXMEGA means that a pin change can wake the device from all sleep modes, included the modes where
no clocks are running (Synchronous sensing requires the presence of the peripheral clock, while asynchronous sensing does not require any
clock.)
See also: ATXMEGA 425

You will find an example below with ATXMEGA, PowerDown and Wake up from asynchronous Port Pin.

Example for Powerdown with ATXMEGA

```
$regfile = "XM256A3BUDEF.DAT"
$crystal = 32000000                                         '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80


Config Osc = Enabled , 32mhzosc = Enabled
```

```
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Power_reduction = Dummy , Aes = Off , Twic = Off , Twid = Off ,
Twie = Off , Aca = Off , Adcb = Off , Tcc0 = Off , Tcc1 = Off , Dma =
Off

' Here you have 5 seconds to measure the current consumption with multi
meter
wait 5

Config Powermode = Powerdown

End
```

Example with ATXMEGA, PowerDown and Wake up from asynchronous Port Pin.

```
' The following example give you 5 seconds to measure the current in
active mode
' Then you have time to measure the current in PowerDown mode
' after this you can wake up the XMEGA from PowerDown with Portf.2 until
the ATXMEGA will
' go to PowerDown mode again after 5 seconds
' The hardware used for this example is XMEGA-A3BU Xplained board from
ATMEL

$regfile = "XM256A3BUDEF.DAT"
$crystal = 32000000                                          '32MHz
$hwstack = 64
$swstack = 40
$framesize = 80

Config Osc = Enabled , 32mhzosc = Enabled
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Power_reduction = Dummy , Aes = Off , Twic = Off , Twid = Off ,
Twie = Off , Aca = Off , Adcb = Off , Tcc0 = Off , Tcc1 = Off , Dma =
Off

Config Priority = Static , Vector = Application , Lo = Enabled , Med =
Enabled , Hi = Enabled

'When a button is pressed it will drive the I/O line to GND.
'We use SW2 (Switch 2) on the A3BU XPLAINED Board
'This Switch is connected to PortF.2 which is an asynchronous Pin (Every
Pin 2 is an asynchronous pin)

'Other Pins can also wake up the XMEGA but only "Both Edges" and "Low
Level is supported and in addition the
'Pin value must be kept unchanged during wake up

On Portf_int0 Wake_up
Enable Portf_int0 , Hi

Config Portf.2 = Input
Config Xpin = Portf.2 , Sense = Falling
Portf_int0mask = &B0000_0100                                 ' Assign pin
F2

Enable Interrupts

Do
' Here you have 5 seconds to measure the current consumption with multi
meter
```

```
wait 5
Config Powermode = Powerdown
Loop

End

Wake_up:

Return
```

IDLE MODE (ATMEGA88)

The Idle mode will stop the CPU but allowing the SPI, USART, Analog Comparator, ADC, 2-wire Serial
Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing the other clocks to run.
Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ACD bit in the Analog Comparator Control and Status Register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

ADC NOISE REDUCTION (ATMEGA88)

This mode will stop the CPU but allowing the ADC, the external interrupts, the 2-wire Serial Interface address watch, Timer/Counter2(1), and the Watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing the other clocks to run.
This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart from the ADC Conversion Complete interrupt, only an External Reset, a Watchdog System Reset, a Watchdog Interrupt, a Brown-out Reset, a 2-wire Serial Interface address match, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an external level interrupt on INT0 or INT1 or a pin change interrupt can wake up the MCU from ADC Noise Reduction mode.

POWERDOWN (ATMEGA88)

In this mode, the external Oscillator is stopped, while the external interrupts, the 2-wire Serial Interface address watch, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog System Reset, a Watchdog Interrupt, a Brown-out Reset, a 2-wire Serial Interface address match, an external level interrupt on INT0 or INT1, or a pin change interrupt can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.
Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU.
When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL Fuses that define the Reset Time-out period, as described in ”Clock Sources”

POWERSAVE (ATMEGA88)

This mode is identical to Power-down, with one exception:
If Timer/Counter2 is enabled, it will keep running during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK2, and the Global Interrupt Enable bit in SREG is set.
If Timer/Counter2 is not running, Power-down mode is recommended instead of Power-save mode.
The Timer/Counter2 can be clocked both synchronously and asynchronously in Power-save mode. If Timer/Counter2 is not using the asynchronous clock, the Timer/Counter Oscillator is stopped during sleep. If Timer/Counter2 is not using the synchronous clock, the clock source is stopped during sleep. Note that even if the synchronous clock is running in Power-save, this

clock is only available for Timer/Counter2.

STANDBY (ATMEGA88)

This mode is identical to Power-down
with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

EXTENDED STANDBY (ATMEGA88)

This mode is identical to
Power-save with the exception that the Oscillator is kept running. From Extended Standby mode, the device wakes up in six clock cycles.

So for standby you would use : POWER STANDBY
It is also possible to use POWERDOWN, IDLE or POWERSAVE. These modes were/are supported by most processors. It is recommended to use the new CONFIG POWERMODE command because it allows to use more modes.

See also

IDLE 1313, POWERDOWN 1398 , POWERSAVE 1399, CONFIG POWER_REDUCTION 1023

Example for Powerdown and wake up with ATTINY

```
' Using the new config powermode = PowerDown function with ATTINY13

' Fuse Bits:
' Disable DWEN (Debug Wire) Fuse Bit
' Disable Brown-Out Detection in Fuse Bits
' Disable Watchdog in Fuse Bits

' You can also just use Config Powermode = Powerdown

' But this example here also considers what the data sheet write under "MINIMIZING POWER CONSUMPTION"
' You need to follow this when you want to achieve the current consumption which you find in the data sheet under Powerdown Mode

' 1. Disable/Switch off ADC
' 2. Disable/Switch off Analog Comparator
' 3. Disable Brown-out Detection when not needed
' 4. Disable internal voltage reference
' 5. Disable Watchdog Timer when not needed
' 6. Disable the digital input buffer
' 7. Enable Pull-up or pull-down an all unused pins


$regfile = "attiny13.dat"
$crystal = 9600000                                          '9.6MHz
$hwstack = 10
$swstack = 0
$framesize = 24


On Int0 Int0_isr                                            'INT0 will be the wake-up source for Powerdown Mode
Config Int0 = Low Level
Enable Int0


' Prepare Powerdown:
' To minimize power consumption, enable pull-up or -down on all unused pins, and
' disable the digital input buffer on pins that are connected to analog sources
Config Portb.0 = Input
Set Portb.0
Config Portb.1 = Input                                      'INT0 --> external 47K pull-up
'Set Portb.1
Config Portb.2 = Input
```

```
Set Portb.2
Config Portb.3 = Input
Set Portb.3
Config Portb.4 = Input
Set Portb.4
Config Portb.5 = Input                                    'External Pull-Up (Reset)

Didr0 = Bits(ain1d , Ain0d)                               'Disable digital input buffer
on the AIN1/0 pin

Set Acsr.acd                                              'Switch off the power to the
Analog Comparator
'alternative:
' Stop Ac

Reset Acsr.acbg                                           'Disable Analog Comparator
Bandgap Select

Reset Adcsra.aden                                         'Switch off ADC
'alternative:
' Stop Adc

'#################################################################################
Do
   Wait 3                                                 ' now we have 3 second to
measure the Supply Current in Active Mode

   Enable Interrupts

   ' Now call Powerdown function
   Config Powermode = Powerdown

   'Here you have time to measure PowerDown current consumption until a Low Level on
Portb.1 which is the PowerDown wake-up
Loop
'#################################################################################
End


Int0_isr:
' wake_up
Return
```

Example for Idle and wake up with ATTINY

```
' Using the new config powermode = Idle function with ATTINY13

' Idle: This sleep mode basically halts clkCPU and clkFLASH, while allowing the other
clocks to run.

' Fuse Bits:
' Disable DWEN (Debug Wire) Fuse Bit
' Disable Brown-Out Detection in Fuse Bits
' Disable Watchdog in Fuse Bits


$regfile = "attiny13.dat"
$crystal = 1200000                                        '1.2MHz (9.6MHz/DIV8 =
1.2MHz)
$hwstack = 10
$swstack = 0
$framesize = 24


On Int0 Int0_isr                                          'INT0 will be the wake-up
source for Idle Mode
Config Int0 = Low Level
Enable Int0


'#################################################################################
Do
   Wait 3                                                 ' now we have 3 second to
measure the Supply Current in Active Mode

   Enable Interrupts

   ' Now call Idle function
   Config Powermode = Idle

   'Here you have time to measure Idle current consumption until a Low Level on Portb.1
which is the Idle wake-up
```

```
Loop
'###############################################################################
End


Int0_isr:
'   wake_up
Return
```

7.21.59 CONFIG POWER_REDUCTION

Action

This option configures the power reduction registers to reduce power consumption.

Syntax

CONFIG POWER_REDUCTION= dummy, device=ON|OFF

Remarks

The Power Reduction (PR) registers provides a method to stop the clock to individual peripherals.
When this is done the current state of the peripheral is frozen and the associated I/O registers cannot be read or written. Resources used by the peripheral will remain occupied;
hence the peripheral should in most cases be disabled before stopping the clock. Enabling the
clock to a peripheral again, puts the peripheral in the same state as before it was stopped. This
can be used in Idle mode and Active mode to reduce the overall power consumption significantly.
In all other sleep modes, the peripheral clock is already stopped.
Not all devices have all the peripherals associated with a bit in the power reduction registers.
Setting a power reduction bit for a peripheral that is not available will have no effect.

Device	A hardware resource of the Xmega. The following hardware resources can be deactivated to reduce power: AES EBI LCD RTC EVSYS DMA DACA, DACB ACA,ACB ADCA,ADCB TWIC,TWID,TWIE,TWIF USARTC0,USARTC1, USARTD0,USARTD1,USARTE0, USARTE1,USARTF0,USARTF1 SPIC,SPID,SPIE,SPIF TCC0,TCC1,TCD0,TCD1,TCE0,TCE1,TCF0,TCF1

	HIRESC,HIRESD,HIRESE,HIRESF XCL A value of ON will leave the resource enabled and a value of OFF will activate the power reduction.

You should use the CONFIG POWER_REDUCTION at start up to disable all unused resources. All the power reduction registers will be set for the provided resources. But the existing configuration will not be preserved. When you need to enable/disable an individual resource at run time, you can manual access the register with a SET or RESET command.
For example, the DMA, EVSYS, RTC, EBI and AES bits are located in the PRGEN register. If you disable DMA and AES the compiler will write a value of 17 (dma +aes) to the PRGEN register.
It will not first read the existing value, and preserve the other bits. That is why this statement should be used once.
When you specify one value, for example DMA, it will write 1 to the PRGEN register and thus overwriting the previous AES bit that was 1, with a 0.

The additional code to mask and set the bits did not seem useful at implementation time. At user request this behaviour can be changed in a future version.

See also
NONE

Example
```
'-----------------------------------------------------------
'             XM128A1-POWER-REDUCTION.BAS
'               (c) 1995-2025 MCS Electronics
' sample provided by MAK3
'-----------------------------------------------------------

' CONFIG POWER_REDUCTION and USING EVENT SYSTEM

' This Example show how to use the config power_reduction and give first
insights to the XMEGA EVENT SYSTEM

' Regarding the Eventsytem this example easy show after event
configuration that one Port Pin is routed to another Port Pin.
' You can see it works even during the WAIT 4 command and there are no
PORT READ OR WRITE commands in the Do .... Loop !
' It also shows how to manual fire an Event

$regfile = "xm128a1def.dat"
$crystal = 2000000                                          ' 2MHz
$hwstack = 64
$swstack = 40
$framesize = 40

Config Osc = Enabled
Config Sysclock = 2mhz                                      ' 2MHz

' YOU CAN MINIMIZE POWER CONSUMPTION FOR EXAMPLE WITH :
' 1. Use Low supply voltage
' 2. Use Sleep Modes
' 3. Keep Clock Frequencys low (also with Precsalers)
' 4. Use Powe Reduction Registers to shut down unused peripherals
```

```
'With Power_reduction you can shut down specific peripherals that are
not used in your application
'Paramters:
aes,dma,ebi,rtc,evsys,daca,dacb,adca,adcb,aca,acb,twic,usartc0,usartc1,s
pic,hiresc,tcc0,tcc1
Config Power_reduction = Dummy , Aes = Off , Twic = Off , Twid = Off ,
Twie = Off , Aca = Off , Adcb = Off , Tcc0 = Off , Tcc1 = Off , Dma =
Off

'For the following we need the EVENT System therefore we do not shut
down EVENT SYSTEM

Config Com1 = 9600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Open "COM1:" For Binary As #1
Waitms 2

Print #1 ,
Print #1 , "----------S T A R T----------------"

'Configure PC0 for input, triggered on falling edge
Config Pinc.0 = Input
Portc_pin0ctrl = &B00_011_010
                 '^   ^
                 '^   React on falling edge (010)
                 '^
                 'enable Pullup

'Select PC0 as input to event channel 0
'select the event source for Event Channel 0
Evsys_ch0mux = &B0110_0_000                    'Event Source for
Event Channel 0  = Portc.0
                '^       ^
                '^       ^
                '^       Pin0
                'portC

Evsys_ch0ctrl = &B0_00_0_0_111                 '8 SAMPLES for Digital
Filter
                          '^
                          'Digital Filter config
Config Pinc.7 = Output
'Event Channel 0 Ouput Configuration
Portcfg_clkevout = &B0_0_01_0_0_00             'Output on PINC.7
/Clock Out must be disabled

Print #1 , "Portcfg_clkevout = " ; Bin(portcfg_clkevout)
Print #1 , "Mainloop -->"


Do
  'IMPORTANT: YOU WILL SEE THE PIN CHANGES ALSO DURING WAIT 4 BECAUSE IT
USE EVENT SYSTEM
  Wait 4

  'This shows how to manual fire an Event
  Set Evsys_strobe.0
Loop

End                                            'end program
```

7.21.60 CONFIG PRIORITY XMEGA

Action
Configures the interrupt system and priority for Xmega

Syntax
CONFIG PRIORITY= prio, **VECTOR**= vector, **HI**= hi, **LO**= lo, **MED**= med

Remarks

prio	STATIC or ROUNDROBIN. In the AVR the lowest interrupt address has the highest priority. When you chose STATIC the interrupts behave as in non-Xmega chips. To prevent that a low priority interrupt never get executed you can select ROUNDROBIN
vector	APPLICATION or BOOT. Application is the default. This will place the interrupt vectors at address 0, the starting address. When you chose BOOT, the interrupt vectors are placed at the beginning of the boot section. This makes it possible to use interrupts in a boot application.
hi	ENABLED or DISABLED. Chose ENABLED to enable the HI priority interrupts.
lo	ENABLED or DISABLED. Chose ENABLED to enable the LO priority interrupts.
med	ENABLED or DISABLED. Chose ENABLED to enable the MED priority interrupts.

In the XMEGA, you must enable HI, LO or MED interrupts before you can use them.
When you enable an interrupt you also must specify the priority.
For example : Enable Usartc0_rxc , Lo
This would enable the USARTC0_RX interrupt and would assign it a low priority.

In this case, at least the LO priority should be enabled :
Config Priority = Static , Vector = Application , Lo = Enabled

When you use LO and MED interrupts, you need to enable the both.

 When you do not specify the priority when enabling an interrupt like : **ENABLE Tcc0_ovf** , the compiler will use the MED interrupt level. This means that you must enable this as well when using CONFIG PRIORITY. When you do NOT use CONFIG PRIORITY, but only ENABLE INTERRUPTS, the compiler will activate the MED interrupt automatically.
So when not using CONFIG PRIORITY all will work out just fine, but when using CONFIG PRIORITY, do not forget to enable the MED priority.

See also
ENABLE 1250 , DISABLE 1240 , ON 1379

Example
```
Config Priority = Static , Vector = Application , Lo = Enabled
```

```
On Usartc0_rxc Rxc_isr
Enable Usartc0_rxc , Lo
Enable Interrupts
```

7.21.61 CONFIG PRIORITY XTINY

Action

Configures the interrupt system and priority for Xmega

Syntax

CONFIG PRIORITY= prio, **VECTOR**= vector, **COMPACT**= compact , **HI**= hi

Remarks

prio	STATIC or ROUNDROBIN. In the AVR the lowest interrupt address has the highest priority. When you chose STATIC the interrupts behave as in non-Xmega chips. To prevent that a low priority interrupt never get executed you can select ROUNDROBIN
vector	APPLICATION or BOOT. Application is the default. This will place the interrupt vectors at address 0, the starting address. When you chose BOOT, the interrupt vectors are placed at the beginning of the boot section. This makes it possible to use interrupts in a boot loader application.
compact	ENABLED or DISABLED. Chose ENABLED to write a compact interrupt table.
hi	The interrupt that should be given a HIGH priority. Only one interrupt can be given a HIGH priority. Possible values : NMI,VLM,PORTA_INT,PORTB_INT,PORTC_INT,RTC_OVF,PIT_OVF, TCA0_LUNF,TCA0_HUNF,TCA0_CMP0,TCA0_CMP1,TCA0_CMP2,TCB0_INT, TCD0_OVF,TCD0_TRIG,AC0_COMP0,ADC0_RDY,ADC0_WCOMP, TWI0_SLAVE,TWI0_MASTER,SPI0_INT,USART0_RXC,USART0_DRE, USART0_TXC,NVM_EE

In the XTINY, you can provide one interrupt to be serviced with a HIGH priority.
By default the interrupt address with the lowest address will get the highest interrupt.
This is NMI with address 0.
After that the PORT interrupts will get priority over all the other interrupts.
When you would like that USART0_RXC would get a HIGH (or the highest) interrupt, you can specify the interrupt name so it will be serviced with the highest priority.

Config Priority = Static , Vector = Application , HI = USART0_RXC

See also

ENABLE 1250 , DISABLE 1240 , ON 1379

Example

```
Config Priority = Static , Vector = Application , Hi = USART0_RXC

On Usartc0_rxc Rxc_isr
Enable Usartc0_rxc
Enable Interrupts
```

7.21.62 CONFIG PRINT

Action
Configure the UART to be used for RS-485

Syntax
CONFIG PRINT0 = pin, mode = mode [, delay=ms]
CONFIG PRINT1 = pin, mode = mode [, delay=ms]
CONFIG PRINT2 = pin, mode = mode [, delay=ms]
CONFIG PRINT3 = pin, mode = mode [, delay=ms]
CONFIG PRINT4 = pin, mode = mode [, delay=ms]
CONFIG PRINT5 = pin, mode = mode [, delay=ms]
CONFIG PRINT6 = pin, mode = mode [, delay=ms]
CONFIG PRINT7 = pin, mode = mode [, delay=ms]

Remarks

pin	The name of the PORT pin that is used to control the direction of an RS-485 driver such as PORTB.1
mode	SET or RESET
delay	Optional delay in mS. This delay is used before the direction is switched back after the transmit buffer is empty. This to compensate for slow RS485 drivers.

Use PRINT or PRINT0 for the first serial port. Use PRINT1 for the second serial port. PRINT2 for the third UART and PRINT3 for the fourth UART.

When you use RS-485 half duplex communication you need a pin for the direction of the data. The CONFIG PRINT automates the manual setting/resetting. It will either SET or RESET the logic level of the specified pin before data is printed with the BASCOM print routines. After the data is sent, it will inverse the pin so it goes into receive mode.

 You need to set the direction of the used pin to output mode yourself.

When CONFIG PRINT is used, the PRINT and PRINTBIN statements will switch the pin logic level, send the data, wait till all data is sent, optional wait the specified time in mS, and then will switch the pin logic level back.
CONFIG PRINT will not work with dynamic Xmega UARTS (BUART). You need to use a constant channel with the Xmega like PRINTBIN #1.
CONFIG PRINT does not work with buffered serial output.

A popular line driver for RS485 communication is the MAX485. But most driver chips are similar.
The driver usually has an /RE pin (/ means inverted) which need to be made low in order to enable the receiver.
The driver also has a DE pin. Which is the driver output enable. This pin is not inverted. You need to make it high in order to enable the data driver output.
So when using the MAX485 as a master in half duplex mode to send data as in the example below, you would connect portb.0 to the DE pin. And you would use SET in the configuration since in order to print the driver must be SET high.

See also
CONFIG PRINTBIN 1029

Example
```
'-----------------------------------------------------------------------
-------
'name                     : rs485.bas
'copyright                : (c) 1995-2054, MCS Electronics
'purpose                  : demonstrates
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-------
$regfile = "m48def.dat"                                    ' we use the
M48
$crystal = 8000000
$baud = 19200

$hwstack = 32
$swstack = 32
$framesize = 32

Config Print0 = Portb.0 , Mode = Set
Config Pinb.0 = Output                                     'set the
direction yourself

Dim Resp As String * 10
Do
   Print "test message"
   Input Resp                                              ' get
response
Loop
```

7.21.63 CONFIG PRINTBIN

Action
Configure PRINTBIN behavior

Syntax
CONFIG PRINTBIN = mode

Remarks

mode	The mode value is either EXTENDED or NORMAL. **EXTENDED** The extended mode is the only mode you can configure. It allows to send packets greater than 255 bytes. For example when you need to send an array with more than 255 elements. The maximum packet size is 64 KB. Because support for big packets requires more code, it is

	made optional. You can not change between normal and extended mode dynamically. If you chose to use extended mode, this will be used for all your PRINTBIN code. The internal constant named _PBIN_EXTENDED will be set to 1. When you do not configure PRINTBIN, it will have the default value of 0. **NORMAL** The normal mode is the default. When you do not use CONFIG PRINTBIN, the default NORMAL mode is selected. You can not switch dynamic between the 2 modes.

See also
CONFIG PRINT 1028 , PRINTBIN 1504

Example
```
$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Config Printbin = Extended
Dim A(1000)
Printbin A(1) ; 1000
```

7.21.64 CONFIG PS2EMU

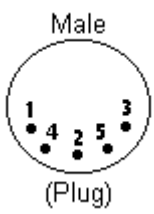
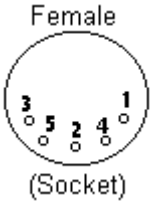
Action
Configures the PS2 mouse data and clock pins.

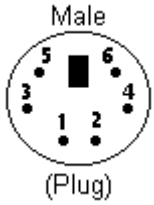
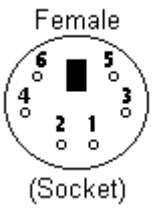
Syntax
CONFIG PS2EMU= int , DATA = data, CLOCK=clock

Remarks

Int	The interrupt used such as INT0 or INT1.
DATA	The pin that is connected to the DATA line. This must be the same pin as the used interrupt.

CLOCK	The pin that is connected to the CLOCK line.

		5-pin DIN (AT/ XT): 1 - Clock 2 - Data 3 - Not Implemented 4 - Ground 5 - +5v

		6-pin Mini-DIN (PS/2): 1 - Data 2 - Not Implemented 3 - Ground 4 - +5v 5 - Clock 6 - Not Implemented

Old PC's are equipped with a 5-pin DIN female connector. Newer PC's have a 6-pin mini DIN female connector.
The male sockets must be used for the connection with the micro.

Besides the DATA and CLOCK you need to connect from the PC to the micro, you need to connect ground. You can use the +5V from the PC to power your microprocessor.

The config statement will setup an ISR that is triggered when the INT pin goes low. This routine you can find in the library.
The ISR will retrieve a byte from the PC and will send the proper commands back to the PC.

The SENDSCAN and PS2MOUSEXY statements allow you to send mouse commands.

Note that the mouse emulator is only recognized after you have booted your PC. Mouse devices can not be plugged into your PC once it has booted. Inserting a mouse or mouse device when the PC is already booted, may damage your PC.

See also
SENDSCAN 1441, PS2MOUSEXY 1399

Example
```
'-------------------------------------------------------------------------
-----------------
'name                          : ps2_emul.bas
'copyright                     : (c) 1995-2025, MCS Electronics
'purpose                       : PS2 Mouse emulator
```

```
'micro                    : 90S2313
'suited for demo          : NO, commercial addon needed
'commercial addon needed  : yes
'-------------------------------------------------------------------------
-----------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

$lib "mcsbyteint.lbx"                                       ' use
optional lib since we use only bytes

'configure PS2 pins
Config Ps2emu = Int1 , Data = Pind.3 , Clock = Pinb.0
'                 ^------------------------ used interrupt
'                              ^------------ pin connected to DATA
'                                        ^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin


Waitms 500                                                  ' optional
delay

Enable Interrupts                                           ' you need
to turn on interrupts yourself since an INT is used

Print "Press u,d,l,r,b, or t"
Dim Key As Byte
Do
    Key = Waitkey()                                         ' get key
from terminal
    Select Case Key
      Case "u" : Ps2mousexy 0 , 10 , 0                      ' up
      Case "d" : Ps2mousexy 0 , -10 , 0                     ' down
      Case "l" : Ps2mousexy -10 , 0 , 0                     ' left
      Case "r" : Ps2mousexy 10 , 0 , 0                      ' right
      Case "b" : Ps2mousexy 0 , 0 , 1                       ' left
button  pressed
                 Ps2mousexy 0 , 0 , 0                       ' left
button released
      Case "t" : Sendscan Mouseup                           ' send a
scan code
      Case Else
    End Select
Loop


Mouseup:
Data 3 , &H08 , &H00 , &H01                                 ' mouse up
by 1 unit
```

7.21.65 CONFIG RAINBOW

Action

This configuration command sets up the number of rainbow channels and their ports & pins.

Syntax

CONFIG RAINBOW=channels, [,RGB=rgb] , RBx_LEN=leds, RBx_PORT=port, RBx_PIN=pin

Remarks

Channels	The number of channels. This is a numeric value in the range from 1-16. Each channel drives a port pin.
RGB	An optional parameter that has to be defined second when used. The WS2812 leds are GRB leds. (green, red, blue). 24 bits of data are sent. RGBW leds have an additional white led and are mapped RGBW. 32 bits of data are sent. The possible options are : 3 - The default. Leds like WS2811/WS2812 with GRB order. 4 - RGBW leds like SK6812RGBW. Notice that 1 more byte internal memory is needed for each led. This option will use RAINBOWBSCN.lib
RB**x**_LEN	The number of LED's for the channel. The minimum number of leds is 1. Each LED is made of 3 colors : R(ed), G(reen), and B(lue). A byte array named RAINBOW0_ will be created with a size of len * 3. Thus RB0_LEN=8 will create an array of RAINBOW0_(24). For RGBW LEDS, the array will have a length of len * 4 to store the additional white color.
RB**x**_PORT	The name of the PORT which is connected to the DI of the rainbow led(stripe). This is a port like PORTB.
RB**x**_PIN	The pin number of the port pin which is connected to the DI of the rainbow led(stripe). This is a number between 0-7.

* The **x** should be replaced by a numeric value from 0-7.

Rainbow leds come in different forms and shapes. There are single LED, stripes with 8 leds, round circles with 24 leds, etc. All have a built in WS2812 RGB controller. The nice thing is that you can cascade leds by connecting the DO (output) to another DI (input). These stripes only requires 5V, GND and DI. You can connect different stripes to different port pins.

The original rainbow library is written by Galahat from the German bascom-forum. It is an excellent example on how to write your own libraries.
The MCS version is for the BASCOM integrated statements and functions. It is named rainbowBSC.lib. The lib uses a few routines from mcs.lib

 A minimum CPU-speed of 8 MHz is required. Tests with WS1812b- types showed, it also works with frequencies down to 6.5 MHz because of the tolerance bandwidth by the chips.

Each LED requires 3 or 4 bytes of memory to store the color. Internally, the color info is stored in RGB order. And for RGBW LEDS in RGBW color.
In version 2081 the library was updated to support RGB**W** LEDS. Some functions in the old lib manipulated the wrong colors. We corrected this in the new library. But to ensure compatibility, we also include the old library.
When you use RGB=4 you will use the new library automatically. Without this option, or when using a value of 3 : RGB=3 , you will use the old library.
In order to use the new library with option 3, you need to include the library in your code using the $LIB directive : $lib "RAINBOWBSCN.lib"
This must be done BEFORE the CONFIG RAINBOW statement.

When using a normal AVR processor the used port must have a low IO address. Most ports have such an address. But processors like the Mega2560 also have some ports with an extended address. PORTH, PORTJ, PORTK and PORTL for example will not work.

See also

RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471, RB_CLEARCOLORS 1472, RB_FILL 1472, RB_FILLCOLORS 1473, RB_FILLSTRIPE 1475, RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463, RB_SWAPCOLOR 1475, RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479, RB_CHANGEPIN 1465, RB_SETTABLECOLOR 1479, RB_GETCOLOR 1483, RB_LOOKUPCOLOR 1484

Example

```
'-----------------------------------------------------------------
---------------
'                     rainbow_ws2812_Knightrider.bas
'                     based on sample from Galahat
'-----------------------------------------------------------------
---------------
$Regfile = "m88pdef.dat"
$Crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 32

Config RAINBOW=1, RB0_LEN=8, RB0_PORT=PORTB,rb0_pin=0
'                                                  ^ connected
to pin 0
'                                       ^------------ connected
to portB
'                           ^-------------------------- 8 leds on
stripe
'               ^----------------------------------- 1 channel
```

```
'Global Color-variables
Dim Color(3) as Byte
R alias Color(_base) : G alias Color(_base + 1) : B alias Color(
_base + 2)

'CONST
const numLeds=8

'----[MAIN]----------------------------------------------------
---------------
Dim n as Byte

RB_SelectChannel 0          ' select first channel
R = 50 : G = 0 : B = 100    ' define a color
RB_SetColor 0 , color(1)    ' update leds
RB_Send

Do
   For n = 1 to Numleds-1
      rb_Shiftright 0 , Numleds  'shift to the right all leds
except the last one
      Waitms 100
      RB_Send
   Next
   For n = 1 to Numleds-1
      rb_Shiftleft 0 , Numleds   'shift to the left all leds
except the last one
      Waitms 100
      RB_Send
   Next
   waitms 500                    'wait a bit
Loop
```

EXAMPLE RGBW

```
'-------------------------------------------------------------------
---------
'                   rainbow_ws2812_KnightriderDual-RGBW.bas
'                      based on sample from Galahat
'-------------------------------------------------------------------
---------
$Regfile = "m88pdef.dat"
$Crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 32

Config RAINBOW = 1 , rgb = 4 , RB0_LEN = 8 , RB0_PORT = PORTB ,
rb0_pin = 0
```

```
'                       ^-- using rgbW leds #### MUST BE FIRST PARAMETER
when defined ###
'                                                  ^ connected to pin
0
'                                        ^------------ connected to
portB
'                           ^-------------------------- 8 leds on stripe
'               ^------------------------------------- 1 channel


'Global Color-variables
Dim Color(4) as Byte
R alias Color(_base) : G alias Color(_base + 1) : B alias Color(_base +
2) : W alias color(_base + 3)

'CONST
const numLeds = 8

'----[MAIN]-----------------------------------------------------------
---------
Dim n as Byte

RB_SelectChannel 0                                                  ' select
first channel
R = 50 : G = 0 : B = 100 : w = 10                                   ' define a
color
RB_SetColor 0 , color(_base)                                        ' update
led on the left
RB_SetColor numleds - 1 , color(_base)                              ' update
led on the right
RB_Send
waitms 2000

Do
   For n = 1 to Numleds / 2 - 1
      rb_Shiftright 0 , Numleds / 2                                 'shift to
the right
      rb_Shiftleft Numleds / 2 , Numleds / 2                        'shift to
the left all leds except the last one
      Waitms 1000
      RB_Send
   Next
   For n = 1 to Numleds/2 - 1
      rb_Shiftleft 0 , Numleds / 2                                  'shift to
the left all leds except the last one
      rb_Shiftright Numleds / 2 , Numleds / 2                       'shift to
the right
      Waitms 1000
      RB_Send
   Next
   waitms 500                     'wait a bit
Loop
```

7.21.66 CONFIG RC5

Action
Overrides the RC5 pin assignment from the Option Compiler Settings 147.

Syntax
CONFIG RC5 = pin [,TIMER=2] [,WAIT=value] [,MODE=BACKGROUND]

Syntax XTINY
CONFIG RC5 = pin [,TIMER=TCAx|TCBx] [,WAIT=value] [,MODE=BACKGROUND]

Remarks

Pin	The port pin to which the RC5 receiver is connected.
TIMER	Must be 2. The micro must have a timer2 when you want to use this option. This additional parameter will cause that TIMER2 will be used instead of the default TIMER0. XTINY : for the normal mode the timer can be TCAx or TCBx. For example TCA0. XTINY : for the background mode the timer can be only a TCBx timer like TCB0.
WAIT	The default value is 100. Each unit is ca. 64 us. This gives a time out of 6.4 ms. Since a start bit is 3.5 ms, you can reduce the value to 56. When you make it lower, it will not work. When you want the old behavior you need to specify a value of 2000 which is ca. 131 ms. The WAIT parameter only has effect on the normal mode. It will not work with the BACKGROUND mode. When no valid RC5 start bit is detected, both command and address will be set to 255.
MODE	The only possible value is BACKGROUND. The MODE parameter is optional. When used, an alternative library will be used to decode the RC5 signals on the background. This means that GETRC5 will not wait for a signal but that a bit will be set to indicate that a valid RC5 signal is received. This is bit : _rc5_bits.4 The variable _rc5_bits is automatically created when you use the MODE=BACKGROUND. This option is not available in the DEMO. The background mode will use a 16 bit timer in capture mode. It also means that you need to connect the IR-transmitter output pin to the ICP capture pin of the timer. When using the background mode, you must specify a 16 bit timer. When you include a constant in your code like : CONST=_RC5_TOGGLE=1 , you will get the toggle bit in the address byte.5. Without this constant you will not get this bit.

	The prescaler value is calculated depending on the used crystal. Some desirable prescale values do not exist is some processors. Such as the 16 divider. In such a case you can override the automatic calculated value by specifying : PRESCALER=64. Typical you would try this when you get a compile error about a missing prescaler value. It is important that the PRESCALER precedes the MODE.
	
	Example : Config Rc5 = Pind.6 , Timer = 1 , PRESCALER=64, Mode = Background
	
	XTINY : For the Xtiny platform a TCBx timer is used. But the advantage of Xtiny is that you can use any processor pin. The Xtiny platform does require an additional configuration : the event system must be configured in a way that the the pin used for RC5 detection is routed to the timer TCB capture event. We can do that like this :
	`'setup RC5 and specify pin to use, and the timer which should be a timer type TCB !!!`
	`Config Rc5 = Pind.1 , Mode = Background , Timer = Tcb1`
	`Config Event_system = Dummy , Ch2 = Pd1 , Evsys_usertcb1capt = Ch2`
	
	The PD1 (PIND.1) pin is the event generator since it will receive RC5 pulses. The timer TCB1 in the above sample, is the event user : it will get the generated event. We use the timer capture unit. In this example channel 2 is used to connect the event and the user. Notice that each channel can access a number of events. Other pins might requires a different channel!
	
	It is important that PIND.1 of config-rc5 matches the PD1 of config event_system.
	And ch2 of PD1 must match the channel of the evsys_userTCBxcapt register.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the RC5 pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. In BASCOM-AVR the settings are also stored in the project. CFG file. We recommend to use the CONFIG commands.

See also

GETRC5 1278 , RC5SEND 1405

Example

```
'-----------------------------------------------------------------------
'                          RC5.BAS
'                 (c) 1995-2025 MCS Electronics
'          based on Atmel AVR410 application note
```

```
'-----------------------------------------------------------------
$RegFile = "m88def.dat"

$Baud = 19200
$Crystal = 16000000

'This example shows how to decode RC5 remote control signals
'with a SFH506-35 IR receiver.

'Connect to input to PIND.2 for this example
'The GETRC5 function uses TIMER0 and the TIMER0 interrupt.
'The TIMER0 settings are restored however so only the interrupt can not
'be used anymore for other tasks


'tell the compiler which pin we want to use for the receiver input

Config Rc5 = PIND.2 , Wait = 2000
Config Timer1 = Timer , Prescale = 1

'the interrupt routine is inserted automatic but we need to make it
occur
'so enable the interrupts
Enable Interrupts

'reserve space for variables
Dim Address As Byte , Command As Byte
Print "Waiting for RC5..."

Do
  'now check if a key on the remote is pressed
  'Note that at startup all pins are set for INPUT
  'so we dont set the direction here
  'If the pins is used for other input just unremark the next line
  'Config Pind.2 = Input
  'Print Timer1       disable this line to see the different with the
various WAIT constants
  GetRC5(Address , Command)

  'we check for the TV address and that is 0
  If Address = 0 Then
    'clear the toggle bit
    'the toggle bit toggles on each new received command
    'toggle bit is bit 7. Extended RC5 bit is in bit 6
    Command = Command And &B01111111
    Print Address ; "  " ; Command
  End If
Loop
End
```

Example MODE=background

```
'-----------------------------------------------------------------------
-----------------------------------
'                                    (c) 1995-2025
'                                 RC5-background.bas
' this sample receives RC5 on the background. it will not block your
code like getrc5
' it requires a 16 bit timer with input capture. you can not use the
timer yourself.
' some processors have multiple 16 bit timers.
'-----------------------------------------------------------------------
-----------------------------------
```

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
$hwstack = 64
$swstack = 64
$framesize = 64

Config Rc5 = Pinb.0 , Timer = 1 , Mode = Background
'                                                ^--- background interrupt
mode
'                       ^--- this must be a 16 bit timer
'              ^---- this is the timer input capture pin

Enable Interrupts                                                ' you must
enable interrupts since input capture and overflow are used


Print "RC5 demo"

Do
  If _rc5_bits.4 = 1 Then                                        ' if there
is RC5 code received
     _rc5_bits.4 = 0                                             ' you MUST
reset this flag in order to receive a new rc5 command

    Print "Address: " ; Rc5_address                              ' Address
    Print "Command: " ; Rc5_command                              ' Command
  End If
Loop
```

Xtiny Background Example

```
'-----------------------------------------------------------------------
----------------
'name                     : avrx128da28-rc5-background.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates GETRC5 in background
mode using timer TCB1
'micro                    : avr128da28
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------------
----------------

$regfile = "AVRX128da28.dat"

$crystal = 24000000
$hwstack = 40
$swstack = 40
$framesize = 64

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24MHZ
```

```
'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'set up the COM por/USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'setup RC5 and specify pin to use, and the timer which should be
a timer type TCB !!!
Config Rc5 = Pind.1 , Mode = Background , Timer = Tcb1
Config Event_system = Dummy , Ch2 = Pd1 , Evsys_usertcb1capt =
Ch2
'it is very important that you also configure the event system
'you need to chose a channel that has access to the PIN used for
the RC5 input in this case PIND.1
'this pin will create an event when it changes. And you need to
connect the TCB CAPTURE user to this channel
'In this example we use TCB1 thus EVSYS_USER will be
evsys_userTCB1CAPT, and since PD1 (pin D.1) is connected to
'channel 2, we also need to select channel 2.
'Now there is a path from PIN2.1 to TCB0, capture event
'The compiler could have created this link too but then it is not
clear to the user that this event channel is used
'for this reason you need to configure it manual

print "RC5 test"

Dim B As Byte
Enable Interrupts
'since interrupts are used we must enable the global interrupt
switch


do
  If _rc5_bits.4 = 1 Then                                   'this
variable is automatically created
     _rc5_bits.4 = 0

     Print "Address: " ; Rc5_address                        'auto
created variable
     Print "Command: " ; Rc5_command                        'auto
created variable
  End If
loop

End
```

7.21.67 CONFIG RC5SEND

Action
Defines the RC5SEND timer and WaveOutput pin.

Syntax
CONFIG RC5SEND = timer, WO=wo

Remarks

TIMER	The TIMER must be TCA0 or when available TCA1 or any oher TCA timer.
WO	The Wave Output pin (WO). This is WO0, WO1 or WO2.

RC5SEND uses a TCA timer in frequency generating mode in order to create a 36 KHz carrier wave.
You can chose any available WO pin. You can also use the PORTMUX to use a different port.
You must set the corresponding pin to OUTPUT mode using : CONFIG pin statement. The pin must also be set to 1.

While the compiler could do all this it would need to deal with the portmux. The portmux is a great piece of hardware that allows you to chose alternative pin locations.
When the compiler performs this automatic it would not be visible to the user. So we have chosen that we leave this to the user.

See also
GETRC5 1278 , RC5SEND 1405

Example
```
'-----------------------------------------------------------------
----------------
'name                        :
avrx128da28-rc5-background-send-receive.bas
'copyright                   : (c) 1995-2025, MCS Electronics
'purpose                     : demonstrates GETRC5 in background
mode using timer TCB1
'                              and RC5 transmission using TCA0
'micro                       : avr128da28
'suited for demo             : no
'commercial addon needed     : yes
'-----------------------------------------------------------------
----------------

$regfile = "AVRX128da28.dat"

$crystal = 24000000
$hwstack = 40
```

```
$swstack = 40
$framesize = 64

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz

'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'set up the COM por/USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'setup RC5 receive and specify pin to use, and the timer which
should be a timer type TCB !!!
Config Rc5 = Pind.1 , Mode = Background , Timer = Tcb1
Config Event_system = Dummy , Ch2 = Pd1 , Evsys_usertcb1capt =
Ch2
'it is very important that you also configure the event system
'you need to chose a channel that has access to the PIN used for
the RC5 input in this case PIND.1
'this pin will create an event when it changes. And you need to
connect the TCB CAPTURE user to this channel
'In this example we use TCB1 thus EVSYS_USER will be
evsys_userTCB1CAPT, and since PD1 (pin D.1) is connected to
'channel 2, we also need to select channel 2.
'Now there is a path from PIN2.1 to TCB0, capture event
'The compiler could have created this link too but then it is not
clear to the user that this event channel is used
'for this reason you need to configure it manual

'for the RC5 transmission we need a TCA0 WOx pin. This pin is
used in output mode.
'we will use WO2 which is connected to PA2. you could use config
port_mux to chose an altenative pin
'the IR diode anode is connected to the power(vcc) and the
cathode is connected to a 220 ohm resistor
' the other end of the resistor is connected to the WO2 pin,
porta.2 in this case
Config Pina.2 = Output : Porta.2 = 1                          'set
the port direction and also set the pin high

'we need this new command to select the timer (tca0 or when
available tca1) and the WO pin
'the timer is operated in frequency generation mode, 36 KHz for
RC5
Config Rc5send = Tca0 , Wo = Wo2

Dim Bcmd As Byte
```

```
Enable Interrupts
'since interrupts are used we must enable the global interrupt
switch

Print "RC5 test,REV:" ; Hex(syscfg_revid)

Do
  Incr Bcmd
  Rc5send 0 , 0 , Bcmd                                    'send
RC5 code
  Waitms 500                                              'wait
500 msec

  'this is the background mode part that receives from the IR led
or a remote control
  If _rc5_bits.4 = 1 Then                                 'this
variable is automatically created
     _rc5_bits.4 = 0
     Print "Address: " ; Rc5_address                      'auto
created variable
     Print "Command: " ; Rc5_command                      'auto
created variable
  End If
loop

End
```

7.21.68 CONFIG RND

Action

This option will set the randomize configuration.

Syntax

CONFIG RND = 16|32

Remarks

By default rnd() is created using 16 bit multiplying and division. This limits the maximum number to a word. The ___Rseed variable is a word.

When you need to have a bigger random number you can use the CONFIG RND = 32 option.
When using 32 bit resolution, only division is used to limit the number with the specified number.
Using 32 bit the ___Rseed will be a DWORD and not a WORD.

See also
RND 1431

Example
```
' Plot
' FT800 platform.
' Original code from http://gameduino2.proboards.com/thread/11/screen-plotting

' Comments by James Bowman:
' Sets up the whole screen as a framebuffer, in PALETTED mode, which should be good for the fractals.
' setpal() sets palette entry 'i' to a 32-bit ARGB color, and plot(x, y, i) sets a single pixel to index 'i'.

' Requires Bascom 2.0.7.8 or greater

$Regfile = "M328pdef.dat"
$Crystal = 8000000
$Baud = 19200
$HwStack = 80
$SwStack = 80
$FrameSize = 300
$NOTYPECHECK

Config ft800=spi , ftsave=0, ftdebug=0 , ftcs=portb.2, ftpd=portb.1

Config Base = 0
Config Submode = New
Config Spi = Hard, Interrupt = Off, Data_Order = Msb, Master = Yes, Polarity = Low, Phase = 0, Clockrate = 4, Noss = 1
SPSR = 1  ' Makes SPI run at 8Mhz instead of 4Mhz


Config RND = 32

$Include "FT800.inc"
$Include "FT800_Functions.inc"

Declare Sub setup
Declare Sub setpal (Byval i As Byte, Byval argb As Long)
Declare Sub plot (Byval x As Integer, Byval y As Integer, Byval i As Long)

dim dw as Dword
dim d1 as Dword
dim d2 as Dword

Spiinit


If FT800_Init() = 1 Then
   print "END"
   END    ' Initialise the FT800
end if

Setup

Do
   d1 = rnd(Ft_DispWidth-1)
   d2 = rnd(Ft_DispHeight-1)
   plot d1, d2, rnd(255)
Loop



END

'-----------------------------------------------------------------------------------------
Sub Setup
'-----------------------------------------------------------------------------------------

   Local i As Byte

   CmdMemset 0, 0, Ft_DispWidth * Ft_DispHeight
   ClearScreen
   BitmapLayout PALETTED, Ft_DispWidth , Ft_DispHeight
   BitmapSize NEAREST, BORDER, BORDER, Ft_DispWidth, Ft_DispHeight
```

```
  BitmapSource 0
  Begin_G BITMAPS
  Vertex2ii 0, 0, 0, 0

  UpdateScreen

  setpal 0, &H00000000

  For i = 1 to 255
    setpal i, rnd(16777216) or &Hff000000
  Next
End Sub ' Setup

'-----------------------------------------------------------------------
Sub SetPal (Byval i As Byte, Byval argb As Long)
'-----------------------------------------------------------------------

  Local Temp1 As Long

  Temp1 =  i * 4
  Temp1 = Temp1 + Ram_Pal
  Wr32  Temp1 , argb

End Sub ' SetPal

'-----------------------------------------------------------------------
Sub Plot(Byval x As Integer, Byval y As Integer, Byval i As Long)
'-----------------------------------------------------------------------

  Local Temp1 As Long

  If x < Ft_DispWidth AND y < Ft_DispHeight Then

    Temp1 = Ft_DispWidth * y
    Temp1 = Temp1 + x
    Wr8 Temp1, i

  End If

End Sub ' Plot
```

7.21.69 CONFIG SDA

Action

Overrides the SDA pin assignment from the Option Compiler Settings 147.

Syntax

CONFIG SDA = pin

Remarks

Pin	The port pin to which the I2C-SDA line is connected.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the SDA pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. In BASCOM-AVR the settings are also stored in the project. CFG file.

When using the Hardware TWI, you only need CONFIG SDA when you use the I2CINIT statement

See also

CONFIG SCL 1049 , CONFIG I2CDELAY 975 , I2CINIT 1300, Using the I2C protocol 297

Example 1

```
CONFIG SDA = PORTB.7  'PORTB.7 is the SDA line
```

Example 2

```
'-------------------------------------------------------------------------
------------------
'name                     : i2c.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: I2CSEND and I2CRECEIVE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'We use here the Software I2C Routines
Config Scl = Portb.4
Config Sda = Portb.5
I2cinit

Config I2cdelay = 10                                       '100KHz

Declare Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
Declare Sub Read_eeprom(byval Adres As Byte , Value As Byte)

Const Addressw = 174                                       'slave write
address
Const Addressr = 175                                       'slave read
address

Dim B1 As Byte , Adres As Byte , Value As Byte             'dim byte

Call Write_eeprom(1 , 3)                                   'write value
of three to address 1 of EEPROM
Call Read_eeprom(1 , Value) : Print Value                  'read it
back
Call Read_eeprom(5 , Value) : Print Value                  'again for
address 5


'-------- now write to a PCF8474 I/O expander -------
I2csend &H40 , 255                                         'all outputs
high
I2creceive &H40 , B1                                       'retrieve
input
Print "Received data " ; B1                                'print it
End
```

```
Rem Note That The Slaveaddress Is Adjusted Automaticly With I2csend &
I2creceive
Rem This Means You Can Specify The Baseaddress Of The Chip.

'sample of writing a byte to EEPROM AT2404
Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
    I2cstart                                                'start
condition
    I2cwbyte Addressw                                       'slave
address
    I2cwbyte Adres                                          'asdress of
EEPROM
    I2cwbyte Value                                          'value to
write
    I2cstop                                                 'stop
condition
    Waitms 10                                               'wait for 10
milliseconds
End Sub


'sample of reading a byte from EEPROM AT2404
Sub Read_eeprom(byval Adres As Byte , Value As Byte)
   I2cstart                                                 'generate
start
   I2cwbyte Addressw                                        'slave
adsress
   I2cwbyte Adres                                           'address of
EEPROM
   I2cstart                                                 'repeated
start
   I2cwbyte Addressr                                        'slave
address (read)
   I2crbyte Value , Nack                                    'read byte
   I2cstop                                                  'generate
stop
End Sub


' when you want to control a chip with a larger memory like the 24c64 it
requires an additional byte
' to be sent (consult the datasheet):
' Wires from the I2C address that are not connected will default to 0 in
most cases!

'   I2cstart                                                'start
condition
'   I2cwbyte &B1010_0000                                    'slave
address
'   I2cwbyte H                                              'high
address
'   I2cwbyte L                                              'low address
'   I2cwbyte Value                                          'value to
write
'   I2cstop                                                 'stop
condition
'   Waitms 10
```

7.21.70 CONFIG SCL

Action
Overrides the SCL pin assignment from the Option Compiler Settings 147.

Syntax
CONFIG SCL = pin

Remarks

Pin	The port pin to which the I2C-SCL line is connected.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the SCL pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. Of course BASCOM-AVR also stores the settings in a project.CFG file.

When using the Hardware TWI, you only need CONFIG SCL when you use the I2CINIT statement

See also
CONFIG SDA 1046 , CONFIG I2CDELAY 975 , I2CINIT 1300, Using the I2C protocol 297

Example 1
```
CONFIG SCL = PORTB.5  'PORTB.5 is the SCL line
```

Example 2
```
'-----------------------------------------------------------------------
------------------
'name                     : i2c.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: I2CSEND and I2CRECEIVE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'We use here the Software I2C Routines
Config Scl = Portb.4
Config Sda = Portb.5
```

```
I2cinit

Config I2cdelay = 10                                        '100KHz

Declare Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
Declare Sub Read_eeprom(byval Adres As Byte , Value As Byte)

Const Addressw = 174                                        'slave write
address
Const Addressr = 175                                        'slave read
address

Dim B1 As Byte , Adres As Byte , Value As Byte              'dim byte

Call Write_eeprom(1 , 3)                                    'write value
of three to address 1 of EEPROM
Call Read_eeprom(1 , Value) : Print Value                   'read it
back
Call Read_eeprom(5 , Value) : Print Value                   'again for
address 5

'-------- now write to a PCF8474 I/O expander -------
I2csend &H40 , 255                                          'all outputs
high
I2creceive &H40 , B1                                        'retrieve
input
Print "Received data " ; B1                                 'print it
End

Rem Note That The Slaveaddress Is Adjusted Automaticly With I2csend &
I2creceive
Rem This Means You Can Specify The Baseaddress Of The Chip.



'sample of writing a byte to EEPROM AT2404
Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
    I2cstart                                                'start
condition
    I2cwbyte Addressw                                       'slave
address
    I2cwbyte Adres                                          'asdress of
EEPROM
    I2cwbyte Value                                          'value to
write
    I2cstop                                                 'stop
condition
    Waitms 10                                               'wait for 10
milliseconds
End Sub


'sample of reading a byte from EEPROM AT2404
Sub Read_eeprom(byval Adres As Byte , Value As Byte)
   I2cstart                                                 'generate
start
   I2cwbyte Addressw                                        'slave
adsress
   I2cwbyte Adres                                           'address of
EEPROM
   I2cstart                                                 'repeated
start
   I2cwbyte Addressr                                        'slave
address (read)
```

```
   I2crbyte Value , Nack                                    'read byte
   I2cstop                                                  'generate
stop
End Sub

' when you want to control a chip with a larger memory like the 24c64 it
requires an additional byte
' to be sent (consult the datasheet):
' Wires from the I2C address that are not connected will default to 0 in
most cases!

'   I2cstart                                                 'start
condition
'   I2cwbyte &B1010_0000                                     'slave
address
'   I2cwbyte H                                               'high
address
'   I2cwbyte L                                               'low address
'   I2cwbyte Value                                           'value to
write
'   I2cstop                                                  'stop
condition
'   Waitms 10
```

7.21.71 CONFIG SERIALIN

Action

Configures the hardware UART to use a buffer for input

Syntax

CONFIG SERIALIN | SERIALIN1 | SERIALIN2 | SERIALIN3 |SERIALx = BUFFERED , SIZE | BIGSIZE = size [, BYTEMATCH=ALL|BYTE|NONE] [,CTS=pin, RTS=pin , Threshold_full=num , Threshold_empty=num]

Remarks

SerialIn	Some chips have multiple HW UARTS. Use the following parameter values: • SERIALIN or SERIALIN0 : first UART/UART0 • SERIALIN1 : second UART/UART1 • SERIALIN2 : third UART/UART2 • SERIALIN3 : fourth UART/UART3 • SERIALIN4 : fifth UART/UART4 • SERIALIN5 : sixth UART/UART5 • SERIALIN6 : seventh UART/UART6 • SERIALIN7 : eight UART/UART7
Size	A numeric constant that specifies how large the input buffer should be. The space is taken from the SRAM. The maximum is 255.
BigSize	Instead of using Size you can use BigSize for COM1. It allows to create a bigger buffer than using Size. While Size works with a byte buffer, BigSize works with a word size buffer. It requires more code and does not handle CTS/RTS. For some applications it can be useful to have a big buffer. This is an overloaded version of the code from mcs.lib. You need to include bigbuf.lib using the $LIB directive in your code.
Bytematch	The ASCII value of the byte that will result in calling a user label. When you specify **ALL**, the user label will be called for every byte

	that is received. You must include the label yourself in your code and end it with a return. The following label names must be used when you check for a specific byte value: • Serial**0**CharMatch (for SERIALIN or the first UART/UART**0**) • Serial**1**CharMatch (for SERIALIN**1** or the second UART/UART**1**) • Serial**2**CharMatch (for SERIALIN**2** or the third UART/UART**2**) • Serial**3**CharMatch (for SERIALIN**3** or the fourth UART/UART**3**) The following label names must be used when you check for any value: • Serial**0**ByteReceived (for SERIALIN or the first UART/UART**0**) • Serial**1**ByteReceived (for SERIALIN**1** or the second UART/UART**1**) • Serial**2**ByteReceived (for SERIALIN**2** or the third UART/UART**2**) • Serial**3**ByteReceived (for SERIALIN**3** or the fourth UART/UART**3**) When you specify **NONE**, it is the same as not specifying this optional parameter.
CTS	The pin used for the CTS.(Clear to send). For example PIND.6. This pin will be used in the INPUT mode since it will be connected to the other parties RTS pin.
RTS	The pin used for RTS. (Ready to send). For example PIND.7 This pin will be used in OUTPUT mode. It is set to 0 to indicate that the other party may send data and it will become 1 to signal to the other party that the buffer is almost full.
Threshold_full	The number of bytes that will cause RTS to be set to '1'. This is an indication to the sender, that the buffer is full. If your buffer is 100 bytes, you could set it to 80 so after receiving 80 bytes, the RTS pin will change and there are still 20 bytes in the buffer to compensate timing at high baud rates.
Threshold_empty	The number of free bytes that must be in the buffer before RTS is enabled (made '0') again. If the buffer is 100 bytes, you could set it to 10.

 The following description is for the normal buffer which uses bytes.
When using the BIGSIZE option instead of SIZE these variables will be of the word type. The mechanism is exactly the same.

The following internal variables will be generated for UART**0**:
_RS_HEAD_PTR**0** , a byte counter that stores the head of the buffer
_RS_TAIL_PTR**0** , a byte counter that stores the tail of the buffer.
_RS232INBUF**0** , an array of bytes that serves as a ring buffer for the received characters.
_RS_BUFCOUNTR**0**, a byte that holds the number of bytes that are in the buffer.

For the other UARTS, the variables are named similar. But they do have a different number.
A **1** for the second UART, a **3** for the third UART and a **4** for the fourth UART. Yes, the '**2**' is skipped.

While you can read and write the internal variables, we advise not to write to them. The variables are updated inside interrupts routines, and just when you write a value to them, an ISR can overwrite the value.

The optional **BYTEMATCH** can be used to monitor the incoming data bytes and call a label when the specified data is found. This label is a fixed label as mentioned in the table above. The label is called after the data is stored in the buffer.
This way you can determine the start of a serial stream when you work with a unique header byte. Or you can determine when the data is received into the buffer when you work with a unique trailer byte.

While bytematch allows you to trap the incoming data bytes, take care that you do not delay the program execution too much. After all the serial input interrupt is used in order not to miss incoming data. When you add delays or code that will delay execution too much you might loose incoming data.

 When using the BYTEMATCH option, you must preserve the registers you alter. If you do not know which one, use PUSHALL 1402 and POPALL 1397.

 When using BYTEMATCH and CTS/RTS, do not print data in the bytematch routine to the same UART. This can disturb the communication when the output buffer becomes full.

 To clear the buffer, use CLEAR 846 SERIALIN. Do not read and write the internal buffer variables yourself.

CTS-RTS is hardware flow control. Both the sender and receiver need to use CTS-RTS when CTS-RTS is used. When one of the parties does not use CTS-RTS, no communication will be possible.
CTS-RTS requires two additional wires. The receiver must check the CTS pin to see if it may send. The CTS pin is an input pin as the receiver looks at the level that the sender can change.

The receiver can set the RTS pin to indicate to the sender that it can accept data.
In the start condition, RTS is made '0' by the receiver. The sender will then check this logic level with it's CTS pin, and will start to send data. The receiver will store the data into the buffer and when the buffer is almost full, or better said, when the Threshold_full is the same as the number of bytes in the receive buffer, the receiver will make RTS '1' to signal to the sender, that the buffer is full. The sender will stop sending data. And will continue when the RTS is made '0' again.

The receiver can send data to the sender and it will check the CTS pin to see if it may send data.

In order to work with CTS-RTS, you need **both** a serial input buffer, and a serial output buffer. So use both CONFIG SERIALIN and CONFIG SERIALOUT to specify the buffers.
The CTS-RTS can only be configured with the CONFIG SERIALIN statement.

The thresholds are needed for high baud rates where it will take some time to react on a CTS-RTS.
You need to experiment with the thresholds but good start values are 80% full, and 20% empty.

 You need to use a pin that is bit addressable. For most chips this is a pin from port A, B, C or D.

 Some serial devices use the RTS pin as an output pin, while other devices use RTS pin as an input pin to indicate that it need to be connected TO an RTS pin. You

always need to have a good look at the data sheet and see in which mode the RTS/CTS pins are used.
In BASCOM RTS is an output pin and CTS is an input pin.

Additional Infos for XMEGA Devices:

Since buffered serial input and output uses interrupts, you must enable the global interrupts in your code with : ENABLE INTERRUPTS.
For the XMEGA, if you set the priority with CONFIG PRIORITY, you must enable the MED priority.
If you only use ENABLE INTERRUPTS, the MED priority is enabled automatically. This means you only need to specify MED when you manually configure the priority.

Buffer full

So what happens when the buffer is full and a new character arrives and cts/rts are not used?
The byte is still read out and the ERR variable is set. But the data is NOT stored in the buffer. It is lost just as when you would have not used any buffering.
When BYTEMATCH is used, this will still be used/called.

ASM

Routines called from MCS.LIB :

_GotChar. This is an ISR that gets called when ever a character is received.
When there is no room for the data it will not be stored.
So the buffer must be emptied periodic by reading from the serial port using the normal statements like INKEY() and INPUT.

Since URXC interrupt is used by _GotChar, you can not use this interrupt anymore.
Unless you modify the _gotchar routine of course.

See also

CONFIG SERIALOUT 1057 , ISCHARWAITING 1498 , CLEAR 846

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : rs232buffer.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : example shows the difference between normal
and buffered
'                           serial INPUT
'micro                    : Mega161
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m161def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 9600                                                ' use baud
rate
$hwstack = 32                                               ' default
```

```
use 32 for the hardware stack
$swstack = 10                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space

'first compile and run this program with the line below remarked
Config Serialin = Buffered , Size = 20


Dim Na As String * 10

'the enabling of interrupts is not needed for the normal serial mode
'So the line below must be remarked to for the first test
Enable Interrupts

Print "Start"
Do
   'get a char from the UART

   If Ischarwaiting() = 1 Then                                   'was there a
char?
      Input Na
      Print Na                                                   'print it
   End If

   Wait 1                                                        'wait 1
second
Loop

'You will see that when you slowly enter characters in the terminal
emulator
'they will be received/displayed.
'When you enter them fast you will see that you loose some chars

'NOW remove the remarks from line 11 and 18
'and compile and program and run again
'This time the chars are received by an interrupt routine and are
'stored in a buffer. This way you will not loose characters providing
that
'you empty the buffer
'So when you fast type abcdefg, they will be printed after each other
with the
'1 second delay

'Using the CONFIG SERIAL=BUFFERED, SIZE = 10 for example will
'use some SRAM memory
'The following internal variables will be generated :
'_Rs_head_ptr0   BYTE , a pointer to the location of the start of the
buffer
'_Rs_tail_ptr0   BYTE , a pointer to the location of tail of the buffer
'_RS232INBUF0 BYTE ARRAY , the actual buffer with the size of SIZE
```

Example2

```
'-----------------------------------------------------------------------------
'name                        :
'copyright                     : (c) 1995-2025, MCS Electronics
'purpose                       : test for M2560 support
'micro                       : Mega2560
'suited for demo             : yes
'commercial addon needed     : no
'-----------------------------------------------------------------------------

$regfile = "m2560def.dat"                                  ' specify the used micro
$crystal = 8000000                                         ' used crystal frequency
$hwstack = 40                                              ' default use 32 for the
hardware stack
```

```
$swstack = 40                                              ' default use 10 for the SW
stack
$framesize = 40                                            ' default use 40 for the frame
space


'$timeout    =    1000000

'The M128 has an extended UART.
'when CO'NFIG COMx is not used, the default N,8,1 will be used
Config Com1 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Clockpol = 0
Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Clockpol = 0
Config Com3 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Clockpol = 0
Config Com4 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Clockpol = 0

Enable Interrupts
Config     Serialin =    Buffered , Size = 20
Config     Serialin1 =    Buffered , Size = 20 , Bytematch = 65
Config     Serialin2 =    Buffered , Size = 20 , Bytematch = 66
Config     Serialin3 =    Buffered , Size = 20 , Bytematch =   All


'Open all UARTS
Open "COM2:" For Binary As #2
Open "COM3:" For Binary As #3
Open "COM4:" For Binary As #4


Print "Hello"                                               'first       uart
Dim B1 As Byte , B2 As Byte , B3 As Byte , B4 As Byte
Dim  Tel As Word , Nm As String * 16

'unremark to test second UART
'Input #2 , "Name ?" , Nm
'Print #2 , "Hello " ; Nm


Do
  Incr Tel
  Print Tel ; "  test serial port 1"
  Print #2 , Tel ; "  test serial port 2"
  Print #3 , Tel ; "  test serial port 3"
  Print #4 , Tel ; "  test serial port 4"

  B1 = Inkey( )                                            'first       uart
  B2 = Inkey( #2)
  B3 = Inkey( #3)
  B4 = Inkey( #4)

  If B1 <> 0 Then
     Print B1 ; "  from port 1"
  End If
  If B2 <> 0 Then
     Print #2 , B2 ; "  from port 2"
  End If
  If B3 <> 0 Then
     Print #3 , B3 ; "  from port 3"
  End If
  If B4 <> 0 Then
     Print #4 , B4 ; "  from port 4"
  End If

  Waitms 500
Loop



'Label called when UART2 received an A
Serial1charmatch:
  Print #2 , "we got an A"
Return


'Label called when UART2 received a B
Serial2charmatch:
```

```
   Print #3 , "we got a B"
Return


'Label called when UART3 receives a char
Serial3bytereceived:
   Print #4 , "we got a char"
Return

End

Close #2
Close #3
Close #4

$eeprom
Data 1 , 2
```

7.21.72 CONFIG SERIALOUT

Action

Configures the hardware UART to use a buffer for output

Syntax

CONFIG SERIALOUT | SERIALOUT1 | SERIALOUT2 | SERIALOUT3 | SERIALOUTx = BUFFERED , SIZE = size

Remarks

SerialOut	Some chips have multiple HW UARTS. Use the following parameter values: • SERIALOUT or SERIALOUT0 : first UART/UART0 • SERIALOUT**1** : second UART/UART1 • SERIALOUT**2** : third UART/UART2 • SERIALOUT**3** : fourth UART/UART3 • SERIALOUT**4** : fifth UART/UART4 • SERIALOUT**5** : sixth UART/UART5 • SERIALOUT**6** : seventh UART/UART6 • SERIALOUT**7** : eight UART/UART7
size	A numeric constant that specifies how large the output buffer should be. The space is taken from the SRAM. The maximum value is 255.

The following internal variables will be used when you use CONFIG SERIALOUT

_RS_HEAD_PTRW**0** , byte that stores the head of the buffer
_RS_TAIL_PTRW**0** , byte that stores the tail of the buffer
_RS232OUTBUF**0**, array of bytes for the ring buffer that stores the printed data.
_RS_BUFCOUNTW**0**, a byte that holds the number of bytes in the buffer.

For the other UARTS, the variables are named similar. But they do have a different number.
A **1** for the second UART, a **3** for the third UART and a **4** for the fourth UART. Yes, the '**2**' is skipped.

Serial buffered output can be used when you use a low baud rate. It would take relatively much time to print all data without a buffer. When you use a buffer, the data is printed on the background when the micro UART byte buffer is empty. It will get a byte from the buffer then and transmit it.
As with any buffer you have, you must make sure that it is emptied at one moment in time.

You can not keep filling it as it will become full. When you do not empty it, you will have the same situation as without a buffer !!! When the roof is leaking and you put a bucket on the floor and in the morning you empty it, it will work. But when you will go away for a day, the bucket will overflow and the result is that the floor is still wet.

Another important consideration is data loss. When you print a long string of 100 bytes, and there is only room in the buffer for 80 bytes, there is still a wait evolved since after 80 bytes, the code will wait for the buffer to become empty. When the buffer is empty it will continue to print the data. The advantage is that you do not loose any data, the disadvantage is that it blocks program execution just like a normal un-buffered PRINT would do.

Since buffered serial output uses interrupts, you must enable the global interrupts in your code with : ENABLE INTERRUPTS.
For the XMEGA, if you set the priority with CONFIG PRIORITY, you must enable the MED priority.

ASM

Routines called from MCS.LIB :
_CHECKSENDCHAR. This is an ISR that gets called when ever the transmission buffer is empty.
Since UDRE interrupt is used , you can not use this interrupt anymore. Unless you modify the _CheckSendChar routine of course.

When you use the PRINT statement to send data to the serial port, the UDRE interrupt will be enabled. And so the _CheckSendChar routine will send the data from the buffer.

See also

CONFIG SERIALIN 1051

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : rs232bufferout.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates how to use a serial output
buffer
'micro                    : Mega128
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m128def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 9600                                                ' use baud
rate
$hwstack = 40                                               ' default
use 32 for the hardware stack
$swstack = 40                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space
```

```
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


'setup to use a serial output buffer
'and reserve 20 bytes for the buffer
Config Serialout = Buffered , Size = 20

'It is important since UDRE interrupt is used that you enable the
interrupts
Enable Interrupts
Print "Hello world"
Print "test1"
Do
Wait 1
'notice that using the UDRE interrupt will slown down execution of
waiting loops like waitms
Print "test"
Loop
End
```

7.21.73 CONFIG SINGLE

Action

Instruct the compiler to use an alternative conversion routine for representation of a single.

Syntax

CONFIG SINGLE = SCIENTIFIC , DIGITS = value

Remarks

Single	SCIENTIFIC for scientific notation. Use NORMAL for the normal default notation. Using both modes will increase your code size.
Digits	A numeric constant with a value between 0 and 7. A value of 0 will result in no trailing zero's. A value between 1-7 can be used to specify the number of digits behind the comma.

When a conversion is performed from numeric single variable, to a string, for example when you PRINT a single, or when you use the STR() function to convert a single into a string, a special conversion routine is used that will convert into human readable output. You will get an output of digits and a decimal point.
This is well suited for showing the value on an LCD display. But there is a downside also. The routine is limited in the way that it can not shown very big or very small numbers correct.

The CONFIG SINGLE will instruct the compiler to use a special version of the conversion routine. This version will use scientific notation such as : 12e3.
You can specify how many digits you want to be included after the decimal point.

It is possible to switch between notations by using multiple CONFIG SINGLE statements. As soon at the compiler encounters a CONFIG SINGLE, it will change to output to the selected format. You should not use CONFIG SINGLE inside a sub/function since this is not a dynamic feature that can be changed at run time.

See also
FUSING 830, STR 836

ASM
Uses single.lbx library

Example
```
'-----------------------------------------------------------------
'                          (c) 1995-2025, MCS
'                  single_scientific.bas
' demonstation of scientific , single output
'-----------------------------------------------------------------

$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200

'you can view the difference by compiling and simulating  this sample
with the
'line below remarked and active
Config Single = Scientific , Digits = 7

Dim S As Single
S = 1
Do
  S = S / 10
  Print S
Loop
```

7.21.74 CONFIG SHIFTIN

Action
Instruct the compiler to use new behaviour of the SHIFTIN statement.

Syntax
CONFIG SHIFTIN = value

Remarks

value	This must be COMPATIBLE or NEW. By default the old behaviour is used. So in order to use the new behaviour you must use : CONFIG SHIFTIN=NEW

The SHIFTOUT has been enhanced with a number of options which make it incompatible to the old SHIFTOUT.
In order to maintain compatibility with your old code, this option has been added so you have control over which SHIFTIN version is used.

See also
SHIFTIN 1449

7.21.75 CONFIG SPI

Action
Configures the SPI mode and pins.

Syntax for software SPI
CONFIG SPI|SPISOFT = SOFT, DIN = PIN, DOUT = PIN , SS = PIN|NONE, CLOCK = PIN , SPIIN=value , MODE=mode, SPEED=speed, SETUP=setup , EXTENDED=ext

Syntax for hardware SPI
CONFIG SPI|SPIHARD = HARD, INTERRUPT=ON|OFF, DATA_ORDER = LSB|MSB , MASTER = YES|NO , POLARITY = HIGH|LOW , PHASE = 0|1, CLOCKRATE = 4|16|64|128 , NOSS=1|0 , SPIIN=value , EXTENDED=ext

Syntax for hardware SPI1
CONFIG SPI1 = HARD, INTERRUPT=ON|OFF, DATA_ORDER = LSB|MSB , MASTER = YES|NO , POLARITY = HIGH|LOW , PHASE = 0|1, CLOCKRATE = 4|16|64|128 , NOSS=1|0 , SPIIN=value

When you just want to use one SPI slave chip using the HW SPI, use this : Config Spi = Hard , Interrupt = Off , Data_Order = Msb , Master = Yes , Polarity = Low , Phase = 0 , Clockrate = 128
When you want more details, read more about the details and options below.

Remarks software SPI

SPI	**SOFT** for software emulation of SPI, this allows you to choose the pins to use. Only works in master mode. **HARD** for the internal SPI hardware, that will use fixed pins of the microprocessor.
DIN	Data input or MISO. Pin is the pin number to use such as PINB.0
DOUT	Data output or MOSI. Pin is the pin number to use such as PORTB.1
SS	Slave Select. Pin is the pin number to use such as PORTB.2 Use NONE when you do not want the SS signal to be generated. See remarks. Or as an alternative you can use : NOSS=1.
CLOCK	Clock. Pin is the pin number to use such as PORTB.3
DATA ORDER	Selects if MSB or LSB is transferred first. For soft SPI you need to use the MODE option as well. Otherwise only MSB order is available.
MASTER	Selects if the SPI is run in master or slave mode.
SPIIN	When reading from the SPI slave, it should not matter what kind of data you send. But some chips require a value of 255 while others require a value of 0. By default, when the SPIIN option is not provided, a value of 0 will be sent to the SPI slave. With this SPIIN option you can override this value.
MODE	A constant in the range from 0-3 which defines the SPI MODE. Without MODE, the default mode 1 will be used.

	Also, when using MODE, new SPI code will be used. When using MODE, you can also specify SPEED and SETUP. MODE is for Software SPI only ! **Mode** **Leading Edge** **Trailing Edge** 0 Rising, Sample Falling, Setup 1 Rising, Setup Falling, Sample 2 Falling, Sample Rising, Setup 3 Falling, Setup Rising, Sample
SPEED	Is a numeric constant for an optional delay. This delay is in us. When you specify 1, it will result in 2 us delay : 1 us before and 1 us after the clock. By default there is no delay. Only slow slave chips might require a delay. SPEED only applies when MODE is specified.
SETUP	Setup is the delay in uS before sampling the MISO pin. A numeric constant must be used. SETUP is for Software SPI only and when MODE is used !
EXTENDED	An optional parameter to extend the maximum data read/write size. A value of 0 is default and will cause the SPIIN, SPIIOUT, SPIMOVE routines to handle a maximum data size of 255 bytes. A value of 1 will extended the data size from bytes to words which means you can move data of 65535 bytes.

Software SPI allows you to chose the processor pins for the SPI operation. Typically you need a MISO, MOSI, CLOCK and SS pin.
While this is an advantage, the disadvantage is that software SPI uses more processor resources.

In software spi mode the SPIINIT 1514 statement will set the SPI pins to the proper logic level. For example to :

```
sbi PORTB,5 ;set latch bit hi (inactive)SS
sbi DDRB,5  ;make it an output SS
cbi PORTB,4 ;set clk line lo
sbi DDRB,4  ;make it an output
cbi PORTB,6 ;set data-out lo MOSI
sbi DDRB,6  ;make it an output MOSI
cbi DDRB,7  ;MISO input
Ret
```

This is just an example. The actual code differs from processor to processor. And also depends on the used port pins.

In most cases, there is just one slave chip to control/address. In such a case you need only one slave select(SS) pin to control this chip. But SPI can also be used to control multiple SPI slaves.
These slaves need to use the same mode. You can not dynamically change the SPI mode at run time.

BASCOM will automatically set the SS pin to logic level 0 when you use a SPI command. And when the SPI command has executed, it will set the SS pin back to a logic 1.
When the slave chip has in inverted SS pin (it requires a 1 to be active) you can not use this automatic SS signal generation.
When you want to address multiple slaves with the software SPI you need multiple pins to select the different slave chips. In this case you also can not use the automatic SS signal generation.

The solution is to specify **NONE** for SS. This will eliminate the automatic SS signal generation. But it also means that you as a user need to handle this. In practice this means :
- choose a port pin to serve as SS pin
- set it to output and to the right logic level (1 in most cases to disable the slave)
- before using a SPI statement, select the slave by making SS logic 0.
- after the SPI statement, set the SS logic level back to 1.

Example user controlled SS pin.

```
Config Spi = Soft , Din = Pinb.0 , Dout = Portb.1 , Ss = NONE ,
Clock = Portb.3
MySS alias portb.2
Config MySS=OUTPUT : MySS=1 ' deactivate
Dim var As Byte
SPIINIT                     ' Init SPI state and pins.
MySS=0                      ' select SS
SPIOUT var , 1              ' send 1 byte
MySS=1                      ' deselect SS
```

Remarks Hardware SPI

SPI	**SOFT** for software emulation of SPI, this allows you to choose the pins to use. Only works in master mode. **HARD** for the internal SPI hardware, that will use fixed pins of the microprocessor.
DATA_ORDER	Selects if MSB or LSB is transferred first.
MASTER	Selects if the SPI is run in master or slave mode.
POLARITY	Select HIGH to make the CLOCK line high while the SPI is idle. LOW will make clock LOW while idle.
PHASE	Refer to a data sheet to learn about the different settings in combination with polarity.
CLOCKRATE	The clock rate selects the division of the of the oscillator frequency that serves as the SPI clock. So with 4 you will have a clock rate of 4.000000 / 4 = 1 MHz , when a 4 MHZ XTAL is used.
NOSS	1 or 0. Use 1 when you do not want the SS signal to be automatically generated in master mode.
INTERRUPT	Specify ON or OFF. ON will enable the SPI interrupts to occur. While OFF disables SPI interrupts. ENABLE SPI and DISABLE SPI will accomplish the same.
SPIIN	When reading from the SPI slave, it should not matter what kind of data you send. But some chips require a value of 255 while others require a value of 0. By default, when the SPIIN option is not provided, a value of 0 will be sent to the SPI slave. With this SPIIN option you can override this value.
EXTENDED	An optional parameter to extend the maximum data read/write size. A value of 0 is default and will cause the SPIIN, SPIIOUT, SPIMOVE

	routines to handle a maximum data size of 255 bytes. A value of 1 will extended the data size from bytes to words which means you can move data of 65535 bytes.

Hardware SPI is the best option when it is available. Hardware SPI can be used in master and slave mode. All BASCOM SPI statements are master mode routines.
The only disadvantage is that you must use the dedicated hardware pins, the SS pin included!

When you use CONFIG SPI = HARD without any other parameter, the SPI will only be enabled. It will work in slave mode then with CPOL =0 and CPH=0.

In hardware spi mode the SPIINIT statement will set the SPI pins to :
```
SCK = Ouput
MISO = Input
MOSI = Output
```
In Master mode, the SS pin will be set to output too.

As explained for Software SPI, it is not always desirable to use the SS pin to control the SPI slave chip. Because you want to use a different pin, use multiple slave, or the slaves has an inverted SS signal.
Since the hardware SPI always has an SS pin, there is an override for this with a different name than for soft spi : **NOSS**=0|1
So where SS=NONE is used for SOFT SPI to disable automatic SPI signal generation, the HARDWARE SPI use the option NOSS=1 to do the same. NOSS means NO SS signal generation.
When NOSS is not used or NOSS=0, the default will be used where the dedicated SS pin will create the slave select signals.

One big difference with software SPI, is that in order to use the SPI in master mode, the SS pin must be set to output mode. Even if you do not use the dedicated SS pin to control a SPI slave chip !
When the SS pin is in input mode, a logic 0 at the input will turn the master mode into slave mode. A pull up resistor could do the same but our advise : use the SS pin as an output pin.
The SS pin is set to output mode when the MASTER mode is selected. So even if NOSS=1, the SS pin is set to output mode when MASTER=YES.

 When using NOSS=1 : In order to use the Hardware SPI in master mode, you need to set the SS pin to output. In input mode, this pin can be used to set the SPI bus into slave mode. You only need to set the pin to output when you use the **NOSS=1** option. With NOSS=0, the compiler will set the SS pin to output and makes SS pin logic 1.
When NOSS=1 is used, the SS pin is only made an output pin in MASTER mode. No logic level is set when NOSS=1.

This table show how SS pin is set with the various options for HW mode.

MODE	NOSS	SS PIN
MASTER	0	output, logic 1
	1	output, logic level unchanged
SLAVE	0	input
	1	input

All SPI routines are SPI-master routines. In the samples directory you will also find a

SPI hardware master and SPI hardware slave sample.
The SPI protocol is explained in the chapter : Using the SPI protocol 314

 When using a processor for both the master and slave : Take in mind that the SPI master processor clock frequency must be 1/4 of the SPI slave processor frequency.

Chips with 2 full SPI ports

Some new processors like the ATMEGA328**PB** have 2 SPI ports. In order to use this second SPI port you have to add a '1' to the statement.
CONFIG SPI1
SPI**1**IN
SPI**1**OUT
SPI**1**INIT
SPI**1**MOVE

See also

SPIIN 1513 , SPIOUT 1518 , SPIINIT 1514 , SPI 314 , SPIMOVE 1515

Example for Software SPI

```
Config Spi = Soft , Din = Pinb.0 , Dout = Portb.1 , Ss = Portb.2 ,
Clock = Portb.3
Dim var As Byte
SPIINIT  'Init SPI state and pins.
SPIOUT var , 1 'send 1 byte
```

Example for Hardware SPI, 1 slave

```
Config Spi = Hard, Interrupt = Off, Data_Order = Msb, Master = Yes,
Polarity = High, Phase = 1, Clockrate = 4, Noss = 0
Spiinit
```

7.21.76 CONFIG SPIx XTINY

Action

Configures the SPI mode of the Xtiny.

Syntax

CONFIG SPIx = HARD, MASTER = YES|NO , MODE=0-3, CLOCKDIV=div, DATA_ORDER = LSB|MSB , EXTENDED=0|1 , SPIPIN=pins

Remarks

SPIx	There is 1 SPI interfaces on the Xtiny. You need to specify **SPI0**. The DB series has 2 SPI interfaces. You can use the second interface with SPI1. The only supported option is HARD for hardware mode.
MASTER	Selects if the SPI is running in master or slave mode. Possible values : YES(1), NO(0).
MODE	The mode of the SPI interface. There are 4 modes in the range from 0-3.

	The mode decides weather the first edge in a clock cycles is rising or falling, and if data setup and sample is on leading or trailing edge. **Mode** **Leading Edge** **Trailing Edge** 0 Rising, Sample Falling, Setup 1 Rising, Setup Falling, Sample 2 Falling, Sample Rising, Setup 3 Falling, Setup Rising, Sample
CLOCKDIV	The SPI is clocked by the system clock which is divided by a the SPI divider. If you select a division factor of 4, and the system clock is 4 MHz, then the SPI clock will be 1 MHz. The possible values are : CLK2, CLK4, CLK8, CLK16, CLK32, CLK64 and CLK128. Some modes use the internal CLK2X bit. **In SLAVE mode, the maximum clock rate is CLK4.**
DATA_ORDER	Selects if MSB or LSB is transferred first. The SPI can send the Least Significant bit (LSB) or the Most Significant Bit(MSB) first.
SS	Slave select option. The possible values are : - NONE, the SS will not be set or used - AUTO, the dedicated pin is used, the pin depends on the used processor.
EXTENDED	An optional parameter to extend the maximum data read/write size. A value of 0 is default and will cause the SPIIN, SPIIOUT, SPIMOVE routines to handle a maximum data size of 255 bytes. A value of 1 will extended the data size from bytes to words which means you can move data of 65535 bytes. When defined for one SPI interface like SPI0, it will also work for all other SPI interfaces like SPI1, SPI2 and SPI3.
SPIPIN	This option allows to select the alternative pin locations. The default option is always the first listed. When you use the default option there is no need to specify it with SPIPIN. The MOSI, CLOCK and SS pin will be set to output mode. The SS pin will only be set to output mode when you use the SS=AUTO option. Otherwise you need to set the pin you use yourself to output mode. When you select the NONE option which means that the SPI is not connected to any of the port pins, no pins will be initialized to output. The mx4809 for example has these options : DEF_PA4567, ALT1_PC0123,ALT2_PE0123,NONE The first listed is DEF_PA4567 which means that this is the default location when you do not use the PORTMUX. PA means port A. And the pins are listed in MOSI,MISO,CLOCK,SS order. Thus PORTA.4 is connected to MOSI, PA.5 to MISO, etc. When you select ALT1_PC0123 it means that you select the alternative pin location 1. This will use PORTC0-3. And the NONE option means that none of the SPI pins are connected. The compiler will set the proper port direction and levels. It will also configure the PORTMUX in case the SPIPIN option is used. So when you use the default location, do not use SPIPIN in order to get less code.

The SPI settings for the Xtiny differs only for the hardware name : SPI0 instead of SPI.
SPIINIT is not required for Xtiny. The pins are initialized as part of the CONFIG statement.
SPIINIT is ignored for Xtiny.

 If you need to use a different pin for SS or when you need to switch the logic level yourself for SS, and thus you use the SS=NONE option, you must setup the SS pin, even if you do not use it yourself. You must prevent that the SS pin will be made low in input mode since that will set the SPI into SLAVE mode, even while it was in MASTER mode.

When SS is in auto mode, the SS pin will be made low before each SPI transfer and be made high when the SPI transfer is finished. SS can be used when multiple slaves are used, or to synchronize data packets.

 The pins are configured before the SPI control register is set. If you do not use the AUTO mode, you must set the pin direction and state yourself before using the CONFIG SPI. The following table shows which pins you have to set when NOT using the AUTO mode.

Pin	Master Mode	Slave Mode
MOSI	User set	Input
MISO	Input	User set
SCK	User set	Input
SS	User set	Input

It is very important that you set the pin direction and level BEFORE you use the CONFIG SPI statement. This because the CONFIG SPI will enable the SPI interface and once enabled you can not change data direction/level.

See Also

SPIOUT 1518, SPIIN 1513, SPIMOVE 1515 , SPI1OUT, SPI1IN, SPI1MOVE

Example

```
'-----------------------------------------------------------------
----------------
'name                     : spi.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates SPI
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24
```

```
'set the system clock and prescaler
Config Sysclock = 20mhz , Prescale = 1

'configure the USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'configure the SPI to master mode
Config Spi0 = Hard , Clockdiv = Clk32 , Data_order = Msb , Mode =
0 , Master = Yes , Ss = Auto


'dimension a variable
Dim B As Word
B = &B1010_1010

Print "Test SPI"
Spiinit
'initialize SPI is not required for Xtiny


Do
   Spiout B , 1                                          'send
some data
   Waitms 1000
Loop

End
```

7.21.77 CONFIG SPIx XMEGA

Action

Configures the SPI mode of the Xmega.

Syntax

CONFIG SPIx = HARD, MASTER = YES|NO , MODE=0-3, CLOCKDIV=div, DATA_ORDER = LSB|MSB , EXTENDED=0|1

Remarks

SPIx	There are 4 SPI interfaces on the Xmega. You need to specify SPIC, SPID, SPIE or SPIF for SPIx. The value must be HARD.
MASTER	Selects if the SPI is running in master or slave mode. Possible values : YES(1), NO(0).
MODE	The mode of the SPI interface. There are 4 modes in the range from 0-3. The mode decides weather the first edge in a clock cycles is rising or falling, and if data setup and sample is on leading or trailing edge.

	Mode	**Leading Edge**	**Trailing Edge**
	0 CPOL=0, CPHA=0	Rising, Sample	Falling, Setup
	1 CPOL=0, CPHA=1	Rising, Setup	Falling, Sample
	2 CPOL=1, CPHA=0	Falling, Sample	Rising, Setup
	3 CPOL=1, CPHA=1	Falling, Setup	Rising, Sample
CLOCKDIV	The SPI is clocked by the system clock which is divided by a the SPI divider. If you select a division factor of 4, and the system clock is 4 MHz, then the SPI clock will be 1 MHz. The possible values are : CLK2, CLK4, CLK8, CLK16, CLK32, CLK64 and CLK128. Some modes use the internal CLK2X bit. **In SLAVE mode, the maximum clock rate is CLK4.**		
DATA ORDER	Selects if MSB or LSB is transferred first. The SPI can send the Least Significant bit (LSB) or the Most Significant Bit(MSB) first.		
SS	Slave select option. The possible values are : - NONE, the SS will not be set or used - AUTO, the dedicated pin is used, this is portC.4 for SPIC, portD.4 for SPID, portE.4 for SPIE and portF.4 for SPIF.		
EXTENDED	An optional parameter to extend the maximum data read/write size. A value of 0 is default and will cause the SPIIN, SPIIOUT, SPIMOVE routines to handle a maximum data size of 255 bytes. A value of 1 will extended the data size from bytes to words which means you can move data of 65535 bytes. When defined for one SPI interface like SPIC, it will also work for all other SPI interfaces like SPID, SPIE and SPIF.		

The SPI settings for the Xmega differ from the SPI settings for normal AVR chips.
In order to be able to use the four different SPI interfaces the Xmega uses a channel which you need to OPEN.
After you have opened the device, you can send/receive data using PRINT and INPUT.

There are 2 manuals available from ATMEL for every ATXMEGA Chip

1. One Family Manual like for example for a ATXMEGA128**A**1 it is Atmel AVR XMEGA **A** Manual
2. Another Manual for the single chips like for example for an ATXMEGA128A1 it is the ATxmega64A1/128A1/192A1/256A1/384A1 Manual. In this Manual you find for example the Alternate Pin Functions. So you can find which Pin MISO, MOSI etc.

The SS pin, MOSI and CLOCK pins are set to output mode automatic in master mode. The SS pin is also made high. The SS pin is only configured when you have selected SS=AUTO.

 If you need to use a different pin for SS or when you need to switch the logic level yourself for SS, and thus you use the SS=NONE option, you must setup the SS pin, even if you do not use it yourself. You must prevent that the SS pin will be made low in input mode since that will set the SPI into SLAVE mode, even while it was in MASTER mode.

When SS is in auto mode, the SS pin will be made low before each SPI transfer and be made high when the SPI transfer is finished. SS can be used when multiple slaves are used, or to synchronize data packets.


The pins are configured before the SPI control register is set. If you do not use the AUTO mode, you must set the pin direction and state yourself before using the CONFIG SPI. The following table shows which pins you have to set when NOT using the AUTO mode.

Pin	Master Mode	Slave Mode
MOSI	User set	Input
MISO	Input	User set
SCK	User set	Input
SS	User set	Input

It is very important that you set the pin direction and level BEFORE you use the CONFIG SPI statement. This because the CONFIG SPI will enable the SPI interface and once enabled you can not change data direction/level.

If you want to change pin levels , you must disable the SPI interface first by clearing bit 6 :

```
Spid_ctrl.6 = 0                          ' disable
Config Portd.4 = Output                   ' set direction
Set Portd.0.4                            ' set level
Spid_ctrl.6 = 1                          ' enable
```

See also

INPUT 1493, PRINT 1501, OPEN 1386
SPIIN 1513 , SPIOUT 1518 , SPIINIT 1514 , SPI 314 , SPIMOVE 1515

Example

```
Dim Bspivar As Byte , Ar(4) As Byte , W As Word
Bspivar = 1
Config Spic = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk2 ,
Data_order = Msb
Config Spid = Hard , Master = Yes , Mode = 1 , Clockdiv = Clk8 ,
Data_order = Lsb
Config Spie = Hard , Master = Yes , Mode = 2 , Clockdiv = Clk4 ,
Data_order = Msb
Config Spif = Hard , Master = Yes , Mode = 3 , Clockdiv = Clk32 ,
Data_order = Msb

Open "SPIC" For Binary As #10
Open "SPID" For Binary As #11
Open "SPIE" For Binary As #12
Open "SPIF" For Binary As #13
Open "SPI" For Binary As #bspivar                              ' use a
dynamic channel
'SPI channel only suppor PRINT and INPUT
Print #10 , "to spi" ; W
Input #10 , Ar(1) , W
Print #bspivar , W
Input #bspivar , W
```

7.21.78 CONFIG SERVOS

Action
Configures how much servo's will be controlled.

Syntax
CONFIG SERVOS = X , Servo**N** = Portb.0 , Reload = rl [, INTERVAL=t]
CONFIG SERVOS = X , Servo**N** = Portb.0 , MODE=mode , PRESCALE=pre

Syntax Xmega
CONFIG SERVOS = X , Servo**N** = Portb.0 , MODE=mode , TIMER= tmr, PRESCALE=pre

Remarks
Servo's need a variable pulse in order to operate. The CONFIG SERVOS directive will set up a byte array with the servo pulse width values and will initialize an ISR that uses TIMER0.

X	The number of servo's you want to control. Each used servo will use one byte of SRAM.
servoN	The port pin the servo is attached too. N represents a value between 1 and 10. When you specify that you will use multiple servo's you need to specify a pin for each servo. Like : config servos=**3**, servo**1**=portb.0, servo**2** =portb.2, servo**3**=portC.4
reload	The reload value for the ISR in uS. This is the overflow rate of the timer. So when 100 is used, it means that each 100 uS an interrupt will occur to update the servo variables.
Interval	The update interval. Using the interval option will result in using alternative servo code optimized for servos.
Mode	The normal default modes use software PWM with a relatively high frequency. This will give a big processor load since the timer ISR is executed many times. It allows to create create precise pulses in small steps. But when controlling a simple RC servo, it is also possible to use a lower refresh rate which will result in lower processor load. MODE=SERVO will work for normal AVR and XMEGA. You do not need to specify the interval or reload value.
Prescale	The prescale value is calculated so that the 8 bit timer interrupt is executed every 2 ms. Inside the interrupt, the servo pin is made high for the value of the servo() array. Then the next time inside the ISR, the pin is set low for the reset of the time. It depends on the processor frequency if you get a good range. In the report you can find the used prescale value as a constant named _SERVO_PRESCALER. When you do not get a full servo swing, you might want to try a higher prescale value. The prescale parameter overrides the automatic calculation.
Timer	This is for XMEGA only. Specify the name of the timer that will be used in interrupt mode.

PWM MODE
When you use for example :

Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 10
The internal ISR will execute every 10 uS.

An arrays named SERVO() will be created and it can hold 2 bytes : servo(1) and servo(2).

By setting the value of the servo() array you control how long the positive pulse will last. After it has reached this value it will be reset to 0.

The reload value should be set to 10. After 20 mS, a new pulse will be generated. You can use other reload values but it will also mean that the repeat value will change.

The PORT pins specified must be set to work as an output pin by the user.
CONFIG PINB.0 = OUTPUT
Will set a pin to output mode.

The CONFIG SERVOS only works with servo's that rotate 180 degrees. These are the servo's found in RC models.
There are also continuous rotation servos which work different. The servo code will NOT work on these servos.

Alternative Servocode

When using the INTERVAL option, you can use alternative code which is optimized for servo's.(this is however not the MODE=SERVO)
You should use a RELOAD value of 100 in that case and an interval of 100 should be used for best results.
Using a reload of 100 uS will give more time to the main application. This does give lower resolution but this is not a problem for most model servos. With an interval of 100, the refresh will be done in 100x100 us which results in 10 mS.
The following test code was used:

```
Config Servos = 2 , Servo1 = Portd.7 , Servo2 = Portb.1 , Reload = 100 , Interval =
100
Servo(1) = 10
Servo(2) = 5
Enable Interrupts
Do
 For J = 8 To 16
   Servo(1) = J
   Waitms 5000 ' some time to check if the servo is stable
 Next
 Waitms 5000
Loop
```

SERVO mode

The MODE=SERVO can be used for normal AVR and XMEGA. It results in a lower processor load.

XMEGA

The Xmega has several timers. You must specify the timer to be used.
The Xmega has 16 bit timers and instead of a byte array, a word array is created for the servo values.

The Xmega can also create pulses with it's timers without the need of interrupts. But this mode demands that you use fixed CCx pins. The software servo pulse mode, allows you to chose any pin.

Resources used

TIMER0 is used to create the ISR. Xmega will use TCxx.

NOTE

The servo() value is not absolute. It will depend on the processor clock. This means that these values might need an adjustment when you alter the $crystal value.

Example PWM mode

```
'-----------------------------------------------------------------------
------------------
'name                     : servos.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates the SERVO option
'micro                    : 90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'Servo's need a pulse in order to operate
'with the config statement CONFIG SERVOS we can specify how many servo's
we
'will use and which port pins are used
'A maximum of 14 servos might be used
'The SERVO statements use one byte for an interrupt counter and the
TIMER0
'This means that you can not use TIMER0 anymore
'The reload value specifies the interval of the timer in uS
'Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 10

Config Servos = 1 , Servo1 = Portb.0 , Reload = 10
'as an option you can use TIMER1
'Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 10 ,
Timer = Timer1

'we use 2 servos with 10 uS resolution(steps)

'we must configure the port pins used to act as output
Config Portb = Output
```

```
'finally we must turn on the global interrupt
Enable Interrupts

'the servo() array is created automatic. You can used it to set the
'time the servo must be on
Servo(1) = 10                                                    '10 times 10
= 100 uS on
'Servo(2) = 20                                                    '20 times
10 = 200 uS on
Do
Loop

Dim I As Byte
Do
For I = 0 To 100
   Servo(1) = I
   Waitms 1000
Next

For I = 100 To 0 Step -1
'  Servo(1) = I
   Waitms 1000
Next
Loop
End
```

Example SERVO mode

```
'-----------------------------------------------------------------------------
'                                    (c) 1995-2025, MCS Electronics
'                                        servos-timer0.bas
'-----------------------------------------------------------------------------
$regfile = "m88def.dat"
$crystal = 8000000
$hwstack = 64
$swstack = 64
$framesize = 64


Config Com1 = 19200 , Parity = None , Stopbits = 1 , Databits = 8
Print "Servo test"

Config Servos = 2 , Mode = Servo , Servo1 = Portb.0 , Servo2 = Portb.1
'Config Servos = 2 , Mode = Servo , Servo1 = Portb.0 , Servo2 = Portb.1 , Prescale= 256

' you need to chose SERVO mode for lowest system resources
Enable Interrupts                                                ' you must enable interrupts
since timer 0 is used in interrupt mode


Dim Key As Byte
'notice that servo() array is a byte array, which is created automatic

Do
   Key = Inkey()                                                 ' get data from serial port
   If Key = "l" Then                                             'left
      Servo(1) = 100
      Servo(2) = 100
   Elseif Key = "m" Then                                         ' middle
      Servo(1) = 170
      Servo(2) = 170
   Elseif Key = "r" Then                                         ' right
      Servo(1) = 255
      Servo(2) = 255
   Elseif Key <> 0 Then                                          ' enter user value
      Input "Servo1 " , Servo(1)
      Servo(2) = Servo(1)
   End If
Loop
```

Example XMEGA SERVO mode

```
'-----------------------------------------------------------------------------
'                                    (c) 1995-2025, MCS Electronics
```

```
'                                       xmega-servo.bas
'-------------------------------------------------------------------------------
$regfile = "xm128a1def.dat"
$crystal =  32000000
$hwstack = 64
$swstack = 64
$framesize = 64


Config Osc =  Enabled , 32mhzosc =  Enabled

'configure     the     systemclock
Config   Sysclock = 32mhz ,   Prescalea = 1 ,    Prescalebc = 1_1

Config Com1 = 19200 , Mode =  Asynchroneous ,  Parity = None ,  Stopbits = 1 ,  Databits = 8
Print "Servo     test"

Config  Servos = 2 , Mode =  Servo , Timer =  Tcc0 ,  Servo1 = Portb.0 ,  Servo2 = Portb.1
' you need to chose SERVO mode and you must provide the name of the timer that will be
used for the system tick
Enable Interrupts                                                ' you must enable interrupts
since timer TCC0 is used in interrupt mode


Dim Key As Byte
'notice that servo() array is a word array, which is created automatic

Do
    Key = Inkey()                                                ' get data from serial port
    If Key = "l" Then                                            'left
        Servo(1) = 12800
        Servo(2) = 12800
    Elseif Key = "m" Then                                        ' middle
        Servo(1) = 19200
        Servo(2) = 19200
    Elseif Key = "r" Then                                        ' right
        Servo(1) = 40000
        Servo(2) = 40000
    Elseif Key <> 0 Then                                         ' enter user value
       Input "Servo1   " ,  Servo(1)
        Servo(2) =  Servo(1)
    End If
Loop
```

7.21.79 CONFIG STRCHECK

Action

Configures string check

Syntax

CONFIG STRCHECK = ON|OFF

Remarks

By default the string check is OFF. You can turn it on for additional string overwrite checking.
Why is it a problem to overwrite a string? A string is in fact a series of bytes that end with a null byte. For this reason a string always uses one more byte than it can store. DIM S As string * 4 , can hold 4 characters. The internal size is 5 bytes. The extra byte could store the size too, but the advantage of using 0 strings is that you can DIM a large string say 1000 bytes, and still need 1 byte to mark the end. And as always there is a disadvantage too : you can not store a character with a 0 value since it marks the end.

Consider code like this :

Dim S as String * 4, b as byte, Z as string * 10

The data is stored after each other. When you write a too long string to S, you will overwrite the variable B since it is allocated after the string S.

In some cases the compiler can check if you overwrite a string. For example when you assign a constant to a string that is too small to hold the content.
You always get an error in such a case. For example : S = "abcd" is ok, but S="abcdE" will give an error.

The problem is when you use another string to assign a string. Code like this :

```
 Z="abcdefg"  : S = Z
```
This will overwrite B.

For this purpose the CONFIG STRCHECK=ON can be used.
It will use an alternative piece of code that will check against overwriting.

When a string will not fit, only the part that will fit will be assigned.
So this option can give unexpected results as well. But at least no other data will be overwritten.

As the example will show it is still not always possible to guard against overwriting of strings.

Example

```
'-----------------------------------------------------------------
----------------
'name                     : string-check.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates string overwrite
protection
'micro                    : avrDA28
'suited for demo          : no
'commercial addon needed  : yes but change the DAT file to test
with any other micro
'-----------------------------------------------------------------
----------------
$regfile = "avrx128da28.dat"
$crystal = 24000000
$hwstack = 40
$swstack = 40
$framesize = 40
'$bigstrings

Const Cigoneerror = 1                                          'make
1 to compile without errors
Const Cstrcheck = 1
'check strings for overwrite
Const Cspeclen = 1
'specify length for better checking

'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1
'select system clock and frequency
```

```
Config Osc = Enabled, FREQUENCY=24MHZ

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

#if Cstrcheck
   Config Strcheck = On                                    'when
ON there will be a check so memory is not overwritten. it will
create slightly mode code since
                                                           'the
size must be passed and checked
#endif
Dim Bdummy As Byte
'Xtiny and Xmega auto create internal variables after the first
DIM
dim idx as byte            'index for array
dim sTarget as string * 10 'this is a string we are going to
assign
Dim Bbeyond As Byte                                        'put
a byte here
dim sSource as string * 20 'this is the source string
dim sAr(5) as string * 10  'test an array as well

Const Csomestring10 = "0123456789"                         ' a
test constant
Const Csomestring11 = Csomestring10 + "A"                  ' and
one string 1 byte longer
Const Csomestring20 = "0123456789abcdefghij"               ' a
test constant
'by default there is no protection against string overwrites.
This is because the first processors had little RAM.
'using large strings on them was not possible. A string is just
an array of bytes with a zero byte at the end.
'for this reason a string can not hold a 0 value.
'A string always takes 1 more byte in memory than the length of
the actual string
'The length info is not stored inside the string. This has pros
and cons. The pro is that you can DIM a string longer than 255
bytes and it will
'still work. The con is that when you pass strings to sub
routines and functions, the maximum length is not passed a long.
So there is no check possible.

'first assigna value to bBeyond which is located after the string
we assign
Bbeyond = 123                                              ' the
idea is that this remains 123
```

```
'There can be a number of problems.
Starget = Csomestring10                                          '
this should be fine
#if Cigoneerror = 0
   Starget = Csomestring11                                       'here
you get an error 119 since you assign a constant with a known
length that is too long to fit
#ENDIF

Ssource = Csomestring11                                          '
this is no problem since it will fit
Starget = "a" + Starget                                          '
this is a problem since this will write beyond the string

print bBeyond
Bbeyond = 123                                                    'the
idea is that this remains 123

'--- array test ---
sAr(2)="0123456789"  'create a string and check if it is not
overwritten
Idx = 1 : Sar(idx) = "ABC" : Sar(idx) = Sar(idx) + Csomestring10
     'lets check if it works for arrays as well
print bBeyond
Bbeyond = 123                                                    'the
idea is that this remains 123

#if Cspeclen
  declare sub somesub(byval s1 as string * 10,s2 as string * 10)
#else
  Declare Sub Somesub(byval S1 As String , S2 As String)
#endif
declare function myfunc() as string

sSource= csomestring11
Somesub Ssource , Ssource
'somesub csomestring11   ,sSource  'will create error in case
length is specified
print bBeyond

sTarget=Myfunc()
print bBeyond

sSource= csomestring20
Idx = 15 : Starget = Left(ssource , Idx)
'check this too

end
```

```
's1 passed by value, s2 by reference
#if cSpecLen
  sub somesub(byval s1 as string * 10,s2 as string * 10)
#else
  sub somesub(byval s1 as string,s2 as string)
#endif
   local test as byte
   test=123

    sTarget=s1
    print bBeyond
    bBeyond=123            'the idea is that this remains 123

    starget=s2
    print bBeyond

    s1="aa"
    s1=s1+csomestring10
    'when you watch the local test value you will see it is
overwritten.
    'so here is a potential problem too
end sub


function myfunc() as string
#IF cIgoneError=0
     myfunc = csomestring11 'this will give an error
#endif
     myfunc= "a"     'this will work
     myfunc=myfunc + csomestring10  'but here we have a problem
!!!
     'despite the test enabled, we can not know the actual size
since
     'this means that when you assign a string with a user string
function you need to be careful
end function
```

7.21.80 CONFIG SUBMODE

Action

This option sets how the compiler deals with Subs, Functions and Declarations.

Syntax

CONFIG SUBMODE = NEW|OLD

Remarks

When the SUBMODE option is not configured, the default 'OLD' will be used.
This is the old mode used in versions up to 2070.

This old mode demands that you DECLARE a function or sub, before you call/use it. It also binds in the sub/function at the same location as in your code.
When working with $include files, this requires that you insert an $include file with the SUBS/FUNCTIONS at the end of your code, and that you insert an $include file with the DECLARE statements at the start of your code.
Or you can put the DECLARE and actual implementation in one file and use a GOTO to jump over the Sub/Function code.
For example consider this code :

```
  print "code here"
  Sub test()
    print
  End Sub
```

When using the OLD method, this will give problems since the code will run into the Sub test, without it actual being called.
We can solve that like this by placing the sub/functions after the END statement:

```
  print "code here"
END
  Sub test()
    print
  End Sub
```

or we can use a GOTO:

```
  print "code here"
GOTO skip
  Sub test()
    print
  End Sub

skip:
 print "more code here"
```

When you use CONFIG SUBMODE=NEW, most behaviour is changed :
- there is no need to DECLARE a sub/function before you call it. But, the actual sub/function code must be placed before the actual call!
- only the used sub/functions are included
- the compiled sub/function code is placed after the main program. this is something you do not need to worry about.
- you can $include the modules without a GOTO to jump over the code because code is stored automatically after the END statement.
- sub/functions behave like macro's : only when used they are included
- Any Dead code or Un-used code will not be Compiled!

This means you can $Include a file with all your collection of Sub or Functions and the Compiler will determine which items are to be used during Compilation saving you unnecessary wastage of Flash space.

See also

DECLARE SUB 1221, SUB 1545, DECLARE FUNCTION 1215, CALL 806

Example

```
$regfile = "m88def.dat"
$crystal = 8000000
config submode=new

declare sub test1()                                          ' not
required

sub test2()                                                  '
this sub is not used and will not be compiled
    print "test2"
end sub

function myfunc() as byte                                    '
called from test1
   myfunc = 1
end function

sub test1()
    print "test1"
    print myfunc()                                           '
uses myfunc
end sub

print "test"
test1                                                        '
call test1
end                                                          '12
```

7.21.81 CONFIG SYSCLOCK XMEGA

Action

Selects the oscillator source for the system clock.

See also ATXMEGA 425

Syntax

CONFIG SYSCLOCK=sysclock , **PRESCALEA=**prescaleA, **PRESCALEBC=**prescaleBC

Remarks

SYSCLOCK	The oscillator used for generation of the system clock. This oscillator must be running. You MUST use CONFIG OSC before you use CONFIG SYSCLOCK. The CONFIG SYSCLOCK will wait till the oscillator is running stable. Possible values: - 2MHZ - 32MHZ - EXTERNAL - PLL
PRESCALEA	The Xmega has 3 prescalers. With PRESCALEA you configure the clock division of the first prescaler. Possible values:

	1 , 2 ,4, 8, 16, 32, 64, 128,256,512
PRESCALEBC	The Xmega has 3 prescalers. With PRESCALEBC you configure the clock division of the second and the third prescaler. Possible values: - 1_1 (1 + 1 division) - 1_2 (1+2 division) - 4_1 (4 + 1 division) - 2_2 (2 + 2 division) This 1_2 will make the second prescaler divide by 1 and the third prescaler divide by 2.

See also
CONFIG OSC 1008

Example
```
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 ' use 32 MHz
```

7.21.82 CONFIG SYSCLOCK XTINY

Action
Selects the oscillator source for the system clock.

Syntax
CONFIG SYSCLOCK=sysclock , **PRESCALE=**prescale , **CLOCKOUT=**clockOtp, **CLOCKOUT_PIN=**pinmode

Remarks

SYSCLOCK	The oscillator used for generation of the system clock. This oscillator must be running. Possible values: - 16_20MHz : internal 20 MHz oscillator or 16 MHz oscillator. This depends on the fuse you set. - 32KHz_INT : internal ultra low power oscillator - 32KHz_EXT : 32 Khz external crystal oscillator - EXTERNAL : external clock
PRESCALE	The Xtiny can divide the oscillator clock with the following values : 1,2,4,8,10,12,16,24,32,48 and 64.
CLOCKOUT	The Xtiny can route the clock output to a pin. Select ENABLED or DISABLED. Even in input mode the clock signal will be present on the designated pin. But the signal is best when the pin is set to output mode.
CLOCKOUT_PIN	This option will set the CLOCKOUT put pin into output mode. The only possible value is : OUTPUT

When using the CLOCKOUT option you can either set the output pin yourself into output mode or use the CLOCKOUT_PIN opiont.
Some processors do not have the CLOCKOUT pin. For these processors this option is not present in the DAT files.

See also
NONE

Example
```
'-----------------------------------------------------------------
----------------
'name                     : serial-osc.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates USART
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24

'set the system clock and prescaler
'the clockout_pin is PB.5
Config Sysclock = 16_20mhz , Prescale = 1 , Clockout = Enabled ,
Clockout_pin = Output

'configure the USART
'use calibrated offset to compensate the BAUD
Config Com1 = 250000 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1 , Baud_offset = Osc20_5v

Waitms 2000

Print "Test USART"
Dim B As Byte

Do
   Print "this is a baud test"
   Print Hex(clkctrl_osc20mcaliba)
   B = Inkey()
   If B = "+" Then
      Cpu_ccp = &HD8
      Incr Clkctrl_osc20mcaliba
```

```
   Elseif B = "-" Then
      Cpu_ccp = &HD8
      Decr Clkctrl_osc20mcaliba
   End If
   Waitms 500
Loop
```

7.21.83 CONFIG TCA0

Action

This configuration statement configures timer TCA0 found in the XTINY.

Syntax

CONFIG TCA0=mode, PRESCALE=prescale, RUN=run, LUPD=lupd , COMPAREx=compareX, RESOLUTION=resolution, EVENT_ACTION=event_action, OVF_INT=int, CMP0_INT=int, CMP1_INT=int, CMP2_INT=int

Remarks

At the moment of writing, all XTINY processors have one TIMER TCA0. This is a 16 bit timer with the following capabilities :

• 16-Bit Timer/Counter
• Three Compare Channels
• Double Buffered Timer Period Setting
• Double Buffered Compare Channels
• Waveform Generation:
– Frequency generation
– Single-slope PWM (pulse-width modulation)
– Dual-slope PWM
• Count on Event
• Timer Overflow Interrupts/Events
• One Compare Match per Compare Channel
• Two 8-Bit Timer/Counters in Split Mode

We do not want to copy the data sheet info. You best read that before you use the timer.
After reading the data sheet the following options will make more sense.

mode	This options sets the Timer and/or Wave Generation mode. Possible values : - NORMAL, no wave generation (NORMAL) - FREQ , frequency generation (FRQ) - PWM , pulse width modulation single slope (SINGLESLOPE) - PWM_TOP, pwm dual slope (DSTOP) - PWM_TOPBOT, pwm dual slope (DSBOTH) - PWM_BOT, pwm dual slope (DSBOTOM) - A value between 0-7 will load the mode. See table 2.
PRESCALE	The pre scaler can divide the system clock that is applied to the timer. The pre scaler will only divide the system clock. Possible values : - 1 , 2, 4, 8, 64, 256, 1024

	- OFF, timer is disabled
RUN	This enables or disables the timer. Possible values : ON : timer will run OFF : timer will stop
LUPD	Lock update. Possible values : MANUAL : LUPD in TCA.CTRLE not altered by system AUTO : LUPD in TCA.CTRLE set and cleared automatically
CompareX	In the FRQ or PWM Waveform Generation mode, the PORT output register for the corresponding pin can be overridden. COMPARE0 will enable/disable WO0 COMPARE1 will enable/disable WO1 COMPARE2 will enable/disable WO2 DISABLE means : Port output settings for the pin with WOn output respected. ENABLE means : Port output settings for pin with WOn output overridden in FRQ or PWM Waveform Generation mode
COMPARExL COMPARExH	In SPLIT mode the counters are split into two 8 bit timers. The name COMPARE0 becomes COMPARE0L and COMPARE0H. Instead of WO0,WO1 and WO2, there are 3 additional outputs : WO3, WO4 and WO5.
RESOLUTION	This option sets the resolution of the timer. Possible value : - NORMAL : 16 bit - SPLIT : two 8 bit timers
EVENT_ACTION	This option defines what kind of event action will increment or decrement. Possible values : - DISABLED : counting on event input is disabled - ENABLED, COUNT_POS_EDGE : count on positive edge event - COUNT_ANY_EDGE : count on any edge event - COUNT_HIGH_LVL : count on prescaled clock while event line is 1 - COUNT_UPDOWN : count on prescaled clock. The event controls the count direction. Up counting when the event line is 0, down counting when the event line is 1.
OVF_INT CMP0_INT CMP1_INT CMP2_INT	You can enable/disable interrupts in BASCOM using the ENABLE/ DISABLE statement. You can also enable interrupts using the CONFIG statement. Possible values : ENABLED and DISABLED Possible interrupt sources you can set : OVF_INT : timer overflow/underflow interrupt CMP0_INT : compare channel 0 interrupt CMP1_INT : compare channel 1 interrupt CMP2_INT : compare channel 2 interrupt

Table 2.

Value	Mode	TOP	UPDATE	EVENT
0	NORMAL	PER	TOP	TOP
1	FREQ	CMP0	TOP	TOP
2	reserved			
3	PWM, single slope	PER	BOTTOM	BOTTOM
4	reserved			
5	PWM, dual slope	PER	BOTTOM	TOP

6	PWM, dual slope	PER	BOTTOM	TOP and BOTTOM
7	PWM, dual slope	PER	BOTTOM	BOTTOM

In normal AVR the ICR register is used to define the PWM frequency. In Xtiny the PER register must be used : TCA0_per = 8000
The duty cycle can be loaded in the TCA0_CMP0 register (or a register of the other channels)

In normal AVR processors the timers had an alias to the counter register named TIMER0, TIMER1 , etc.
Thus TIMER1 would access TIMER1 registers TCNT1L and TCNT1H.

In the Xtiny, megaX. AVRX these aliases do not exist. There is however an alias to access word registers like a word.
You can find these aliases in the DAT file under the [WIO] section.
For TCA0 you will find :
- TCA0_CNT the timer counter register
- TCA0_PER the period register
- TCA0_CMP0 the compare 0 register
- TCA0_CMP1 the compare 1 register
- TCA0_CMP2 the compare 2 register

All relevant registers that form a word register like :
TCA0_CNTL=2592 ; 0A20 byte alias LSB see WIO
TCA0_CNTH=2593 ; 0A21 byte alias MSB see WIO
Will have an entry under the WIO section.
This is simply the name without the L/H
And the address is always the low register address.
Because the name is under the WIO section the variable/register will be treated as a 16 bit word. The correct read/write order will be used by the compiler which is different for AVR/XMEGA/XTINY

When you like to use your own definition or alias you could add an alias. Just take care that an update will replace the DAT files.
For example if you like TIMER1 or TCA0 you can add it to the WIO section like this :
TCA0_CNT=2592 ; 0A20 word ## EXISTING ENTRY
TCA0 = 2592 ; NEW ENTRY

If you like an alias to be used you best write to support. When there is enough demand we add it.

See also
NONE

Example
```
'-----------------------------------------------------------------
----------------
'name                    : TCA0-PWM.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates TCA0
```

```
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-------------------------------------------------------------
---------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24
'set the system clock and prescaler

Config Sysclock = 16_20mhz , Prescale = 1 , Clockout = Enabled

'configure the USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

Waitms 2000

Print "Test TCA0"

Config Portb.0 = Output                                    'WO0
output

Config Tca0 = Pwm_bot , Prescale = 1 , Resolution = Normal ,
Compare0 = Enabled , Run = On
Tca0_per = 8000                                            'PWM
frequency (period)
Tca0_cmp0 = 2000                                           '25%
duty cycle on pin PB0 (WO0)

Do
  nop
Loop
```

7.21.84 CONFIG TCB0-TCB1

Action

This configuration statement configures timer TCB0/TCB1 found in the XTINY.

Syntax

CONFIG TCB0|TCB1=mode, RUN=run, PRESCALE=prescale, RUNMODE=runmode , SYNCUPDATE=syncupdate, ASYNC=async, CCMP_INIT=ccmp_init, CCMP_OTP=ccmp_otp, FILTER=filter, EDGE=edge, CAPT_EVENT=ecapt_event, CAPT_INT=capt_int

Remarks

At the moment of writing, all XTINY processors have one TIMER TCB0. Some processors have 2 TCB timers like the tiny3216.
The second TCB timer is named TCB1.
The TCB is is a 16 bit timer with the following capabilities :

• 16-Bit Counter Operation Modes:
– Periodic interrupt
– Time-out check
– Input capture
• On event
• Frequency measurement
• Pulse-width measurement
• Frequency and pulse-width measurement
– Single shot
– 8-bit Pulse-Width Modulation (PWM)
• Noise Canceler on Event Input
• Optional: Operation Synchronous with TCA0

You best read that before you use the timer.
After reading the data sheet the following options will make more sense.

mode	This options sets the Timer mode. Possible values : - PERIODIC_INT : Periodic interrupt - TIME_OUT_CHECK : time out check - INP_CAP_EVENT : input capture event - INP_CAP_FREQ : input capture frequency - INP_CAP_PWM : input capture pulse with measurement - INP_CAP_FREQ_PWM : input capture frequency width measurement - SINGLE_SHOT : single shot - PWM : 8 bit PWM - A value between 0-7 will load the mode. See table 2.
PRESCALE	The pre scaler can divide the system clock that is applied to the timer. The pre scaler will divide the system clock. Possible values : - 1 , 2 - TCA0 : uses CLK_TCA from timer TCA0 - OFF, timer is disabled
RUN	This enables or disables the timer. Possible values : ON : timer will run OFF : timer will stop
RUNMODE	Run in standby mode. ENABLED : the timer runs in standby sleep mode. Except when PRESCALE is set to TCA0. DISABLED : timer is stopped in standby sleep mode.
SYNCUPDATE	Synchronize Update. ENABLED : TCB will restart whenever the TCA0 counter is restarted or overflows. This can be used to synchronize capture with the PWM period DISABLED : no sync
ASYNC	Asynchronous Enabling. ENABLED : asynchronous updates of the TCB signal in single shot mode The output will go HIGH when an event arrives

<table>
<tr><td></td><td>DISABLED : The output will go HIGH when the counter starts after synchronization.</td></tr>
<tr><td>CCMP_INIT</td><td>Compare/Capture PIN initial value. This setting is used to set the initial output value of the pin when an pin output is used. This bit has no effect in 8 bit PWM and single shot mode.
LOW : initial pin state is low
HIGH : initial pin state is high</td></tr>
<tr><td>CCMP_OTP</td><td>Compare/Capture output enable. This option is used to set the output value of the compare/capture output
DISABLED : Compare/capture output is zero.
ENABLED : Compare/capture output has a valid value</td></tr>
<tr><td>FILTER</td><td>Filter capture noise cancellation filter.
ENABLED : the input capture noise cancellation unit is enabled
DISABLED : input capture noise cancellation unit is disabled.</td></tr>
<tr><td>EDGE</td><td>Event Edge. This selects the event edge. The effect of this depends on the selected count mode.
<table>
<tr><th>Mode</th><th>Edge</th><th>Positive Edge</th><th>Negative Edge</th></tr>
<tr><td>PERIODIC_INT</td><td>0
1</td><td>NA
NA</td><td>NA
NA</td></tr>
<tr><td>TIME_OUT_CHECK</td><td>0
1</td><td>Start counter
Stop counter</td><td>Stop counter
Start counter</td></tr>
<tr><td>INP_CAP_EVENT</td><td>0

1</td><td>Input capture freq and pulse with measurement mode
NA</td><td>NA

capture=count</td></tr>
<tr><td>INP_CAP_FREQ</td><td>0
1</td><td>capture=count,init,int
NA</td><td>NA
capture=count, init, int</td></tr>
<tr><td>INP_CAP_PWM</td><td>0
1</td><td>init
capture=count, int</td><td>capture=count,int
init</td></tr>
<tr><td>SINGLE_SHOT</td><td>0
1</td><td>Start counter
Start counter</td><td>NA
Start counter</td></tr>
<tr><td>PWM</td><td>0
1</td><td>NA
NA</td><td>NA
NA</td></tr>
<tr><td>INP_CAP_FREQ_PWM</td><td>0</td><td colspan="2">On 1st positive : init
On following negative : capture
On second positive : stop, int</td></tr>
<tr><td></td><td>1</td><td colspan="2">On 1st negative : init
On following positive : capture
On second negative : stop, int</td></tr>
</table>
</td></tr>
<tr><td>CAPT_EVENT</td><td>Capture Event input enable.
ENABLED : event input capture is enabled.
DISABLED : event input capture is disabled</td></tr>
<tr><td>CAPT_INT</td><td>All interrupts can be enabled/disabled using the ENABLE/DISABLE statements.
The Capture interrupt enable can be enabled/disabled using the configuration parameter.
ENABLED : capture interrupt is enabled
DISABLED : capture interrupt is disabled</td></tr>
</table>

See also

CONFIG TCA0 1084, CONFIG TCD0 1090

Example

7.21.85 CONFIG TCD0

Action

This configuration statement configures timer TCD0 found in the XTINY.

Syntax

CONFIG TCD0=mode, PRESCALE=prescale , CLOCK_SOURCE=clock_source , SYNC_PRESCALER=sync_prescaler, RUN=run , CMPD_SEL=cmpd_sel , CMPC_SEL=cmpc_sel, FIFTY=fifty , AUT_UPDATE=auto_update , CMP_OVR=cmp_over , CMP_VAL=cmp_val , EVENTA_CONFIG=eventA_config , EVENTB_CONFIG=eventb_config , EVENTA_ACTION=eventa_action , EVENTB_ACTION=eventb_action , EVENTA_TRIG=eventa_trig, EVENTB_TRIG=eventb_trig , TRIGA_INT=triga_int , TRIGB_INT=trigb_int , OVF_INT=over_int , INP_MODEA=inp_modea ,INP_MODEB=inp_modeb ,CMPAEN=cmpaen, CMPBEN=cmpben ,CMPCEN=cmpcen ,CMPDEN=cmpden, CMPA=cmpa, CMPB=cmpb , CMPC=cmpc ,CMPD=cmpd, DLY_PRESCALER=dly_prescaler , DLY_TRIGGER=dly_trigger DLY_SEL=dly_sel ,DLY_VAL=dly_val , DIT_CTRL=dit_ctrl , DIT_VAL=dit_val , DIS_EOC=dis_eoc , SOFT_CAPB=soft_capb , SOFT_CAPA=soft_capa , RESTART_STROBE=restart_strobe , SYNC_STROBE=sync_strobe, SYNC_EOC=sync_eoc

Remarks

The timer TCD0 is found in a number of XTINY processors.
The TCD0 is is a 12 bit timer with the following capabilities :

- 12-bit timer/counter
- Programmable prescaler
- Double buffered compare registers
- Waveform generation
  – One ramp mode
  – Two ramp mode
  – Four ramp mode
  – Dual-slope mode
- Two separate input capture, double buffered
- Connection to event system
  – Programmable filter
- Conditional waveform on external events
  – Fault handling
  – Input blanking
  – Overload protection function
  – Fast emergency stop by hardware
- Supports both half bridge and full bridge output

You best read that before you use the timer.

After reading the data sheet the following options will make more sense.

<table>
<tr><td>mode</td><td>This options sets the Timer wave generation mode. Possible values :

- ONE_RAMP : One ramp mode
- TWO_RAMP : Two ramp mode
- FOUT_RAMP : Four ramp mode
- DUAL_SLOPE : dual slope mode

- A value between 0-3 will load the mode.</td></tr>
<tr><td>PRESCALE</td><td>The counter prescaler selects the division factor of the TCD counter clock.
Possible values :
- 1 , 4 and 32</td></tr>
<tr><td>CLOCK_SOURCE</td><td>The clock source for the TCD clock
- OSC16_20MHZ : the internal 16/20 MHz oscillator
- EXTERNAL : an external clock signal
- SYSTEM : system clock</td></tr>
<tr><td>SYNC_PRESCALER</td><td>The synchronization prescaler select the division factor of the TCD clock.
Possible values :
1, 2 , 4 and 8</td></tr>
<tr><td>RUN</td><td>This enables or disables the timer. Possible values :
ENABLED : TCD is enabled and running
DISABLED : TCD is disabled</td></tr>
<tr><td>CMPD_SEL</td><td>Compare D output select
- PWMA : Waveform A
- PWMB : Waveform B</td></tr>
<tr><td>CMPC_SEL</td><td>Compare C output select
- PWMA : Waveform A
- PWMB : Waveform B</td></tr>
<tr><td>FIFTY</td><td>Fifty percent waveform.
- ENABLED : chose this when two waveforms have identical characterics. This will cause any values written to CMPBSET/CLR register also to be written to register CMPASET/CLR</td></tr>
<tr><td>AUT_UPDATE</td><td>Automatic Update.
ENABLED : A synchronization at the end of the TCD cycle is automatically requested after the compare B Clear High register (CMPBCLRH) is written,
DISABLED : no sync</td></tr>
<tr><td>CMP_OVR</td><td>Compare output value override.
ENABLED : default values of the waveform outputs A and B are overridden by the values written in the compare X value in active state bit fields in the control D register CTRLD.CMPnxVAL.
DISABLED : no action</td></tr>
<tr><td>CMP_VAL</td><td>The CMPVAL register contains compare values for compare A and B. This is only used when CMP_OVR is enabled.
A numeric value must be used between 0-255.
The upper nible writes to CMPBVAL. The lower nibble writes to CMPAVAL.

<table>
<tr><td>CMPxVAL</td><td>A_off</td><td>A_on</td><td>B_off</td><td>B_on</td></tr>
<tr><td>PWMA</td><td>CMPAVAL[0]</td><td>CMPAVAL[1]</td><td>CMPAVAL[2]</td><td>CMPAVAL[3]</td></tr>
<tr><td>PWMB</td><td>CMPBVAL[0]</td><td>CMPBVAL[1]</td><td>CMPBVAL[2]</td><td>CMPBVAL[3]</td></tr>
</table>

In One Ramp mode, PWMA will only use A_off and A_on values and PWMB will only use B_off and B_on values.
This is due to possible overlap between the values A_off, A_on, B_off and B_on.</td></tr>
</table>

	The following config options are intended to be used alone. For example : config tcd0=mode,DIS_EOC=ENABLED The datasheet does not make it clear if these commands can be combined.
DIS_EOC	Disable at end of TCD cycle strobe. When ENABLED the ENRDY bit in TCD0_STATUS will keep low until the TCD is disabled. Writing to this bit only has effect if there no ongoing synchronization of Enable. (RUN=ENABLED) The ENRDY bit tells when the ENABLE value is synchronized to the TCD domain and is ready to be written again. The following clears the ENRDY bit : - writing to the ENABLE bit (RUN=ENABLED|DISABLED) - DIS_EOC strobe=ENABLED
SOFT_CAP**B**	Software Capture B strobe. When ENABLED a software capture to capture register B is done as soon as the strobe is synchronized to the TCD domain. Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.
SOFT_CAP**A**	Software Capture A strobe. When ENABLED a software capture to capture register A is done as soon as the strobe is synchronized to the TCD domain. Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.
RESTART_STROBE	Restart Strobe. When ENABLED a restart of the TCD counter is executed as soon as this bit is synchronized to the TCD domain. Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.
SYNC_STROBE	Synchronize Strobe When ENABLED the double buffered registers will be loaded to the TCD domain as soon as this bit is synchronized to the TCD domain. Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.
SYNC_EOC	Synchronize end of TCD cycle strobe. When ENABLED the double buffered registers will be loaded to the TCD domain at the end of the next TCD cycle. Writing to this bit only has effect if there is no ongoing synchronization of a command. See also CMDRDY bit in TCD.STATUS.
EVENT**A**_CONFIG EVENT**B**_CONFIG	Event A|B configuration. When the Input Capture Noise canceler is activated (FILTERON), the Event input is filtered. The filter function requires four successive equal valued samples of the Retrigger pin for changing its output. The Input Capture is therefore delayed by four clock cycles when the noise canceler is enabled. When the Asynchronous Event is enabled (ASYNCON), the Event input will qualify the output directly. Possible values : - NEITHER : Neither Filter nor Asynchronous Event is enabled. - FILTER_ON : Input Capture Noise Cancellation Filter enabled. - ASYNC_ON : Asynchronous Event output qualification enabled.

EVENT**A**_EDGE EVENT**B**_EDGE	Edge Selection This bit is used to select the active edge or level of the event interrupt - FALL_LOW : The falling edge or low level of the Event input generates Retrigger or Fault action. - RISE_HIGH : The rising edge or high level of the Event input generates Retrigger or Fault action.
EVENT**A**_ACTION EVENT**B**_ACTION	Event Action. This bit enables Capture on Event input. By default, the input will trigger a Fault, depending on the Input x register input mode (TCD.INPUTx). It is also possible to trigger a Capture on the Event input. Possible values : - FAULT : FAULT Event triggers a Fault. - CAPTURE : CAPTURE Event triggers a Fault and Capture.
EVENT**A**_TRIG EVENT**B**_TRIG	Trigger Event Input Enable This options enabled or disables the Event as a trigger to input A|B
TRIG**A**_INT TRIG**B**_INT OVF_INT	Interrupts can be enabled/disabled by using the ENABLE and DISABLE statements. Using the timer interrupt names you can also enable/disable them using the CONFIG statement. - ENABLED : the interrupt will be enabled - DISABLED : the interrupt is disabled TRIGA_INT : event is executed when trigger input A is received TRIGB_INT : event is executed when trigger input B is received OVF_INT : event is executed when the timer overflows or restarts
INP_MODE**A** INP_MODE**B**	Input mode options. Possible values : - NONE : input has no action - JMPWAIT : stop output, jump to opposite compare cycle and wait - EXECWAIT : stop output, execute opposite compare cycle and wait - EXECFAULT : stop output , execute opposite compare cycle while fault active - FREQ : stop all outputs, maintain frequency - EXECDT : stop all outputs, execute dead time while fault active - WAIT : stop all outputs, jump to next compare cycle and wait - WAITSW : stop all outputs, wait for software action - EDGETRIG : stop all output on edge, jump to next compare cycle - EDGETRIGFREQ : stop output on edge, maintain frequency - LVLTRIGFREQ : stop output at level, maintain frequency
CMP**A**EN CMP**B**EN CMP**C**EN CMP**D**EN	Compare Enable output pin. ENABLED : The compare output pin is enabled. DISABLED : The compare output pin disabled (no output) At reset the settings are loaded from FUSE.TCDFG. So configuration should not be required.
CMP**A**	Compare value. These bits set the default state from Reset or when a input

CMP**B** CMP**C** CMP**D**	event triggers a fault causing changes to the output. At reset the content is kept and during the reset sequence loaded from the TCD configuration fuse so configuration should not be required. ENABLED : set the bit to 1. DISABLED : reset the bit to 0.
DLY_PRESCALER	Delay Prescaler. This option controls the prescaler setting for the blanking or output event delay Possible prescaler values : 1,2,4 and 8
DLY_TRIGGER	Delay Trigger. These option control what should trigger the blanking or output event delay. - CMPASET : CMPASET triggers delay - CMPACLR : CMPACLR triggers delay - CMPBSET : CMPBSET triggers delay - CMPBCLR : CMPASET troggers delay (end of cycle)
DLY_SEL	Delay Select. This option controls what function should be used by the delay trigger the blanking or output event delay - OFF : delay function not used - INBLANK : input blanking enabled - EVENT : event delay enabled
DLY_VAL	Delay value. This value specifies the blanking output event delay time or event output synchronization in number of prescaled TCD cycles. This is an 8 bit value from 0-255. The default is 0.
DIT_CTRL	Dither Control. This option configures which compare register is using the dither function. - ONTIMEB : On-time ramp B - ONTIMEAB : On-time ramp A and B - DEADTIMEB : Dead-time ramp B - DEADTIMEAB : Dead-time ramp A and B
DIT_VAL	Dither value. These bits configure the fractional adjustment of the on-time or off-time according to Dither Selection bits (DITHERSEL) in the Dither Control register (TCD.DITCTRL). The DITHER value is added to a 4-bit accumulator at the end of each TCD cycle. When the accumulator overflows the frequency adjustment will occur. The DITHER bits are doubled buffered so the new value is copied in at an update condition. The value has a range from 0-15. Default value is 0.

See also
CONFIG TCA0 1084 , CONFIG TCB 1087

Example

7.21.86 CONFIG TCXX

Action
Configures the Xmega TIMER.

Syntax

CONFIG TCxx = wg , PRESCALE=pre, COMPAREA=ca, COMPAREB=cb, COMPAREC=cc, COMPARED=cd, EVENT_SOURCE= event, EVENT_ACTION=act, EVENT_DELAY=ed, RESOLUTION=res

Remarks

Depending on the Xmega processor of your choice, there are one or more timers. The Xmega uses the name of the port as part of the name. The first port that has a timer is portC. The first timer is named TC**C0**. Most timer ports have 2 timers. The next timer is named TC**C1**. Xmega timers are 16 bit but can be cascaded to 32 bit timers or be set to 8 bit mode.
The possible timer names are : TCC0, TCC1, TCD0, TCD1, TCE0, TCE1, TCF0 and TCF1.

WG	This options sets the Timer and/or Wave Generation mode. Possible values : - NORMAL, no wave generation - FREQ , frequency generation - PWM , pulse width modulation single slope - PWM_TOP, pwm dual slope - PWM_BOT, pwm dual slope - PWM_TOPBOT, pwm dual slope - A value between 0-7 will load the mode. See table 2. **- TIMER2. This will set the timer into byte mode.**
PRESCALE or *CLOCKSEL*	The prescaler can divide the system clock that is applied to the timer. Possible values : - 1 , 2, 4, 8, 64, 256, 1024 - OFF, timer is disabled - E0, E1, E2, E3, E4, E5, E6, E7 . Event channel 0-7 - value between 0-15. This will write the value to the CTRLA register. In the XMEGA, CLOCKSEL (clock selection) describes the parameter better than PRESCALE because of the additional options. But the coded explorer will use PRESCALE from the DAT files. In order not to break code the CLOCKSEL name will be dropped in a future version.
COMPAREx	Where x is A, B, C, or D. This is the COMPARE or CAPTURE register setup. You may use either COMPARE or CAPTURE since the same registers are used. Each COMPARE/CAPTURE pin must be enabled if the input/ output pin is used. By default they are disabled. Each TCx0 timer has 4 compare registers/pins. The TCx1 timer has two capture registers/pins. Possible values : ENABLED : this will enable the capture/compare register DISABLED : this will disable the capture/compare register 0 : this will set the logic level of the compare output pin to 0. 1 : this will set the logic level of the compare output pin to 1. In FREQ and PWM modes the compare pins will be set to output mode. In CAPTURE mode, the capture pin will be set to input mode. **NOTE : NOT valid in TIMER2 mode.**

COMPAREx TIMER2 mode	In TIMER2 mode, there are 8 compare outputs. They have the names : CAPTURE**A**L , CAPTURE**A**H ,CAPTURE**B**L , CAPTURE**B**H, CAPTURE**C**L, CAPTURE**C**H,CAPTURE**D**L and CAPTURE**D**H. The last character indicates the Low or High byte. Each COMPARE/CAPTURE pin must be enabled if the input/output pin is used. By default they are disabled. Possible values : ENABLED : this will enable the capture/compare ouput pin DISABLED : this will disable the capture/compare output pin 0 : this will set the logic level of the compare output pin to 0. 1 : this will set the logic level of the compare output pin to 1.
EVENT_SOURCE	The event channel source. Possible values : - OFF (default) - E0-E7 - A value between 0-15 **NOTE : NOT valid in TIMER2 mode.**
EVENT_ACTION	The event action the timer will perform. Possible values : - OFF - CAPTURE, input capture - UPDOWN, external controlled up/down count - QDEC, quadrature decode - RESTART , restart waveform period - FREQ, frequency capture - PWC, pulse width capture **NOTE : NOT valid in TIMER2 mode.**
EVENT_DELAY	Enabled, or disabled(default). When this bit is set, the selected event source is delayed by one peripheral clock cycle. This feature is intended for 32-bit input capture operation. Adding the event delay is necessary for compensating for the carry propagation delay that is inserted when cascading two counters via the Event System. **NOTE : NOT valid in TIMER2 mode.**
RESOLUTION	Valid options : NORMAL, BYTE, SPLIT. Timer resolution is 16 by default (NORMAL). A value of BYTE will set the timer to 8 bit resolution. SPLIT is reserved for future use. (cascading 32 bit timers). When WG mode TIMER2 is chosen, the timer will be set into BYTE mode automatically.

Table 2.

Value	Mode	TOP	UPDATE	EVENT
0	NORMAL	PER	TOP	TOP
1	FREQ	CCA	TOP	TOP
2	reserved			
3	PWM, single slope	PER	BOTTOM	BOTTOM
4	reserved			
5	PWM, dual slope	PER	BOTTOM	TOP
6	PWM, dual slope	PER	BOTTOM	TOP and BOTTOM
7	PWM, dual slope	PER	BOTTOM	BOTTOM

A CONFIG TCxx statement will update the timer control registers immediately. A pre

scale value other than OFF will also START the timer at once.

 CONFIG TCxx statement must be placed in the main code. Or you may include it in the main code using $INCLUDE.
- you can use CONFIG TCxx multiple times
- do not use CONFIG TCxx in a SUB/FUNCTION in combination with SUBMODE=NEW.

See Also
START , STOP

Example 1:
```
'Counter/Timer D1 is used for overflow counter at --> 400ms
'32MHz/256 = 125000
'32MHz/256 = 125000 --> 125000/2.5 = 50000  '400ms
'Or in other words: 50000 counts at 125Khz (8µSec per tick) = 50000 *
8µSec = 400mSec = 0.4 sec
Config Tcd1 = Normal , Prescale = 256
Tcd1_per = 50000
```

You could use the overflow for example now as an interrupt (every 400ms) or feed it to the Event System (every 400ms).

Example 2:
The following example configuration counts the incoming events from Event Channel 7. You can use the **Tcd0_cnt** register to analyze the number of events.

```
Config Tcd0 = Normal , Prescale = E7 , Event_source = 7 , Event_action =
Capture
```

Example 3:
```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                   xm128-TIMER-S1.bas
'  This sample demonstrates the TIMER sample 1 from AVR1501
'  This sample uses TIMER TCD0 since TCC0 isused for the UART
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 64
$framesize = 64
'include the following lib and code, the routines will be replaced since
they are a workaround

'First Enable The Osc Of Your Choice , make sure to enable 32 KHz clock
or use an external 32 KHz clock
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

'connect portE bit 0 and 1 to some LED
Config Porte = Output
```

```
'config timer to normal mode
Config Tcd0 = Normal , Prescale = 64
Tcd0_per = &H30                                          ' period
register

Do
  If Inkey() <> 0 Then
     Tcd0_per = Tcd0_per + 100          ' increase period
     Print "period:" ; Tcd0_per         ' you will see that a larger
PERIOD value will cause the TIMER to
                                        ' overflow later and this
generating a bigger delay
  End If
  Bitwait Tcd0_intflags.0 , Set         ' wait for overflow
  Tcd0_intflags.0 = 1                   ' clear flag by writing 1
  Toggle Porte                          ' toggle led
Loop
```

7.21.87 CONFIG TCPIP

Action

Configures the TCP/IP chip's from WIZNET (http://www.wiznet.co.kr/).
This chip's can be found on various modules and shields but the **Config** Tcpip is always depending on the WIZNET chip.
Supported chip's are W3100A, W5100, W5200 and W5300.

Syntax W3100A

CONFIG TCPIP = int , MAC = mac , IP = ip, SUBMASK = mask, GATEWAY = gateway, LOCALPORT= port, TX= tx, RX= rx , NOINIT= 0|1 [, TWI=address] [, Clock = speed] [, baseaddress = address] [,TimeOut=tmOut] [,CHIP=W3100A]

Syntax W5100

CONFIG TCPIP = int , MAC = mac , IP = ip, SUBMASK = mask, GATEWAY = gateway, LOCALPORT= port, TX= tx, RX= rx , NOINIT= 0|1 [, baseaddress = address] [,TimeOut=tmOut] [,CHIP=5100] [,SPI=spi] [,INT=imsg] [,CS=cs] [, NOUDP=noudp]

Syntax W5200

CONFIG TCPIP = int , MAC = mac , IP = ip, SUBMASK = mask, GATEWAY = gateway, LOCALPORT= port, NOINIT= 0|1 [,TimeOut=tmOut] [,CHIP=W5200] [, SPI=spi] [,INT=imsg] [,CS=cs] [,NOUDP=noudp] [TXn= tx] [, RXn= rx]

Syntax W5300

CONFIG TCPIP = int , MAC = mac , IP = ip, SUBMASK = mask, GATEWAY = gateway, LOCALPORT= port, NOINIT= 0|1 [, baseaddress = address] [, TimeOut=tmOut] [,CHIP=W5300] [,INT=imsg] [,NOUDP=noudp] [align=align] [TX**n**= tx] [, RX**n**= rx] [SOCKMEM=sockmem]

Syntax W5500

CONFIG TCPIP = NOINT , MAC = mac , IP = ip, SUBMASK = mask, GATEWAY = gateway, LOCALPORT= port, NOINIT= 0|1 [,TimeOut=tmOut] [,CHIP=W5500] [, SPI=spi] [,INT=imsg] [,CS=cs] [,NOUDP=noudp] [TXn= tx] [, RXn= rx]

Remarks

Int	The interrupt to use such as INT0, INT1 or INTn. For the Easy TCP/IP PCB, use INT0. **W5100,W5200,W5300** also support the NOINT option. This option will not use any interrupt. The internal status array *s_status* will not be created and is not available either. When you do use interrupts, the *s_status* array will contain the status of each socket. s_status(1) will contain the status of the first socket. In interrupt mode you can also get a notification that a socket was updated when you use the INT=1 option. Using interrupts does use more code and resources. **W5500** only supports the NOINT option.
MAC	The MAC address you want to assign to the ethernet chip. The MAC address is a unique number that identifies your chip. You must use a different address for every ethernet chip in your network. Example : 00.00.12.34.56.78 You need to specify 6 bytes that must be separated by dots. The bytes must be specified in decimal notation. For some networks it is important that the MAC address starts with a zero. So we advise to start the MAC address with a 0.
IP	The IP address you want to assign to the chip. The IP address must be unique for every ethernet chip in your network. When you have a LAN, 192.168.0.10 can be used. 192.168.0.x is used for LAN's since the address is not an assigned internet address. The same applies to 10.0.0.0.
SUBMASK	The sub mask you want to assign to the ethernet chip. The sub mask is in most cases 255.255.255.0
GATEWAY	This is the gateway address of the ethernet chip. The gateway connects your LAN with the internet. The gateway address you can determine with the **IPCONFIG** command at the command prompt : C:\>ipconfig Windows 2000 IP Configuration Ethernet adapter Local Area Connection 2: Connection-specific DNS Suffix . : IP Address. . . . . : 192.168.0.3 Subnet Mask . . . . . : 255.255.255.0 Default Gateway . . . . . : 192.168.0.1 Use 192.168.0.1 in this case.
LOCALPORT	A word value that is assigned to the LOCAL_PORT internal variable. See also Getsocket 1553. As a default you can assign a value of 5000.
TX	**W3100A,W5100**

<table>
<tr><td></td><td>A byte which specifies the transmit buffer size of the W3100A/W5100. The W3100A/W5100 has 4 sockets.

A value of 00 will assign 1024 bytes, a value of 01 will assign 2048 bytes. A value of 10 will assign 4096 bytes and a value of 11 will assign 8192 bytes.

This is binary notation. And the Most Significant bits (bit 6 and 7) specify the size of socket 3.

For example, you want to assign 2048 bytes to each socket for transmission : TX = &B01010101

Since the transmission buffer size may be 8KB in total, you can split them up in 4 parts of 2048 bytes : 01.

When you want to use 1 socket with 8KB size, you would use : TX = &B11. You can use only 1 socket in that case : socket 0.

Consult the W3100A/W5100 pdf for more info.</td></tr>
<tr><td>RX</td><td>W3100A,W5100
A byte which specifies the receive buffer size of the W3100A/W5100. The W3100A/W5100 has 4 sockets.

A value of 00 will assign 1024 bytes, a value of 01 will assign 2048 bytes. A value of 10 will assign 4096 bytes and a value of 11 will assign 8192 bytes.

This is binary notation. And the Most significant bits specify the size of socket 3.

For example, you want to assign 2048 bytes to each socket for reception : RX = &B01010101

Since the receive buffer size may be 8KB in total, you can split them up in 4 parts of 2048 bytes : 01.

When you want to use 1 socket with 8KB size, you would use : RX = &B11. You can use only 1 socket in that case : socket 0.

Consult the W3100A/W5100 pdf for more info.</td></tr>
<tr><td>TXn</td><td>W5200, W5300,w5500
A constant which specifies the socket size of the transmit buffer of socket n. N is in range of 1-8.
This notation is only used by W5200 and W5300 where you can define the size in KB.
By default the W5200 sockets are 2 KB each and the W5300 are 8 KB each.
The following values are possible :
<table>
<tr><th>Value</th><th>W5200,W5500</th><th>W5300</th></tr>
<tr><td>1</td><td>1 KB</td><td>1 KB</td></tr>
<tr><td>2</td><td>2 KB default</td><td>2 KB</td></tr>
<tr><td>4</td><td>4 KB</td><td>4 KB</td></tr>
<tr><td>8</td><td>8 KB</td><td>8 KB default</td></tr>
<tr><td>15</td><td>16 KB</td><td>15 KB</td></tr>
<tr><td>any other value between 1-64</td><td>invalid</td><td>size in KB</td></tr>
</table></td></tr>
</table>

	The total amount may not exceed the available socket memory. For example the W5200 can use 8x2=16 KB of TX memory. But you can also use 2 sockets with 8 KB each.
RXn	**W5200,W5300,W5500** This will set the socket receive buffer size similar as described above for TXn.
sockmem	**W5300** The w5300 allows to configure how much of the memory is used for the transmit and receive buffers. The default is &HFF00 which will split the memory in even parts. See the W5300 datasheet for more details.
Noinit	Make this option 1 when you want to configure the TCP, MAC, Subnetmask and GateWay dynamic. Noinit will only make some important settings and you need to use SETTCP 1559 in order to finish the setup.
TWI	**W3100A only** The slave address of the W3100A/NM7010. When you specify TWI, your micro must have a TWI interface such as Mega128, Mega88, Mega32. TWI is only supported by the W3100A.
Clock	**W3100A only** The clock frequency to use with the TWI interface. Use this in combination with the TWI option.
Baseaddress	**W3100A,W5100,W5300** An optional value for the chip select of the ethernet chip. This is default &H8000 when not specified. When you create your own board, you can override it. See also: Adding XRAM with External Memory Interface 251
TimeOut	**W3100A** You can specify an optional timeout when sending UDP data. The Wiznet API does wait for the CSEND status. But it means that it will block your application. In such cases, you can use the timeout value. The timeout constant is a counter which decreases every time the status is checked. When it reaches 0, it will get out of the loop. Thus a higher value will result in a longer delay. Notice that it has nothing to do with the chip timeout registers/values. Without the software timeout, the chip will also time out. W5100,W5200 and W5300 have a time out option in the hardware.
CHIP	The wiznet chip you use. By default this is W3100. Specify W5100 for the W5100 chip. This chip has 4 sockets and a SPI interface instead of an I2C/TWI interface. Specify W5200 for the W5200 chip. This chip has 8 sockets and only a SPI interface. This SPI interface has a high speed. Specify W5300 for the W5300 chip. This chip has 8 sockets and can work in bus mode only. Specify W5500 for the W5500 chip. This chip has 8 sockets and only a SPI interface. This SPI interface supports high speed and blockmode.
SPI	This option is intended to be used with the W5100/W5200 chips. When you want to use the W5100 or W5200 in SPI mode, make this parameter value 1. When you do not specify his parameter, or set it to 0, the external memory mode will be used. For the Xmega you can specify SPIC, SPID, SPIE of SPIF. For normal AVR with multiple SPI such as M328PB you can specify SPI1

	When using SPI, you must configure it before configuring the TCPIP. SPI must be configured in mode 0.
	Example :
	
	`Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes , Polarity = Low , Phase = 0 , Clockrate = 4 , Noss = 0`
	
	`'Init the spi pins`
	`Spiinit`
	`Config Tcpip = Noint , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 , Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 , Tx = $55 , Rx = $55 , Chip = W5100 , Spi = 1 , Cs = Portb.4`

imsg	In interrupt mode, you can get a notification about changed socket status such as new data arrived, or socket closed. Use INT=1 for this option. The library will call a routine named TCP_INT. So your code need to include this label or sub routine. You can test the s_status() array but you can also test the _tcp_intflags variable. This variable contains the flags from the IR register. You must dimension the variable _tcp_intflags if you want to use this option.
cs	This is an optional parameter used in combination with the SPI option. By default the compiler will use the standard SS pin for the SPI. But if you have multiple SPI slaves, or want to use a different pin to control the CS of the W5100/W5200, you can add this parameter. The name of a port pin is expected such as PORTB.4  You should use a normal port register. Do not use an extended address port like PORTL.
noudp	By default UDP variables PEERADDRESS, PEERPORT and PEERSIZE are created by the compiler. If you do not use any UDP statement, you can use NOUDP=1. This will save 8 bytes of memory.
align	The **W5300** has an align option. Align is ignored for all other chips. The align modes : **0** - this will disable alignment. This will add a header packet for TCP data with the size. You must use TCPREADHEADER to read the actual data size. Socketstat 1571 will not return the actual data size. After you have determined there is data in the receive buffer, you must use TCPREADHEADER to get the actual size. You may only use TCPREADHEADER once since it will read 2 bytes from the receive buffer. **1**- this will enable alignment. This will not add the header packet to TCP data. SocketStat will return the actual data size. You must not use TCPREADHEADER in this case. **2**- since using alignment caused some unexpected problems in tcp traffic, (see wiznet forum) there is also the smart and default option which makes tcp reading compatible to the other chips. When using mode 2, the mode 0 will be used, and socketstat will automatic read the buffer size packet in case there is data in the received buffer and this it will return the correct size. Since it will read from the receive buffer, you must empty the buffer with tcpread, after you have determined that there is data waiting. You must not call socketstat 1571 again before you have read all the pending data.

The CONFIG TCPIP statement may be used only once.
If you do use interrupts, you must enable them before you use CONFIG TCPIP. When using the NOINT option this is not required.
Configuring the ethernet chip will initialize the chip.

After the CONFIG TCPIP, you can already PING the chip!

 As all the samples show, the CONFIG TCPIP must be used in the main program. The CONFIG TCPIP should be used early as possible in your code. This is especially important for processors with multiple pages. (>64KB). The reason is that the configuration data is stored in flash and read with LPM instruction. LPM can only reach page 0.

W3100A

The TWI mode works only when your micro support the TWI mode. You need to have 4k7 pull up resistors.
MCS Electronics has a small adapter PCB and KIT available that can be connected easily to your microprocessor.
The TWI mode makes your PCB design much simpler. TWI is not as fast as bus mode. While you can use every supported TCP/IP function, it will run at a lower speed.

W5100

The W5100 is the successor of the W3100A. It is an improved chip without shadow registers. This means that less code is required to use the chip.
Because the W5100 has different constants compared to the W3100A, the constants are removed from the samples. The constants are automatically created with a value depending on the chip you use.

From the user perspective the W5100 library is almost the same as the W3100 library. But there are some differences.
- The peersize, peerport and peeraddress have a different order in the W5100. To avoid mistakes, the compiler will create these variables automatic in the proper order. The NOUDP=1 option can disable this feature if you do not use UDP.
- When reading UDP, you need to use the UDPREADHEADER 1585 statement to read the UDP header. After reading the header, the peersize, peerport and peeraddress variables are set. You then should use the peersize variable to determine the number of bytes to retrieve. You must read all these bytes.
- The W5100 has a command to disconnect the socket in TCP/IP mode. It is named SOCKETDISCONNECT 1570.
- The CLOSESOCKET statement has been renamed into SOCKETCLOSE 1564. You can use both names.
The MCS web shop offers the WIZ810MJ ethernet module and the TCPADB5100 adapter board.

W5200

The W5200 is a SPI only version of the W5100 so read the comment above about the W5100 first.
The W5200 chip has less pins and is smaller and simpler to use. It has 8 sockets instead of 4 and it has a faster SPI mode. One example where the W5200 is used is the Wiz820io module. See example below.

This Chip need specific reset times before you can use config TCPIP (see example below).
It has been reported that when the RETRY_TIME and RETRY_COUNT registers are altered, sending UDP data can have a variable delay the first time the data will actually be sent.

W5300

The W5300 is a bus mode **only** version of the W5100 so read the comment above about the W5100 first
The W5300 chip has a fast 8/16 bit bus and has 8 sockets with increased socket size.

See also the W5300 examples in: Adding XRAM with External Memory Interface 251 regarding base address.

W5500

The W5500 is a SPI only version of the W5100 so read the comment above about the W5100 first.
The W5500 chip has less pins and is smaller and simpler to use. It has 8 sockets instead of 4 and it has a faster SPI mode. It is similar to W5200.
For samples, use the W5200 samples and change CHIP to W5500.
The W5500 library has specific provision to be used in a boot loader.

WIZ810

REV 1.0 of the WIZ810 leaves the SPI_EN Pin floating (REV1.1 has an internal pulldown). When using REV1.0 in parallel mode, you will have to tie that pin to ground.

See also

GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, TCPREAD 1576, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , SOCKETDISCONNECT 1570 , SETTCP 1559 , UDPREAD 1582, UDPWRITE 1588, UDPWRITESTR 1589 , UDPREADHEADER 1585, TCPREADHEADER 1577 , TCPCHECKSUM 1574, SNTP 1562 , , GETTCPREGS 1555 , SETTCPREGS 1560

Syntax Example using W3100:

```
Config Tcpip = Int0 , Mac = 00.00.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx
= $55 , Rx = $55

Now use PING at the command line to send a ping:

PING 192.168.0.8
```

Or use the easytcp application to ping the chip.

Syntax Example using W5100

```
$regfile = "m88def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 80                                              ' default
use 32 for the hardware stack
$swstack = 128                                             ' default
use 10 for the SW stack
$framesize = 80                                            ' default
use 40 for the frame space
$lib "datetime.lbx"                                        ' this
```

```
example uses date time routines

Print "Init TCP"                                          ' display a
message
Enable Interrupts                                        ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int1 , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 , Tx
= $55 , Rx = $55 ,  Chip = W5100 , Spi = 1
Print "Init done"

Dim Var As Byte                                          ' for i2c
test
Dim Ip As Long                                           ' IP number
of time server
Dim Idx As Byte                                          ' socket
number
Dim Lsntp As Long                                        ' long SNTP
time

Print "SNTP demo"

'assign the IP number of a SNTP server
Ip = Maketcp(64.90.182.55 )                              ' assign IP
num NIST time.nist.gov  port 37
Print "Connecting to : " ; Ip2str(ip)

'we will use Dutch format
Config Date = Dmy , Separator = -

'we need to get a socket first
'note that for UDP we specify sock_dgram
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)             ' get socket
for UDP mode, specify port 5000
Print "Socket " ; Idx ; " " ; Idx

'UDP is a connection less protocol which means that you can not listen,
connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the addition
of the IP and PORT
'The SNTP uses port 37 which is fixed in the tcp asm code

Do
   Waitms 5000

   Lsntp = Sntp(idx , Ip)                                ' get time
from SNTP server
 '  Print Idx ; Lsntp
   'notice that it is not recommended to get the time every sec
   'the time server might ban your IP
   'it is better to sync once or to run your own SNTP server and update
that once a day

   'what happens is that IP number of timer server is send a diagram too
   'it will put the time into a variable lsntp and this is converted to
BASCOM date/time format
   'in case of a problem the variable is 0
   Print Date(lsntp) ; Spc(3) ; Time(lsntp)
Loop
```

Example for using W5200 Chip on a WIZ820io module with ATXMEGA:

Hardware connections:

```
WIZ820io [SCLK]  <-----> ATXMEGA128A1 PortC.7 [SCK]
WIZ820io [MOSI]  <-----> ATXMEGA128A1 PortC.5 [MOSI]
WIZ820io [MISO]  <-----> ATXMEGA128A1 PortC.6 [MISO]
WIZ820io [nSS]   <-----> ATXMEGA128A1 PortC.4 [SS]
WIZ820io [nReset]<-----> ATXMEGA128A1 PortC.2
WIZ820io [nINT]  <-----> ATXMEGA128A1 PortC.3
```

Because it is a SPI based communication interface to the W5200 you need to setup the SPI interface (SPI on Port C is used in this example):

```
Config Spic = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk2 ,
Data_order = Msb , Ss = Auto

Config Pinc.2 = Output
W5200_nreset Alias Portc.2
Set W5200_nreset

Config Pinc.3 = Input
W5200_nint Alias Portc.3
```

Reset the WIZ820io Module:

```
Reset W5200_nreset
Waitms 1
Set W5200_nreset
Waitms 150
```

Config TCP Syntax Example for WIZ820io (using SPI on Port C and Port.4 as Slave Select (Chip Select)):

```
Config Tcpip = Noint , _
           Mac = 0.11.22.33.44.55 , _
           Ip = 192.168.1.254 , _
           Submask = 255.255.255.0 , _
           Gateway = 192.168.1.1 , _
           Localport = 80 , _
           Chip = W5200 , _
           Spi = Spic , _
           Cs = Portc.4
```

Now use PING at the command line to send a ping:

```
PING 192.168.1.254
```

Example for using W5300 Chip:

```
Config Tcpip = Noint , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.253 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 ,
Chip = W5300 , Baseaddress = &HFC00
```

Now use PING at the command line to send a ping:

```
PING 192.168.1.253
```

See also the W5300 examples in: Adding XRAM with External Memory Interface 251 regarding base address.

Example for using W5500 Chip:

```
'-----------------------------------------------------------------------------------
'name                    : sntp_W5500.bas   RFC 2030
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : test SNTP() function
'micro                   : xMega128A1
'suited for demo         : no, needs library only included in the full
version
'commercial addon needed  : no
'-----------------------------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64                                   ' default use 32 for the
hardware stack
$swstack = 128                                  'default use 10 for the
SW stack
$framesize = 64                                 'default use 40 for the
frame space

'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
'configure UART
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8


Config Spie = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk32 ,
Data_order = Msb , Ss = Auto
 'SPI on Port E is used
 'portx.7 - SCK
 'portx.6 - MISO
 'portx.5 - MOSI
 'portx.4 - SS

Waitms 1000
Print "Init , set IP to 192.168.1.88"              ' display a message
Config Tcpip = Noint , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.88 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport =
1000 , Chip = W5500 , Spi = Spie , Cs = Porte.4
```

```
Print "Init Done"


$lib "datetime.lbx"                          'this example uses date
time routines


Dim Ip As Long                               ' IP number of time
server
Dim Idx As Byte                              ' socket number
Dim Lsntp As Long                              ' long SNTP time

Print "SNTP demo"

'assign the IP number of a SNTP server
Ip = Maketcp(129.6.15.30 )                       'assign IP num NIST
time.nist.gov  port 37
Print "Connecting to : " ; Ip2str(ip)


'we will use Dutch format
Config Date = Dmy , Separator = Minus


'we need to get a socket first
'note that for UDP we specify sock_dgram
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)              ' get socket for
UDP mode, specify port 5000
Print "Socket " ; Idx

'UDP is a connection less protocol which means that you can not
listen, connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the
addition of the IP and PORT
'The SNTP uses port 37 which is fixed in the tcp asm code


Do

  Waitms 5000
```

```
   Lsntp = Sntp(idx , Ip)                          ' get time from SNTP
server
   'notice that it is not recommended to get the time every sec
   'the time server might ban your IP
   'it is better to sync once or to run your own SNTP server and update
that once a day

   'what happens is that IP number of timer server is send a diagram
too
   'it will put the time into a variable lsntp and this is converted to
BASCOM date/time format
   'in case of a problem the variable is 0
   Print Date(lsntp) ; Spc(3) ; Time(lsntp)
Loop



End
```

7.21.88 CONFIG TIMER0

Action

Configure TIMER0.

Syntax

CONFIG TIMER0 = COUNTER , EDGE=RISING/FALLING , CLEAR_TIMER = 1|0 [, CONFIGURATION=NAME]
CONFIG TIMER0 = TIMER , PRESCALE= 1|8|64|256|1024 [,CONFIGURATION=NAME]
CONFIG TIMER2 = TIMER | PWM , ASYNC=ON |OFF,PRESCALE = 1 | 8 | 32 | 64 | 128 | 256 | 1024 ,COMPARE = CLEAR | SET | TOGGLE | DISCONNECT ,PWM = ON | OFF , COMPARE_PWM = CLEAR_UP| CLEAR_DOWN | DISCONNECT ,CLEAR_TIMER = 1|0 [, CONFIGURATION=NAME]

Remarks

TIMER0 is an 8 bit counter. See the hardware description of TIMER0.

When configured as a COUNTER:

EDGE	You can select whether the TIMER will count on the falling or rising edge.

When configured as a TIMER:

PRESCALE	The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter. Valid values are 1 , 8, 64, 256 or 1024

Note that some new AVR chips have different pre scale values. You can use these.

CONFIGURATION is optional. When you add configuration=mysetting, you can use this setting when you start the timer : START TIMER0 , mysetting
If you have multiple settings, you can start the timer with these different settings.

 Notice that the Help was written with the AT90S2313 and AT90S8515 timers in mind.

When you use the CONFIG TIMER0 statement, the mode is stored by the compiler and the TCCRO register is set.
When you use the STOP TIMER0 statement, the TIMER is stopped.
When you use the START TIMER0 statement, the TIMER TCCR0 register is loaded with the last value that was configured with the CONFIG TIMER0 statement.

So before using the START 1538 and STOP 1544 TIMER0 statements, use the CONFIG statement first.

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : timer0.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows how to use TIMER0 related statements
'micro                    : 90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


'First you must configure the timer to operate as a counter or as a
timer
'Lets configure it as a COUNTER now
'You must also specify if it will count on a rising or falling edge

Config Timer0 = Counter , Edge = Rising
'Config Timer0 = Counter , Edge = falling
'unremark the line aboven to use timer0 to count on falling edge

'To get/set the value from the timer access the timer/counter register
'lets reset it to 0
Tcnt0 = 0
```

```
Do
  Print Tcnt0
Loop Until Tcnt0 >= 10
'when 10 pulses are count the loop is exited
'or use the special variable TIMER0
Timer0 = 0


'Now configire it as a TIMER
'The TIMER can have the systemclock as an input or the systemclock
divided
'by 8,64,256 or 1024
'The prescale parameter excepts 1,8,64,256 or 1024
Config Timer0 = Timer , Prescale = 1

'The TIMER is started now automaticly
'You can STOP the timer with the following statement :
Stop Timer0

'Now the timer is stopped
'To START it again in the last configured mode, use :
Start Timer0

'Again you can access the value with the tcnt0 register
Print Tcnt0
'or
Print Timer0
'when the timer overflows, a flag named TOV0 in register TIFR is set
'You can use this to execute an ISR
'To reset the flag manual in non ISR mode you must write a 1 to the bit
position
'in TIFR:
Set Tifr.1



'The following code shows how to use the TIMER0 in interrupt mode
'The code is block remarked with '(  en ')

'(

'Configute the timer to use the clock divided by 1024
Config Timer0 = Timer , Prescale = 1024

'Define the ISR handler
On Ovf0 Tim0_isr
'you may also use TIMER0 for OVF0, it is the same

Enable Timer0                                                ' enable the
timer interrupt
Enable Interrupts                                            'allow
interrupts to occur
Do
   'your program goes here
Loop

'the following code is executed when the timer rolls over
Tim0_isr:
  Print "*";
Return

')
End
```

7.21.89 CONFIG TIMER1

Action
Configure TIMER1.

Syntax
CONFIG TIMER1 = COUNTER | TIMER | PWM ,
EDGE=RISING | FALLING , PRESCALE= 1|8|64|256|1024 ,
NOISE_CANCEL=0 |1, CAPTURE_EDGE = RISING | FALLING ,
CLEAR_TIMER = 1|0,
COMPARE_A = CLEAR | SET | TOGGLE | DISCONNECT ,
COMPARE_B = CLEAR | SET | TOGGLE | DISCONNECT ,
PWM = 8 | 9 10 ,
COMPARE_A_PWM = CLEAR_UP| CLEAR_DOWN | DISCONNECT
COMPARE_B_PWM = CLEAR_UP| CLEAR_DOWN | DISCONNECT
[,CONFIGURATION=NAME]

Remarks
The TIMER1 is a 16 bit counter. See the hardware description of TIMER1.
It depends on the chip if COMPARE_B is available or not.
Some chips even have a COMARE_C.

The syntax shown above must be on one line. Not all the options need to be selected.

Here is the effect of the various options.

EDGE	You can select whether the TIMER will count on the falling or rising edge. Only for COUNTER mode.
CAPTURE_ EDGE	You can choose to capture the TIMER registers to the INPUT CAPTURE registers With the CAPTURE_EDGE = FALLING/RISING, you can specify to capture on the falling or rising edge of pin ICP
NOISE_ CANCELING	To allow noise canceling you can provide a value of 1.
PRESCALE	The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter. Valid values are 1 , 8, 64, 256 or 1024 PRESCALE can't be used in COUNTER mode.

The TIMER1 also has two compare registers A and B

When the timer value matches a compare register, an action can be performed

COMPARE_ A	The action can be: SET will set the OC1X pin CLEAR will clear the OC1X pin TOGGLE will toggle the OC1X pin DISCONNECT will disconnect the TIMER from output pin OC1X

And the TIMER can be used in PWM mode.

You have the choice between 8, 9 or 10 bit PWM mode

Also you can specify if the counter must count UP or down after a match to the compare registers
Note that there are two compare registers A and B

PWM	Can be 8, 9 or 10.
COMPARE_A_PWM	PWM compare mode. Can be CLEAR_UP or CLEAR_DOWN

Using COMPARE_A, COMPARE_B, COMPARE_A_PWM or COMPARE_B_PWM will set the corresponding pin for output. When this is not desired you can use the alternative NO_OUTPUT version that will not alter the output pin.

For example : COMPARE_A_NO_OUTPUT , COMPARE_A_PWM NO_OUTPUT

CONFIGURATION is optional. When you add configuration=mysetting, you can use this setting when you start the timer : START TIMER0 , mysetting
If you have multiple settings, you can start the timer with these different settings.

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : timer1.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : show using Timer1
'micro                    : 90S8515
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "8515def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


Dim W As Word

'The TIMER1 is a versatile 16 bit TIMER.
'This example shows how to configure the TIMER

'First like TIMER0 , it can be set to act as a TIMER or COUNTER
'Lets configure it as a TIMER that means that it will count and that
'the input is provided by the internal clock.
'The internal clock can be divided by 1,8,64,256 or 1024
Config Timer1 = Timer , Prescale = 1024


'You can read or write to the timer with the COUNTER1 or TIMER1 variable
```

```
W = Timer1
Timer1 = W


'To use it as a COUNTER, you can choose on which edge it is trigereed
Config Timer1 = Counter , Edge = Falling
'Config Timer1 = Counter , Edge = Rising

'Also you can choose to capture the TIMER registers to the INPUT CAPTURE
registers
'With the CAPTURE EDGE = , you can specify to capture on the falling or
rising edge of
'pin ICP
Config Timer1 = Counter , Edge = Falling , Capture_Edge = Falling
'Config Timer1 = Counter , Edge = Falling , Capture Edge = Rising

'To allow noise canceling you can also provide :
Config Timer1 = Counter , Edge = Falling , Capture_Edge = Falling ,
Noise_Cancel = 1

'to read the input capture register :
W = Capture1
'to write to the capture register :
Capture1 = W




'The TIMER also has two compare registers A and B
'When the timer value matches a compare register, an action can be
performed
Config Timer1 = Counter , Edge = Falling , Compare_A = Set , Compare_B =
 Toggle , Clear_Timer = 1
'SET , will set the OC1X pin
'CLEAR, will clear the OC1X pin
'TOGGLE, will toggle the OC1X pin
'DISCONNECT, will disconnect the TIMER from output pin OC1X
'CLEAR TIMER will clear the timer on a compare A match

'To read write the compare registers, you can use the COMPARE1A and
COMPARE1B variables
Compare1a = W
W = Compare1a


'And the TIMER can be used in PWM mode
'You have the choice between 8,9 or 10 bit PWM mode
'Also you can specify if the counter must count UP or down after a match
'to the compare registers
'Note that there are two compare registers A and B
Config Timer1 = Pwm , Pwm = 8 , Compare_A_Pwm = Clear_Up , Compare_B_Pwm
= Clear_Down , Prescale = 1

'to set the PWM registers, just assign a value to the compare A and B
registers
Compare1a = 100
Compare1b = 200

'Or for better reading :
Pwm1a = 100
Pwm1b = 200
End
```

7.21.90 CONFIG TIMER2

Action
Configure TIMER2.

Syntax for the 8535
CONFIG TIMER2 = TIMER | PWM , ASYNC=ON |OFF,
PRESCALE = 1 | 8 | 32 | 64 | 128 | 256 | 1024 ,
COMPARE = CLEAR | SET | TOGGLE | DISCONNECT ,
PWM = ON | OFF ,
COMPARE_PWM = CLEAR_UP| CLEAR_DOWN | DISCONNECT ,
CLEAR_TIMER = 1|0
[,CONFIGURATION=NAME]

Syntax for the M103
CONFIG TIMER2 = COUNTER| TIMER | PWM ,
EDGE= FALLING |RISING,
PRESCALE = 1 | 8 | 64 | 256 | 1024 ,
COMPARE = CLEAR | SET | TOGGLE | DISCONNECT ,
PWM = ON | OFF ,
COMPARE_PWM = CLEAR UP| CLEAR DOWN | DISCONNECT ,
CLEAR _TIMER = 1|0
[,CONFIGURATION=NAME]

Remarks
The TIMER2 is an 8 bit counter.
It depends on the chip if it can work as a counter or not.
The syntax shown above must be on one line. Not all the options need to be selected.
Some chips support multiple COMPARE outputs. Use COMPARE_A, COMPARE_B, COMPARE_C , etc.

Here is the effect of the various options.

EDGE	You can select whether the TIMER will count on the falling or rising edge. Only for COUNTER mode.

PRESCALE	The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter. Valid values are 1 , 8, 64, 256 or 1024 or 1 , 8, 32 , 64 , 256 or 1024 for the M103 Prescale can not be used in COUNTER mode.

The TIMER2 also has a compare registers

When the timer value matches a compare register, an action can be performed

COMPARE	The action can be: SET will set the OC2 pin

	CLEAR will clear the OC2 pin TOGGLE will toggle the OC2 pin DISCONNECT will disconnect the TIMER from output pin OC2

And the TIMER can be used in 8 bit PWM mode

You can specify if the counter must count UP or down after a match to the compare registers

COMPARE PWM	PWM compare mode. Can be CLEAR_UP or CLEAR_DOWN

CONFIGURATION is optional. When you add configuration=mysetting, you can use this setting when you start the timer : START TIMER0 , mysetting
If you have multiple settings, you can start the timer with these different settings.

Example

```
Dim W As Byte
Config Timer2 = Timer , ASYNC = 1 , Prescale = 128
On TIMER2 Myisr
ENABLE INTERRUPTS
ENABLE TIMER2
DO

LOOP


MYISR:
'get here every second with a 32768 Hz xtal
RETURN


'You can read or write to the timer with the COUNTER2 or TIMER2 variable
W = Timer2
Timer2 = W
```

7.21.91 CONFIG TWI, TWIx

Action

Configure the TWI (two wire serial interface) when using hardware I2C/TWI.

Syntax

CONFIG TWI = clockspeed
CONFIG TWI1 = clockspeed

Syntax XMEGA

CONFIG TWIC | TWID | TWIE | TWIF = clockspeed

(Config TWI and TWI1 is for ATMEGA and Config TWI**X** is for ATXMEGA chips)

Syntax XTINY

CONFIG TWI|TWI0|TWI1 = clockspeed

The XTINY uses TWI0. TWI and TWI0 are similar and can be exchanged. For devices with an additional TWI interface you can use TWI1.

Remarks

clockspeed	The desired clock frequency for SCL

CONFIG TWI will set TWSR pre scaler bits 0 and 1, and TWBR depending on the used $CRYSTAL 625 frequency and the desired SCL clock speed.
Typical you need a speed of 400 KHz. Some devices will work on 100 KHz as well.

When TWI is used in SLAVE mode, you need to have a faster clock speed as the master.

 There is no dynamic channel support for I2C

 To use the hardware I2C routines and not the Software I2C routines you need to use the `$lib "i2c_twi.lbx"`! (NOT FOR XMEGA/XTINY)

XMEGA

The XMEGA can contain up to 4 TWI units. When not specifying TWIC, TWID, TWIE or TWIF, the **TWIC** will be used as the default.
Because the XMEGA can contains multiple TWI busses, a channel identifier **MUST** be used when addressing TWID,TWIE or TWIF.
This means that your normal I2C code is fully compatible but only with TWIC. Thus omitting the channel identifiers, will automatically use TWIC.

You MUST dimension a variable named **TWI_START** as a byte. It is used by the xmega TWI library code. Without it, you will get an error.

There are 2 manuals available from ATMEL for every ATXMEGA Chip

1. One Family Manual like for example for a ATXMEGA128**A**1 it is Atmel AVR XMEGA **A** Manual
2. Another Manual for the single chips like for example for an ATXMEGA128A1 it is the ATxmega64A1/128A1/192A1/256A1/384A1 Manual. In this Manual you find for example the Alternate Pin Functions. So you can find which Pin on Port C is the SDA and SCL Pin when you want to use the I2C/TWI Interface of this Port.

 It is important that you specify the proper crystal frequency. Otherwise it will result in a wrong TWI clock frequency.

XTINY

The XTINY can contain up to 2 TWI units.
Because the XTINY can contains multiple TWI busses, a channel identifier MUST be used when addressing TWI1 or up.
This means that your normal I2C code is fully compatible but only with TWI/TWI0.
Thus omitting the channel identifiers, will automatically use TWI0.

You MUST dimension a variable named **TWI_START** as a byte. It is used by the xtiny

TWI library code. Without it, you will get an error.

Some processors support multiple TWI interfaces like the MEGA328PB. Use CONFIG TWI1 to configure the second TWI named TWI1. The first TWI which is named TWI0 is referred to as TWI.

See also

$CRYSTAL 625 , OPEN 1386, Using the I2C protocol 297, I2CINIT 1300

Example using Hardware I2C Pin's over Library: i2c_twi.lbx

```
'-----------------------------------------------------------------------
' (c) 1995-2025 MCS Electronics
' This demo shows an example of the TWI
' Not all AVR chips have TWI (hardware I2C)
'-----------------------------------------------------------------------
-

'The chip will work in TWI/I2C master mode
'Connected is a PCF8574A 8-bits port extender


$regfile="M8def.dat"' the used chip
$crystal= 4000000 ' frequency used
$baud = 19200                                               ' baud rate
$hwstack = 40
$swstack = 30
$framesize = 40


$lib "i2c_twi.lbx"                                          ' we do not
use software emulated I2C but the TWI
Config Scl = Portc.5                                        ' we need to
provide the SCL pin name
Config Sda = Portc.4                                        ' we need to
provide the SDA pin name
I2cinit                                                     ' we need to
set the pins in the proper state

'On the Mega8, On the PCF8574A
'scl=PC5 , pin 28 pin 14
'sda=PC4 , pin 27 pin 15

Config Twi = 100000                                         ' wanted
clock frequency when using $lib "i2c_twi.lbx"
'will set TWBR and TWSR
'Twbr = 12 'bit rate register
'Twsr = 0 'pre scaler bits

Dim B As Byte , X As Byte
Print "TWI master"

Do
   Incr B                                                   ' increase
value
   I2csend &B01110000 , B                                   ' send the
value
```

```
   Print "Error : " ; Err                                    ' show error
status
   I2creceive &B01110000 , X                                 ' get a byte
   Print X ; " " ; Err                                       ' show error
   Waitms 500                                                ' wait a bit
Loop
End
```

XMEGA SAMPLE

```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                       xm128-TWI.bas
'  This sample demonstrates the Xmega128A1 TWI
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


Dim S As String * 20

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Dim N As String * 16 , B As Byte
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Config Input1 = Cr , Echo = Crlf                             ' CR is used
for input, we echo back CR and LF

Open "COM1:" For Binary As #1
'       ^^^^ change from COM1-COM8

Print #1 , "Xmega revision:" ; Mcu_revid                     ' make sure
it is 7 or higher !!! lower revs have many flaws

Const Usechannel = 1


Dim B1 As Byte , B2 As Byte
Dim W As Word At B1 Overlay


Open "twic" For Binary As #4                                 ' or use
TWID,TWIE oR TWIF
Config Twic = 100000                                          'CONFIG TWI
will ENABLE the TWI master interface
'you can also use TWIC, TWID, TWIE of TWIF
'!!!!!!!!!!!   WITHOUT a channel identifier, TWIC will be used
!!!!!!!!!!!!!

#if Usechannel = 1
  I2cinit #4
```

```
#else
  I2cinit
#endif


Do
  I2cstart                                          'since not #
is used, TWIC will be used
  Waitms 20
  I2cwbyte &H70                                     ' slave
address write
  Waitms 20
  I2cwbyte &B10101010                               ' write
command
  Waitms 20
  I2cwbyte 2
  Waitms 20
  I2cstop
  Print "Error : " ; Err                            ' show error
status

  'waitms 50
  Print "start"
  I2cstart
  Print "Error : " ; Err                            ' show error
  I2cwbyte &H71
  Print "Error : " ; Err                            ' show error
  I2crbyte B1 , Ack
  Print "Error : " ; Err                            ' show error
  I2crbyte B2 , Nack
  Print "Error : " ; Err                            ' show error
  I2cstop
  Print "received A/D : " ; W ; "-" ; B1 ; "-" ; B2
  Waitms 500                                        'wait a bit
Loop



Dim J As Byte , C As Byte , K As Byte
Dim Twi_start As Byte                               ' you MUST
dim this variable since it is used by the lib

'determine if we have an i2c slave on the bus
For J = 0 To 200 Step 2
  Print J
  #if Usechannel = 1
    I2cstart #4
  #else
    I2cstart
  #endif

  I2cwbyte J
  If Err = 0 Then                                  ' no errors
    Print "FOUND : " ; Hex(j)
    'write some value to the pcf8574A
    #if Usechannel = 1
    I2cwbyte &B1100_0101 , #4
    #else
    I2cwbyte &B1100_0101
    #endif
    Print Err
    Exit For
  End If
```

```
  #if Usechannel = 1
    I2cstop #4
  #else
    I2cstop
  #endif
Next
#if Usechannel = 1
  I2cstop #4
#else
  I2cstop
#endif

#if Usechannel = 1
  I2cstart #4
  I2cwbyte &H71 , #4                                    'read
address
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack , #4
  Print Bin(j) ; " err:" ; Err
  I2cstop #4
#else
  I2cstart
  I2cwbyte &H71                                         'read
address
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack
  Print Bin(j) ; " err:" ; Err
  I2cstop
#endif

'try a transaction
#if Usechannel = 1
  I2csend &H70 , 255 , #4                               ' all 1
  Waitms 1000
  I2csend &H70 , 0 , #4                                 'all 0
#else
  I2csend &H70 , 255
  Waitms 1000
  I2csend &H70 , 0
#endif
Print Err


'read transaction
Dim Var As Byte
Var = &B11111111
#if Usechannel = 1
  I2creceive &H70 , Var , 1 , 1 , #4                    ' send and
receive
  Print Bin(var) ; "-" ; Err
  I2creceive &H70 , Var , 0 , 1 , #4                    ' just
receive
  Print Bin(var) ; "-" ; Err
#else
  I2creceive &H70 , Var , 1 , 1                         ' send and
receive
  Print Bin(var) ; "-" ; Err
```

```
  I2creceive &H70 , Var , 0 , 1                              ' just
receive
  Print Bin(var) ; "-" ; Err
#endif

End
```

XTINY SAMPLE

```
'-----------------------------------------------------------------
--
'                    (c) 1995-2025 MCS
'                    xtiny-TWI-scanner.bas
'purpose : scan all i2c addresses to find slave chips
'Micro: tiny816
'-----------------------------------------------------------------
--
$regfile = "atxtiny816.dat"                                  ' the
used chip
$crystal = 20000000                                          '
frequency used
$hwstack = 40
$swstack = 40
$framesize = 40

Config Sysclock = 20mhz , Prescale = 1
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

Waitms 3000
'small delay
Print "XTINY:" ; Hex(rstctrl_rstfr)
'print reset cause

Config Twi0 = 100000
'CONFIG TWI will ENABLE the TWI master interface
I2cinit

Dim Twi_start As Byte , B As Byte
Do
   Print "Scan start"
   For B = 0 To 254 Step 2                                   'for
all odd addresses
      I2cstart                                               'send
start
      I2cwbyte B                                             'send
address
      If Err = 0 Then                                        'we
got an ack
         Print "Slave at : " ; B ; " hex : " ; Hex(b) ; " bin : "
 ; Bin(b)
```

```
      End If
      I2cstop                                                    'free
bus
   Next
   Print "End Scan"
   Waitms 2000                                                   'some
delay and then repeat
Loop
```

7.21.92 CONFIG TWISLAVE

Action

Configure the TWI Slave address and bit rate

Syntax

CONFIG TWISLAVE = address , BTR = value , BITRATE = value , SAVE=option [, GENCALL=value] [,USERACK=ack]

(I2C TWI Slave is part of the I2C-Slave library. This is an add-on library that is not included in Bascom-AVR by default. It is a commercial add on library. It is available from MCS Electronics)

See also: I2C TWI Slave 1831, USING I2C Protocol 297, Using USI 321, CONFIG I2CSLAVE 978 , CONFIG USI 1138

Remarks

Address	The slave address that is assigned to the slave chip. This must be an Even number. Bit 0 of the address is used to activate the general call address. The GENCAL option will set this bit automatic. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
BTR	Bytes to receive. With this constant you specify how many bytes will be expected when the master reads data from the slave. And thus how many bytes will be sent to the master.
Bit rate	This is the I2C/TWI clock frequency. Most chips support 400 KHz (400000) but all I2C chips support 100000.
SAVE	SAVE = NOSAVE : this can be used when you do not change a lot of registers in the interrupt. SAVE = SAVE : this is best to be used when you do not use ASM in the TWI interrupt. See the explanation below. When you do not specify SAVE, the default will be SAVE=SAVE.
GENCALL	General call address activated or not. When you specify 1 or YES, the General call address will be activated which mean that the slave will respond not only to it's own address, but also to the general call address

	0. When you omit the option or specify 0 or NO, the general call address will not be honored.
USERACK	Default is OFF. When you use ON, an alternative library will be used. This library will create a variable named TWI_ACK. Each time your code is called this variable is filled with the value 255. If you do not alter the value, the slave will send an ACK as it is supposed to. If you reset the value to 0, the slave will send a NACK. You can use this to send data with variable length to the slave. In this case, BTR only serves as an index. You must make sure to reset TWI_ACK when you have send the last byte to the master.

The variables Twi , Twi_btr and Twi_btw are created by the compiler. These are all bytes
The TWI interrupt is enabled but you need to enabled the global interrupt

The TWI Slave code is running as an interrupt process. Each time there is a TWI interrupt some slave code is executed. Your BASIC code is called from the low level slave code under a number of events. You must include all these labels in your Slave application. You do not need to write code in all these sub routines. All the time your user code is executed, the clock line is stretched. This will reduce the TWI bus speed. So it is important that you do not put delays in your code.

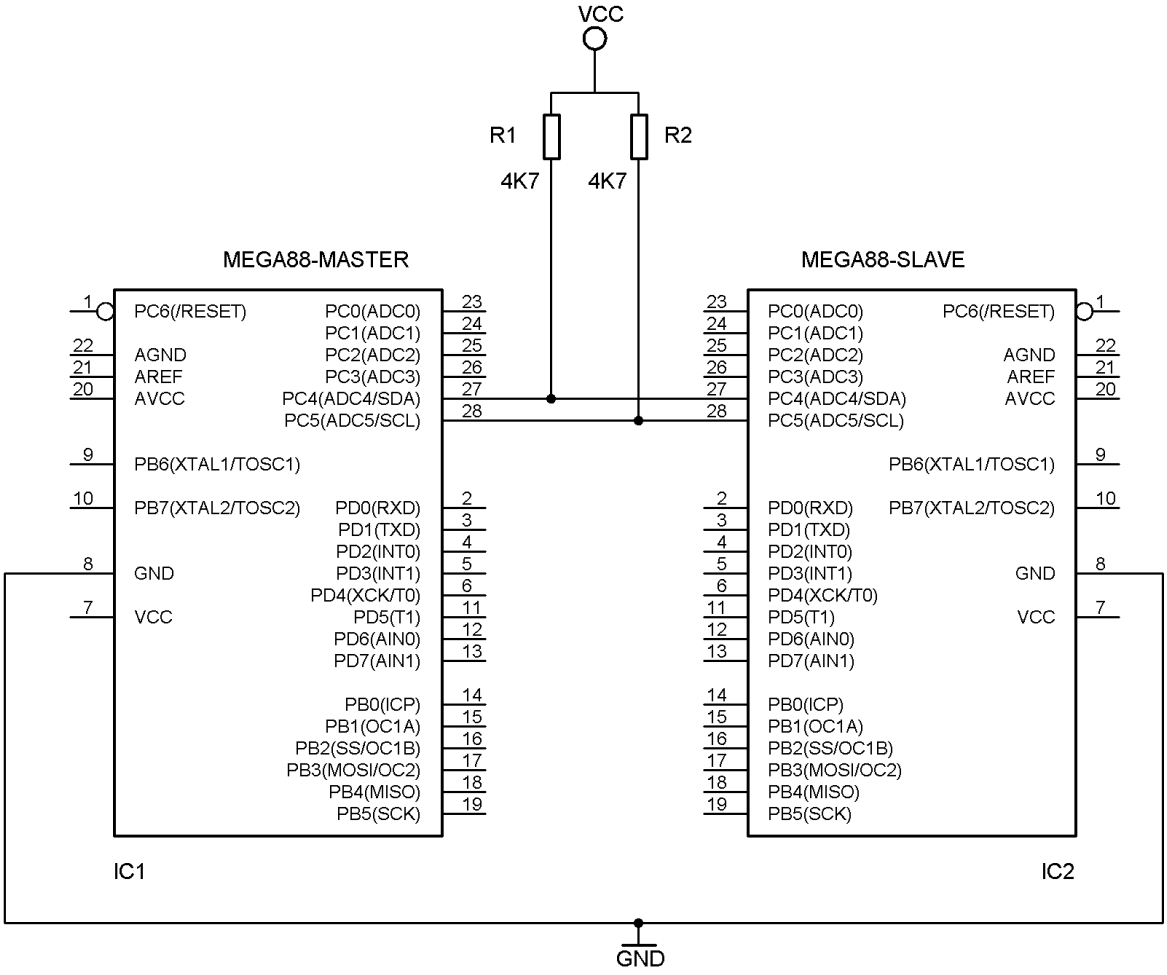
Label	**Event**
Twi_stop_rstart_received	The Master sent a stop(i2CSTOP) or repeated start. Typical you do not need to do anything here.
Twi_addressed_goread	The master has addressed the slave and will now continue to send data to the slave. You do not need to take action here.
Twi_addressed_gowrite	The master has addressed the slave and will now continue to receive data from the slave. You do not need to take action here.
Twi_gotdata	The master has sent data. The variable **TWI** holds the received value. The byte **TWI_BTW** is an index that holds the value of the number of received bytes. The first received byte will have an index value of 1.
Twi_master_needs_byte	The master reads from the slave and needs a value. The variable TWI_BTR can be inspected to see which index byte was needed. With the CONFIG **BTR**, you specify how many bytes the master will read.

In most cases your main application is just an empty DO LOOP. But when you write a slave that performs other tasks on the background these other tasks are interrupted by the TWI traffic.
Take in mind that the interrupt with the lowest address has the highest priority.
So do NOT write blocking code inside an interrupt. While servicing another interrupt, the TWI interrupt can not be serviced.

The TWI Slave code will save all used registers.
But since it will call your BASIC application when the TWI interrupt occurs, your BASIC code could be in the middle of say a PRINT statement.
When you then execute another PRINT statement , you will destroy registers.
So keep the code in the sub routines to a minimum, and use SAVE option to save all registers. This is the default.
While two printing commands will give odd results (print 12345 and 456 in the middle

of the first print will give 1234545) at least no register is destroyed.

A typical configuration is shown below.



To test the above hardware, use the samples : twi-master.bas and twi-slave.bas
Optional you can use i2cscan.bas to test the general call address.

When you want to change the address of the slave at run time you need to write to the **TWAR** register.
The TWAR register contains the slave address. Bit 0 which is used to indicate a read or write transaction should be cleared. When you set it, the slave will also recognize the general call address. The GENCALL option just sets bit 0 of the slave.

See also

CONFIG TWI 1116 , CONFIG SCL 1049 , CONFIG SDA 1046 , I2C TWI Slave 1831, Using the I2C protocol 297

ASM

NONE

Example1(master)

```
'-----------------------------------------------------------------------
'                          (c) 1995-2025 MCS Electronics
```

```
'                       This demo shows an example of the TWI
'                      Not all AVR chips have TWI (hardware I2C)
'-----------------------------------------------------------------------------

'The chip will work in TWI/I2C master mode
'Connected is a PCF8574A 8-bits port extender


$regfile = "M88def.dat"                                     ' the used chip
$crystal = 8000000                                          ' frequency used
$baud = 19200                                               ' baud rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


$lib "i2c_twi.lbx"                                          ' we do not use softwar

Config Scl = Portc.5                                        ' we need to provide th
Config Sda = Portc.4                                        ' we need to provide th

'On the Mega88,          On the PCF8574A
'scl=PC5 , pin 28            pin 14
'sda=PC4 , pin 27            pin 15


I2cinit                                                     ' we need to set the pi


Config Twi = 100000                                         ' wanted clock frequenc
'will set TWBR and TWSR
'Twbr = 12                                                   'bit rate register
'Twsr = 0                                                    'pre scaler bits

Dim B As Byte , X As Byte
Print "TWI master"
Do
  Incr B                                                    ' increase value
  I2csend &H0 , B                                           ' send the value to ger

  I2csend &H70 , B                                          ' send the value
  Print "Error : " ; Err                                    ' show error status
  I2creceive &H70 , X                                       ' get a byte
  Print X ; " " ; Err                                       ' show error
  Waitms 500                                                'wait a bit
Loop
End
```

Example2(slave)

```
'-----------------------------------------------------------------------------
'                          (c) 1995-2025 MCS Electronics
'                This demo shows an example of the TWI in SLAVE mode
'                       Not all AVR chips have TWI (hardware I2C)
' IMPORTANT : this example ONLY works when you have the TWI slave library
'             which is a commercial add on library, not part of BASCOM
'Use this sample in combination with i2cscan.bas and/or twi-master.bas
'-----------------------------------------------------------------------------
$regfile = "M88def.dat"                                     ' the chip we use
$crystal = 8000000                                          ' crystal oscillator va
$baud = 19200                                               ' baud rate
$hwstack = 32                                               ' default
```

```
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Print "MCS Electronics TWI-slave demo"

Config Twislave = &H70 , Btr = 1 , Bitrate = 100000 , Gencall = 1
'In i2c the address has 7 bits. The LS bit is used to indicate read or write
'When the bit is 0, it means a write and a 1 means a read
'When you address a slave with the master in bascom, the LS bit will be set/reset a
'The TWAR register in the AVR is 8 bit with the slave address also in the most left
'This means that when you setup the slave address as &H70, TWAR will be set to &H01
'And in the master you address the slave with address &H70 too.
'The AVR TWI can also recognize the general call address 0. You need to either set
'by using &H71 as a slave address, or by using GENCALL=1

'as you might need other interrupts as well, you need to enable them all manual
Enable Interrupts

'this is just an empty loop but you could perform other tasks there
Do
  nop
Loop
End

'A master can send or receive bytes.
'A master protocol can also send some bytes, then receive some bytes
'The master and slave must match.

'the following labels are called from the library
Twi_stop_rstart_received:
  Print "Master sent stop or repeated start"
Return


Twi_addressed_goread:
  Print "We were addressed and master will send data"
Return


Twi_addressed_gowrite:
  Print "We were addressed and master will read data"
Return


'this label is called when the master sends data and the slave has received the byt
'the variable TWI holds the received value
Twi_gotdata:
   Print "received : " ; Twi
Return

'this label is called when the master receives data and needs a byte
'the variable twi_btr is a byte variable that holds the index of the needed byte
'so when sending multiple bytes from an array, twi_btr can be used for the index
Twi_master_needs_byte:
  Print "Master needs byte : " ; Twi_btr
  Twi = 65                                                   ' twi must be filled wi
Return


'when the mast has all bytes received this label will be called
Twi_master_need_nomore_byte:
```

```
  Print "Master does not need anymore bytes"
Return
```

7.21.93 CONFIG TWIxSLAVE

Action

Configure the Xmega TWIC,TWID,TWIE or TWIF hardware to be used a a slave.

Syntax

CONFIG TWICSLAVE = address , BTR = value ,GENCALL=value
CONFIG TWIDSLAVE = address , BTR = value ,GENCALL=value
CONFIG TWIESLAVE = address , BTR = value ,GENCALL=value
CONFIG TWIFSLAVE = address , BTR = value ,GENCALL=value

(I2C TWI Slave is part of the I2C-Slave library. This is an add-on library which is not included with Bascom-AVR by default. It is a commercial add on library. It is available from MCS Electronics)

See also: I2C TWI Slave 1831, USING I2C Protocol 297, Using USI 321, CONFIG I2CSLAVE 978 , CONFIG USI 1138

Remarks

Address	The slave address which is assigned to the slave chip. This must be an Even number. Bit 0 of the address is used to activate the general call address. The GENCAL option will set this bit automatic. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
BTR	Bytes to receive. With this constant you specify how many bytes will be expected when the master reads data from the slave. And thus how many bytes will be sent to the master. This value can be changed dynamically.
GENCALL	General call address activated or not. When you specify **1**, the General call address will be activated which mean that the slave will respond not only to it's own address, but also to the general call address **0**. When you omit the option or specify **0**, the general call address will not be honored.

The variables TwiX , TwiX_btr, TwiX_CBTR and TwiX_btw are created by the compiler. These are all byte variables.
The X represents the TWI interface letter which can be C, D, E or F.

The TWIx interrupt is enabled as well but you need to enabled the global interrupt

The TWI Slave code is running as an interrupt process. Each time there is a TWI interrupt some slave code is executed. Your BASIC code is called from the low level slave code by a number of events. You must include all these labels in your Slave application. You do not need to write code in all these sub routines.

Label	Event
Twi_stop_rstart_received TwiD_stop_rstart_received TwiE_stop_rstart_received TwiF_stop_rstart_received	The Master sent a stop(i2CSTOP) or repeated start. Typical you do not need to do anything here.
Twi_addressed_goread TwiD_addressed_goread TwiE_addressed_goread TwiF_addressed_goread	The master has addressed the slave and will now continue to send data to the slave. You do not need to take action here.
Twi_addressed_gowrite TwiD_addressed_gowrite TwiE_addressed_gowrite TwiF_addressed_gowrite	The master has addressed the slave and will now continue to receive data from the slave. You do not need to take action here.
Twi_gotdata TwiD_gotdata TwiE_gotdata TwiF_gotdata	The master has sent data. The variable **TWIx** holds the received value. The byte **TWIx_BTW** is an index that holds the value of the number of received bytes. The first received byte will have an index value of 1.
Twi_master_needs_byte TwiD_master_needs_byte TwiE_master_needs_byte TwiF_master_needs_byte	The master reads from the slave and needs a value. The variable **TWIx_BTR** can be inspected to see which index byte was requested. With the CONFIG parameter **BTR**, you specify how many bytes the master will read. This value is stored in the variable TWIx_CBTR. You can alter this value but you should not do that in the middle of a transaction.

The name of the label called depends on the used TWI interface. TWIC is the default TWI interface. All I2C commands work with TWIC by default.
In order to make the normal slave code compatible with the Xmega, the TWIC interface uses the same label names as used for normal AVR TWI interface.
This means that your BASCOM slave code for the M32 should work for the TWIC interface without much changes.

 It is important that you do not use the MASTER TWI routines when using the TWI as a slave. Just supply or read data at the provided routines.

In most cases your main application is just an empty DO LOOP. But when you write a slave that performs other tasks on the background these other tasks are interrupted by the TWI traffic.
Do NOT write blocking code inside an interrupt. While servicing another interrupt, the TWI interrupt can not be serviced.
Also, do not block execution by putting delays in the called routines such as TWI_GOTDATA. All these labels are called from the TWIX SLAVE library which is an interrupt routine that will halt the main application and other interrupts.

The TWI Slave code will save all used registers.

In order to get a working slave it is important that the slave matches the protocol used by the master. Thus if the slave reads data from the master and only expects 2 bytes, the master should not send less or more. We advise to make a simple slave first like a PCF8574 clone.

See also
CONFIG TWIX 1116

Example

The following example uses two TWI interfaces. TWID is used in master mode while TWIC is used as the slave.

```
'-------------------------------------------------------------------------
-------
'name                     : xmega-twi-slave.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates Xmega TWI slave add on
'micro                    : Xmega128A1
'suited for demo          : yes
'commercial addon needed  : yes
'-------------------------------------------------------------------------
-------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
'Config Serialin = Buffered , Size = 50

'Enable Interrupts
Open "COM1:" For Binary As #1

Open "twid" For Binary As #4                                ' or use
TWIC,TWIE oR TWIF
Config Twid = 100000                                        'CONFIG TWI
will ENABLE the TWI master interface
'you can also use TWIC, TWID, TWIE of TWIF
'!!!!!!!!!!   WITHOUT a channel identifier, TWIC will be used
!!!!!!!!!!!!!
'SCL is on pin 1
'SDA is on pin 0
'This demo uses TWID as master and TWIC as SLAVE
'Thus portc.0 connects with portD.0 and
'     portc.1 connects with portD.1

'The TWIC when used as a slave has megaAVR compatible labels
'The TWID,TWIE and TWIF have unique new labelnames
'These labels are the labels in your code which are called from the
slave ISR.
'For example : Twi_addressed_gowrite  is named TwiD_addressed_gowrite
for TWID


Dim Twi_start As Byte , j as byte , b as byte
I2cinit #4                                                  'init the
master
config TWIcslave = &H70 , btr = 2                           'use address
&H70 which is &H38 in 7-bit i2c notation
```

```
Enable INTERRUPTS                                          'for the
slave to work we must enable global interrupts

do
   Print #1 , "test xmega"

   For J = 0 To 120 Step 1                                 'notice that
we scan odd and even addresses
      I2cstart #4                                          'send start
      I2cwbyte J , #4                                      'send value
of J
      If Err = 0 Then                                        ' no
errors
         Print #1 , "FOUND : " ; Hex(j)
         if j.0 = 0 then                                   'ONLY if R/W
bit is not set we may write data !!!
            I2cwbyte 100 , #4                              'just write
to values to the slave
            I2cwbyte 101 , #4
         else                                              'read
            I2crbyte b , Ack , #4 : print #1 , "GOT : " ; b 'read 2
bytes
            I2crbyte b , nAck , #4 : print #1 , "GOT : " ; b
         end if
      End If
      I2cstop #4                                           'done
   Next
   waitms 2000                                             'wait some
time
loop


'the following labels are called from the library  when master send stop
or start
'notice that these label names are valid for TWIC.
'for TWID the name would be TWID_stop_rstart_received:
Twi_stop_rstart_received:
   Print #1 , "Master sent stop or repeated start"
Return

'master sent our slave address and will not send data
Twi_addressed_goread:
   Print #1 , "We were addressed and master will send data"
Return


Twi_addressed_gowrite:
   Print #1 , "We were addressed and master will read data"
Return

'this label is called when the master sends data and the slave has
received the byte
'the variable TWIx holds the received value
'The x is the TWI interface letter
Twi_gotdata:
   Print #1 , "received : " ; Twic ; " byte no : " ; Twic_btw
   'here you would do something with the received data
   '   Select Case Twic_btw
   '     Case 1 : Portb = Twi                                  ' first
byte
   '     Case 2:                                               'you can
set another port here for example
   '   End Select
```

```
Return

'this label is called when the master receives data and needs a byte
'the variable twix_btr is a byte variable that holds the index of the
needed byte
'so when sending multiple bytes from an array, twix_btr can be used for
the index
'again the variable name depends on the twi interface
Twi_master_needs_byte:
   Print #1 , "Master needs byte : " ; Twic_btr
   Select Case Twic_btr
      Case 1:                                                  ' first
byte
         twic = 66                                          'we assign a
value but this could be any value you want
      Case 2                                                   ' send
second byte
         twic = 67
   End Select
Return


'when the mast has all bytes received this label will be called
Twi_master_need_nomore_byte:
   Print #1 , "Master does not need anymore bytes"
Return

End
```

7.21.94 CONFIG USB

Action

Create settings related to USB.

Syntax

CONFIG USB = dev, Language= lang, Manufact= "man", Product="prod" , Serial="serial"

Remarks

Dev	The possible options are Device and Host. Host is not supported yet.
Lang	A language identifier. &H0409 for US/English
Man	A string constant with the manufacture name.
Prod	A string constant with the product name.
Serial	A string constant with the serial number.

The above settings determine how your device is displayed by the operating system. Since these settings end up in flash code space, it is best to chose short names. There is no limit to the length other then the USB specifications impose, but keep it short as possible. Strings in USB are UNI coded. Which mean that a word is used for each character. with normal ASCII coding, only a byte is used for each character.

For a commercial USB device you need to give it a unique VID & PID combination. When you plan to use it at home, this is not needed.
You can buy a Vendor ID (VID) from the USB organization. This cost 2000 $.
As a service MCS offers a PID in the on line shop. This cost little and it gives you a

unique Product ID(PID) but with the MCS Electronics VID.


Notice that using CONFIG USB will include a file named **USBINC.BAS**. This file is not part of the BASCOM setup/distribution. It is available as a commercial add on.
The add on package includes 3 samples , the include file, and a special activeX for the HID demo.
None of the samples require a driver. A small UB162 module with normal pins is available from the on line shop too.
The first supported USB devices are USB1287, USB162.

See also
NONE

Example
```
$regfile = "usb162.dat"
$crystal = 8000000
$baud = 19200

Const Mdbg = 1


Config Clockdiv = 1
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Const Vendor_id = &H16D0                                    ' MCS Vendor
ID
Const Product_id = &H201D                                   ' MCS
product ID, you can buy a VID&PID in the MCS shop

Const Ep_control_length = 32
Const User_conf_size = 41
Const Size_of_report = 53
Const Device_class = 0
Const Device_sub_class = 0
Const Device_protocol = 0
Const Release_number = &H1000
Const Length_of_report_in = 8
Const Length_of_report_out = 8
Const Interface_nb = 0
Const Alternate = 0
Const Nb_endpoint = 2
Const Interface_class = 3                                   ' HID
Const Interface_sub_class = 0
Const Interface_protocol = 0
Const Interface_index = 0

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Print "USB GENERIC test"

Declare Sub Usb_user_endpoint_init
Declare Sub Hid_test_hit()
Declare Sub Hid_task()
Declare Sub Hid_task_init()
```

```
Const Usb_config_attributes_reserved = &H80
Const Usb_config_buspowered = Usb_config_attributes_reserved
Const Usb_config_selfpowered = Usb_config_attributes_reserved Or &H40
Const Usb_config_remotewakeup = Usb_config_attributes_reserved Or &H20

Const Nb_interface = 1
Const Conf_nb = 1
Const Conf_index = 0
Const Conf_attributes = Usb_config_buspowered
Const Max_power = 50                                        ' 100 mA


Const Interface_nb_mouse = 0
Const Alternate_mouse = 0
Const Nb_endpoint_mouse = 1
Const Interface_class_mouse = 3                             ' HID Class
Const Interface_sub_class_mouse = 1                         ' Sub Class
is Mouse
Const Interface_protocol_mouse = 2                          ' Mouse
Const Interface_index_mouse = 0

Const Nb_endpoints = 2                                      ' number of
endpoints in the application including control endpoint
Const Ep_kbd_in = 1                                         ' Number of
the mouse interrupt IN endpoint
Const Ep_hid_in = 1
Const Ep_hid_out = 2

Const Endpoint_nb_1 = Ep_hid_in Or &H80
Const Ep_attributes_1 = 3                                   ' BULK =
0x02, INTERUPT = 0x03
Const Ep_in_length_1 = 8
Const Ep_size_1 = Ep_in_length_1
Const Ep_interval_1 = 20                                    ' Interrupt
polling interval from host

Const Endpoint_nb_2 = Ep_hid_out
Const Ep_attributes_2 = 3                                   ' BULK =
0x02, INTERUPT = 0x03
Const Ep_out_length = 8
Const Ep_size_2 = Ep_out_length
Const Ep_interval_2 = 20                                    ' interrupt
polling from host


Config Usb = Device , Language = &H0409 , Manufact = "MCS" , Product =
"MCSHID162" , Serial = "MC0001"


'Dim some user vars
Dim Usb_kbd_state As Byte , Usb_key As Byte , Usb_data_to_send As Byte
Dim Dummy As Byte , Dummy1 As Byte , Dummy2 As Byte

Print "task init"
Usb_task_init
Hid_task_init
Do
  Usb_task
  Hid_task
  'you can call your sub program here
Loop

'nothing needed to init
```

```
Sub Hid_task_init()
  'nothing
end sub


'HID task must be checked regular
Sub Hid_task()
    If Usb_connected = 1 Then                                   ' Check USB
HID is enumerated
       Usb_select_endpoint Ep_hid_out                           ' Get Data
Repport From Host
       If Ueintx.rxouti = 1 Then                                '
Is_usb_receive_out())
          Dummy1 = Uedatx : Print "Got : " ; Dummy1
          Dummy2 = Uedatx : Print "Got : " ; Dummy2
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Usb_ack_receive_out
      End If

      If Dummy1 = &H55 And Dummy2 = &HAA Then                   '  Check if
we received DFU mode command from host
         Usb_detach                                             '  Detach
Actual Generic Hid Application
         Waitms 500
         Goto &H1800                                            'goto
bootloader
         'here you could call the bootloader then
      End If

      Usb_select_endpoint Ep_hid_in                             ' Ready to
send these information to the host application
      If Ueintx.txini = 1 Then                                  '
Is_usb_in_ready())
         Uedatx = 1
         Uedatx = 2
         Uedatx = 3
         Uedatx = 4
         Uedatx = 5
         Uedatx = 6
         Uedatx = 7
         Uedatx = 8
         Usb_ack_fifocon                                        ' Send data
over the USB
      End If
    End If
End Sub


Function Usb_user_read_request(type As Byte , Request As Byte) As Byte
  #if Mdbg
    Print "USB_USER_READ_REQ"
  #endif
  Usb_string_type = Uedatx
'Usb_read_byte();
  Usb_descriptor_type = Uedatx
'Usb_read_byte();
  Usb_user_read_request = 0
  Select Case Request
     Case Get_descriptor:
```

```
         Select Case Usb_descriptor_type
            Case Report : Call Hid_get_report()
                          Usb_user_read_request = 1
            Case Hid : Call Hid_get_hid_descriptor()
                       Usb_user_read_request = 1
            Case Else
               Usb_user_read_request = 0
         End Select
     Case Set_configuration:
         Select Case Usb_descriptor_type
          Case Set_report : Call Hid_set_report()
                 Usb_user_read_request = 1
          Case Else
                Usb_user_read_request = 0
         End Select
    Case Get_interface:
         '//      usb_hid_set_idle();
       Call Usb_hid_get_interface()
       Usb_user_read_request = 1
    Case Else
      Usb_user_read_request = 0
 End Select
End Function

'usb_init_device.
'This function initializes the USB device controller and
'configures the Default Control Endpoint.
Sub Usb_init_device()
   #if Usbfunc
      Usb_select_device
   #endif
   #if Usbfunc
   If Usbsta.id = 1 Then                                    'is it an
USB device?
   #endif
      Uenum = Ep_control                                    ' select USB
endpoint
      If Ueconx.epen = 0 Then                               ' usb
endpoint not enabled yet
         Call Usb_configure_endpoint(ep_control , Type_control ,
Direction_out , Size_32 , One_bank , Nyet_disabled)
      End If
 #if Usbfunc
     End If
 #endif
End Sub



Sub Usb_user_endpoint_init(byval Nm As Byte)
  Call Usb_configure_endpoint(ep_hid_in , Type_interrupt , Direction_in
, Size_8 , One_bank , Nyet_enabled)
  Call Usb_configure_endpoint(ep_hid_out , Type_interrupt ,
Direction_out , Size_8 , One_bank , Nyet_enabled)
End Sub


Usb_dev_desc:
Data 18 , Device_descriptor                                 'size and
device_descriptor
Data 0 , 2
'Usb_write_word_enum_struc(USB_SPECIFICATION)
Data Device_class , Device_sub_class                        '
DEVICE_CLASS  and  DEVICE_SUB_CLASS
```

```
Data Device_protocol , Ep_control_length                    ' device
protol and ep_control_length
Data Vendor_id%                                             '
Usb_write_word_enum_struc(VENDOR_ID)
Data Product_id%                                            '
Usb_write_word_enum_struc(PRODUCT_ID)
Data Release_number%                                        '
Usb_write_word_enum_struc(RELEASE_NUMBER)
Data Man_index , Prod_index                                 ' MAN_INDEX
and  PROD_INDEX
Data Sn_index , Nb_configuration                            ' SN_INDEX
and NB_CONFIGURATION


Usb_conf_desc:
Data 9 , Configuration_descriptor                           ' length ,
CONFIGURATION descriptor
Data User_conf_size%                                        ' total
length of data returned
Data Nb_interface , Conf_nb                                 ' number of
interfaces for this conf. ,  value for SetConfiguration resquest
Data Conf_index , Conf_attributes                           ' index of
string descriptor  , Configuration characteristics
Data Max_power                                              ' maximum
power consumption


Data 9 , Interface_descriptor                               'length ,
INTERFACE descriptor type
Data Interface_nb , Alternate                               'Number of
interface   ,   value to select alternate setting
Data Nb_endpoint , Interface_class                          'Number of
EP except EP 0  ,Class code assigned by the USB
Data Interface_sub_class , Interface_protocol               'Sub-class
code assigned by the USB  , Protocol code assigned by the USB
Data Interface_index                                        'Index Of
String Descriptor


Data 9 , Hid_descriptor                                     'length  ,
HID descriptor type
Data Hid_bdc% , 8                                           ' Binay
Coded Decimal Spec. release  , Hid_country_code
Data Hid_class_desc_nb , Hid_descriptor_type                'Number of
HID class descriptors to follow ,  Report descriptor type
Data Size_of_report%                                        'HID
KEYBOARD LENGTH

Data 7 , Endpoint_descriptor                                ' Size Of
This Descriptor In Bytes    ,  ENDPOINT descriptor type
Data Endpoint_nb_1 , Ep_attributes_1                        ' Address of
the endpoint   ,Endpoint's attributes
Data Ep_size_1%                                             ' Maximum
packet size for this EP  ,   Interval for polling EP in ms
Data Ep_interval_1

Data 7 , Endpoint_descriptor                                ' Size Of
This Descriptor In Bytes    ,  ENDPOINT descriptor type
Data Endpoint_nb_2 , Ep_attributes_2                        ' Address of
the endpoint  ,  Endpoint's attributes
Data Ep_size_2%                                             ' Maximum
packet size for this EP
Data Ep_interval_2                                          ' Interval
for polling EP in ms
```

```
Usb_hid_report:
Data &H06 , &HFF , &HFF                                    ' 04|2   ,
Usage Page (vendordefined?)
Data &H09 , &H01                                           ' 08|1   ,
Usage      (vendordefined
Data &HA1 , &H01                                           ' A0|1   ,
Collection (Application)
'      // IN report
Data &H09 , &H02                                           ' 08|1   ,
Usage      (vendordefined)
Data &H09 , &H03                                           ' 08|1   ,
Usage      (vendordefined)
Data &H15 , &H00                                           ' 14|1   ,
Logical Minimum(0 for signed byte?)
Data &H26 , &HFF , &H00                                    ' 24|1   ,
Logical Maximum(255 for signed byte?)
Data &H75 , &H08                                           ' 74|1   ,
Report Size(8) = field size in bits = 1 byte
Data &H95 , Length_of_report_in                            '
94|1:ReportCount(size) = repeat count of previous item
Data &H81 , &H02                                           ' 80|1: IN
report (Data,Variable, Absolute)
'      // OUT report
Data &H09 , &H04                                           ' 08|1   ,
Usage      (vendordefined)
Data &H09 , &H05                                           ' 08|1   ,
Usage      (vendordefined)
Data &H15 , &H00                                           ' 14|1   ,
Logical Minimum(0 for signed byte?)
Data &H26 , &HFF , &H00                                    ' 24|1   ,
Logical Maximum(255 for signed byte?)
Data &H75 , &H08                                           ' 74|1   ,
Report Size(8) = field size in bits = 1 byte
Data &H95 , Length_of_report_out                           '
94|1:ReportCount(size) = repeat count of previous item
Data &H91 , &H02                                           ' 90|1: OUT
report (Data,Variable, Absolute)
'      // Feature report
Data &H09 , &H06                                           ' 08|1   ,
Usage      (vendordefined)
Data &H09 , &H07                                           ' 08|1   ,
Usage      (vendordefined)
Data &H15 , &H00                                           ' 14|1   ,
LogicalMinimum(0 for signed byte)
Data &H26 , &HFF , &H00                                    ' 24|1   ,
Logical Maximum(255 for signed byte)
Data &H75 , &H08                                           ' 74|1   ,
Report Size(8) =field size in bits = 1 byte
Data &H95 , &H04                                           '
94|1:ReportCount
Data &HB1 , &H02                                           ' B0|1:
Feature report
Data &HC0                                                  ' C0|0    ,
End Collection
```

7.21.95 CONFIG USI

Action

Configures the hardware USI.

Syntax

CONFIG USI=usimode , Address=adr , ALTPIN=port
CONFIG USI=usimode , Mode=mode , ALTPIN=port

Remarks

The USI(universal serial Interface) is found in most atTiny processors. It can be used for various tasks. At the moment only the TWI slave and TWI master modes are supported. The other modes you need to configure/code yourself.
The CONFIG USI = TWISLAVE mode requires a library that is part of the i2c slave add on which is a commercial add on.
The CONFIG USI = TWIMASTER also requires a library which is included in the commercial distribution.

usiMode	The supported mode is : - TWISLAVE. This will set the USI in TWI slave mode. The USI works in interrupt mode on the background. The library i2c_usi_slave.lib contains the USI slave code. -TWIMASTER. This will set the USI in TWI master mode. This mode does not use interrupts. The library i2c_usi.lib contains the USI master code.
Address	This is the I2C/TWI slave address. Notice that bascom uses the 8-bit address notation. The address is only required when using the USI as a slave.
Mode	The mode is only intended to be used with the USI in master mode. The options are : FAST and NORMAL. Normal will result in a 100 KHz clock signal. And FAST will use a 400 KHz signal if possible.
Altpin	Some processor have an option to swap the USI pins. For example tiny261 can swap from default portB to portA. When not specified, the default pins will be used. When a different port is defined than the default, a constant will be created that is used inside the library. The USIPP register will be set to swap the pins. It is not possible to swap pins dynamically.

TWI SLAVE MODE

When USI is used in TWI/I2C mode, it does require that SCL and SDA have pull up resistors. You can not freely choose the SCL and SDA pins : you must use the fixed SCL en SDA pins.

The variables TWI_USI_OVS , TWI_slaveAddress, Twi , Twi_btr and Twi_btw are created by the compiler. These are all bytes.
The USI interrupts are enabled but you need to enabled the global interrupt using ENABLE INTERRUPTS

The USI Slave code is running as an interrupt process. Each time there is an USI interrupt some slave code is executed. Your BASIC code is called from the low level slave code at a number of events.
You must include all these labels in your Slave application. You do not need to write code in all these sub routines.

Label	Event

Twi_stop_rstart_received	The Master sent a stop(i2CSTOP) or repeated start. Typical you do not need to do anything here.
Twi_addressed_goread	The master has addressed the slave and will now continue to send data to the slave. You do not need to take action here.
Twi_addressed_gowrite	The master has addressed the slave and will now continue to receive data from the slave. You do not need to take action here.
Twi_gotdata	The master has sent data. The variable **TWI** holds the received value. The byte **TWI_BTW** is an index that holds the value of the number of received bytes. The first received byte will have an index value of 1.
Twi_master_needs_byte	The master reads from the slave and needs a value. The variable TWI_BTR can be inspected to see which index byte was needed.

TWI MASTER MODE

When USI is used in TWI/I2C mode, it does require that SCL and SDA have pull up resistors. You can not freely choose the SCL and SDA pins : you must use the fixed SCL en SDA pins.

The master mode does NOT require or use any variables. It also does not use any interrupts.

See also

Using USI 321 , CONFIG TWISLAVE 1123, CONFIG TWIXSLAVE 1128

Example, USI SLAVE

```
'-----------------------------------------------------------------------
--------
'                              (c) 1995-2025 MCS Electronics
'          This demo demonstrates the USI I2C slave
'          Not all AVR chips have an USI !!!!
'-----------------------------------------------------------------------
--------

$regfile = "attiny2313.dat"

$crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 24

const cPrint = 0                                                'make 0 for
chips that have NO UART, make 1 when the micro has a UART and you want
to show data on the terminal

#if cPrint
   $baud = 19200                                                'only when
the processor has a UART
#endif

config usi = twislave , address = &H40                          'bascom uses
8 bit i2c address (7 bit shifted to the left with one bit)
```

```
#if cPrint
   print "USI DEMO"
#endif

'do not forget to enable global interrupts since USI is used in
interrupt mode
enable interrupts                                          'it is
important you enable interrupts

do
   !  nop                                                  ; nothing to
do here
loop



'The following labels are called from the library. You need to insert
code in these subroutines
'Notice that the PRINT commands are remarked.
'You can unmark them and see what happens, but it will increase code
size
'The idea is that you write your code in the called labels. And this
code must execute in as little time
'as possible. So when you slave must read the A/D converter, you can
best do it in the main program
'then the data is available when the master requires it, and you do not
need to do the conversion which cost time.


'A master can send or receive bytes.
'A master protocol can also send some bytes, then receive some bytes
'The master and slave address must match.

'the following labels are called from the library  when master send stop
or start
Twi_stop_rstart_received:
   '  Print "Master sent stop or repeated start"
Return

'master sent our slave address and will not send data
Twi_addressed_goread:
   ' Print "We were addressed and master will send data"
Return


Twi_addressed_gowrite:
   '  Print "We were addressed and master will read data"
Return

'this label is called when the master sends data and the slave has
received the byte
'the variable TWI holds the received value
Twi_gotdata:
   '   Print "received : " ; Twi ; " byte no : " ; Twi_btw
   Select Case Twi_btw
      Case 1 :                                              'Portd =
Twi                                    ' first byte
      Case 2:                                               'you can
set another port here for example
   End Select
Return
```

```
'this label is called when the master receives data and needs a byte
'the variable twi_btr is a byte variable that holds the index of the
needed byte
'so when sending multiple bytes from an array, twi_btr can be used for
the index
Twi_master_needs_byte:
    '  Print "Master needs byte : " ; Twi_btr
   Select Case Twi_btr
      Case 1 : twi = 68                                          ' first
byte
      Case 2 : twi = 69                                          ' send
second byte
   End Select                                                   'you could
also return the state of a port pin or A/D converter
Return
```

Example, USI Master

```
'-----------------------------------------------------------------------
'              (c) 1995-2025 MCS Electronics
'                USI-MASTER.bas
' USI used as TWI master demo
'-----------------------------------------------------------------------

$regfile = "attiny2313.dat"
$crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 24
$baud = 19200

config usi = twimaster , mode = fast

dim b as byte

i2cinit

do
  i2cstart
  i2cwbyte &H40                       'send slave WRITE address for PCF8574
  i2cwbyte &B10101010                     'send a pattern
  i2crepstart                       'repeated start

  i2cwbyte &H41                       'send slave READ address
  i2crbyte b , ack                  'read a byte
  i2crbyte b , nack                  'and again
  i2cstop                          'end transaction and free bus

  waitms 100                        'some delay not required only when you print
loop
```

7.21.96 CONFIG VARPTRMODE

Action

This options sets the behavior of the VARPTR() function.

Syntax

CONFIG VARPTRMODE= Relave | Absolute

Remarks

Different AVR processors have different memory architectures.
Plan old AVR like atmega8 start with the 32 registers, then IO registers with address 0-&H3F then a gap of 32 bytes after which the SRAM memory starts.

The registers R0-R31 are accessible using memory pointer ST/LD. They occupy the absolute address 0-31.
The IO registers with address 0-&H3F can be indexed as well. But an offset of 32 must be added. So IO register TWBR which is at location 0 requires an offset of 32 when using a pointer like LD/ST. Port operations like IN/OUT requires the address from the DAT file thus 0-&H3F.

The Xmega works different. Here the registers R0-R31 can not be accessed by pointers. So here we use the relative address.
And the same applies to the Xtiny platorm processors. There is no way you can address registers by a pointer either.

When you use VARPTR with an IO register to assign a constant the absolute address was returned up to version 2086.
This was wrong since the absolute address was returned instead of the relative address.
Now this is fixed in 2087. But you need to add a CONFIG statement in order to fix this.
In order not to break code this bug can be corrected with a new CONFIG directive.
You may change the behavior between the 2 modes.
The default is the wrong absolute mode.

CONFIG VarptrMode= Relative will set the mode to the relative address. This will give the desired output which will match that of the report file and the Code Explorer. The Code Explorer will always show the relative value which will match the value from the DAT file.

CONFIG VarptrMode = Absolute will set the mode to absolute address so it will be ideal for BASIC INP/OUT operations. Do not confuse with ASM IN/OUT instructions which will always require the relative address !

See also
VARPTR 1604

Example
```
$Regfile="m2560def.dat"
$Crystal=16000000
$hwstack=40
$swstack=32
$framesize=32

 ' the default is the old 2086 mode which gives a wrong result
for normal AVR IO registers
Config VarptrMode=relative

'pina=0
Const Test1 = VarPtr(PINA)    ' IO &h00
Const Test2 = VarPtr(EECR)    ' IO &h1F
Const Test3 = VarPtr(EEDR)    ' IO &h20
Const Test4 = VarPtr(SREG)    ' IO &h3F
Const Test5 = VarPtr(WDTCSR) ' extIO &h60
```

```
Const Test6 = VarPtr(UDR3)    ' extIO &h136
'(
Report:
TEST1                           &H00
TEST2                           &H1F
TEST3                           &H20
TEST4                           &H3F
TEST5                           &H60
TEST6                           &H136
')

Print "&h";Hex(VarPtr(PINA))    ' IO &h00
Print "&h";Hex(VarPtr(EECR))    ' IO &h1F
Print "&h";Hex(VarPtr(EEDR))    ' IO &h20
Print "&h";Hex(VarPtr(SREG))    ' IO &h3F
Print "&h";Hex(VarPtr(WDTCSR)) ' extIO &h60
Print "&h";Hex(VarPtr(UDR3))    ' extIO &h136
Print
'(
Output:
&h0020
&h003F
&h0040
&h005F
&h0060
&h0136
')

 ' when you set the mode to relative you get the correct value
Config VarptrMode=relative
'pina=0
Const Test11 = VarPtr(PINA)    ' IO &h00
print hex(test11)             'prints 00
End
```

7.21.97 CONFIG VPORT

Action

Maps an XMEGA port to a virtual port.

Syntax

CONFIG VPORT0 = port [, VPORT1=port, VPORT2=port, VPORT3=port]

Remarks

VPORT	There are 4 virtual port registers. When setting up these registers, you need to use VPORTx, where X is 0,1,2 or 3, indicating the virtual port. The virtual port itself is accesed via it's registers PORTy, PINy and DDRy where Y is a 0,1 ,2 or 3.

	The normal ports have named like PORTA, PORTB, etc. A virtual port will access the same port but using a different register.
port	The last letter of the real port name. For example A for PORTA, B for PORTB, C for PORTC etc.

 You must specify multiple virtual ports on one CONFIG line. You should not split up the lines in multiple statements because a new CONFIG VPORT will write a new value, erasing the previous setting. When you need to configure 2 virtual ports, put them on one config line like : `Config VPort0 = D , VPort1 = E`
When you split the command like :

```
Config VPort0 = D
Config VPort1 = E
```

The second config will erase the setting of the first config.

 Some processors like the ones from the E5 series have a fixed relation. These chips have virtual port registers (port,ddr,pin) and do not need a CONFIG VPORT).For the E5 this relation is :
PORT0 - Virtual port A
PORT1 - Virtual port C
PORT2 - Virtual port D
PORT3 - Virtual port R

All ports in the Xmega are located in the extended address area. This space can only be accessed with instructions like LDS,STS, LD and ST.
Special bit instructions only work on the lower IO-registers.

Xmega example :

```
again:
Lds r24, PINA      ; read port input value
sbrs r24,7         ; skip next instruction if bit 7 is set (1)
rjmp again         ; try again
```

Now the same code for a normal AVR

```
again:
sbis PINA,7        ; skip if pina.7 is set
rjmp again
```

Not only less code is required, but the LDS takes 3 cycles

With the virtual mapping, you can access any PORT register (PORT,PIN and DDR) via it's virtual name PORT0, PIN0 or DDR0.
Since there are 4 virtual mapping registers, you can define PORT0, PORT1, PORT2 and PORT3.
When you write to PORT**n**, the compiler can use the smaller/quicker code.

Devices like graphical LCD can benefit from this.

XTINY

Xtiny also have virtual port registers. And these are fixed as well. There is no config required. The benefit of the virtual port is again that it is located in lower memory so shorter/faster assembly instructions are possible.
We would recommend to use the virtual port names.

See Also

CONFIG PORT 1011

Example

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'           Mapping Real Ports to Virtual Ports.bas
'  This sample demonstrates mapping ports to virtual ports
'  based on MAK3's sample
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40

'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

Print "Map VPorts"
'map portD to virtual port0, map portE to virtual port1, map portC to
virtual port2
'map portR to virtual port 3
Config VPort0 = D , VPort1 = E , VPort2 = C , VPort3 = R

'Each virtual port is available as PORT0, PORT1, PORT2 and PORT3
'      data direct is available as DDR0 , DDR1,  DDR2  and DDR3
'        PIN input is available as PIN0 , PIN1,  PIN2  and PIN3

'The advantage of virtual port registers is that shorter asm instruction
can be used which also use only 1 cycle
Dim Var As Byte


'Real Port Direction
Ddr1 = &B0000_0000                                          ' Port E =
INPUT
Ddr0 = &B1111_1111                                          ' Port D =
OUTPUT


'Continously copy the value from PORTE to PORTD using the virtual ports.
  Do
    Var = Pin1                                              'Read
Virtual Port 0
    Port0 = Var                                             'Write
Virtual Port 1
  Loop

End                                                         'end program
```

7.21.98 CONFIG VREF

Action
This configuration statement will configure the XTINY voltage reference.

Syntax
CONFIG VREF=Dummy, ADC**x**=ref1,DAC**x**|AC**0**=ref2, force_adc**X**=opt1,force_dac**X**|force_ac**X**=opt2

Remarks

dummy	There is no actual global setting for VREF so the only option is dummy
ADC**x**	This will set the voltage reference for the ADC0/ADC1 to the specified value of ref1. The X represents the number 0 or 1 which represents ADC0 and ADC1. The voltage reference for the ADC ref1 can be : - 0.55V - 1.1V - 4.3V - 1.5V The default is 0.55V The voltage reference can not exceed the supply voltage. So 4.3 is only possible when VCC is 5V
force_adc**X**	This option allows to force the reference to be running even if it is not requested. A value of ENABLED will force the reference to be on. A value of DISABLED which is the default will set the reference in automatic mode. In this mode the reference is turned on when requested. This will save power.
DAC**x** AC**0**	This will set the voltage reference for the DAC**x** and/or AC**x** to the specified value of ref2. Note that ADC**x** and DAC**x**/AC**x** reference can be set in depended of each other. The X indicates a number 0,1 or 2 which represents DAC0, DAC1 and DAC2. The voltage reference for the DAC which can be : - 0.55V - 1.1V - 4.3V - 1.5V The voltage reference can not exceed the supply voltage. So 4.3 is only possible when VCC is 5V
force_dac**X** force_ac**X**	This option allows to force the reference to be running even if it is not requested. A value of ENABLED will force the reference to be on. A value of DISABLED which is the default will set the reference in automatic mode. In this mode the reference is turned on when requested. This will save power. As for the other options, the X represents the peripheral DAC/AC.

Note that not all processors have an ADC and/or DAC. It depends on the processor. Some processors have multiple ADC and/or DAC.
The DAC and AC are grouped together. The IDE will show the proper options depending on the chosen processor when using CTRL+SPACE

See also
CONFIG ADC0 880

Example

7.21.99 CONFIG VREGPWR

Action
Configures the voltage regulator.

Syntax
CONFIG VREGPWR= AUTO|FULL , HTMP_LOWLEAK=DISABLED|ENABLED , REGMODE=OVERWRITE|PRESERVE

Remarks

The CONFIG VREGPWR sets the power controller related to the sleep controller options.

VREGPWR	Configures the mode. - AUTO : the regulator will run at the full performance unless the 32 KHz oscillator is selected, then it runs in lower power mode - FULL : full performance voltage regulator drive strength in all modes
HTMP_LOWLEAK	Selects the high temperature low leakage mode - DISABLED : high temperature low leakage disabled - ENABLED : high temperature low leakage enabled When enabled the leakage is reduced when operating at temperature above 70 Celsius. This setting has an effect only in power down mode when the VREGPWR mode is set to AUTO. It must be configured before the CONFIG POWERMODE 1017 option.
REGMODE	- OVERWITE : the entire register is updated. - PRESERVE : the register bits are preserved. See also the AVRX 503 topic.

See also
CONFIG POWERMODE 1017

Example
NONE

7.21.10 CONFIG WAITSUART

Action
Compiler directive that specifies that software UART waits after sending the last byte.

Syntax
CONFIG WAITSUART = value

Remarks

value	A numeric value in the range of 1-255. A higher value means a longer delay in mS.

When the software UART routine are used in combination with serial LCD displays it can be convenient to specify a delay so the display can process the data.

See also
OPEN 1386

Example
See OPEN 1386 example for more details.

7.21.10 CONFIG WATCHDOG

Action
Configures the watchdog timer.

Syntax
CONFIG WATCHDOG = time

Syntax XTINY
CONFIG WATCHDOG = time [,window=time]

Remarks

Time	The interval constant in ms the watchdog timer will count to before it will reset your program. Possible settings : 16 , 32, 64 , 128 , 256 , 512 , 1024 and 2048. Some newer chips : 4096, 8192. The **XMEGA** has a 1 KHz clocked watchdog. For Xmega the following value in millisecond need to be used : 8 ,16,32,64,125,250,500,1000,2000,4000,8000 So 2000 will sets a timeout of 2 seconds. The **XTINY** platform accepts the following values :

	0 - will turn **off** the WD 8 - 8 clock cycles which is 7.8 ms 16 - 16 clock cycles which is 16.625 ms 32 - 32 clock cycles which is 32.25 ms 64 - 64 clock cycles which is 62.5 ms 128 - 128 clock cycles which is 0.125 ms 256 - 256 clock cycles which is 0.250 ms 512 - 512 clock cycles which is 0.500 ms 1000 - 1000 clock cycles which is 1 sec 2000 - 2000 clock cycles which is 2 sec 4000 - 4000 clock cycles which is 4 sec 8000 - 8000 clock cycles which is 8 sec The Xtiny also has an optional window value that can be set.

Note that some new AVR's might have additional reset values such as 4096 and 8192.

Normal AVR

When the **W**atch**D**og is started, a reset will occur after the specified number of mS. With a value of 2048, a reset will occur after 2 seconds, so you need to reset the WD in your programs periodically with the **RESET WATCHDOG** statement.

Some AVR's might have the WD timer enabled by default. You can change this by changing the Fuse Bits.

 Global Interrupts should be disabled when they are active. The reason is that changing the WD, a special timed sequence is required. An interrupt could extend the time, making the timed sequence fail.

 After the CONFIG WATCHDOG statement, the watchdog timer is disabled. You can also use CONFIG WATCHDOG to change the time out value. This will stop the watchdog timer and load the new value.
After a CONFIG WATCHDOG, you always need to start the Watchdog with the **START WATCHDOG** statement.

Most new AVR chips have an MCUSR register that contains some flags. One of the flags is the WDRF bit. This bit is set when the chip was reset by a Watchdog overflow. The CONFIG WATCHDOG will clear this bit, provided that the register and bit are available in the micro.
When it is important to examine at startup if the micro was reset by a Watchdog overflow, you need to examine this MCUSR.WDRF flag before you use CONFIG WATCHDOG, since that will clear the flag.

ALL PLATFORMS

 For chips that have an enhanced WD timer, the WD timer is cleared as part of the chip initialize procedure. This because otherwise the WD timer will only work once. If it is important to know the cause of the reset, you can read the register **R0** before you run other code.

When the chip resets, the status registers with the reset cause bits is saved into register **R0**.
This is done because the compiler need to reset these flags since otherwise they can

not occur again. And before clearing the bits, the status is saved into register **R0**.

The sample below demonstrates how to store the WDRF bit if you need it, and print it later.
The compiler will read R0 from the correct register which depends on the used platform (normal AVR, Xmega, Xtiny)

XTINY

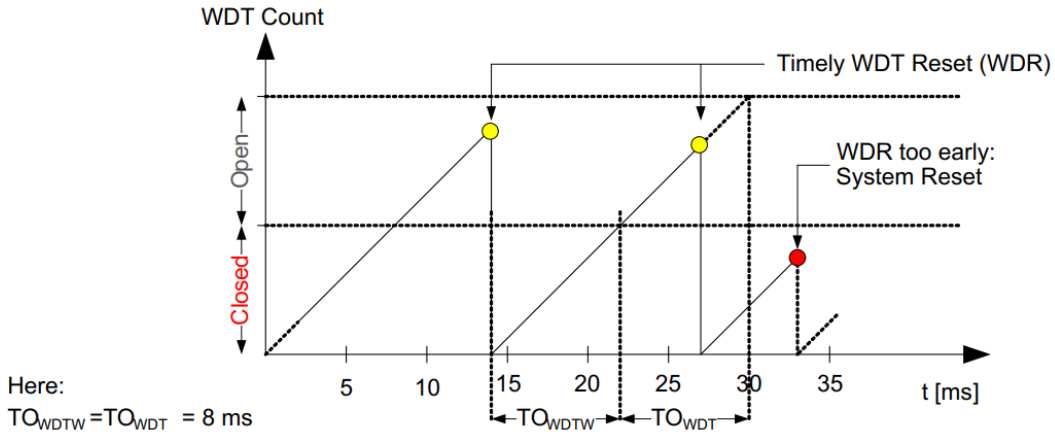
The XTINY platform differs from the normal AVR. The WD timer will be turned off with a value of 0. Normal AVR use a bit for that.
When you use STOP WATCHDOG, a value of 0 will be written to the control register to turn off the WD.
This also means that when you configure the Watchdog with a value other than 0, it will start immediately.
Since there is no control bit it means that START WATCHDOG will only work when the WD timer has been previously configured with a value other than 0.
This value will be written to the Watchdog in order to start it.

When there is no window value provided, the watchdog timer is in the normal mode. In the normal mode a single time out period is set for the WDT.
If the WDT is not reset during the defined time-out period, the WDT will issue a system reset.
In order to prevent overflow of the WDT a reset must be done using : RESET WATCHDOG statement.

When a window value is defined the WDT will be in window mode. In window mode the WDT uses two different time-out periods :
- the closed window time-out period (TOwdtw) define a duration from 8 ms to 8 sec where the WDT can not be reset. If the WDT is reset during this period, the WDT will issue a system reset.
- the open window time-out period (TOwdt) which is also 8 ms to 8s sec, defines the open period during which the WDT can (and needs to) be reset. The open period will always follow the closed period, so the total duration of the time-out perio is the sum of the closed window and the open window time-out periods.

The following picture is taken from the datasheet

Figure 22-3. Window Mode Operation



 When setting the Watchdog FUSE , this value is loaded when the processor boots. After that this value can not be changed since a LOCK bit is set.
This also means that when using the FUSE you can not stop the watchdog.

See also

START WATCHDOG 1538, STOP WATCHDOG 1544, RESET WATCHDOG 1428

Example

```
'-----------------------------------------------------------------------------
'name                     : watchd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates the watchdog timer
'micro                    : Mega88
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------

$regfile = "m88def.dat"                                    ' specify the used micr
$crystal = 8000000                                         ' used crystal frequenc
$baud = 19200                                              ' use baud rate
$hwstack = 32                                              ' default use 32 for th
$swstack = 32                                              ' default use 32 for th
$framesize = 40                                            ' default use 40 for th
Dim B As Byte
Dim Wdbit As Bit
Dim bWD As Byte

bWD= R0                                                    ' read the wd flag
Print "Watchdog test"
If bwd.wdrf = 1 Then                                       ' there was a WD overfl
   Wdbit = 1                                               'store the flag
End If

Config Watchdog = 2048                                     'reset after 2048 mSec
If Wdbit = 1 Then                                          'just print it now sinc
    Print "Micro was reset by Watchdog overflow"
End If

Start Watchdog                                             'start the watchdog tim
Dim I As Word
For I = 1 To 1000
  Waitms 100
  Print I                                                  'print value
  B = Inkey()                                              ' get a key from the se
  If B = 65 Then                                           'letter A pressed
     Stop Watchdog                                         ' test if the WD will s
  Elseif B = 66 Then                                       'letter B pressed
     Config Watchdog = 4096                                'reconfig to 4 sec
     Start Watchdog                                        'CONFIG WATCHDOG will c
  Elseif B = 67 Then                                       'C pressed
     Config Watchdog = 8192                                ' some have 8 sec timer
     'observe that the WD timer is OFF
  Elseif B = 68 Then                                       'D pressed
     Start Watchdog                                        ' start it
  End If
  'Reset Watchdog
  'you will notice that the for next doesnt finish because of the reset
  'when you unmark the RESET WATCHDOG statement it will finish because the
```

```
  'wd-timer is reset before it reaches 2048 msec
  'When you press 'A' you will see that the WD will stop
  'When you press 'B' you will see that the WD will time out after 4 Sec
  'When you press 'C' you will see the WD will stop
  'When you press 'D' you will see the WD will start again timing out after 8 secs
Next
End
```

And this shows how to read the register r0:

```
Dim Breset As Byte
Breset = R0
```

When you show this value on an LCD display you will see a value of 7 the first time, and later a va

Xmega Sample

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-WD.bas
'  This sample demonstrates the Xmega128A1 Watchdog
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 64
$framesize = 64

'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Config Input1 = Cr , Echo = Crlf                              ' CR is used
for input, we echo back CR and LF

Open "COM1:" For Binary As #1
'       ^^^^ change from COM1-COM8

Print #1 , "Xmega revision:" ; Mcu_revid                      ' make sure
it is 7 or higher !!! lower revs have many flaws

Config Watchdog = 4000                                        'after 4
seconds a reset will occur if the watchdog is enabled
'possible value :  8 ,16,32,64,125,250,500,1000,2000,4000,8000
'these values are clock cycles, based on a 1 KHz clock !!!

Dim W As Word , B As Byte
Do
  W = W + 1
  Print W
  Waitms 500
  B = Inkey()
  If B = "a" Then
     Start Watchdog
     Print "start"
  Elseif B = "b" Then
     Stop Watchdog
     Print "stop"
  Elseif B = "c" Then
     Config Watchdog = 8000
```

```
    Print "8 sec"
  Elseif B = "d" Then
    Reset Watchdog
    Print "reset"
  End If
Loop
```

XTINY Sample

```
'-----------------------------------------------------------------
---------------
'name                     : watchdog-avrx128da28.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates Watchdog
'micro                    : avra128DA28
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
---------------
$regfile = "AVRX128da28.dat"

$crystal = 24000000
$hwstack = 64
$swstack = 64
$framesize = 64

Config Submode = New

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz
'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'configure the USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

Config Clock = Soft , Rtc = 32khz_32khz_intosc
'the RTC requires that global interrupts are enabled
Enable Interrupts


Print "Test WD"

'time out after 8 sec
'a configuration other then 0 will start the watchdog
'a configuration of 0 will turn off the watchdog
Config Watchdog = 8000
Dim B As Byte
```

```
Do
  B = Inkey()
  Select Case B
     Case "r" : Reset Watchdog
'reset WD
     Case "s" : Config Watchdog = 8000                    'set
back to value used in config above
     Case "q" : Config Watchdog = 0                       'turn
off WD
     Case "1" : Config Watchdog = 1000                    'set
time out to 1 sec
     Case "2" : Config Watchdog = 2000                    'set
time out to 2 sec
     Case "4" : Config Watchdog = 4000                    'set
time out to 4 sec
     Case "8" : Config Watchdog = 8000                    'set
time out to 8 sec
  End Select

  Print "WD:" ; Time$                                      'now
watch the time
  Waitms 1000
Loop

End
```

Xtiny Example 2

```
'-----------------------------------------------------------------
----------------
'name                     : watchdog.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates Watchdog
'micro                    : xtiny816
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------
$regfile = "atxtiny816.dat"
$crystal = 20000000
$hwstack = 40
$swstack = 40
$framesize = 40

Dim Bwd As Byte
R0 = Bwd
'store R0 into variable so we can check the reset cause
'set the system clock and prescaler
Config Sysclock = 16_20mhz , Prescale = 1
```

```
'configure the USART
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

Config Clock = Soft , Rtc = 32khz_32khz_intosc
'the RTC requires that global interrupts are enabled
Enable Interrupts


Print "Test WD, reset cause:"
If Bwd.0 = 1 Then
  Print "power on reset"
End If
If Bwd.1 = 1 Then
  Print "brown out reset"
End If
If Bwd.2 = 1 Then
  Print "external reset"
End If
If Bwd.3 = 1 Then
  Print "watchdog reset"
End If
If Bwd.4 = 1 Then
  Print "software reset"
End If
If Bwd.5 = 1 Then
  Print "UPDI reset"
End If

'time out after 8 sec
Config Watchdog = 8000

Do
  Print "WD:" ; Time$
  Waitms 1000
Loop

End
```

7.21.10 CONFIG X10

Action

Configures the pins used for X10.

Syntax

CONFIG X10 = pinZC , TX = portpin

Remarks

PinZC	The pin that is connected to the zero cross output of the TW-523. This is a pin that will be used as INPUT.
Portpin	The pin that is connected to the TX pin of the TW-523. TX is used to send X10 data to the TW-523. This pin will be used in output mode.

The TW-523 RJ-11 connector has the following pinout:

Pin	Description	Connect to micro
1	Zero Cross	Input pin. Add 5.1K pull up.
2	GND	GND
3	RX	Not used.
4	TX	Output pin. Add 1K pull up.

See also

X10DETECT 1613 , X10SEND 1615

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : x10.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : example needs a TW-523 X10 interface
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


'define the house code
Const House = "M"                                           ' use code
A-P

Waitms 500                                                  ' optional
delay not really needed

'dim the used variables
Dim X As Byte

'configure the zero cross pin and TX pin
Config X10 = Pind.4 , Tx = Portb.0
```

```
'             ^--zero cross
'                           ^--- transmission pin

'detect the TW-523
X = X10detect()
Print X                                                    ' 0 means
error, 1 means 50 Hz, 2 means 60 Hz

Do
   Input "Send (1-32) " , X
   'enter a key code from 1-31
   '1-16 to address a unit
   '17 all units off
   '18 all lights on
   '19 ON
   '20 OFF
   '21 DIM
   '22 BRIGHT
   '23 All lights off
   '24 extended code
   '25 hail request
   '26 hail acknowledge
   '27 preset dim
   '28 preset dim
   '29 extended data analog
   '30 status on
   '31 status off
   '32 status request

   X10send House , X                                       ' send the
code
Loop

Dim Ar(4) As Byte
X10send House , X , Ar(1) , 4                              ' send 4
additional bytes
End
```

7.21.10 CONFIG XPIN

Action

Configures additional features of a processor port or pin.

Syntax

CONFIG XPIN=PORT|PIN, OUTPULL=pull

Syntax Xmega

CONFIG XPIN=PORT|PIN, INVERTIO=invio, SLEWRATE=slew, PULLUP=pull, SENSE=sense

Syntax Xtiny

CONFIG XPIN=PORT|PIN, INVERTIO=invio, PULLUP=pull, SENSE=sense

Remarks

Normal AVR port pins can be configured as an input or output. When configured as an input (CONFIG PIN=INPUT) they can also be set to tri-state (write a 0 to the PORT

register) or to activate the pull up resistor(write a 1 to the PORT register).
Some new AVR processors use a special PUD register to control the pull up. The CONFIG XPIN automatically uses the proper registers to control the pull up state.

 The XPIN option was added for the Xmega which uses the term Outpull instead of Pullup. The compiler will accept both names but the Code Explorer and Intellisense expect PULLUP.

Normal AVR

PORT PIN	The pin to be configured. For example PORTC.0 When configuring the whole port (all the pins must have the same functionality), use PORT. For example : PORTD
PULLUP	Sets the output or pull mode. The following options are available: - OFF :no pull up - PULLUP : input pull up

Normal AVR processors (tiny,mega) have only one option : PULLUP.
The compiler will either write a 1 or 0 to the PORT register or the PUEx register.

You can control a single pin using a port pin name like PORTB.0 or the whole register like PORTB.
Normal AVR code that use : PORTX.Y=1 to activate the pull up, should be written as : CONFIG XPIN=PORTX.Y,PULLUP=PULLUP

XMEGA

You still need to use PORTx = state or PINx.y = state to configure the data direction of that port or pin in addition to CONFIG XPIN.

The xmega has many more options. The Xmega manual explains all the options.
The CONFIG XPIN statement will set the proper registers.

PORT PIN	The pin to be configured. For example PORTC.0 When configuring the whole port (all the pins must have the same functionality), use PORT. For example : PORTD
INVERTIO	This option will invert the data for both input and output modes. Possible values : ENABLED (will invert data), DISABLED(normal mode)
SLEWRATE	Will enable or disable the slewrate. Enabling the slew rate will increase the rise/fall time by 50%-150%. Possible values : ENABLED, DISABLED  **For the Xmega E-series, the slewrate is set for the whole port. While the other Xmega series allow setting of sleware for an individual pin.**
PULLUP	Sets the output or pull mode. The following options are available: - TOTEM : output totem pole - BUSKEEPER : output totem pole, input bus keeper - PULLDOWN : output totem pole, input pull down - PULLUP : output totem pole, input pull up - WIREDOR : output wired OR - WIREDAND: output wired AND -WIREDORPULL : output wired OR, input pull down

	-WIREDANDPULL : output wired AND, input pull up
SENSE	In input mode, the trigger sense can be configured. Possible values : - BOTH : sense both edges - RISING : sense rising edge -FALLING : sense falling edge -LOW_LEVEL :sense low level -INP_DISABLED : digital input buffer disabled (only PORTA-PORTF)

Xtiny

You still need to use PORTx = state or PINx.y = state to configure the data direction of that port or pin in addition to CONFIG XPIN.

The xtiny has many more options.
The CONFIG XPIN statement will set the proper registers.

PORT PIN	The pin to be configured. For example PORTC.0 When configuring the whole port (all the pins must have the same functionality), use PORT. For example : PORTD
INVERTIO	This option will invert the data for both input and output modes. Possible values : ENABLED (will invert data), DISABLED(normal mode)
PULLUP	Sets the output or pull mode. The following options are available: - DISABLED or OFF : pull up disabled - PULLUP : output totem pole, input pull up
SENSE	In input mode, the trigger sense can be configured. Possible values : - INT_DISABLED : interrupt disabled but input buffer enabled - BOTH : sense both edges - RISING : sense rising edge -FALLING : sense falling edge -LOW_LEVEL :sense low level -INP_DISABLED : digital input buffer disabled

See also

CONFIG PIN 1011, CONFIG INT 985

Example:

```
Config Porte.5 = Input
Config Xpin = Porte.5 , Pullup = Pullup , Sense = Falling  'enable Pull
up and reaction on falling edge
```

Example

```
$regfile = "xm256a3budef.dat"
$Crystal = 32000000                                 '32MHz

Config Xpin = Portc.0 , Slewrate = Enabled , Pullup = Buskeeper , Sense
= Low_level
Config Xpin = Portc.1 , Slewrate = Enabled , Pullup = Buskeeper , Sense
= Low_level

'setup the whole port at once
Config Xpin = Portd , Slewrate = Enabled , Pullup = Buskeeper , Sense =
```

```
Low_level
```

7.21.10 CONFIG XRAM

Action
Instruct the compiler to set options for external memory access.

Syntax
CONFIG XRAM = mode [, WaitstateLS=wls] [, WaitStateHS=whs]

Syntax Older chips
CONFIG XRAM = mode , Waitstate=wls

Syntax Xmega
CONFIG XRAM = mode, sdbus=sdbus,lpc=lpc,sdcol=sdcol,sdcas=sdcas, sdrow=sdrow,refresh=refresh,initdelay=initdelay,modedelay=modedelay, rowcycledelay=rowcycledelay,rowprechargedelay=rowprechargedelay, wrdelay=wrdelay,ersdelay=esrdelay, rowcoldelay=rowcoldelay,modesel0=sel, adrsize0=adr,baseadr0=base,modesel1=sel,adrsize1=adr,baseadr1=base, modesel2=sel,adrsize2=adr,baseadr2=base,modesel3=sel,adrsize3=adr, baseadr3=base

See also: Adding XRAM with External Memory Interface 251

Remarks AVR

Mode	The memory mode. This is either enabled or disabled. By default, external memory access is disabled.
Wls	When external memory access is enabled, some chips allow you to set a wait state. The number of modes depend on the chip. A modern chip such as the Mega8515 has 4 modes : 0 - no wait states 1 - 1 cycle wait state during read/write 2 - 2 cycle wait state during read/write 3 - 2 cycle wait state during read/write and 1 before new address output WLS works on the lower sector. Provided that the chip supports this.
Whs	When external memory access is enabled, some chips allow you to set a wait state. The number of modes depend on the chip. A modern chip such as the Mega8515 has 4 modes : 0 - no wait states 1 - 1 cycle wait state during read/write 2 - 2 cycle wait state during read/write 3 - 2 cycle wait state during read/write and 1 before new address output WHS works on the high sector. Provided that the chip supports this.

Wait states are needed in case you connect equipment to the bus, that is relatively slow. Especial older electronics/chips.
Some AVR chips also allow you to divide the memory map into sections. By default the total XRAM memory address is selected when you set a wait state.

Older chips like the 90S8515 do not have a lower and upper sector. The setting is for

all the memory in that case.

The $XA directive should not be used anymore. It is the same as CONFIG XRAM=Enabled.

 When using IDLE or another power down mode, it might be needed to use CONFIG XRAM again, after the chip wakes from the power down mode.

[See also Adding XRAM] 251

XMEGA

Mode	The memory mode. There are 4 options: - DISABLED, this will turn off the EBI and is the default - 3PORT. For using EBI in 3 PORT mode. - 4PORT. For using EBI in 4 PORT mode. - 2PORT. For using EBI in 2 PORT mode. The EBI uses specific ports for each of the modes.
sdbus	When using SDRAM, you need to configure 4 bit or 8 bit data width. For the 3 PORT mode you need to use 4 bit SDRAM. Options are : 4 and 8.
sdcol	When using SDRAM, you need to configure the number of columns of the chip. This depends on the chip. You can find this info in the datasheet of the SDRAM chip. For example a chip with column address A0-A9 would use 10 bits. Options : 8 ,9, 10 or 11.
sdrow	When using SDRAM, you need to configure the number of rows of the chip. This depends on the chip. You can find this info in the datasheet of the SDRAM chip. Options : 11 or 12.
sdcas	When using SDRAM you can configure the CAS latency as a number of Peripheral 2x Clock cycles. By default this is two Peripheral 2x Clock cycles. Options are : -2 : CAS latency is two Peripheral 2x Clock cycles -3 : CAS latency is three Peripheral 2x Clock cycles
refresh	When using SDRAM this value sets the refresh period as a number of peripheral clock cycles. Use a value between 0-1023. The value depends on the chip.
initdelay	When using SDRAM this value sets the delay of the initialization sequence that is sent after the voltages have been stabilized and the SDRAM clock is stable. The value is in the range of 0-16384
modedelay	When using SDRAM this value select the delay between Mode Register command and an Activate command in number of Peripheral 2x clock (CLK_{PER2}) cycles. The range is between 0-3
rowcycledelay	When using SDRAM this value select the delay between a refresh an and Activate command in number of Peripheral 2x clock (CLK_{PER2}) cycles. The range is between 0-7
rowprechargedelay	When using SDRAM this value select the delay between a pre-charge command and another command in number of Peripheral 2x clock (CLK_{PER2}) cycles. The range is between 0-7

wrdelay	When using SDRAM this value selects the write recovery time in number of Peripheral 2x clock (CLK_{PER2}) cycles. The range is between 0-3
esrdelay	When using SDRAM this value selects the delay between CKE set high and activate command in number of Peripheral 2x clock (CLK_{PER2}) cycles. The range is between 0-7
rowcoldelay	When using SDRAM this value selects the delay between an activate command and a read/write command as a number of Peripheral 2x clock (CLK_{PER2}) cycles. The range is between 0-7
	*The options ending with **x**, are available multiple times.(0-3)* *So there is an option named selfrefresh0, selfrefresh1, selfrefresh2 and selfrefresh3.*
selfrefresh**X**	When using SDRAM this options can turn on/off self refresh of the SDRAM. Not all SDRAM have this capability. Valid options are : - ENABLED - DISABLED. This is the default.
sdmode**X**	When using SDRAM this option sets the SDRAM mode. This is either NORMAL (default) or LOAD.
modesel**X**	This option selects the MODE of the CS line. There are 4 CS lines and modes. When using SDRAM you can only select modesel3 to configure the SDRAM. The following options are possible: - DISABLE - SRAM - LPC (this is SRAM in low pin count mode) - SDRAM
adrsize**X**	This options sets the address size for the chip select. This is the size of the block above the base address and determines which address lines are compared to generate the CS. Options are: 256b , 256 bytes, address 8:23 512b, 512 bytes, address 9:23 1K , 1 KB , address 10:23 2K , 2 KB , address 11:23 4K , 4 KB , address 12:23 8K, 8 KB , address 13:23 16K , 16 KB , address 14:23 32K , 32 KB , address 15:23 64K , 64 KB , address 16:23 128K , 128 KB, address 17:23 256K , 256 KB , address 18:23 512K , 512 KB , address 19:23 1M , 1 MB, address 20:23 2M , 2 MB , address 21:23 4M , 4 MB , address 22:23 8M , 8 MB, address 23 16M , 16 MB
baseadr**X**	This option sets the chip base address which is the lowest address in the address space enabled by the chip select. The value is a word and sets address bits 12:23. Bits 0:11 are unused and need to be 0. For an 8 MB SDRAM the valid values are 0 and &H800000. Since the lower bits are not used the address is divided by 256 by the compiler. When using 0, the memory overlaps the SRAM which is not a big problem with 8MB of ram!

	In SRAM mode there are some other options you must set
lpc	This sets the ALE mode in LPC SRAM mode. Options are : ALE1 : data multiplexed with address byte 0 ALE12 : data multiplexed with address byte 0 and 1
ale	This sets the ALE mode in normal SRAM mode. Options are : ALE1 : address byte 0 and 1 multiplexed ALE2 : address byte 0 and 2 multiplexed ALE12 : address byte 0, 1 and 2 multiplexed NOALE : No address multiplexing
waitstate**X**	The wait state selects the wait states for SRAM and SRAM LPC access as a number of peripheral 2x clock cycles. This is a value in the range from 0-7

 While the EBI (External Bus Interface) can be configured to use a big 8 MB or 16 MB SDRAM, the compiler was changed in order to support more then 64KB of RAM (you need BASCOM-AVR Verison 2.0.7.4 or higher).

For 3PORT , 4-bit SDRAM mode the ports are set to the right direction and level. For all other modes you need to do this.
An example on how to determine the columns and rows is shown below:

Table 1: Address Table

	16 Meg x 4	8 Meg x 8	4 Meg x 16
Configuration	4 Meg x 4 x 4 banks	2 Meg x 8 x 4 banks	1 Meg x 16 x 4 banks
Refresh count	4K	4K	4K
Row addressing	4K (A0–A11)	4K (A0–A11)	4K (A0–A11)
Bank addressing	4 (BA0, BA1)	4 (BA0, BA1)	4 (BA0, BA1)
Column addressing	1K (A0–A9)	512 (A0–A8)	256 (A0–A7)

In 4 bit data mode, you use 16 Meg x 4, the row addressing is A0-A11 thus 12 bit and the column addressing is A0-A9 thus 10 bit.

See also
$XA 713 , $WAITSTATE 712, Memory Usage 267, Adding Xram 251

ASM
NONE

Example
```
CONFIG XRAM = Enabled, WaitstateLS=1 , WaitstateHS=2
```

Xmega SRAM Example
```
CONFIG XRAM=3PORT , MODESEL3=SRAM, ADRSIZE3=1M , BASEADR3=&h100000 , ALE
= ALE1 , WAITSTATE3 = 0
```

Xmega Example

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-XRAM-SDRAM-XPLAIN.bas
'  This sample demonstrates the Xmega128A1 XRAM SDRAM
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 64
$framesize = 64
$xramsize = &H800000

'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled


'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

'for xplain we need 9600 baud
Config Com1 = 9600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

Dim B As Byte , B1 As Byte , B2 As Byte
Config Porte = Output
For B = 1 To 5
  Toggle Porte
  Waitms 1000
Next

Print "Xplain SDRAM test"
'the XPLAIN has a 64 MBit SDRAM which is 8 MByte, it is connected in 3
port, 4 bit databus mode
'in the PDF of the SDRAM you can see it is connected as 16 Meg x 4.
Refreshcount is 4K and the row address is A0-A11, column addressing is
A0-A9
Config Xram = 3port , Sdbus = 4 , Sdcol = 10 , Sdcas = 3 , Sdrow = 12 ,
Refresh = 500 , Initdelay = 3200 , Modedelay = 2 , Rowcycledelay = 7 ,
Rowprechargedelay = 7 , Wrdelay = 1 , Esrdelay = 7 , Rowcoldelay = 7 ,
Modesel3 = Sdram , Adrsize3 = 8m , Baseadr3 = &H0000
'the config above will set the port registers correct. it will also wait
for Ebi_cs3_ctrlb.7
'for all other modes you need to do this yourself !

Dim X(65000) As Xram Byte , B as byte

Print "SRAM"
X(10000) = 100                                              ' this will
use normal SRAM
B = X(10000)
Print "result : " ; B

End
```

Another ATXMEGA Example:

'Example to copy a SRAM Array to a XRAM Array over Direct Memory Access (DMA)

```
$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

' for xplain you need 9600 baud
' Config Com1 = 9600 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8

Config Com5 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #1

'SRAM Variables
Dim Ar(100) As Byte , J As Word , W As Word
Dim B As Byte


' Demoboards like XPLAIN has a 64 MBit SDRAM (MT48LC16M4A2TG) which is 8 MByte, it is connected in 3 port, 4 bit databus mode
'
http://www.micron.com/products/ProductDetails.html?product=products/dram/sdram/MT48LC16M4A2TG-75
' in the PDF of the SDRAM you can see it is connected as 16 Meg x 4. Refreshcount is 4K and the row address is A0-A11, column addressing is A0-A9
'SDRAM=SYNCHRONOUS DRAM
Config Xram = 3port , Sdbus = 4 , Sdcol = 10 , Sdcas = 3 , Sdrow = 12 , Refresh = 500 , Initdelay = 3200 , Modedelay = 2 , Rowcycledelay = 7 , Rowprechargedelay = 7 , Wrdelay = 1 , Esrdelay = 7 , Rowcoldelay = 7 , Modesel3 = Sdram , Adrsize3 = 8m , Baseadr3 = &H0000
' the config above will set the port registers correct. it will also wait for Ebi_cs3_ctrlb.7
' for all other modes you need to do this yourself !

$xramsize = 8000000                                         ' 8 MByte

'XRAM Variables
Dim Dummy(100000) As Xram Byte                              'Xram Variable with 100000 Bytes to ensure we are working above 64KByte
Dim Dest(100) As Xram Byte                                  'Next Xram Var with 100 Byte


For J = 1 To 100
   Ar(j) = J                                                ' create an array and assign a value
Next

Print #1 , "Start DMA DEMO --> copy SRAM Array to XRAM Array"
Config Dma = Enabled , Doublebuf = Disabled , Cpm = Rr      ' enable DMA


'you can configure 4 DMA channels
Config Dmach0 = Enabled , Burstlen = 8 , Chanrpt = Enabled , Tci = Off , Eil = Off , Sar = None , Sam = Inc , Dar = None , Dam = Inc , Trigger = 0 , Btc = 100 , Repeat = 1 , Sadr = Varptr(ar(1)) , Dadr = Varptr(dest(1))

Start Dmach0                                                ' this will do a manual/software DMA transfer, when trigger<>0 you can use a hardware event as a trigger source

'--------------------------------------------------------------------------------
For J = 1 To 50
   B = Dest(j)                                              'This step is needed to work with XRAM above 64KByte
   Print #1 , J ; " - " ; Ar(j) ; " - " ; B                 ' print the values
Next

'--------------------------------------------------------------------------------


End

                                    'end program



' (
Terminal Output of example:
```

```
Start DMA DEMO --> copy SRAM Array to XRAM Array
1-1-1
2-2-2
3-3-3
4-4-4
5-5-5
6-6-6
7-7-7
8-8-8
9-9-9
10-10-10
11-11-11
12-12-12
13-13-13
14-14-14
15-15-15
16-16-16
17-17-17
18-18-18
19-19-19
20-20-20
21-21-21
22-22-22
23-23-23
24-24-24
25-25-25
26-26-26
27-27-27
28-28-28
29-29-29
30-30-30
31-31-31
32-32-32
33-33-33
34-34-34
35-35-35
36-36-36
37-37-37
38-38-38
39-39-39
40-40-40
41-41-41
42-42-42
43-43-43
44-44-44
45-45-45
46-46-46
47-47-47
48-48-48
49-49-49
50-50-50
' )
```

7.21.10 CONFIG ZCDx

Action

This statement configures the ZCD(Zero Cross Detector) of the AVRX.

Syntax

CONFIG ZCDx = mode , RUNMODE=runmode, INVERT=invert, OUT_ENABLE=out_enable [, REGMODE=regmode]

Remarks

There can be up to 3 Zero Cross Detectors. The first detector is 0. CONFIG ZCD0 configures the first ZCD.

OPTION	DESCRIPTION
x	Number that identifies the ZCD. Typical from 0-2. Depends on the processor.
mode	- DISABLED : The ZCD is disabled.

	- ENABLED : The ZCD is enabled. Enabling the ZCD
runmode	- DISABLED : The ZCD is only active when required. - ENABLED : the ZCD remains active when the device enters standby sleep mode.
invert	- DISABLED : The output pin is normal. - ENABLED : The output pin has inverted output
out_enable	- DISABLED : The output pin is not connected to the supported pin - ENABLED : The output pin is connected to the supported pin
regmode	- OVERWRITE : this is the default mode. When using 1 option, the other bits are not preserved and set back to 0(disable). - PRESERVE : bits that are not changed are preserved. See also the AVRX 503 description.

You can check the zero cross output in the ZCDx_STATUS register bit 4.

The ZCD has an interrupt as well. It can be triggered on the rising or falling edge. Or on both. By default interrupts are disabled.
Since there is only one interrupt with 3 settings, you can use ENABLE ZCD0_RISING , ZCD0_FALLING or ZCD0_BOTH. They all trigger the same interrupt.
When you use DISABLE ZCD0_RISING it will disable the interrupt. It does not matter which name you use. We do advise to keep them the same for clarity.
Thus when you ENABLE ZCD0_RISING, use DISABLE ZCD0_RISING to disable when required.

See also
NONE

Example

7.22 CONTINUE

Action
The CONTINUE statement will skip code inside a loop till the end of the loop.

Syntax
CONTINUE

Remarks
CONTINUE must be used inside a DO-LOOP, WHILE-WEND or FOR-NEXT loop.
The code jump is always inside the current loop.
Some times you want to skip some code without leaving a loop. You can solve this with a GOTO and a label but use of GOTO creates hard to understand code. For this reason some languages have the CONTINUE statement.

DO-LOOP
```
DO
  some code here
  some code here
```

CONTINUE_WILL_JUMP_TO_THIS_POINT
LOOP

WHILE-WEND

WHILE <CONDIITON>
  some code here
  some code here
CONTINUE_WILL_JUMP_TO_THIS_POINT
WEND

FOR-NEXT

FOR VAR=START TO END
  some code here
  some code here
CONTINUE_WILL_JUMP_TO_THIS_POINT
NEXT

See also

EXIT 1257 , REDO 1422

Example

```
'-----------------------------------------------------------------------
---------------------------------------
'                                    REDO and CONTINUE example
'
'-----------------------------------------------------------------------
---------------------------------------
$regfile = "m128def.dat"
$hwstack = 32
$swstack = 16
$FrameSize = 24


dim b as byte
const test = 0

#if test = 0
   for b = 1 to 10
'when REDO is used, the code will continue here
      print b
      if b = 3 then
         continue                                                ' when b
becomes 3,  the code will continue at the NEXT statement
      end if
      if b = 9 then exit for
      if b = 8 then
         redo                                                  ' when b
becomes 8, the code will continue after the FOR statement, it will not
increase the variable B !
      'so in this example the loop will be forever
```

```
      end if
      print b
    'code continues here when CONTINUE is used
   next

#elseif test = 1
   b = 0
   do
      incr b
      if b = 2 then
         continue
      elseif b = 3 then
         redo
      end if
   loop until b > 5

#elseif test = 2
   b = 0
   while b < 5
      incr b
      if b = 2 then
         continue
      elseif b = 3 then
         redo
      end if
   wend
#endif
end
```

7.23 CONST

Action

Declares a symbolic constant.

Syntax

CONST symbol = numconst
CONST symbol = stringconst
CONST symbol = expression

Remarks

Symbol	The name of the symbol.
Numconst	The numeric value to assign to the symbol.
Stringconst	The string to assign to the symbol
Expression	An expression that returns a value to assign the constant

Assigned constants consume no program memory because they only serve as a reference to the compiler.
The compiler will replace all occurrences of the symbol with the assigned value.

You can use a constant to give a value a more meaningful name.
For example :

```
variable = 1
const optHeaterOn = 1
variable = optHeaterOn
```

The source code is better to read when you assign a constant. Even better when the values change later, for example when HeaterOn becomes 2, you only need to replace 1 line of code.

See also

ALIAS 735

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : const.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo for constants
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'dimension some variables
Dim Z As String * 10
Dim B As Byte

'assign some constants
'constants dont use program memory
Const S = "test"
Const A = 5                                                'declare a
as a constant
Const B1 = &B1001

'or use an expression to assign a constant
Const X =(b1 * 3) + 2
Const Ssingle = Sin(1)


Print X
Print Ssingle


B = A
'the same as b = 5
```

```
Z = S
'the same as Z = "test"

Print A
Print B1
Print S


'you can use constants with conditional compilation
#if A = 5                                                    ' note there
is no then
  Print "constant a is 5"
  #if S = "test"
    Print "nested example"
  #else                                                      ' else is
optional
  #endif
#else
#endif
End
```

7.24 COUNTER0 and COUNTER1

Action

Set or retrieve the internal 16 bit hardware register.

Syntax

COUNTER0 = var var = COUNTER0	TIMER0 can also be used
COUNTER1 = var var = COUNTER1	TIMER1 can also be used
CAPTURE1 = var var = CAPTURE1	TIMER1 capture register
COMPARE1A = var var = COMPARE1A	TIMER1 COMPARE A register
COMARE1B = var var = COMPARE1B	TIMER1 COMPARE B register
PWM1A = var var = PWM1A	TIMER1 COMPAREA register. (Is used for PWM)
PWM1B = var var = PRM1B	TIMER1 COMPARE B register. (Is used for PWM)

Remarks

Var	A byte, Integer/Word variable or constant that is assigned to the register or is read from the register.

Because the above 16 bit register pairs must be accessed somewhat differently than you may expect, they are implemented as variables.

The exception is TIMER0/COUNTER0, this is a normal 8 bit register and is supplied for compatibility with the syntax.

When the CPU reads the low byte of the register, the data of the low byte is sent to

the CPU and the data of the high byte is placed in a temp register. When the CPU reads the data in the high byte, the CPU receives the data in the temp register.

When the CPU writes to the high byte of the register pair, the written data is placed in a temp register. Next when the CPU writes the low byte, this byte of data is combined with the byte data in the temp register and all 16 bits are written to the register pairs. So the MSB must be accessed first.

All of the above is handled automatically by BASCOM when accessing the above registers.
Note that the available registers may vary from chip to chip.

The BASCOM documentation used the 90S8515 to describe the different hardware registers.

7.25 CPEEK

Action
Returns a byte stored in code memory.

Syntax
var = **CPEEK**(address)

Remarks

Var	Numeric variable that is assigned with the content of the program memory at **address.** The cpeek() function returns one BYTE.
Address	Numeric variable or constant with the byte address location.

So what is code memory? Code memory is the same as the flash memory where your program code is stored.
That is not the same memory as the EEPROM memory!
The code memory is exactly the same as the BIN file that the compiler creates.
So why is Cpeek() useful ? You could read the memory and perform a checksum to see if the code is valid.
Or you could check if a boot loader is present in the code.

There is no CPOKE statement because you can not write into program/code memory. Only a boot loader(a piece of code in a special area of the code memory) can write to the normal code memory.

Cpeek(0) will return the first byte of the flash code memory. Cpeek(1) will return the second byte of the flash code memory.
Cpeek() is limited to the first 64 KB of the code memory. For processors that have larger flash code memory like the Mega128 (128KB) you can use CpeekH 1174().

While the AVR uses word addresses since all instructions are 2 bytes long, the Cpeek () function uses a byte address. You need to take that in consideration with for example a boot loader address. The Atmel data sheet will only mention word addresses. For example boot loader address $1000 in the data sheet is $2000 and $2001 byte address for Cpeek().

See also

PEEK 1395 , CPEEKH 1174 , POKE 1396 , INP 1315 , OUT 1394, SETREG 1441, GETREG 1284

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : peek.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates PEEk, POKE, CPEEK, INP and OUT
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31                                            'only 32
registers in AVR
  B1 = Peek(i)                                             'get byte
from internal memory
  Print Hex(b1) ; "  ";
  'Poke I , 1                              'write a value into memory
Next
Print                                                      'new line
'be careful when writing into internal memory !!

'now dump a part ofthe code-memory(program)
For I = 0 To 255
  B1 = Cpeek(i)                                            'get byte
from internal memory
  Print Hex(b1) ; "  ";
Next
'note that you can not write into codememory!!

Out &H8000 , 1                                             'write 1
into XRAM at address 8000
B1 = Inp(&H8000)                                           'return
value from XRAM
Print B1
End
```

7.26 CPEEKH

Action

Returns a byte stored in code memory of micro processors with more then 64KB such as M103, M128.

Syntax
var = **CPEEKH**(address [,page])

Remarks

Var	Numeric variable that is assigned with the content of the program memory at address. One byte is returned by the function.
address	Numeric variable or constant with the byte address location.
page	A numeric variable or constant with the page address. Each page is 64 KB. Thus for the first 64 KB you would specify 0. For the second 64 KB you would specify 1.

The similar Cpeek() function only works on the first 64 KB page. It was intended for processors with memory up to 64 KB.
When processors were made by Atmel with larger memory like the Mega128 (128 KB) the cpeekH() function was added.
The CpeekH() function uses the ELPM instruction instead of the LPM instruction that Cpeek() uses.

Since the memory is broken up in page of 64 KB, the cpeekH() function also access the memory in pages.
You can also omit the page number in which case the compiler will calculate the proper page address.

CpeekH(address,0) will work on the first page (first 64 KB)
CpeekH(address,1) will work on the second page (second 64 KB)

 When omitting the page, the compiler will calculate and load the page register automatically.

While the AVR uses word addresses since all instructions are 2 bytes long, the Cpeek () function uses a byte address. You need to take that in consideration with for example a boot loader address. The Atmel data sheet will only mention word addresses. For example boot loader address $1000 in the data sheet is $2000 and $2001 byte address for Cpeek().

See also
PEEK 1395 , POKE 1396 , INP 1315 , OUT 1394 , CPEEK 1173

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : peek.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates PEEk, POKE, CPEEK, INP and OUT
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
```

```
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31                                             'only 32
registers in AVR
  B1 = Peek(i)                                              'get byte
from internal memory
  Print Hex(b1) ; "  ";
  'Poke I , 1                              'write a value into memory
Next
Print                                                       'new line
'be careful when writing into internal memory !!

'now dump a part ofthe code-memory(program)
For I = 0 To 255
  B1 = Cpeek(i)                                             'get byte
from internal memory
  Print Hex(b1) ; "  ";
Next
'note that you can not write into codememory!!

Out &H8000 , 1                                              'write 1
into XRAM at address 8000
B1 = Inp(&H8000)                                            'return
value from XRAM
Print B1
End
```

7.27 CRYSTAL

Action

Special byte variable that can be used with software UART routine to change the baud rate during runtime.

Syntax

CRYSTAL = var (old option do not use !!)

```
___CRYSTAL1 = var
BAUD #1, 2400
```

Remarks

With the software UART you can generate good baud rates. But chips such as the ATtiny22 have an internal 1 MHz clock. The clock frequency can change during runtime by influence of temperature or voltage.

The crystal variable can be changed during runtime to change the baud rate.

The above has been changed in version 1.11
Now you still can change the baud rate with the crystal variable.
But you don't need to dimension it. And the name has been changed:

___CRYSTALx where x is the channel number.

When you opened the channel with #1, the variable will be named ___CRYSTAL1

But a better way is provided now to change the baud rate of the software uart at run time. You can use the BAUD option now:

Baud #1 , 2400 'change baud rate to 2400 for channel 1

When you use the baud # option, you must specify the baud rate before you print or use input on the channel. This will dimension the ___CRYSTALx variable and load it with the right value.

When you don't use the BAUD # option the value will be loaded from code and it will not use 2 bytes of your SRAM.

The ___CRYSTALx variable is hidden in the report file because it is a system variable. But you may assign a value to it after BAUD #x, zzzz has dimensioned it.

The old CRYSTAL variable does not exist anymore.

Some values for 1 MHz internal clock :
66 for 2400 baud
31 for 4800 baud
14 for 9600 baud

See also
OPEN 1386 , CLOSE 1386

Example
```
Dim B as byte
Open "comd.1:9600,8,n,1,inverted" For Output As #1
Print #1 , B
Print #1 ,"serial output"
baud #1, 4800 'use 4800 baud now
Print #1,"serial output"
___CRYSTAL1 = 255
Close#1
End
```

7.28 DATA

Action
Specifies constant values to be read by subsequent READ statements.

Syntax
DATA var [, varn]

Remarks

Var	Numeric or string constant.

The DATA related statements use the internal registers pair R8 and R9 to store the data pointer.

To store a " sign on the data line, you can use :
```
DATA $34
```

The $-sign tells the compiler that the ASCII value will follow.
You can use this also to store special characters that can't be written by the editor such as chr(7)

Another way to include special ASCII characters in your string constant is to use {XXX}. You need to include exactly 3 digits representing the ASCII character. For example 65 is the ASCII number for the character A.

```
DATA "TEST{065}"

Will be read as TESTA.


While :
DATA "TEST{65}" will be read as :

TEST{65}. This because only 2 digits were included instead of 3.

{xxx} works only for string constants. It will also work in a normal string assignm

s = "{065}" . This will assign A to the string s.
```

Because the DATA statements allow you to generate an EEP file to store in EEPROM, the $DATA 626 and $EEPROM 631 directives have been added. Read the description of these directives to learn more about the DATA statement.

The DATA statements must not be accessed by the flow of your program because the DATA statements are converted to the byte representation of the DATA.

When your program flow enters the DATA lines, unpredictable results will occur.
So as in QB, the DATA statement is best be placed at the end of your program or in a place that program flow will no enter.

For example this is fine:

```
Print "Hello"
Goto jump
DATA "test"

Jump:
'because we jump over the data lines there is no problem.

The following example will case some problems:
Dim S As String * 10
Print "Hello"
Restore lbl
Read S

DATA "test"

Print S
```

When the END statement is used it must be placed BEFORE the DATA lines.

When you have multiple labels with data you need to be aware that each time a label is used, previous data will be aligned to a word. This because the AVR has a word address. This means that :
abc:
DATA 1
klm:
DATA 2

Will consume not 2 bytes but 2 words.

But :
abc:
DATA 1,2
klm:
DATA 3,4

Will also consume 4 bytes. When RESTORE is used, the label address is used which is a word. So take care to put labels only at places which need to be RESTORED/READ.

Difference with QB

Integer and Word constants must end with the **%**-sign.
Long and Dword constants must end with the **&**-sign.
Single constants must end with the **!**-sign.
Double constants must end with the **#**-sign.

See also

READ 1413 , RESTORE 1429 , $DATA 626 , $EEPROM 631 , LOOKUP 1365, LOOKUPSTR 1366 , LOOKDOWN 1363 , $USER 711

Example

```
'-----------------------------------------------------------------------
-----------------
'name                    : readdata.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demo : READ,RESTORE
'micro                   : Mega48
'suited for demo         : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space
```

```
Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long
Restore Dta1                                                    'point to
stored data
For Count = 1 To 3                                              'for number
of data items
   Read B1 : Print Count ; "  " ; B1
Next

Restore Dta2                                                    'point to
stored data
For Count = 1 To 2                                              'for number
of data items
   Read A : Print Count ; "  " ; A
Next

Restore Dta3
Read S : Print S
Read S : Print S


Restore Dta4
Read L : Print L                                                'long type


'demonstration of readlabel
Dim W As Iram Word At 8 Overlay                                 ' location
is used by restore pointer
'note that W does not use any RAM it is an overlayed pointer to the data
pointer
W = Loadlabel(dta1)                                             ' loadlabel
expects the labelname
Read B1
Print B1
End


Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%

Dta3:
Data "Hello" , "World"
'Note that integer values (>255 or <0) must end with the %-sign
'also note that the data type must match the variable type that is
'used for the READ statement

Dta4:
Data 123456789&
'Note that LONG values must end with the &-sign
'Also note that the data type must match the variable type that is used
'for the READ statement
```

7.29 Date and Time

7.29.1 DAYOFWEEK

Action
Returns the Day of the Week of a Date.

Syntax
Target = **DayOfWeek**()
Target = **DayOfWeek**(bDayMonthYear)
Target = **DayOfWeek**(strDate)
Target = **DayOfWeek**(wSysDay)
Target = **DayOfWeek**(lSysSec)

Remarks

Target	A Byte – variable, that is assigned with the day of the week
BDayMonthYear	A Byte – variable, which holds the Day-value followed by Month (Byte) and Year (Byte)
StrDate	A String, which holds a Date-String in the format specified in the CONFIG DATE statement
WSysDay	A Word – variable, which holds the System Day (SysDay)
LSysSec	A Long – variable, which holds the System Second (SysSec)

The Function can be used with five different kind of Input:

1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Week can be calculated of every date.
3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement
4. With a System Day – Number.
5. With a System Second - Number

The Return-Value is in the range of 0 to 6, Monday starts with 0.

The Function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See Also
Date and Time routines 1835 , CONFIG DATE 925 , CONFIG CLOCK 895, SYSDAY 1206, SYSSEC 1204

Example
```
'-----------------------------------------------------------------------
-----------------
'name                    : datetime_test1,bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : show how to use the Date-Time routines from
the DateTime.Lib
'micro                   : Mega103
'suited for demo         : no
```

```
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Const Clockmode = 1
'use i2c for the clock

#if Clockmode = 1
  Config Clock = Soft                                       ' we use
build in clock
  Disable Interrupts
#else
  Config Clock = User                                       ' we use I2C
for the clock
  'configure the scl and sda pins
  Config Sda = Portd.6
  Config Scl = Portd.5

  'address of ds1307
  Const Ds1307w = &HD0                                      ' Addresses
of Ds1307 clock
  Const Ds1307r = &HD1
#endif


'configure the date format
Config Date = Ymd , Separator = -                           ' ANSI-
Format
'This sample does not have the clock started so interrupts are not
enabled
' Enable Interrupts

'dim the used variables
Dim Lvar1 As Long
Dim Mday As Byte
Dim Bweekday As Byte , Strweekday As String * 10
Dim Strdate As String * 8
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Bday As Byte , Bmonth As Byte , Byear As Byte
Dim Lsecofday As Long
Dim Wsysday As Word
Dim Lsyssec As Long
Dim Wdayofyear As Word



' ================== DayOfWeek
============================================
' Example 1 with internal RTC-Clock
```

```
_day = 4 : _month = 11 : _year = 2                              ' Load RTC-
Clock for example - testing
Bweekday = Dayofweek()
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of " ; Date$ ; " is " ; Bweekday ; " = " ;
Strweekday


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 26 : Bmonth = 11 : Byear = 2
Bweekday = Dayofweek(bday)
Strweekday = Lookupstr(bweekday , Weekdays)
Strdate = Date(bday)
Print "Weekday-Number of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " is " ; Bweekday ; " (" ; Date(bday) ; ") = " ; Strweekday


' Example 3 with System Day
Wsysday = 2000                                                  ' that is
2005-06-23
Bweekday = Dayofweek(wsysday)
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
 ") is " ; Bweekday ; " = " ; Strweekday


' Example 4 with System Second
Lsyssec = 123456789                                             ' that is
2003-11-29 at 21:33:09
Bweekday = Dayofweek(lsyssec)
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of System Second " ; Lsyssec ; " (" ; Date(lsyssec
) ; ") is " ; Bweekday ; " = " ; Strweekday



' Example 5 with Date-String
Strdate = "04-11-02"                                            ' we have
configured Date in ANSI
Bweekday = Dayofweek(strdate)
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of " ; Strdate ; " is " ; Bweekday ; " = " ;
Strweekday



' ================= Second of Day
=============================================
' Example 1 with internal RTC-Clock
_sec = 12 : _min = 30 : _hour = 18                              ' Load RTC-
Clock for example - testing

Lsecofday = Secofday()
Print "Second of Day of " ; Time$ ; " is " ; Lsecofday


' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
Bsec = 20 : Bmin = 1 : Bhour = 7
Lsecofday = Secofday(bsec)
Print "Second of Day of Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour
```

```
; " (" ; Time(bsec) ; ") is " ; Lsecofday


' Example 3 with System Second
Lsyssec = 1234456789
Lsecofday = Secofday(lsyssec)
Print "Second of Day of System Second " ; Lsyssec ; "(" ; Time(lsyssec)
; ") is " ; Lsecofday


' Example 4 with Time - String
Strtime = "04:58:37"
Lsecofday = Secofday(strtime)
Print "Second of Day of " ; Strtime ; " is " ; Lsecofday



' ================== System Second
=========================================

' Example 1 with internal RTC-Clock
                          ' Load RTC-Clock for example - testing
_sec = 17 : _min = 35 : _hour = 8 : _day = 16 : _month = 4 : _year = 3

Lsyssec = Syssec()
Print "System Second of " ; Time$ ; " at " ; Date$ ; " is " ; Lsyssec


' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day /
Month / Year)
Bsec = 20 : Bmin = 1 : Bhour = 7 : Bday = 22 : Bmonth = 12 : Byear = 1
Lsyssec = Syssec(bsec)
Strtime = Time(bsec)
Strdate = Date(bday)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec


' Example 3 with System Day

Wsysday = 2000
Lsyssec = Syssec(wsysday)
Print "System Second of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
" 00:00:00) is " ; Lsyssec


' Example 4 with Time and Date String
Strtime = "10:23:50"
Strdate = "02-11-29"                                        ' ANSI-Date
Lsyssec = Syssec(strtime , Strdate)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec       ' 91880630



' ==================== Day Of Year
=========================================
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2                         ' Load RTC-
Clock for example - testing
Wdayofyear = Dayofyear()
Print "Day Of Year of " ; Date$ ; " is " ; Wdayofyear
```

```
' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wdayofyear = Dayofyear(bday)
Print "Day Of Year of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wdayofyear


' Example 3 with Date - String
Strdate = "04-10-29"                                        ' we have
configured ANSI Format
Wdayofyear = Dayofyear(strdate)
Print "Day Of Year of " ; Strdate ; " is " ; Wdayofyear


' Example 4 with System Second

Lsyssec = 123456789
Wdayofyear = Dayofyear(lsyssec)
Print "Day Of Year of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
 ") is " ; Wdayofyear


' Example 5 with System Day
Wsysday = 3000
Wdayofyear = Dayofyear(wsysday)
Print "Day Of Year of System Day " ; Wsysday ; " (" ; Date(wsysday) ; ")
is " ; Wdayofyear


' ================== System Day ======================================
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2                         ' Load RTC-
Clock for example - testing
Wsysday = Sysday()
Print "System Day of " ; Date$ ; " is " ; Wsysday


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wsysday = Sysday(bday)
Print "System Day of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wsysday


' Example 3 with Date - String
Strdate = "04-10-29"
Wsysday = Sysday(strdate)
Print "System Day of " ; Strdate ; " is " ; Wsysday

' Example 4 with System Second
Lsyssec = 123456789
Wsysday = Sysday(lsyssec)
Print "System Day of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
") is " ; Wsysday


' ================== Time
================================================
' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour)
to Time - String
Bsec = 20 : Bmin = 1 : Bhour = 7
```

```
Strtime = Time(bsec)
Print "Time values: Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; "
converted to string " ; Strtime


' Example 2: Converting System Second  to Time - String
Lsyssec = 123456789
Strtime = Time(lsyssec)
Print "Time of Systemsecond " ; Lsyssec ; " is " ; Strtime


' Example 3: Converting Second of Day to Time - String
Lsecofday = 12345
Strtime = Time(lsecofday)
Print "Time of Second of Day " ; Lsecofday ; " is " ; Strtime


' Example 4: Converting System Second to defined Clock - Bytes (Second /
Minute / Hour)

Lsyssec = 123456789
Bsec = Time(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Sec=" ; Bsec ; " Min="
 ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsyssec) ; ")"



' Example 5: Converting Second of Day to defined Clock - Bytes (Second /
Minute / Hour)
Lsecofday = 12345
Bsec = Time(lsecofday)
Print "Second of Day " ; Lsecofday ; " converted to Sec=" ; Bsec ; "
Min=" ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsecofday) ; ")"

' Example 6: Converting Time-string to defined Clock - Bytes (Second /
Minute / Hour)
Strtime = "07:33:12"
Bsec = Time(strtime)
Print "Time " ; Strtime ; " converted to Sec=" ; Bsec ; " Min=" ; Bmin ;
 " Hour=" ; Bhour



' ============================== Date
=========================================

' Example 1: Converting defined Clock - Bytes (Day / Month / Year) to
Date - String
Bday = 29 : Bmonth = 4 : Byear = 12
Strdate = Date(bday)
Print "Dat values: Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear
; " converted to string " ; Strdate


' Example 2: Converting from System Day to Date - String
Wsysday = 1234
Strdate = Date(wsysday)
Print "System Day " ; Wsysday ; " is " ; Strdate


' Example 3: Converting from System Second to Date String
Lsyssec = 123456789
Strdate = Date(lsyssec)
Print "System Second " ; Lsyssec ; " is " ; Strdate
```

```
' Example 4: Converting SystemDay to defined Clock - Bytes (Day /
Month / Year)

Wsysday = 2000
Bday = Date(wsysday)
Print "System Day " ; Wsysday ; " converted to Day=" ; Bday ; " Month="
; Bmonth ; " Year=" ; Byear ; " (" ; Date(wsysday) ; ")"


' Example 5: Converting Date - String to defined Clock - Bytes (Day /
Month / Year)
Strdate = "04-08-31"
Bday = Date(strdate)
Print "Date " ; Strdate ; " converted to Day=" ; Bday ; " Month=" ;
Bmonth ; " Year=" ; Byear


' Example 6: Converting System Second to defined Clock - Bytes (Day /
Month / Year)
Lsyssec = 123456789
Bday = Date(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Day=" ; Bday ; "
Month=" ; Bmonth ; " Year=" ; Byear ; " (" ; Date(lsyssec) ; ")"



' ================ Second of Day elapsed

Lsecofday = Secofday()
_hour = _hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1

Lsyssec = Syssec()
_day = _day + 1
Lvar1 = Syssecelapsed(lsyssec)
Print Lvar1


Looptest:

' Initialising for testing
_day = 1
_month = 1
_year = 1
_sec = 12
_min = 13
_hour = 14



Do
   If _year > 50 Then
      Exit Do
   End If

  _sec = _sec + 7
  If _sec > 59 Then
     Incr _min
     _sec = _sec - 60
  End If
```

```
      _min = _min + 2
      If _min > 59 Then
         Incr _hour
         _min = _min - 60
      End If

      _hour = _hour + 1
      If _hour > 23 Then
         Incr _day
         _hour = _hour - 24
      End If

      _day = _day + 1


      If _day > 28 Then
         Select Case _month
            Case 1
               Mday = 31
            Case 2
               Mday = _year And &H03
               If Mday = 0 Then
                  Mday = 29
               Else
                  Mday = 28
               End If
            Case 3
               Mday = 31
            Case 4
               Mday = 30
            Case 5
               Mday = 31
            Case 6
               Mday = 30
            Case 7
               Mday = 31
            Case 8
               Mday = 31
            Case 9
               Mday = 30
            Case 10
               Mday = 31
            Case 11
               Mday = 30
            Case 12
               Mday = 31
         End Select
         If _day > Mday Then
            _day = _day - Mday
            Incr _month
            If _month > 12 Then
               _month = 1
               Incr _year
            End If
         End If
      End If
      If _year > 99 Then
         Exit Do
      End If

   Lsecofday = Secofday()
   Lsyssec = Syssec()
   Bweekday = Dayofweek()
```

```
Wdayofyear = Dayofyear()
Wsysday = Sysday()


Print Time$ ; " " ; Date$ ; " " ; Lsecofday ; " " ; Lsyssec ; " " ;
Bweekday ; " " ; Wdayofyear ; " " ; Wsysday


Loop
End


'only when we use I2C for the clock we need to set the clock date time
#if Clockmode = 0
'called from datetime.lib
Dim Weekday As Byte
Getdatetime:
  I2cstart                                                ' Generate
start code
  I2cwbyte Ds1307w                                        ' send
address
  I2cwbyte 0                                              ' start
address in 1307

  I2cstart                                                ' Generate
start code
  I2cwbyte Ds1307r                                        ' send
address
  I2crbyte _sec , Ack
  I2crbyte _min , Ack                                     ' MINUTES
  I2crbyte _hour , Ack                                    ' Hours
  I2crbyte Weekday , Ack                                  ' Day of
Week
  I2crbyte _day , Ack                                     ' Day of
Month
  I2crbyte _month , Ack                                   ' Month of
Year
  I2crbyte _year , Nack                                   ' Year
  I2cstop
  _sec = Makedec(_sec) : _min = Makedec(_min) : _hour = Makedec(_hour)
  _day = Makedec(_day) : _month = Makedec(_month) : _year = Makedec(
_year)
Return

Setdate:
  _day = Makebcd(_day) : _month = Makebcd(_month) : _year = Makebcd(
_year)
  I2cstart                                                ' Generate
start code
  I2cwbyte Ds1307w                                        ' send
address
  I2cwbyte 4                                              ' starting
address in 1307
  I2cwbyte _day                                           ' Send Data
to SECONDS
  I2cwbyte _month                                         ' MINUTES
  I2cwbyte _year                                          ' Hours
  I2cstop
Return

Settime:
  _sec = Makebcd(_sec) : _min = Makebcd(_min) : _hour = Makebcd(_hour)
  I2cstart                                                ' Generate
start code
```

```
  I2cwbyte Ds1307w                                          ' send
address
  I2cwbyte 0                                                ' starting
address in 1307
  I2cwbyte _sec                                             ' Send Data
to SECONDS
  I2cwbyte _min                                             ' MINUTES
  I2cwbyte _hour                                            ' Hours
  I2cstop
Return

#endif


Weekdays:
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" ,
"Saturday" , "Sunday"
```

7.29.2 DAYOFYEAR

Action

Returns the Day of the Year of a Date

Syntax

Target = **DayOfYear**()
Target = **DayOfYear**(bDayMonthYear)
Target = **DayOfYear**(strDate)
Target = **DayOfYear**(wSysDay)
Target = **DayOfYear**(lSysSec)

Remarks

Target	A Integer, that is assigned with the Day of the Year
BDayMonthYear	A Byte, which holds the Day-value followed by Month(Byte) and Year (Byte)
StrDate	A String, which holds a Date-String in the format specified in the CONFIG DATE statement
WSysDay	A Variable (Word) which holds a System Day (SysDay)
LsysSec	A Variable (Long) which holds a System Second (SysSec)

The Function can be used with five different kind of Input:

1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Year can be calculated of every date.
3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement.
4. With a System Day Number (WORD)
5. With a System Second Number (LONG)

The Return-Value is in the Range of 0 to 364 (365 in a leap year). January the first starts with 0.

The function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See also
Date and Time Routines 1835 , SysSec 1204 , SysDay 1206

Example
See DayOfWeek 1181

7.29.3 DATE$

Action
Internal variable that holds the date.

Syntax
DATE$ = "mm/dd/yy"
var = **DATE$**

Remarks
The DATE$ variable is used in combination with the CONFIG CLOCK 895 directive.

The CONFIG CLOCK 895 statement will create an interrupt that occurs every second. In this interrupt routine the *_Sec*, *_Min* and *_Hour* variables are updated. The *_dat*, *_month* and *_year* variables are also updated. The date format is in the same format as in VB.

When you assign DATE$ to a string variable these variables are assigned to the DATE$ variable.
When you assign the DATE$ variable with a constant or other variable, the _day, _month and _year variables will be changed to the new date.
The only difference with VB is that all data must be provided when assigning the date. This is done for minimal code. You can change this behavior of course.

 Do not confuse DATE$ with the DATE function !

ASM
The following ASM routines are called.
When assigning DATE$: _set_date (calls _str2byte)
When reading DATE$: _make_dt (calls _byte2str)

See also
TIME$ 1208 , CONFIG CLOCK 895 , DATE 1193

Example
```
'-----------------------------------------------------------------------
-----------------
'name                    : megaclock.bas
```

```
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : shows the new TIME$ and DATE$ reserved
variables
'micro                   : Mega103
'suited for demo         : yes
'commercial addon needed : no
'--------------------------------------------------------------------------
-----------------

$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'With the 8535 and timer2 or the Mega103 and TIMER0 you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code

'This example is written for the STK300 with M103
Enable Interrupts

'[configure LCD]
$lcd = &HC000                                               'address for
E and RS
$lcdrs = &H8000                                             'address for
only E
Config Lcd = 20 * 4                                         'nice
display from bg micro
Config Lcdbus = 4                                           'we run it
in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus                                        'tell about
the bus mode

'[now init the clock]
Config Date = Mdy , Separator = /                           ' ANSI-
Format

Config Clock = Soft                                         'this is how
simple it is
'The above statement will bind in an ISR so you can not use the TIMER
anymore!
'For the M103 in this case it means that TIMER0 can not be used by the
user anymore

'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/00"

'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used

Time$ = "02:20:00"

'-----------------------------------------------
```

```
'clear the LCD display
Cls

Do
  Home                                                    'cursor home
  Lcd Date$ ; "  " ; Time$                                'show the
date and time
Loop

'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
_day = 1
'For the _year variable only the year is stored, not the century
End
```

7.29.4 DATE

Action

Returns a date-value (String or 3 Bytes for Day, Month and Year) depending of the data type of the Target

Syntax

bDayMonthYear = **Date**(lSysSec)
bDayMonthYear = **Date**(lSysDay)
bDayMonthYear = **Date**(strDate)

strDate = **Date**(lSysSec)
strDate = **Date**(lSysDay)
strDate = **Date**(bDayMonthYear)

Remarks

StrDate	A Date-String in the format specified in the CONFIG DATE statement
LsysSec	A LONG – variable which holds the System Second (SysSec = TimeStamp)
LsysDay	A WORD – variable, which holds then System Day (SysDay)
BDayMonthYear	A BYTE – variable, which holds Days, followed by Month (Byte) and Year (Byte). You can use a byte array, or 3 bytes dimensioned after each other.

Converting to String:

 The target string must have a length of at least 8 Bytes, otherwise SRAM after the target-string will be overwritten.

Converting to Soft clock date format (3 Bytes for Day, Month and Year):

Three Bytes for Day, Month and Year must follow each other in SRAM. The variable-name of the first Byte, the one for Day must be passed to the function.

See also

Date and Time Routines 1835 , DAYOFYEAR 1190, SYSDAY 1206

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : datetime_test1,bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : show how to use the Date-Time routines from
the DateTime.Lib
'micro                    : Mega103
'suited for demo          : no
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m103def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Const Clockmode = 1
'use i2c for the clock

#if Clockmode = 1
  Config Clock = Soft                                       ' we use
build in clock
  Disable Interrupts
#else
  Config Clock = User                                       ' we use I2C
for the clock
  'configure the scl and sda pins
  Config Sda = Portd.6
  Config Scl = Portd.5

  'address of ds1307
  Const Ds1307w = &HD0                                      ' Addresses
of Ds1307 clock
  Const Ds1307r = &HD1
#endif


'configure the date format
Config Date = Ymd , Separator = -                           ' ANSI-
Format
'This sample does not have the clock started so interrupts are not
enabled
' Enable Interrupts

'dim the used variables
Dim Lvar1 As Long
Dim Mday As Byte
Dim Bweekday As Byte , Strweekday As String * 10
Dim Strdate As String * 8
```

```
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Bday As Byte , Bmonth As Byte , Byear As Byte
Dim Lsecofday As Long
Dim Wsysday As Word
Dim Lsyssec As Long
Dim Wdayofyear As Word



' ================== DayOfWeek
=============================================
' Example 1 with internal RTC-Clock

_day = 4 : _month = 11 : _year = 2                          ' Load RTC-
Clock for example - testing
Bweekday = Dayofweek()
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of " ; Date$ ; " is " ; Bweekday ; " = " ;
Strweekday


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 26 : Bmonth = 11 : Byear = 2
Bweekday = Dayofweek(bday)
Strweekday = Lookupstr(bweekday , Weekdays)
Strdate = Date(bday)
Print "Weekday-Number of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " is " ; Bweekday ; " (" ; Date(bday) ; ") = " ; Strweekday


' Example 3 with System Day
Wsysday = 2000                                              ' that is
2005-06-23
Bweekday = Dayofweek(wsysday)
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
 ") is " ; Bweekday ; " = " ; Strweekday


' Example 4 with System Second
Lsyssec = 123456789                                         ' that is
2003-11-29 at 21:33:09
Bweekday = Dayofweek(lsyssec)
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of System Second " ; Lsyssec ; " (" ; Date(lsyssec
) ; ") is " ; Bweekday ; " = " ; Strweekday



' Example 5 with Date-String
Strdate = "04-11-02"                                        ' we have
configured Date in ANSI
Bweekday = Dayofweek(strdate)
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of " ; Strdate ; " is " ; Bweekday ; " = " ;
Strweekday
```

```
' ================= Second of Day
=============================================
' Example 1 with internal RTC-Clock
_sec = 12 : _min = 30 : _hour = 18                          ' Load RTC-
Clock for example - testing

Lsecofday = Secofday()
Print "Second of Day of " ; Time$ ; " is " ; Lsecofday


' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
Bsec = 20 : Bmin = 1 : Bhour = 7
Lsecofday = Secofday(bsec)
Print "Second of Day of Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour
; " (" ; Time(bsec) ; ") is " ; Lsecofday


' Example 3 with System Second
Lsyssec = 1234456789
Lsecofday = Secofday(lsyssec)
Print "Second of Day of System Second " ; Lsyssec ; "(" ; Time(lsyssec)
; ") is " ; Lsecofday


' Example 4 with Time - String
Strtime = "04:58:37"
Lsecofday = Secofday(strtime)
Print "Second of Day of " ; Strtime ; " is " ; Lsecofday



' ================== System Second
=============================================

' Example 1 with internal RTC-Clock
                          ' Load RTC-Clock for example - testing
_sec = 17 : _min = 35 : _hour = 8 : _day = 16 : _month = 4 : _year = 3

Lsyssec = Syssec()
Print "System Second of " ; Time$ ; " at " ; Date$ ; " is " ; Lsyssec


' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day /
Month / Year)
Bsec = 20 : Bmin = 1 : Bhour = 7 : Bday = 22 : Bmonth = 12 : Byear = 1
Lsyssec = Syssec(bsec)
Strtime = Time(bsec)
Strdate = Date(bday)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec


' Example 3 with System Day

Wsysday = 2000
Lsyssec = Syssec(wsysday)
Print "System Second of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
" 00:00:00) is " ; Lsyssec


' Example 4 with Time and Date String
Strtime = "10:23:50"
Strdate = "02-11-29"                                        ' ANSI-Date
```

```
Lsyssec = Syssec(strtime , Strdate)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec       ' 91880630



' ==================== Day Of Year
=========================================
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2                          ' Load RTC-
Clock for example - testing
Wdayofyear = Dayofyear()
Print "Day Of Year of " ; Date$ ; " is " ; Wdayofyear


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wdayofyear = Dayofyear(bday)
Print "Day Of Year of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wdayofyear



' Example 3 with Date - String
Strdate = "04-10-29"                                         ' we have
configured ANSI Format
Wdayofyear = Dayofyear(strdate)
Print "Day Of Year of " ; Strdate ; " is " ; Wdayofyear


' Example 4 with System Second

Lsyssec = 123456789
Wdayofyear = Dayofyear(lsyssec)
Print "Day Of Year of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
 ") is " ; Wdayofyear


' Example 5 with System Day
Wsysday = 3000
Wdayofyear = Dayofyear(wsysday)
Print "Day Of Year of System Day " ; Wsysday ; " (" ; Date(wsysday) ; ")
is " ; Wdayofyear



' ==================== System Day ====================================
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2                          ' Load RTC-
Clock for example - testing
Wsysday = Sysday()
Print "System Day of " ; Date$ ; " is " ; Wsysday


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wsysday = Sysday(bday)
Print "System Day of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wsysday


' Example 3 with Date - String
Strdate = "04-10-29"
```

```
Wsysday = Sysday(strdate)
Print "System Day of " ; Strdate ; " is " ; Wsysday

' Example 4 with System Second
Lsyssec = 123456789
Wsysday = Sysday(lsyssec)
Print "System Day of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
") is " ; Wsysday



' ================== Time
================================================
' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour)
to Time - String
Bsec = 20 : Bmin = 1 : Bhour = 7
Strtime = Time(bsec)
Print "Time values: Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; "
converted to string " ; Strtime


' Example 2: Converting System Second  to Time - String
Lsyssec = 123456789
Strtime = Time(lsyssec)
Print "Time of Systemsecond " ; Lsyssec ; " is " ; Strtime


' Example 3: Converting Second of Day to Time - String
Lsecofday = 12345
Strtime = Time(lsecofday)
Print "Time of Second of Day " ; Lsecofday ; " is " ; Strtime


' Example 4: Converting System Second to defined Clock - Bytes (Second /
Minute / Hour)

Lsyssec = 123456789
Bsec = Time(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Sec=" ; Bsec ; " Min="
 ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsyssec) ; ")"



' Example 5: Converting Second of Day to defined Clock - Bytes (Second /
Minute / Hour)
Lsecofday = 12345
Bsec = Time(lsecofday)
Print "Second of Day " ; Lsecofday ; " converted to Sec=" ; Bsec ; "
Min=" ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsecofday) ; ")"

' Example 6: Converting Time-string to defined Clock - Bytes (Second /
Minute / Hour)
Strtime = "07:33:12"
Bsec = Time(strtime)
Print "Time " ; Strtime ; " converted to Sec=" ; Bsec ; " Min=" ; Bmin ;
 " Hour=" ; Bhour



' ============================ Date
========================================

' Example 1: Converting defined Clock - Bytes (Day / Month / Year) to
Date - String
```

```
Bday = 29 : Bmonth = 4 : Byear = 12
Strdate = Date(bday)
Print "Dat values: Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear
; " converted to string " ; Strdate


' Example 2: Converting from System Day to Date - String
Wsysday = 1234
Strdate = Date(wsysday)
Print "System Day " ; Wsysday ; " is " ; Strdate


' Example 3: Converting from System Second to Date String
Lsyssec = 123456789
Strdate = Date(lsyssec)
Print "System Second " ; Lsyssec ; " is " ; Strdate


' Example 4: Converting SystemDay to defined Clock - Bytes (Day /
Month / Year)

Wsysday = 2000
Bday = Date(wsysday)
Print "System Day " ; Wsysday ; " converted to Day=" ; Bday ; " Month="
; Bmonth ; " Year=" ; Byear ; " (" ; Date(wsysday) ; ")"


' Example 5: Converting Date - String to defined Clock - Bytes (Day /
Month / Year)
Strdate = "04-08-31"
Bday = Date(strdate)
Print "Date " ; Strdate ; " converted to Day=" ; Bday ; " Month=" ;
Bmonth ; " Year=" ; Byear


' Example 6: Converting System Second to defined Clock - Bytes (Day /
Month / Year)
Lsyssec = 123456789
Bday = Date(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Day=" ; Bday ; "
Month=" ; Bmonth ; " Year=" ; Byear ; " (" ; Date(lsyssec) ; ")"



' ================ Second of Day elapsed

Lsecofday = Secofday()
_hour = _hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1

Lsyssec = Syssec()
_day = _day + 1
Lvar1 = Syssecelapsed(lsyssec)
Print Lvar1


Looptest:

' Initialising for testing
_day = 1
_month = 1
_year = 1
_sec = 12
```

```
_min = 13
_hour = 14



Do
   If _year > 50 Then
      Exit Do
   End If

  _sec = _sec + 7
  If _sec > 59 Then
     Incr _min
     _sec = _sec - 60
  End If

  _min = _min + 2
  If _min > 59 Then
     Incr _hour
     _min = _min - 60
  End If

  _hour = _hour + 1
  If _hour > 23 Then
     Incr _day
     _hour = _hour - 24
  End If

  _day = _day + 1


  If _day > 28 Then
     Select Case _month
        Case 1
           Mday = 31
        Case 2
           Mday = _year And &H03
           If Mday = 0 Then
              Mday = 29
           Else
              Mday = 28
           End If
        Case 3
           Mday = 31
        Case 4
           Mday = 30
        Case 5
           Mday = 31
        Case 6
           Mday = 30
        Case 7
           Mday = 31
        Case 8
           Mday = 31
        Case 9
           Mday = 30
        Case 10
           Mday = 31
        Case 11
           Mday = 30
        Case 12
           Mday = 31
     End Select
     If _day > Mday Then
```

```
            _day = _day - Mday
            Incr _month
            If _month > 12 Then
               _month = 1
               Incr _year
            End If
         End If
      End If
      If _year > 99 Then
         Exit Do
      End If

Lsecofday = Secofday()
Lsyssec = Syssec()
Bweekday = Dayofweek()
Wdayofyear = Dayofyear()
Wsysday = Sysday()


Print Time$ ; " " ; Date$ ; " " ; Lsecofday ; " " ; Lsyssec ; " " ;
Bweekday ; " " ; Wdayofyear ; " " ; Wsysday


Loop
End


'only when we use I2C for the clock we need to set the clock date time
#if Clockmode = 0
'called from datetime.lib
Dim Weekday As Byte
Getdatetime:
  I2cstart                                                ' Generate
start code
  I2cwbyte Ds1307w                                        ' send
address
  I2cwbyte 0                                              ' start
address in 1307

  I2cstart                                                ' Generate
start code
  I2cwbyte Ds1307r                                        ' send
address
  I2crbyte _sec , Ack
  I2crbyte _min , Ack                                     ' MINUTES
  I2crbyte _hour , Ack                                    ' Hours
  I2crbyte Weekday , Ack                                  ' Day of
Week
  I2crbyte _day , Ack                                     ' Day of
Month
  I2crbyte _month , Ack                                   ' Month of
Year
  I2crbyte _year , Nack                                   ' Year
  I2cstop
  _sec = Makedec(_sec) : _min = Makedec(_min) : _hour = Makedec(_hour)
  _day = Makedec(_day) : _month = Makedec(_month) : _year = Makedec(
_year)
Return

Setdate:
  _day = Makebcd(_day) : _month = Makebcd(_month) : _year = Makebcd(
_year)
  I2cstart                                                ' Generate
```

```
start code
  I2cwbyte Ds1307w                                          ' send
address
  I2cwbyte 4                                                ' starting
address in 1307
  I2cwbyte _day                                             ' Send Data
to SECONDS
  I2cwbyte _month                                           ' MINUTES
  I2cwbyte _year                                            ' Hours
  I2cstop
Return

Settime:
  _sec = Makebcd(_sec) : _min = Makebcd(_min) : _hour = Makebcd(_hour)
  I2cstart                                                  ' Generate
start code
  I2cwbyte Ds1307w                                          ' send
address
  I2cwbyte 0                                                ' starting
address in 1307
  I2cwbyte _sec                                             ' Send Data
to SECONDS
  I2cwbyte _min                                             ' MINUTES
  I2cwbyte _hour                                            ' Hours
  I2cstop
Return

#endif


Weekdays:
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" ,
"Saturday" , "Sunday"
```

7.29.5 SECELAPSED

Action

Returns the elapsed Seconds to a former assigned time-stamp.

Syntax

Target = **SECELAPSED**(TimeStamp)

Remarks

Target	A variable (LONG), that is assigned with the elapsed Seconds
TimeStamp	A variable (LONG), which holds a timestamp like the output of an earlier called SecOfDay()

The Function works with the SOFTCLOCK variables _sec, _min and _hour and considers a jump over midnight and gives a correct result within 24 hour between two events.

The Return-Value is in the range of 0 to 86399.

See also

Date and Time Routines 1835 , SecOfDay 1203 , SysSecElapsed 1206

Partial Example

```
Lsecofday = Secofday()
_hour = _hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1
```

7.29.6 SECOFDAY

Action

Returns the Seconds of a Day.

Syntax

Target = **SECOFDAY**()
Target = **SECOFDAY**(bSecMinHour)
Target = **SECOFDAY**(strTime)
Target = **SECOFDAY**(lSysSec)

Remarks

Target	A variable (LONG), that is assigned with the Seconds of the Day
bSecMinHour	A Byte, which holds the Second-value followed by Minute(Byte) and Hour(Byte)
strTime	A String, which holds the time in the format „hh:mm:ss"
LSysSec	A Variable (Long) which holds the System Second

The Function can be used with 4 different kind of inputs:

1. Without any parameter. The internal Time of SOFTCLOCK (_sec, _min, _hour) is used.
2. With a user defined time array. It must be arranged in same way (Second, Minute, Hour) as the internal SOFTCLOCK time. The first Byte (Second) is the input by this kind of usage. So the Second of Day can be calculated of every time.
3. With a time-String. The time-string must be in the Format „hh:mm:ss".
4. With a System Second Number (LONG)

The Return-Value is in the range of 0 to 86399 from 00:00:00 to 23:59:59.
No validity-check of input is made.

See also

Date and Time Routines 1835 , SysSec 1204

Partial Example

```
' ================ Second of Day
============================================
' Example 1 with internal RTC-Clock
_sec = 12 : _min = 30 : _hour = 18                          ' Load RTC-
Clock for example - testing
```

```
Lsecofday = Secofday()
Print "Second of Day of " ; Time$ ; " is " ; Lsecofday


' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
Bsec = 20 : Bmin = 1 : Bhour = 7
Lsecofday = Secofday(bsec)
Print "Second of Day of Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour
; " (" ; Time(bsec) ; ") is " ; Lsecofday


' Example 3 with System Second
Lsyssec = 1234456789
Lsecofday = Secofday(lsyssec)
Print "Second of Day of System Second " ; Lsyssec ; "(" ; Time(lsyssec)
; ") is " ; Lsecofday

' Example 4 with Time - String
Strtime = "04:58:37"
Lsecofday = Secofday(strtime)
Print "Second of Day of " ; Strtime ; " is " ; Lsecofday
```

7.29.7 SYSSEC

Action

Returns a Number, which represents the System Second

Syntax

Target = **SYSSEC**()
Target = **SYSSEC**(bSecMinHour)
Target = **SYSSEC**(strTime, strDate)
Target = **SYSSEC**(wSysDay)

Remarks

Target	A Variable (LONG), that is assigned with the System-Second
BSecMinHour	A Byte, which holds the Sec-value followed by Min(Byte), Hour (Byte), Day(Byte), Month(Byte) and Year(Byte)
StrTime	A time-string in the format „hh:mm:ss"
StrDate	A date-string in the format specified in the Config Date statement
wSysDay	A variable (Word) which holds the System Day (SysDay)

The Function can be used with 4 different kind of inputs:

1. Without any parameter. The internal Time and Date of SOFTCLOCK (_sec, _min, _hour, _day, _month, _year) is used.
2. With a user defined time and Date array. It must be arranged in same way (Second, Minute, Hour, Day, Month, Year) as the internal SOFTCLOCK time/ date. The first Byte (Second) is the input by this kind of usage. So the System Second can be calculated of every time/date.
3. With a time-String and a date-string. The time-string must be in the Format „hh:mm:ss". The date-string must be in the format specified in the Config Date statement
4. With a System Day Number (Word). The result is the System Second of this day at 00:00:00.

The Return-Value is in the Range of 0 to 2147483647. 2000-01-01 at 00:00:00 starts with 0.
The Function is valid from 2000-01-01 to 2068-01-19 03:14:07. In the year **2068** a LONG – overflow will occur.

Unix time stamp starts 1-1-1970 which will limit the use till 2038.
Bascom time stamp starts 1-1-2000 giving longer working time.

If you wish to convert to NTP which starts at 1.1.1970, which is 30 years earlier, you need to subtract a value of 946684800
BASCOM DATE_TIME = NTP - 946684800

See also

Date and Time Routines 1835 , SYSSECELAPSED 1206, SYSDAY 1206

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator =.' ANSI-Format

Dim Strdate As String * 8
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Bday As Byte , Bmonth As Byte , Byear As Byte
Dim Wsysday As Word
Dim Lsyssec As Long

' Example 1 with internal RTC-Clock
' Load RTC-Clock for example - testing
_sec = 17 : _min = 35 : _hour = 8 : _day = 16 : _month = 4 : _year = 3
Lsyssec = Syssec()
Print "System Second of " ; Time$ ; " at " ; Date$ ; " is " ; Lsyssec
' System Second of 08:35:17 at 03.04.16 is 103797317

' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day /
Month / Year)
Bsec = 20 : Bmin = 1 : Bhour = 7 : Bday = 22 : Bmonth = 12 : Byear = 1
Lsyssec = Syssec(bsec)
Strtime = Time_sb(bsec) : Strdate = Date_sb(bday)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec
' System Second of 07:01:20 at 01.12.22 is 62319680

' Example 3 with Time and Date - String
Strtime = "04:58:37"
strDate ="02.09.18"
Lsyssec = Syssec(strtime , Strdate)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec
' System Second of 04:58:37 at 02.09.18 is 85640317

' Example 4 with System Day
Wsysday = 2000
Lsyssec = Syssec(wsysday)
Print "System Second of System Day " ; Wsysday ; " (00:00:00) is " ;
Lsyssec
' System Second of System Day 2000 (00:00:00) is 172800000
```

7.29.8 SYSSECELAPSED

Action
Returns the elapsed Seconds to a earlier assigned system-time-stamp.

Syntax
Target = **SysSecElapsed**(SystemTimeStamp)

Remarks

Target	A variable (LONG), that is assigned with the elapsed Seconds
SystemTimeStamp	A variable (LONG), which holds a Systemtimestamp like the output of an earlier called SysSec()

The Return-Value is in the Range of 0 to 2147483647. The Function is valid from 2000-01-01 to 2068-01-19 at 03:14:07. In the year 2068 a LONG – overflow will occur.

The difference to the pair DayOfSec and SecElapsed is, that SysSec and SysSecElapsed can be used for event distances larger than 24 hours.

See also
Date and Time Routines 1835 , SECELAPSED 1202, SYSSEC 1204

Example
```
Enable Interrupts
Config Clock = Soft

Dim Lsystemtimestamp As Long
Dim Lsystemsecondselapsed As Long

Lsystemtimestamp = Syssec()
Print "Now it's " ; Lsystemtimestamp ; " seconds past 2000-01-01
00:00:00"

' do other stuff
' some time later

Lsystemsecondselapsed = Syssecelapsed(lsystemtimestamp)
Print "Now it's " ; Lsystemsecondselapsed ; " seconds later"
```

7.29.9 SYSDAY

Action
Returns a number, which represents the System Day

Syntax
Target = **SysDay**()
Target = **SysDay**(bDayMonthYear)
Target = **SysDay**(strDate)
Target = **SysDay**(lSysSec)

Remarks

Target	A Variable (WORD), that is assigned with the System-Day
bDayMonthDay	A Byte, which holds the Day-value followed by Month(Byte) and Year (Byte)
strDate	A String, which holds a Date-String in the format specified in the CONFIG DATA statement
lSysSec	A variable, which holds a System Second (SysSec)

The Function can be used with 4 different kind of inputs:

1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Year can be calculated of every date.
3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement.
4. With a System Second Number (LONG)

The Return-Value is in the Range of 0 to 36524. 2000-01-01 starts with 0.
The Function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See also

Date and Time Routines 1835 , Config Date 925 , Config Clock 895 , SysSec 1204

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator =.' ANSI-Format

Dim Strdate As String * 8
Dim Bday Asbyte , Bmonth As Byte , Byear As Byte
Dim Wsysday As Word
Dim Lsyssec As Long


' Example 1 with internal RTC-Clock
_day = 20 : _Month = 11 : _Year = 2 ' Load RTC-Clock for example -
testing
Wsysday = Sysday()
Print "System Day of " ; Date$ ; " is " ; Wsysday

' System Day of 02.11.20 is 1054


' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wsysday = Sysday(bday)
Print "System Day of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " is " ; Wsysday
' System Day of Day=24 Month=5 Year=8 is 3066


' Example 3 with Date - String
```

```
Strdate = "04.10.29"
Wsysday = Sysday(strdate)
Print "System Day of " ; Strdate ; " is " ; Wsysday
' System Day of 04.10.29 is 1763

' Example 4 with System Second
Lsyssec = 123456789
Wsysday = Sysday(lsyssec)
Print "System Day of System Second " ; Lsyssec ; " is " ; Wsysday
' System Day of System Second 123456789 is 1428"Now it's " ;
Lsystemsecondselapsed ; " seconds later"
```

7.29.10 TIME$

Action
Internal variable that holds the time.

Syntax
TIME$ = "hh:mm:ss"
var = **TIME$**

Remarks
The TIME$ variable is used in combination with the CONFIG CLOCK and CONFIG DATE directive.

See CONFIG CLOCK 895 statement for further information. In this interrupt routine the _Sec, _Min and _Hour variables are updated. The time format is 24 hours format.

When you assign TIME$ to a string variable these variables are assigned to the TIME$ variable.
When you assign the TIME$ variable with a constant or other variable, the _sec, _Hour and _Min variables will be changed to the new time.

The only difference with VB is that all digits must be provided when assigning the time. This is done for minimal code. You can change this behavior of course.

 Do not confuse TIME$ with the TIME function !

ASM
The following asm routines are called from mcs.lib.
When assigning TIME$: _set_time (calls _str2byte)
When reading TIME$: _make_dt (calls _byte2str)

See also
DATE$ 1191 , CONFIG CLOCK 895 , CONFIG DATE 925

Example
See the sample of DATE$ 1191

7.29.11 TIME

Action
Returns a time-value (String or 3 Byte for Second, Minute and Hour) depending of the Type of the Target

Syntax
bSecMinHour = **Time**(lSecOfDay)
bSecMinHour = **Time**(lSysSec)
bSecMinHour = **Time**(strTime)

strTime = **Time**(lSecOfDay)
strTime = **Time**(lSysSec)
strTime = **Time**(bSecMinHour)

Remarks

bSecMinHour	A BYTE – variable, which holds the Second-value followed by Minute (Byte) and Hour (Byte)
strTime	A Time – String in Format „hh:mm:ss"
lSecOfDay	A LONG – variable which holds Second Of Day (SecOfDay)
lSysSec	A LONG – variable which holds System Second (SysSec)

Converting to a time-string:
The target string strTime must have a length of at least 8 Bytes, otherwise SRAM after the target-string will be overwritten.

Converting to Softclock format (3 Bytes for Second, Minute and Hour):
Three Bytes for Seconds, Minutes and Hour must follow each other in SRAM. The variable-name of the first Byte, that one for Second must be passed to the function.

 Time not to be confused with Time**$** !

See also
Date and Time Routines 1835 , SECOFDAY 1203, SYSSEC 1204

Partial Example
```
Enable Interrupts
Config Clock = Soft

Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte AT Bsec + 1 , Bhour As Byte AT Bmin +1
Dim Lsecofday As Long
Dim Lsyssec As Long


' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour)
to Time - String
Bsec = 20 : Bmin = 1 : Bhour = 7
Strtime = Time(bsec)
Print "Time values: Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; "
converted to string " ; Strtime
```

```
' Time values: Sec=20 Min=1 Hour=7 converted to string 07:01:20


' Example 2: Converting System Second to Time - String
Lsyssec = 123456789
Strtime = Time(lsyssec)
Print "Time of Systemsecond " ; Lsyssec ; " is " ; Strtime

' Time of Systemsecond 123456789 is 21:33:09


' Example 3: Converting Second of Day to Time - String
Lsecofday = 12345
Strtime = Time(lsecofday)
Print "Time of Second of Day " ; Lsecofday ; " is " ; Strtime
' Time of Second of Day 12345 is 03:25:45


' Example 4: Converting System Second to defined Clock - Bytes (Second /
Minute / Hour)
Lsyssec = 123456789
Bsec = Time(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Sec=" ; Bsec ; " Min="
 ; Bmin ; " Hour=" ; Bhour

' System Second 123456789 converted to Sec=9 Min=33 Hour=21


' Example 4: Converting Second of Day to defined Clock - Bytes (Second /
Minute / Hour)
Lsecofday = 12345
Bsec = Time(lsecofday)
Print "Second of Day " ; Lsecofday ; " converted to Sec=" ; Bsec ; "
Min=" ; Bmin ; " Hour=" ; Bhour
' Second of Day 12345 converted to Sec=45 Min=25 Hour=3
```

7.30 DBG

Action

Prints debug info to the hardware UART

Syntax

DBG

Remarks

See $DBG 628 for more information

7.31 DCF77TIMEZONE

Action

This function will return the offset to Greenwich Time.

Syntax

res = **DCF77TimeZone()**

Remarks

Res	The target variable that is assigned with the result. The result will be: - 0: when there is no valid DCF77 data yet - 1: when in "Middle Europe Normal Time" - 2: when in "Middle Europe daylight saving Time"

In Middle Europe, daylight saving is used to make better use of the day light in the summer.
The last Sunday in March at 02:00 AM the Daylight Saving will start. All clocks are set from 2:00 to 3:00.
Your weekend, is one hour shorter then.

But the last Sunday of October is better : at 03:00 AM, the Daylight Saving will end and all clocks are set from 03:00 to 02:00.

When you have a lot of clocks in your house, you can understand why DCF77 synchronized clocks are so popular.

See also
CONFIG DCF77 927

Example
```
Print = DCF77TimeZone()
```

7.32 DEBUG

Action
Instruct compiler to start or stop debugging, or print variable to serial port

Syntax
DEBUG ON | OFF | var

Remarks

ON	Enable debugging
OFF	Disable debugging
var	A variable which values must be printed to the serial port

During development of your program a common issue is that you need to know the value of a variable.
You can use PRINT to print the value but then it will be in the application as well.
You can use conditional compilation such as :
```
CONST TEST=1
#IF TEST
  print var
#ENDIF
```

But that will result in a lot of typing work. The DEBUG option is a combination of conditional compilation and PRINT. Whenever you activate DEBUG with the ON parameter, all 'DEBUG var' statements will be compiled.
When you turn DEBUG OFF, all 'DEBUG var' statements will not be compiled.

You can not nest the ON and OFF. The last statements wins.
Typical you will have only one DEBUG ON statement. And you set it to OFF when your program is working.

An example showing nesting is NOT supported:
```
DEBUG ON
DEBUG ON ' it is still ON
DEBUG OFF' it is OFF now
```

An example showing multiple DEBUG:
```
DEBUG ON
DEBUG var   ' this is printed
DEBUG var2 ' this is also printed

DEBUG OFF
DEBUG var3  'this is NOT printed
DEBUG var4  ' this is not printed

DEBUG ON    ' turn DEBUG ON
If A = 2 Then
  DEBUG A ' this is printed when A is 2
End If
```

 When DEBUG ON is used, the UART is initialized. This means that TX and RX pins are set to UART mode where they can not be altered by the user with simple SET/RESET statements.

See also
DBG

ASM
NONE

Example
```
DEBUG ON
Dim A As Byte
DEBUG A
End
```

7.33 DEBOUNCE

Action
Debounce a port pin connected to a switch.

Syntax
DEBOUNCE Px.y , state , label [, SUB]

Remarks

Px.y	A port pin like PINB.0 , to examine.
State	0 for jumping when PINx.y is low , 1 for jumping when PINx.y is high
Label	The label to GOTO when the specified state is detected

SUB	The label to GOSUB when the specified state is detected

When you specify the optional parameter SUB, a GOSUB to label is performed instead of a GOTO.

The DEBOUNCE statement tests the condition of the specified pin and if true there will be a delay for 25 mS and the condition will be checked again. (eliminating bounce of a switch)

When the condition is still true and there was no branch before, it branches to specified the label.

When the condition is not true, or the logic level on the pin is not of the specified level, the code on the next line will be executed.

When DEBOUNCE is executed again, the state of the switch must have gone back in the original position before it can perform another branch. So if you are waiting for a pin to go low, and the pin goes low, the pin must change to high, before a new low level will result in another branch.

Each DEBOUNCE statement, which uses a different port, uses 1 BIT of the internal memory to hold its state. And as the bits are stored in SRAM, it means that even while you use only 1 pin/bit, a byte is used for storage of the bit.

DEBOUNCE will not wait for the input value to met the specified condition. You need to use BITWAIT if you want to wait until a bit will have a certain value.

So DEBOUNCE will not halt your program while a BITWAIT can halt your program if the bit will never have the specified value. You can combine BITWAIT and DEBOUNCE statements by preceding a DEBOUNCE with a BITWAIT statement.

See also

CONFIG DEBOUNCE 935 , BITWAIT 803

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : deboun.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates DEBOUNCE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space
```

```
Config Debounce = 30                                        'when the
config statement is not used a default of 25mS will be used but we
override to use 30 mS


  'Debounce Pind.0 , 1 , Pr 'try this for branching when high(1)
  Debounce Pind.0 , 0 , Pr , Sub
  Debounce Pind.0 , 0 , Pr , Sub
  '                    ^----- label to branch to
  '                ^---------- Branch when PIND.0 goes low(0)
  '          ^---------------- Examine PIND.0

  'When Pind.0 goes low jump to subroutine Pr
  'Pind.0 must go high again before it jumps again
  'to the label Pr when Pind.0 is low

  Debounce Pind.0 , 1 , Pr                                  'no branch
  Debounce Pind.0 , 1 , Pr                                  'will result
in a return without gosub
End

Pr:
  Print "PIND.0 was/is low"
Return
```

7.34 DECR

Action

Decrements a variable by one.

Syntax

DECR var

Remarks

There are often situations where you want a number to be decreased by 1. It is simpler to write :

```
DECR var
```

compared to :

```
var = var - 1
```

See also

INCR 1314

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : decr.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demostrate decr
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------
```

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte , I As Integer

A = 5                                                      'assign
value to a
Decr A                                                     'decrease
(by one)
Print A                                                    'print it


I = 1000
Decr I
Print I
End
```

7.35 DECLARE FUNCTION

Action

Declares a user function.

Syntax

DECLARE FUNCTION TEST[([BYREF/BYVAL] prm as type)] As type

Remarks

test	Name of the function.
prm	Name of the optional parameters.
Type	Type of the parameter(s) and of the result. Byte,Word, Dword, Integer, Long, Single, Double or String. Bits are not supported. When passing a string it is recommended to also pass the maximum length of the string : SomeString As String * 30 would indicate that the string will have a maximum length of 30 characters. Please notice that you need to specify the string length in both the DECLARE and the actual implementation. Unless you use CONFIG SUBMODE=NEW in which case you only write the implementation.

When BYREF or BYVAL is not specified, the parameter will be passed by reference.
Use BYREF to pass a variable by reference with its address. This means that you work on the string you pass. Any change you make in the sub/function you will make on the original string.
Use BYVAL to pass a copy of the variable. This means that a copy is created and the address of this copy is passed to the sub/function. If you change the string, the

original sting remains the same.
Use BYLABEL to pass the address of a label.

See the CALL 806 and DECLARE SUB 1221 statements for more details.
See also Memory usage 267

 SUB and FUNCTION are the same with 1 difference : a function returns a result which can be assigned to a variable.

ARRAYS

Arrays can be passed by reference only. You need to add empty parenthesis**()** after the variable to indicate that you pass an array.
Inside the sub/function you also need to use **()** when accessing the variable.
Let's have a look at an example which calls a SUB. Functions and Subs are similar with the difference that a functions returns a result.

```
Declare Sub TestArray(ar() as byte, b as byte)
Dim a(10) as byte , q as byte
TestArray a(1) , q
```

As you can see, we add () after the variable to indicate that it is an array we pass.
When we call the sub program, we pass the first address or the base address of the array. That is a(1) in this case. See also CONFIG BASE. 886
Inside the sub module, we also refer to the variable using ().

```
Sub TestArray(ar() as byte, b as byte)
   print ar(1)
   print ar(b)
End Sub
```

In older BASCOM versions, it was not required to use (). You only needed to pass the base address. But that is potential unsafe : if you reference a variable as an array while it is actually a single variable, then you can write to the wrong address. When using (), the compiler knows when an array is expected and can inform you about a possible error.
If you have old code you can use CONFIG ERROR=IGNORE,380=IGNORE to ignore errors as a result of the updated syntax.

 You must declare each function before writing the function or calling the function. And the declaration must match the function.
Bits are global and can not be passed to functions or subs.

When you want to pass a string, you pass it with it's data type : string. So the size is not important. For example :
Declare function Test(s as string, byval z as string) as byte
You may and should specify the optional maximum length. In fact it is highly recommended that you do so.

 When you set the function result, you need to take care that no other code is executed after this.
So a good way to set the result would be this :

```
Function Myfunc(b as byte) as Byte
   local bDummy as byte
   'some code here
   Myfunc=3 ' assign result
   ' no other code is executed
End Function
```

Also good would be:

```
Function Myfunc(b as byte) as Byte
   local bDummy as byte
   'some code here
   Myfunc=1 ' assign default result
   Print "this is a test " ; b
   Myfunc=4 ' now again the result is the last code
   ' no other code is executed
End Function
```

If you execute other code after you assigned the function result, registers will be trashed. This is no problem if you assigned the function result to a variable. But when you use a function without assigning it to a variable, some temporarily registers are used which might be trashed.

Thus this special attention is only needed when you use the function like :
If Myfunc()=3 then 'myfunc is not assigned to a variable but the result is needed for the test

When you use :
`myvar=Myfunc()`
Then you will not trash the registers. So in such a case there is no problem to run code after the function assignment.

To keep it safe, assign the result just before you exit the function.

 **IMPORTANT**
In order to prevent memory overwrites there are some things you need to be aware of.
All data types have a fixed length. A byte takes 1 byte, a word takes 2 bytes, etc.
Strings have a variable length. When you dimension the string you specify the maximum amount of memory that will be used by the string.
DIM S as string * 10 means that 10 bytes + 1 trailer byte, will be reserved in RAM for string named S.
When you assign this string with a constant the compiler will check if the assigned string is dimensioned large enough to hold the string constant. You will get an error if it does not fit.
But when you assign a string with a function or other string, or perform a string concatenation (+ string) there is no such check. This means that you can overwrite the memory that is placed behind the string.
Consider the following examples.
Example 1, this will give an error since the constant will not fit.

```
$RegFile = "m88def.dat"
$Crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100
```

```
dim s as string *10
s="0123456789A"
End
```

Example 2, this will be ok since the string is large enough.

```
$RegFile = "m88def.dat"
$Crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100

dim s as string *10
s="0123456789"
End
```

Example 3, this will give no error but will overwrite memory since data is added to the string which is not large enough

```
$RegFile = "m88def.dat"
$Crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100

dim s as string *10
s="0123456789"
s=s+"abc"
End
```

Example 4, this is the same as sample 3 but will demonstrate that variable B will be overwritten

```
$RegFile = "m88def.dat"
$Crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100

dim s as string *10 ,b as byte
b=123
s="0123456789"
s=s+"abc"
print b

End
```

Example 5, this is when you reserve 100 bytes for the frame/temp space, but you

pass data which would take more than that. You get an error in that case.

```
 $regfile = "m128def.dat"
$crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100 'notice that it is 100

Declare Sub test(ByVal S As String * 160)
                                        ' ^^^^ will not fit
Call test("012")

Sub test(ByVal S As String * 160)
   Local vs As String * 3
   vs = "AAA"
End Sub
```

In example 6 we pass a constant but we specified a maximum length, and we will get error 119 : constant too big to fit

```
$regfile = "m128def.dat"
$crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100

Declare Sub test(ByVal S As String * 10)

Call test("0123456789a") 'notice that the size is 11 so we get an
error

Sub test(ByVal S As String * 10)
   Local vs As String * 3
   vs = "AAA"
End Sub
```

In example 7 we use $FRAMECHECK to see if the data will fit. We call the same sub (recursive) and we never exit which mean the memory is never released. So after a few calls we run out of space and ERR becomes 1

```
$regfile = "m128def.dat"
$crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100
$FrameCheck

Declare Sub test(ByVal S As String * 10)

Call test("0123456789")
```

```
Sub test(ByVal S As String * 10)
   Local vs As String * 3
   print err 'will be 1 when there is not enough frame space
   'you will see that ERR will be 0 but becomes 1 when there is
not enough space
   vs = "AAA"
   test "abc"    'call ourselves which is not smart but will demo
the check
End Sub
```

In example 8 we see how unsafe it can be when you do not specify the length of the string

```
$regfile = "m128def.dat"
$crystal = 8000000
$hwstack = 50
$swstack = 40
$framesize = 100
$FrameCheck

Declare Sub test(ByVal S As String)

Call test("0123")

Sub test(ByVal S As String)
   Local vs As String * 3
   print err 'will be 1 when there is not enough frame space
   'you will see that ERR will be 0 but becomes 1 when there is
not enough space
   vs = "AAA"
   S = "012345"  '<--- This overwrites local vs
   print err'there was enough space so we get 0, still there is
an overwrite
End Sub
```

See also

CALL 806, SUB 1545 , CONFIG SUBMODE 1079 , EXIT 1257 , $FRAMECHECK 651

Example

```
'-------------------------------------------------------------------
-----------------
'name                     : function.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstration of user function
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------
-----------------
```

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'A user function must be declare before it can be used.
'A function must return a type
Declare Function Myfunction(byval I As Integer , S As String) As Integer
'The byval paramter will pass the parameter by value so the original
value
'will not be changed by the function

Dim K As Integer
Dim Z As String * 10
Dim T As Integer
'assign the values
K = 5
Z = "123"

T = Myfunction(k , Z)
Print T
End


Function Myfunction(byval I As Integer , S As String) As Integer
  'you can use local variables in subs and functions
  Local P As Integer
  P = I
  'because I is passed by value, altering will not change the original
  'variable named k
  I = 10

  P = Val(s) + I

  'finally assign result
  'Note that the same data type must be used !
  'So when declared as an Integer function, the result can only be
  'assigned with an Integer in this case.
  Myfunction = P
End Function
```

7.36 DECLARE SUB

Action

Declares a subroutine.

Syntax

DECLARE SUB TEST[([BYREF|BYVAL|BYLABEL|BYREG|BYSTACK] var as type)]

Remarks

test	Name of the procedure.

Var	Name of the parameter(s).
Type	Type of the parameter(s) : Byte, Word, Dword, Integer, Long, Single, Double or String. When passing a string it is recommended to also pass the maximum length of the string : SomeString As String * 30 would indicate that the string will have a maximum length of 30 characters. Please notice that you need to specify the string length in both the DECLARE and the actual implementation. Unless you use CONFIG SUBMODE=NEW in which case you only write the implementation.

ARRAYS

Arrays can be passed by reference only. You need to add empty parenthesis**()** after the variable to indicate that you pass an array.
Inside the sub/function you also need to use () when accessing the variable.
Let's have a look at an example.

*Declare Sub TestArray(**ar()** as byte, b as byte)*
Dim a(10) as byte , q as byte
*TestArray **a()** , q*

As you can see, we add () after the variable to indicate that it is an array we pass.
When we call the sub program, we pass the first address or the base address of the array. That is a(1) in this case.
Inside the sub module, we also refer to the variable using ().

```
Sub TestArray(ar() as byte, b as byte)
   print ar(1)
   print ar(b)
End Sub
```

In older BASCOM versions, it was not required to use (). You only needed to pass the base address. But that is potential unsafe : if you reference a variable as an array while it is actually a single variable, then you can write to the wrong address. When using (), the compiler knows when an array is expected and can inform you about a possible error.
If you have old code you can use CONFIG ERROR=IGNORE,380=IGNORE to ignore errors as a result of the updated syntax.

Parameter Passing

When BYREF | BYVAL | BYREG | BYLABEL or BYSTACK is not provided, the parameter will be passed by reference (BYREF).

BYREF

Use BYREF to pass a variable by reference with its address. When using the referenced address, you work on the original variable. So a change of the variable inside the sub routine, will change the passed variable outside the routine as well.

BYVAL

Use BYVAL to pass a copy of the variable. Passing a copy of the variable allows to alter the copy in the sub routine while leaving the original variable unaltered. BYVAL

will not change the original passed variable but it requires more code since a copy of the parameter must be created.

BYREG

Use BYREG to pass a copy of the variable using a register. The value will be passed to the register(s) you specify. When multiple bytes need to be passed, multiple registers will be used. Registers are named from R0-R31. When you pass a WORD to register R16, you will also use R17 since a word requires 2 bytes.
You can not pass strings. Only numeric variables and constants.
Using BYREG requires some knowledge of the routines you call. The current implementation does not protect already loaded registers. This means that when you pass multiple registers you could destroy some already loaded registers just because a parameter will destroy the register.
Example : declare Sub MySub(byreg R16 as Word, byreg R18 as long, byreg R22 as dword)

```
mysub 1000, var(J+100), var(j)
```

In this example, R16 and R17 are loaded, after this the array index of variable var() need to be calculated which uses the ML16 routine which uses R16-R21
Numeric constants and expression do not alter registers but functions might. A future version will track and protect registers.
Why would you want to use BYREG ? Using BYREG is equivalent to using ASM. It is intended to be used with ASM code inside subs. The FT800 include files use BYREG and BYSTACK.

Example from FT800:

```
Sub Stencilfunc(byreg  r18 As Byte , Byreg r17 As Byte , Byreg R16 As Byte)
  Cmd32 _stencilfunc(r18 , R17 , r16)
End Sub
```

BYSTACK

Use BYSTACK to pass a copy of the variable by the soft stack (Y-pointer). BYSTACK will not create a copy of the variable but instead will pass the data directly to the soft stack.
The first parameter is passed first , LSB first. You will find BYSTACK used in the FT800 include files. BYSTACK has the advantage compared to BYREG that no registers are altered. But it has the disadvantage that it requires an optional step to pass the data to the stack.
The SUB/FUNCTION need to clean up the stack. Typically you would use LD reg, y+ to pop data from the stack.
The FT800 uses CMDFTSTACK 1651 to pop data from the stack and send it to the FT800.

BYLABEL

Use BYLABEL to pass the address of a label. BYLABEL will pass the **word** address. It will not work for processors with multiple 64 KB pages.

Using BYLABEL on the EEPROM is possible but the EEPROM image must proceed the call with the label name.

See also READEEPROM 1415, LOADLABEL 1359 and Memory usage 267

If you pass a string you may specify the length of the string. This length will be the

maximum length the string may grow. This is important when you pass a string BYVAL.
For example, when you pass a string like "ABC" to a subroutine or function using BYVAL, the compiler will create a copy with a length of 3. This is sufficient to pass it to the sub routine.
But if the sub routine adds data to the string, it will not fit since the string is too short. In such a case you can specify the length. **s as string * 10**, will create a string with a size of 10.

See the CALL 806 statement for more details.

 You must declare each function before writing the function or calling the function. And the declaration must match the function. Optional you can use CONFIG SUBMODE 1079=NEW so DECLARE is not required.
Bits are global and can not be passed to functions or subs.

 See DECLARE FUNCTION 1211 paragraph named IMPORTANT

See also

CALL 806, SUB 1545 , FUNCTION 1215 , CONFIG SUBMODE 1079

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : declare.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrate using declare
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
' Note that the usage of SUBS works different in BASCOM-8051
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


' First the SUB programs must be declared

'Try a SUB without parameters
Declare Sub Test2()

'SUB with variable that can not be changed(A) and
'a variable that can be changed(B1), by the sub program
```

```
'When BYVAL is specified, the value is passed to the subprogram
'When BYREF is specified or nothing is specified, the address is passed
to
'the subprogram

Declare Sub Test(byval A As Byte , B1 As Byte)
Declare Sub Testarray(byval A As Byte , B1 As Byte)
'All variable types that can be passed
'Notice that BIT variables can not be passed.
'BIT variables are GLOBAL to the application
Declare Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S
As String)

'passing string arrays needs a different syntax because the length of
the strings must be passed by the compiler
'the empty () indicated that an array will be passed
Declare Sub Teststr(b As Byte , Dl() As String)

Dim Bb As Byte , I As Integer , W As Word , L As Long , S As String * 10
      'dim used variables
Dim Ar(10) As Byte
Dim Sar(10) As String * 8                                   'strng array

For Bb = 1 To 10
  Sar(bb) = Str(bb)                                         'fill the
array
Next
Bb = 1
'now call the sub and notice that we always must pass the first address
with index 1
Call Teststr(bb , Sar(1))


Call Test2                                                  'call sub
Test2                                                       'or use
without CALL
'Note that when calling a sub without the statement CALL, the enclosing
parentheses must be left out
Bb = 1
Call Test(1 , Bb)                                          'call sub
with parameters
Print Bb                                                    'print value
that is changed

'now test all the variable types
Call Testvar(bb , I , W , L , S )
Print Bb ; I ; W ; L ; S

'now pass an array
'note that it must be passed by reference
Testarray 2 , Ar(1)
Print "ar(1) = " ; Ar(1)
Print "ar(3) = " ; Ar(3)

$notypecheck                                                ' turn off
type checking
Testvar Bb , I , I , I , S
'you can turn off type checking when you want to pass a block of memory
$typecheck                                                  'turn it
back on
End

'End your code with the subprograms
'Note that the same variables and names must be used as the declared
```

```
ones

Sub Test(byval A As Byte , B1 As Byte)                          'start sub
    Print A ; " " ; B1                                          'print
passed variables
    B1 = 3                                                      'change
value
    'You can change A, but since a copy is passed to the SUB,
    'the change will not reflect to the calling variable
End Sub

Sub Test2                                                       'sub without
parameters
    Print "No parameters"
End Sub



Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As
String)
    Local X As Byte
    X = 5                                                       'assign
local
    B = X
    I = -1
    W = 40000
    L = 20000
    S = "test"
End Sub

Sub Testarray(byval A As Byte , B1 As Byte)                     'start sub
    Print A ; " " ; B1                                          'print
passed variables
    B1 = 3                                                      'change
value of element with index 1
    B1(1) = 3                                                   'specify the
index which does the same as the line above
    B1(3) = 3                                                   'modify
other element of array
    'You can change A, but since a copy is passed to the SUB,
    'the change will not reflect to the calling variable
End Sub

'notice the empty() to indicate that a string array is passed
Sub Teststr(b As Byte , Dl() As String)
  Dl(b) = Dl(b) + "add"
End Sub
```

Example BYLABEL

```
$regfile = "m88def.dat"
$hwstack = 40
$swstack = 80
$framesize = 80

Dim B As Byte , W As Word

Declare Sub Somesub(bylabel Mylabel As Word)
Somesub Alabel
End
```

```
Sub Somesub(bylabel Mylabel As Word)
   W = Mylabel                    ' this points to the BYTE address of the
data
   !lds _dptrl,{W }               ' point to
   !LDS _dptrh,{W+1}
   Read B : Print B
End Sub

Alabel:
Data 1 , 2 , 3
```

7.37 DEFxxx

Action

Declares all variables that are not dimensioned of the DefXXX type.

Syntax

DEFBIT b	Define BIT
DEFBYTE c	Define BYTE
DEFINT I	Define INTEGER
DEFWORD x	Define WORD
DEFLNG l	Define LONG
DEFSNG s	Define SINGLE
DEFDBL z	Define DOUBLE

Remarks

While you can DIM each individual variable you use, you can also let the compiler handle it for you.
All variables that start with a certain letter will then be dimmed as the specified type.

Example

```
Defbit b : DefInt c     ' default type for bit and integers

Set b1                  ' set bit to 1

c = 10                  ' let c = 10
```

7.38 DELAY

Action

Delay program execution for a short time.

Syntax

DELAY

Remarks

Use DELAY to wait for a short time.
The delay time is ca. 1000 microseconds.

 Interrupts that occur frequently and/or take a long time to process, will let the delay last longer.
When you need a very accurate delay, you need to use a timer.

See also
WAIT 1607 , WAITMS 1607

Example
```
'-----------------------------------------------------------------------
-----------------
'name                     : delay.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: DELAY, WAIT, WAITMS
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Ddrb = &HFF                                                'port B as
output
Portb = 255
Print "Starting"
Delay                                                      'lets wait
for a very short time
Print "Now wait for 3 seconds"
Portb = 0
Wait 3
Print "Ready"
Waitms 10                                                  'wait 10
milliseconds
Portb = 255
End
```

7.39 DIM

Action
Dimension a variable.

Syntax
DIM var[,varn] AS [XRAM/SRAM/ERAM]type [AT location/variable] [OVERLAY] [SAFE]

Remarks

Var	Any valid variable name such as b1, i or longname. var may also be an array : ar(10) for example. You can also use a list and created a number of variables of the same data type : DIM A1,A2, BVAR AS BYTE. This will create 3 BYTE variables. When using a list, you may not use identifiers such as #%!&. You may also not use the optional OVERLAY. It is also possible to define the data type by ending the variable name with an identifier : % for Integer & for Long # for Double ! for Single Dim A!, b# would create a variable A! of the SINGLE data type and a variable B# with the DOUBLE data type When a variable is dimensioned with an identifier, the variable must be referenced with that identifier as well. We encourage the use of Hungarian Notation where you use a prefix instead : Dim bVar As Byte ' the b indicates a BYTE Dim iMyInt As Integer 'the i indicates an INTEGER common used prefixes : b - BYTE w - WORD dw - DWORD i - INTEGER l - LONG s - STRING dbl - DOUBLE sng - SINGLE The IDE can show the data type of the variable when you hover the mouse above the variable name and keep the SHIFT key pressed.
Type	Bit/Boolean, Byte, Word, Integer, Long, Dword, Single, Double, String or Type.
XRAM	Specify XRAM to store variable into external memory
SRAM	Specify SRAM to store variable into internal memory (default)
ERAM	Specify ERAM to store the variable into EEPROM
OVERLAY	Specify that the variable is overlaid in memory.
location	The address or name of the variable when OVERLAY is used.
SAFE	An optional specifier to indicate that access to this variable must be done in a safe way. See the full explanation below.

A string variable needs an additional length parameter:
*Dim s As XRAM String * 10*

In this case, the string can have a maximum length of 10 characters. Internally one additional byte is needed to store the end of string marker. Thus in the example above, 11 bytes will be used to store the string.

BITS
Note that BITS can only be stored in internal memory.

You may also specify IRAM. IRAM is the place in memory where the registers are located : absolute address 0 - 31. BASCOM uses most of these addresses, depending on the instructions/options you use. For a $TINY 710 chip it makes sense to use IRAM since there is NO SRAM in most tiny AVR chips (TINY15 for example). You may also use to IRAM to overlay registers in memory.

See also Memory usage 267

Multiple variables on one line

You may Dimension multiple variables using one DIM statement when you separate them by a comma. There are 2 ways to do so :
Dim A As Byte, B As Byte, C As Word

The second method is even simpler :
Dim A, B, C As Byte
Here all variables are bytes. They are only separated by a comma. In the sample above, C is a word, so the equivalent would need :
Dim A, B As Byte, C As Word

Depending on which method you use, the variables might end up at a different memory location. When not using AT, you should not depend on the memory location of a variable.
Variables are usually stored in the same memory order as they are dimensioned. But you should not depend on it. Some optimization techniques requite that some variables are stored in a certain order. Use VARPTR 1604 to get the address of a variable in memory.

The Data/TIme routines require that sec,min and hour variables are in a specific order. For those you need to be explicit using AT :
Dim b as byte , m as byte at b + 1 , h as byte at m + 1
This will ensure that the bytes are placed in the specified order.

SCOPE

The scope for DIM is global. So no matter where you use the DIM statements, the variable will end up as a global visible variable that is visible in all modules, procedures and functions.
When you need a LOCAL variable that is local to the procedure or function, you can use LOCAL 1360.
Since LOCAL variables are stored on the frame, it takes more code to dynamic generate and clean up these variables. This because all functions and subs are fully re-entrant. (re-entrant means they can call themselves recursively)

AT

The optional **AT** parameter lets you specify where in memory the variable must be stored. When the memory location already is occupied, the first free memory location will be used. You need to look in the report file to see where the variable is located in memory. In general it is a bad idea to use fixed locations. The SRAM starts at different locations in various processors. Some use &H60, &H100, or &H2000 for Xmega. When you have hard coded that a variable will start at &H60, and you port your code to an XMEGA this location is not usable.

OVERLAY

The **OVERLAY** option will not use any variable space. It will create a sort of phantom

variable.

```
Dim x as Long at &H60                'long uses 60,61,62 and 63 hex of
SRAM

Dim B1 As Byte At &H60 Overlay       'overlay at the same address at &H60
Dim B2 As Byte At &H61 Overlay
```

B1 and B2 are no real variables! They refer to a place in memory. In this case to &H60 and &H61. By assigning the phantom variable B1, you will write to memory location &H60 that is used by variable X.
So to define it better, OVERLAY does create a normal usable variable, but it will be stored at the specified memory location which could be already be occupied by another OVERLAY variable, or by a normal variable.
You can not overlay BIT/Boolean variables. These are global variables stored in bytes which can not be overlayed. You can however use an ALIAS : Mybit ALIAS SomeByte.0

 Take care with the OVERLAY option. Use it only when you understand it. Refer to a variable if possible, not to an absolute address.

You can also read the content of B1:

```
Print B1
```

This will print the content of memory location &H60.

By using a phantom variable you can manipulate the individual bytes of real variables.

Overlay example 2

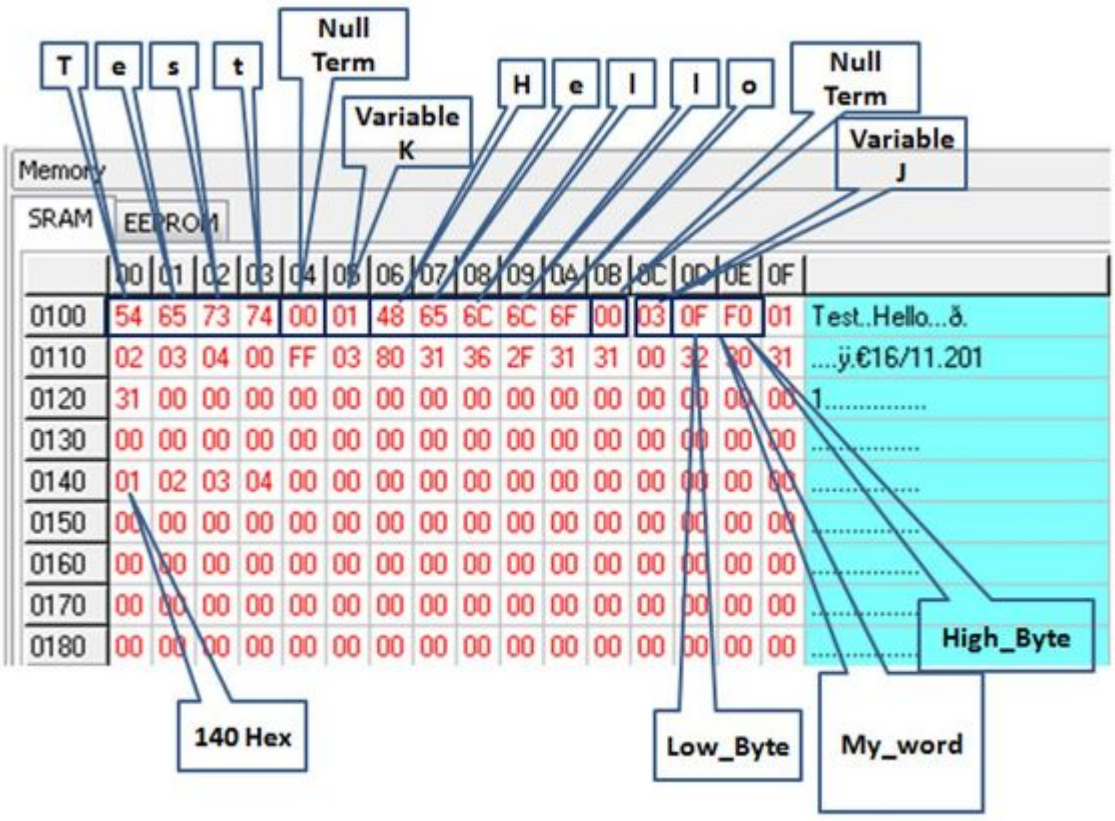
```
Dim L as Long at &H60
Dim W as Word at &H62 OVERLAY
```

W will now point to the upper two bytes of the long.

Overlay example 3

Following you find the Bascom-AVR Simulator Memory status when you run the following example in Bascom-AVR Simulator. This example is intended to be used with the simulator. You need to uncomment the **$sim** when you want to test it on an real AVR.

 Strings need an additional byte (Null termination). So you need an overlay of 8 bytes when you overlay a string with 7 bytes.




```
$regfile = "m644pdef.dat"
$crystal = 4000000
$hwstack = 60
$swstack = 60
$framesize = 60   'frame space can grow rapid when using it on variables
with a big size (strings)
$baud = 9600
$sim              '$sim to use this example in Bascom-AVR simulator


Print "-------------------------"

Dim Array(5) As Byte
Dim My_string As String * 4 At Array Overlay
Dim K As Byte

K = 1

My_string = "Test"

' ---> 4 ASCII but 5 Bytes because of 0 Termination of String which is
another byte
' This is how it will be stored in SRAM
'  Array(1) Array(2) Array(3) Array(4) Array(5)
' +--------+--------+--------+--------+--------+
' |   T    |    e   |    s   |    t   |   00   |
' +--------+--------+--------+--------+--------+

Print Chr(array(1))
Print Chr(array(2))

Print "-------------------------"
```

```
Dim Teststring As String * 5
Dim Ar(6) As Byte At Teststring Overlay
Dim J As Byte
J = &H03

Ar(5) = 47

Teststring = "Hello"

'  ---> 5 ASCII but 6 Bytes because of 0 Termination of String
' This is how it will be stored in SRAM
'    Ar(1)    Ar(2)    Ar(3)    Ar(4)    Ar(5)    Ar(6)
' +--------+--------+--------+--------+--------+--------+
' |   H    |    e   |    l   |    l   |   o    |   00   |
' +--------+--------+--------+--------+--------+--------+

For K = 1 To 5
  Print Chr(ar(k)) ;
Next
Print

K = 1

Print "-------------------------"

Dim My_word As Word
Dim Low_byte As Byte At My_word Overlay
Dim High_byte As Byte At My_word + 1 Overlay

Low_byte = &B0000_1111
High_byte = &B1111_0000

' This is how it will be stored in SRAM
' <-------my_word-------->
' +-----------+----------+
' | Low_byte  |High_byte |
' +-----------+----------+

'But when you print it with print bin(Variable) you will see it as

' <-------my_word-------->
'    11110000   00001111
' +-----------+----------+
' | High_byte |Low_byte  |
' +-----------+----------+

Print "My_word = " ; Bin(my_word)

Print "-------------------------"

Dim My_long_1 As Long
Dim Byte_1 As Byte At My_long_1 Overlay
Dim Byte_2 As Byte At My_long_1 + 1 Overlay
Dim Byte_3 As Byte At My_long_1 + 2 Overlay
Dim Byte_4 As Byte At My_long_1 + 3 Overlay

Byte_1 = 1
Byte_2 = 2
Byte_3 = 3
Byte_4 = 4

Print Bin(my_long_1)

' This is how it will be stored in SRAM
' <-------my_long_1------------>
```

```
' +-------+------+------+------+
' | Byte_1|Byte_2|Byte_3|Byte_4|
' +-------+------+------+------+

'But when you print it with print bin(Variable) you will see it as

' <-------my_long_1------------>
' +-------+------+------+------+
' | Byte_4|Byte_3|Byte_2|Byte_1|
' +-------+------+------+------+

Print "-------------------------"

Dim My_dword As Dword At $140       ' This places the my_long_2 variable
at a fixed SRAM address  starting at HEX 140
Dim Byte__1 As Byte At $140 Overlay ' NOTICE: because this will be
stored at the specified memory location
Dim Byte__2 As Byte At $141 Overlay ' which could be already be occupied
by another OVERLAY variable, or by a normal variable the
Dim Byte__3 As Byte At $142 Overlay ' compiler generate an ERROR
"Address already occupied" in this case.
Dim Byte__4 As Byte At $143 Overlay


Byte__1 = 1
Byte__2 = 2
Byte__3 = 3
Byte__4 = 4

'This is how it will be stored in SRAM
' <----------my_dword---------->
' +-------+------+------+------+
' | Byte_1|Byte_2|Byte_3|Byte_4|
' +-------+------+------+------+

'But when you print it with print bin(Variable) you will see it as
' <----------my_dword---------->
' +-------+------+------+------+
' | Byte_4|Byte_3|Byte_2|Byte_1|
' +-------+------+------+------+

Print "my_dword = " ; Bin(my_dword)

Print "-------------------------"

Dim My_dword_2 As Dword
Dim My_word_2 As Word At My_dword_2 Overlay
Dim My_byte3 As Byte At My_dword_2 + 2 Overlay
Dim My_byte4 As Byte At My_dword_2 + 3 Overlay

My_word_2 = &B11111111_00000000
My_byte3 = &B00000011
My_byte4 = &B10000000

'This is how it will be stored in SRAM
' <-------------my_dword_2------------>
' +---------+--------+--------+--------+
' |     my_word_2    |my_byte3|my_byte4|
' +---------+--------+--------+--------+

'But when you print it with print bin(Variable) you will see it as
' <-------------my_dword_2------------>
' +---------+--------+--------+--------+
' | my_byte4|my_byte3|    my_word_2     |
' +---------+--------+--------+--------+
```

```
Print Bin(my_dword_2)

Print "-------------------------"

' Now we examine the Null terminator in Strings

Dim My_date(11) As Byte                                    ' 8 strings
+ 3 Null terminator = 11 Byte
Dim Day As String * 2 At My_date(1) Overlay
Dim Null_terminator As Byte At My_date(1) + 2 Overlay      ' Null
terminator
Dim Month As String * 2 At My_date(1) + 3 Overlay
Dim Null_terminator_2 As Byte At My_date(1) + 5 Overlay    ' Null
terminator
Dim Year As String * 4 At My_date(1) + 6 Overlay
Dim Null_terminator_3 As Byte At My_date(1) + 10 Overlay   ' Null
terminator

Day = "16"
Month = "11"
Year = "2011"

Print "Day= " ; Day
Print "Month= " ; Month
Print "Year= " ; Year

'For example the print function use the Null Terminator to check the end
of the string
'When we set now the Null_terminator to "/"  (forward slash) instead of
0 then the print function print until a Null terminator is recognised
Null_terminator = 47                                       ' 47 = "/"
(forward slash

Print Day   ' This will now print  "16/11" because the first Null
terminator will be found after the "11"



End                                                        ' end
program
```

Using variable name instead of address

As variables can be moved though the program during development it is not always convenient to specify an address. You can also use the name of the variable :

```
DIM W as WORD
Dim B as BYTE AT W OVERLAY
```

Now B is located at the same address as variable W.

For XRAM variables, you need additional hardware 251: an external RAM and address decoder chip.

ERAM

For ERAM variables, it is important to understand that these are not normal variables. ERAM variables serve as a way to simple read and write the EEPROM memory. You can use READEEPROM and WRITEEEPROM for that purpose too.

To write to an ERAM variable you have to use an SRAM variable as the source :

eramVAR= sramVAR
To read from an ERAM variable you have to use an SRAM variable as the targer :
sramVAR=eramVAR
Both variables need to be of the same data type. So when writing to an ERAM double, the source variable need to be of the double type too.
ERAM can be assigned with a numeric value too : eramVAR= 123

You can not use an ERAM variable as you would use a normal variable.

Also keep in mind that when you write to ERAM, you write to EEPROM, and that after 100.000 times, the EEPROM will not erase properly.

```
Dim b as byte, bx as ERAM byte
B = 1
Bx = b ' write to EEPROM
B = bx ' read from EEPROM
```

Updateeprom

When you define a constant named Updateeprom in your code, the EEPROM will only be updated when the value differs. In order to do so, the EEPROM is read before the new value is written. This will take some extra time/code. The constant only need to be defined, the value itself is not important. Like : CONST Updateeprom=1

Xmega

The XMEGA need an additional configuration command : CONFIG EEPROM 952 = MAPPED, in order to use ERAM. Or use the QUICK option.

Arrays

An array is a sequential collection of elements with the same data type. Till version 2077, arrays could have only 1 index or dimension. But in 2078 this has been changed and while there are no technical limits for unlimited indexes, the limit has been set to 5. This means that you can create a variable array like : *Dim ar(5,10,5) As Byte*
This will create a BYTE variable named AR, and it has 3 indexes. Each index requires space, in this sample the amount of bytes required would be : 5 * 10 * 5 = 250 * lengthOfByte = 250 bytes.
For a WORD, which uses 2 bytes, the required space would have been 250 * 2 = 500 bytes.
While using multiple indexes might be a nice feature, it comes with a penalty : the processor need to calculate the address in memory based on the indexes. The more indexes you add, the more calculations/code is required.
When you use a single index, the old calculation method is used. When using multiple indexes, a new method is used which calls array calculator code in mcs.lib.
As mentioned, the maximum number of indexes is 5 so : Dim ar(5,5,5,10,10) As Byte would work.
Multiple indexes is a new feature in 2078. The simulator does not support this option yet, so for the simulator, only 1 array exists.

Lets see how memory is organized when using multiple indexes. For the sample we use an array of 5x3 bytes.
Dim ar(5,3) as byte.
This gives us the following possible index values :
1,1 1,2 1,3

2,1 2,2 2,3
3,1 3,2 3,3
4,1 4,2 4,3
5,1 5,2 5,3

Since the memory of the processor is linear, we have 15 cells.

Address	Cell
n	1,1
n+1	1,2
n+2	1,3
n+3	1,4
n+4	1,5
n+5	2,1
n+6	2,2
n+7	2,3
n+8	2,4
n+9	2,5
n+10	3,1
n+11	3,2
n+12	3,3
n+13	3,4
n+14	3,5

Arrays start with element 1 by default. Thus DIM ar(5) will create 5 elements and the first element is ar(**1**).
Some times it is more convenient to start with element 0. For this you can use the **CONFIG BASE=0** option.
When CONFIG BASE 886 is set to 0, and not the default 1, the first element will be 0 : DIM ar(5), will make ar(0) the first element, and ar(4) the last element.

 Multi DIM arrays are not supported for bootloaders with an address > 64KB. This means that for an Mega128 bootloader it will not work. The reason is that the compiler places a type/index table in the first segment for faster calculation.

Size

The maximum size of an array depends on the available memory and the data type.
The XMEGA supports up to 8 MB of external memory. BASCOM supports this but the implementation is still considered BETA. It should not be used for production. The only thing you need to do to activate the big memory is to specify the size with $XRAMSIZE.
For example : $XRAMSIZE=8000000 will tell the compiler that you use 8 MB of external memory.
Additional registers must be set to pass the 24 bit address. This will create more code.
There is only one restriction : you can/may not pass variables located in the external memory to a sub or function.
The compiler will always pass a word address and does not support to pass the additional byte.

SAFE

The optional SAFE attribute can be used to specify that access to a variable must be done in a safe way. So when would it be unsafe?
Imagine that you DIM a BYTE and access this byte in your main code but also inside

an interrupt service routine (ISR).
A typical piece of code would be for the main code :
1 - LDS r24, address of BYTE load into register R24 the value of the byte
2 - Inc R24 increase the value of the register R24
3 - STS address, R24 write the updated value of the register back to the byte

The ISR will save the register R24. So that will not cause a problem.
Imagine that the ISR must read the value of the byte, depending on the step in main, it will load a different value.
When interrupted at step 1, the ISR will load the same value
When interrupted at step 2, the ISR will load the increased value
When interrupted at step 3, the ISR will also load the new increased value

The case above will probably not cause much problems to your code.

Now what if the ISR will alter the value? It could write for example a zero to the byte.
The ISR will have similar code like the main code :
1 - LDS r24, address of BYTE load into register R24 the value of the byte
2 - CLR R24 clear the value of the register R24
3 - STS address, R24 write the updated value of the register back to the byte

Now depending on the step in the main code, the following will happen :
When interrupted at step 1, the register is loaded , the ISR clears the variable to 0, but since R24 was saved and restored it holds the previous value. It then will continue to be increased and saved. The other steps will have a similar outcome. This is normal since you interrupted a process.
It is like printing to the serial port "Hello world" and an ISR will print a *. This star will appear in the other data.

Beside writing at the same time to the same variable, there is another problem with variables that have a longer data length than 1 byte.
Lets say you work on a WORD that is 2 bytes and has a value of &H1234
The code :
1- LDS R24, address of LSB
2- LDS R25, address of MSB
3- add some value to R24 like 1
4- add some value to R25 like 1
5- STS address of LSB, R24
6- STS address of MSB, R25

The normal outcome would be &H1335 since both registers were increased by 1.

When this code is interrupted between step 5 and 6, and the WORD variable is read, it will not have the right value. This because one of the registers is not written yet.
When you read a BYTE, it will always have the value that you have written to it. (either in the main or ISR).
But in this sample you would read &H1235 because the MSB register R25 is not written to memory yet.

BITS

Bits are stored in bytes. This can cause the same problems when you manipulate bits that have the same byte address.

So what do we do to prevent problems? The normal solution is to disable interrupts in

your code and re-enable them when you write to variables that are also access inside the ISR :
DISABLE INTERRUPTS
SomeWord = 0
ENABLE INTERRUPTS

Now the ISR can not interrupt the updating of the variable. So it can not read the wrong value.
The SAFE option will disable interrupts and enables them, whenever you write or read a safe variable.
Please notice that this will not be done inside an ISR since when an ISR is executed, the global interrupts are disabled by default at the hardware level.
~~And also notice that when interrupts were not enabled yet, they will be as soon you access a variable with the SAFE flag !~~
From version 2086 off, at the cost of some extra code, the I flag is saved and restored.

Using the SAFE flag on a BIT will make ALL the bits of the byte of that group SAFE as well.
For example :
Dim b1 as BIT SAFE , b2 as BIT
Now b1 and b2 will be placed inside the same byte. And since b1 is marked with SAFE, b2 will be safe too since it shares the same memory location.

You can not use the SAFE attribute on an OVERLAY variable.

TYPES

A type is a container for a number of normal data types. See also TYPE 1597.
Almost all rules apply to types. Dim rec as klm 'where klm is a defined TYPE

See Also

CONST 1170 , LOCAL 1360, Memory usage 267 , CONFIG BASE 886 , TYPE 1597

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : dim.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: DIM
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
```

```
use 40 for the frame space

Dim B1 As Bit                                                 'bit can be
0 or 1
Dim A As Byte                                                 'byte range
from 0-255
Dim C As Integer                                              'integer
range from -32767 - +32768
Dim L As Long
Dim W As Word
Dim S As String * 11                                          'length can
be up to 11 characters

'new feature : you can specify the address of the variable
Dim K As Integer At &H120
'the next dimensioned variable will be placed after variable s
Dim Kk As Integer


'Assign bits
B1 = 1                                                        'or
Set B1                                                        'use set

'Assign bytes
A = 12
A = A + 1

'Assign integer
C = -12
C = C + 100
Print C

W = 50000
Print W

'Assign long
L = 12345678
Print L

'Assign string
S = "Hello world"
Print S
End
```

7.40 DISABLE

Action

Disable specified interrupt.

Syntax

DISABLE interrupt [device]

Remarks

Interrupt	Description
INT0	External Interrupt 0
INT1	External Interrupt 1
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt

OVF1,TIMER1, COUNTER1	TIMER1 overflow interrupt
CAPTURE1, ICP1	INPUT CAPTURE TIMER1 interrupt
COMPARE1A,OC1A	TIMER1 OUTPUT COMPARE A interrupt
COMPARE1B,OC1B	TIMER1 OUTPUT COMPARE B interrupt
SPI	SPI interrupt
URXC	Serial RX complete interrupt
UDRE	Serial data register empty interrupt
UTXC	Serial TX complete interrupt
SERIAL	Disables URXC, UDRE and UTXC
ACI	Analog comparator interrupt
ADC	A/D converter interrupt

By default all interrupts are disabled.
To disable all interrupts specify INTERRUPTS.

To enable the enabling and disabling of individual interrupts use ENABLE INTERRUPTS.
The ENABLE INTERRUPTS serves as a master switch. It must be enabled/set in order for the individual interrupts to work.

The interrupts that are available will depend on the used microprocessor. The available interrupts are shown automatically in the editor.

 To disable the JTAG you can use DISABLED JTAG. The JTAG is not an interrupt but a device.

XTINY/MEGAX/AVRX

The newest processors can configure each pin of a port. This is done using the CONFIG XPIN statement.
The sense parameter controls how the pin is used :

INT_DISABLED : no interrupt will occur but input buffer is enabled
BOTH : on a rising or falling edge an interrupt will occur
RISING : a rising edge will trigger an interrupt
FALLING : a falling edge will trigger an interrupt
INP_DISABLED : input and digital input buffer are disabled
LOW_LEVEL : interrupt will occur on a low level

So instead of using ENABLE you need to use CONFIG XPIN and select the proper trigger mode.
Instead of DISABLE you need to use CONFIG XPIN and select INT_DISABLED for the sense mode.

See also

ENABLE 1250

Example

```
'-----------------------------------------------------------------------
```

```
------------------
'name                     : serint.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : serial interrupt example for AVR
'micro                    : 90S8535
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
------------------

$regfile = "8535def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Const Cmaxchar = 20                                         ' number of
characters

Dim B As Bit                                                ' a flag for
signalling a received character
Dim Bc As Byte                                              ' byte
counter
Dim Buf As String * Cmaxchar                                ' serial
buffer
Dim D As Byte

'Buf = Space(20)
'unremark line above for the MID() function in the ISR
'we need to fill the buffer with spaces otherwise it will contain
garbage

Print "Start"

On Urxc Rec_isr                                             ' define
serial receive ISR
Enable Urxc                                                 ' enable
receive isr

Enable Interrupts                                           ' enable
interrupts to occur

Do
  If B = 1 Then                                             ' we
received something
     Disable Serial
     Print Buf                                              ' print
buffer
     Print Bc                                               ' print
character counter


     'now check for buffer full
     If Bc = Cmaxchar Then                                  ' buffer
full
        Buf = ""                                            ' clear
        Bc = 0                                              ' rest
```

```
character counter
     End If

     Reset B                                                    ' reset
receive flag
     Enable Serial
  End If
Loop

Rec_isr:
  Print "*"
  If Bc < Cmaxchar Then                                         ' does it
fit into the buffer?
     Incr Bc                                                    ' increase
buffer counter


     If Udr = 13 Then                                           ' return?
        Buf = Buf + Chr(0)
        Bc = Cmaxchar
     Else
        Buf = Buf + Chr(udr)                                    ' add to
buffer
     End If


    ' Mid(buf , Bc , 1) = Udr
    'unremark line above and remark the line with Chr() to place
    'the character into a certain position
     'B = 1                                                     ' set flag
  End If
  B = 1                                                         ' set flag
Return
```

7.41 DO-LOOP

Action

Repeat a block of statements until condition is true.

Syntax

```
DO
  statements
LOOP [ UNTIL expression]
```

Remarks

You can exit a DO..LOOP with the EXIT DO statement.
The DO-LOOP is always performed at least once.

The main part of your code can best be executed within a DO.. LOOP.
You could use a GOTO also but it is not as clear as the DO LOOP.

```
Main:
  '  code
GOTO Main

Do
  ' Code
Loop
```

Of course in the example above, it is simple to see what happens, but when the code consist of a lot of lines of code, it is not so clear anymore what the GOTO Main does.

See also
EXIT 1257 , WHILE-WEND 1609 , FOR-NEXT 1260

Example
```
'-----------------------------------------------------------------------
-----------------
'name                     : do_loop.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: DO, LOOP
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte

A = 1                                                      'assign a
var
Do                                                         'begin a
do..loop
  Print A                                                  'print var
  Incr A                                                   'increase by
one
Loop Until A = 10                                          'do until
a=10
End

'You can write a never-ending loop with the following code
Do
  'Your code goes here
Loop
```

7.42 DTMFOUT

Action
Sends a DTMF tone to the compare1 output pin of timer 1.

Syntax
DTMFOUT number, duration
DTMFOUT string , duration

Remarks

Number	A variable or numeric constant that is equivalent with the number of your phone keypad.
Duration	Time in mS the tone will be generated.
string	A string variable that holds the digits to be dialed.

The DTMFOUT statement is based on an Atmel application note (314).

It uses TIMER1 to generate the dual tones. As a consequence, timer1 can not be used in interrupt mode by your application. You may use it for other tasks.

Since the TIMER1 is used in interrupt mode you must enable global interrupts with the statement ENABLE INTERRUPTS 1250. The compiler could do this automatic but when you use other interrupts as well it makes more sense that you enable them at the point where you want them to be enabled.

The working range is from 4 MHz to 10 MHz system clock(xtal).

The DTMF output is available on the TIMER1 OCA1 pin. For a 2313 this is PORTB.3.

Take precautions when connecting the output to your telephone line.

 Ring voltage can be dangerous!

System Resources used

TIMER1 in interrupt mode

See also

NONE

ASM

The following routine is called from mcs.lib : _DTMFOUT
R16 holds the number of the tone to generate, R24-R25 hold the duration time in mS.
Uses R9,R10,R16-R23

The DTMF table is remarked in the source and shown for completeness, it is generated by the compiler however with taking the used crystal in consideration.

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : dtmfout.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates DTMFOUT statement based on AN
314 from Atmel
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
```

```
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'since the DTMFOUT statement uses the TIMER1 interrupt you must enable
'global interrupts
'This is not done by the compiler in case you have more ISRs
Enable Interrupts


'the first sample does dtmfout in a loop
Dim Btmp As Byte , Sdtmf As String * 10

Sdtmf = "12345678"                                         ' number to
dial

Do

Dtmfout Sdtmf , 50                                        ' lets dial a
number
'              ^ duration is 50 mS for each digit
Waitms 1000                                               ' wait for
one second


' As an alternative you can send single digits
' there are 16 dtmf tones
  For Btmp = 0 To 15
   Dtmfout Btmp , 50                                       ' dtmf out
on PORTB.3 for the 2313 for 500 mS
    'output is on the OC1A output pin
    Waitms 500                                             ' wait 500
msec
Next
Loop
End

'the keypad of most phones looks like this :
'1  2  3    optional are A
'4  5  6                 B
'7  8  9                 C
'*  0  #                 D

'the DTMFOUT translates a numeric value from 0-15 into :
' numeric value    phone key
'   0                0
'   1                1
'   2                2
'   3                3
' etc.
'   9                9
'  10                *
```

```
'  11                #
'  12                A
'  13                B
'  14                C
'  15                D
```

7.43 ECHO

Action
Turns the ECHO on or off while asking for serial INPUT.

Syntax
ECHO value

Remarks

Value	ON to enable ECHO and OFF to disable ECHO.

When you use INPUT to retrieve values for variables, all info you type can be echoed back. In this case you will see each character you enter. When ECHO is OFF, you will not see the characters you enter.

In versions 1.11.6.2 and earlier the ECHO options were controlled by an additional parameter on the INPUT statement line like : INPUT "Hello " , var NOECHO

This would suppress the ECHO of the typed data. The new syntax works by setting ECHO ON and OFF. For backwards compatibility, using NOECHO on the INPUT statement line will also work. In effect it will turn echo off and on automatic.

By default, ECHO is always ON.

See also
INPUT 1493

ASM
The called routines from mcs.lib are _ECHO_ON and _ECHO_OFF

The following ASM is generated when you turn ECHO OFF.
Rcall Echo_Off
This will set bit 3 in R6 that holds the ECHO state.

When you turn the echo ON the following code will be generated
Rcall Echo_On

Example
```
'-------------------------------------------------------------------------
-----------------
'name                    : input.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demo: INPUT, INPUTHEX
'micro                   : Mega48
```

```
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15

Input "Use this to ask a question " , V
Input B1                                                   'leave out
for no question

Input "Enter integer " , C
Print C


Inputhex "Enter hex number (4 bytes) " , C
Print C
Inputhex "Enter hex byte (2 bytes) " , D
Print D

Input "More variables " , C , D
Print C ; " " ; D

Input C Noecho                                             'supress
echo

Input "Enter your name " , S
Print "Hello " ; S

Input S Noecho                                             'without
echo
Print S
End
```

7.44 ELSE

Action

Executed if the IF-THEN expression is false.

Syntax

ELSE

Remarks

You don't have to use the ELSE statement in an IF THEN .. END IF structure.

You can use the ELSEIF statement to test for another condition.

```
IF a = 1 THEN
...
ELSEIF a = 2 THEN
..
ELSEIF b1 > a THEN
...
ELSE
...
END IF
```

See also

IF 1313 , END IF 1313 , SELECT-CASE 1437

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : if_then.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: IF, THEN, ELSE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte , B1 As Byte

Input "Number " , A                                        'ask for
number
If A = 1 Then                                              'test number
   Print "You got it!"
End If

If A = 0 Then                                              'test again
   Print "Wrong"                                           'thats wrong
Else                                                       'print this
if a is not 0
   Print "Almost?"
End If

Rem You Can Nest If Then Statements Like This
B1 = 0
If A = 1 Then
  If B1 = 0 Then
```

```
      Print "B1=0"
  End If
Else
  Print "A is not 0"
End If

Input "Number " , A
If A = 1 Then                                                    '
  Print "Ok"
Elseif A = 2 Then                                                'use elseif
for more tests
  Print "2" : A = 3
Elseif A = 3 Then
  Print "3"
End If

If A.1 = 1 Then Print "Bit 1 set"                                'test for  a
bit
End
```

7.45 ENABLE

Action

Enable specified interrupt.
(ATTINY, ATMEGA, ATXMEGA)

Syntax

ENABLE interrupt [, prio]

[, prio] is only for ATXMEGA

Remarks

Interrupt	Description
INT0	External Interrupt 0
INT1	External Interrupt 1
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt
OVF1,TIMER1, COUNTER1	TIMER1 overflow interrupt
CAPTURE1, ICP1	INPUT CAPTURE TIMER1 interrupt
COMPARE1A,OC1A or COMPARE1, OC1	TIMER1 OUTPUT COMPARE A interrupt In case of only one compare interrupt
COMPARE1B,OC1B	TIMER1 OUTPUT COMPARE B interrupt
SPI	SPI interrupt
URXC	Serial RX complete interrupt
UDRE	Serial data register empty interrupt
UTXC	Serial TX complete interrupt
SERIAL	Disables URXC, UDRE and UTXC
ACI	Analog comparator interrupt
ADC	A/D converter interrupt
XMEGA ONLY	

prio	The priority you want to assign to the interrupt. Specify Lo, Hi or Med. In the Xmega you must provide the priority of the interrupts. Lo=Low priority. Hi=High priority and Med=Medium priority. If you do not specify a priority, **MED** will be used.

By default all interrupts are disabled.
The global interrupts master switch is also disabled by default.
If you enable an interrupt, it will only fire if the master interrupt switch is enabled.
You enable this master switch with ENABLE INTERRUPTS.
You can disable it with DISABLE INTERRUPTS.

If an interrupt is executed, the global master switch will be disabled automatically by the hardware.
This is to prevent other interrupts to occur.
When the interrupt routine returns, the processor hardware will automatically enable the master switch so new interrupts may occur.

The following schematic demonstrates the interrupt master switch. It forms an AND with the other interrupts. This means that both the interrupt of a hardware source and the master switch interrupt must be enabled. ENABLE interrupts will enable the I flag in SREG.

It depends on the processor how many and which interrupts it has. If you type ENABLE in the editor, you will get a pop up with a list of interrupts you can chose from.

ATTINY & ATMEGA Interrupt List

In normal AVR chips the priority is determined by the interrupts address. The lower the address, the higher the priority.
In the DAT file you can find a list with interrupts and their address.
For example , taken from the m1280def.dat file :

```
[INTLIST]
count=56
INTname1=INT0,$002,EIMSK.INT0,EIFR.INTF0
INTname2=INT1,$004,EIMSK.INT1,EIFR.INTF1
INTname3=INT2,$006,EIMSK.INT2,EIFR.INTF2
INTname4=INT3,$008,EIMSK.INT3,EIFR.INTF3
INTname5=INT4,$00a,EIMSK.INT4,EIFR.INTF4
INTname6=INT5,$00c,EIMSK.INT5,EIFR.INTF5
```

INT0 has the highest priority since it has the lowest address (address 2)

Following an Overview where **INT0** is used as an example.

Overview

1. You configure an Interrupt

```
On Int0 Int0_isr
Config Int0 = Low Level
```

2. You enable the specific Interrupt

```
Enable Int0
```

3. Enable all Interrupts

```
Enable Interrupts
```

4. You have an Interrupt Service Routine after the **"End"** with an **Return**

```
End

'    Interrupt    Service    Routine
Int0_isr:
'    so    someting.....
Return
```

XMEGA

The XMEGA has a priority system. You can specify if an interrupt has a low, medium or high priority.
But you MUST enable these priorities with CONFIG PRIORITY 1026

Please read the topic CONFIG PRIORITY 1026 in order to understand which interrupt to enable.

In the DAT file you can find a list with interrupts and their address.
For example , taken from the "xm128A4Udef.dat file "

```
INTLIST]
count=95
INTname1=OSCFAIL,$0002,OSC_XOSCFAIL.0,OSC_XOSCFAIL.1         ; XOSC Failure Detectio
INTname2=PORTC_INT0,$0004,#PORTC_INTCTRL.0,PORTC_INTFLAGS.0
INTname3=PORTC_INT1,$0006,#PORTC_INTCTRL.2,PORTC_INTFLAGS.1
INTname4=PORTR_INT0,$0008,#PORTR_INTCTRL.0,PORTR_INTFLAGS.0
INTname5=PORTR_INT1,$000A,#PORTR_INTCTRL.2,PORTR_INTFLAGS.1
INTname6=DMA_CH0,$000C,#,DMA_CH0_CTRLB.0,DMA_CH0_CTRLB.4
INTname7=DMA_CH1,$000E,#,DMA_CH1_CTRLB.0,DMA_CH1_CTRLB.4
INTname8=DMA_CH2,$0010,#,DMA_CH2_CTRLB.0,DMA_CH2_CTRLB.4
INTname9=DMA_CH3,$0012,#,DMA_CH3_CTRLB.0,DMA_CH3_CTRLB.4
INTname10=RTC_OVF,$0014,#RTC_INTCTRL.0,RTC_INTFLAGS.0
INTname11=RTC_COMP,$0016,#RTC_INTCTRL.2,RTC_INTFLAGS.1
```

Example with **PORTC_INT0**

```
On Portc_int0 portc_isr
Enable Portc_int0 , Hi
```

In ATXMEGA 425 there is an example for Pin Interrupt.

XTINY/MEGAX/AVRX

The newest processors can configure each pin of a port. This is done using the CONFIG XPIN statement.
The sense parameter controls how the pin is used :

INT_DISABLED : no interrupt will occur but input buffer is enabled

BOTH : on a rising or falling edge an interrupt will occur
RISING : a rising edge will trigger an interrupt
FALLING : a falling edge will trigger an interrupt
INP_DISABLED : input and digital input buffer are disabled
LOW_LEVEL : interrupt will occur on a low level

So instead of using ENABLE you need to use CONFIG XPIN and select the proper trigger mode.
Instead of DISABLE you need to use CONFIG XPIN and select INT_DISABLED for the sense mode.

See also

DISABLE 1240 , ON 1379 , CONFIG PRIORITY 1026, ATXMEGA 425

Example

```
$regfile = "attiny25.dat"
$crystal = 1000000                                          ' 1MHz
$hwstack = 10
$swstack = 0
$framesize = 24


On Int0 Int0_isr                                            ' INT0 will
be the wake-up source for Powerdown Mode
Config Int0 = Low Level                                     ' External
Pull-up (47K) on Portb.2
Enable Int0

'###############################################################################
########
Do
   Wait 3                                                   ' now we
have 3 second to measure the Supply Current in Active Mode

   Enable Interrupts

   ' Now call Powerdown function
   Config Powermode = Powerdown

   'Here you have time to measure PowerDown current consumption until a
Low Level on Portb.1 which is the PowerDown wake-up
Loop
'###############################################################################
########
End


Int0_isr:
' wake_up
Return
```

7.46 ENCODER

Action
Reads pulses from a rotary encoder.

Syntax
Var = **ENCODER**(pin1, pin2, LeftLabel, RightLabel , wait)

Remarks

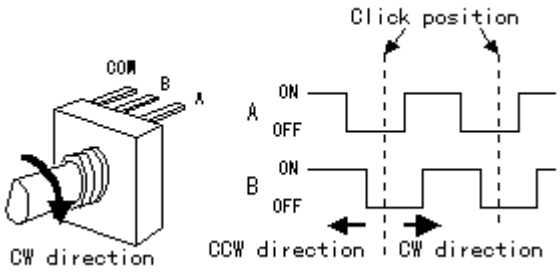
Var	The target variable that is assigned with the result. This should be a byte. This byte is used to maintain the state.
Pin1 and pin2	These are the names of the PIN registers to which the output of the encoder is connected. Both pins must be on the same PIN register. So Pinb.0 and Pinb.7 is valid while PinB.0 and PinA.0 is not.
LeftLabel	The name of the label that will be called/executed when a transition to the left is encountered.
RightLabel	The name of the label that will be called/executed when a transition to the right is encountered.
wait	A value of 0 will only check for a rotation/pulse. While a value of 1 will wait until a user actual turns the encoder. A value of 1 will thus halt your program.

There are some conditions you need to fulfill :

- The label that is called by the encoder must be terminated by a RETURN statement.
- The pin must work in the input mode. By default all pins work in input mode.
- The pull up resistors must be activated by writing a logic 1 to the port registers as the example shows.

Rotary encoders come in many flavors. Some encoders also have a build in switch.

A sample of an encoder



Since the microprocessor has internal pull up resistors, you do not need external pull up resistors for most encoders.
An encoder has 2 output pins which change state when you turn the knob. For one 'click' you can get one or more pulses. This depends on the model of the encoder. Both output pins are sampled and compared with their previous value.

RIGHT	STATE	01	11	10	00
OLD		0000_0000	0001_0000	0011_0000	0010_0000
NEW		0000_0001	0000_0011	0000_0010	0000_0000
VALUE		1	19	50	32
LEFT	STATE	10	11	01	00
OLD		0000_0000	0010_0000	0011_0000	0001_0000
NEW		0000_0010	0000_0011	0000_0001	0000_0000
VALUE		2	35	49	16

The table above show the states when rotating left and right. For example, when you turn left, the encoder will change state from 00 to 10 to 11 to 01 to 00 etc.
The software loads the pin values and compares the value with the previous value. Only if you turn the knob there will be a different value.
Next the old state nibbles are swapped so that for example state 0000_0011 becomes 0011_0000 and the new state is added to this value. For a left rotation you get the values 2,35,49 and 16. In all other cases, the rotation was right.

When you call the encoder routine often enough, you will not miss any pulses. Most new processors support the pin change interrupt. This means that an interrupt occurs when the logic level of a pin changes. you can use this interrupt to call the encoder function. This way you can be sure you will not miss a pulse.

The example will just show the direction but the idea is that you can increase or decrease a variable in these routines. For example for volume.

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : encoder.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstration of encoder function
'micro                    : Mega128
'suited for demo          : yes
'commercial addon needed  : no
```

```
'An encoder has 2 outputs and a ground
'We connect the outputs to pinb.0 and pinb.1
'You may choose different pins as long as they are at the same PORT
'The pins must be configured to work as input pins
'This function works for all PIN registers
'-----------------------------------------------------------------------
------------------

$regfile = "m128def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Print "Encoder test"
Dim B As Byte
'we have dimmed a byte because we need to maintain the state of the
encoder

Portb = &B11                                                ' activate
pull up registers

Do
   B = Encoder(pinb.0 , Pinb.1 , Links , Rechts , 1)
   '                                                  ^--- 1 means wait for
change which blocks programflow
   '                                  ^--------^---------- labels which are
called
   '              ^-------^---------------------------- port PINs
   Print B
  Waitms 10
Loop
End

'so while you can choose PINB0 and PINB7,they must be both member of
PINB
'this works on all PIN registers

Links:
  Print "left rotation"
Return

Rechts:
  Print "right rotation"
Return
End
```

7.47 END

Action

Terminate program execution.

Syntax

```
END
```

Remarks
STOP can also be used to terminate a program.

When an END statement is encountered, all interrupts are disabled and a never-ending loop is generated.
When a STOP is encountered the interrupts will not be disabled. Only a never ending loop will be created.

In an embedded application you probably do not want to end the application. But there are cases where you do want to end the application. For example when you control some motors, and you determine a failure, you do not want to use a Watchdog reset because then the failure will occur again. In that case you want to display an error, and wait for service personal to fix the failure.

It is important to notice that without the END statement, your program can behave strange in certain cases. For example :
Print "Hello"

Note that there is no END statement. So what will happen? The program will print "Hello". But as the compiler places the library code behind the program code, the micro will execute the library code ! But without being called. As most library code are assembler sub routines that end with a RET, your program will most likely crash, or reset and repeat for ever.

See also
STOP 1544

Example
```
Print "Hello"        ' print this
End                  ' end program execution and disable all interrupts
```

7.48 EXIT

Action
Exit a FOR..NEXT, DO..LOOP , WHILE ..WEND, SUB..END SUB or FUNCTION..END FUNCTION.

Syntax
EXIT FOR
EXIT DO
EXIT WHILE
EXIT SUB
EXIT FUNCTION

Remarks
With the EXIT statement you can exit a structure at any time.

Remarks about EXIT SUB/FUNCTION
It is important that you exit a SUB or FUNCTION with EXIT. Do not use a RETURN. A

return can be used inside a sub routine to return from a sub routine located inside the sub routine.
For example:

```
Sub Test()
  gosub label1
  Exit Sub

label1:
  print "test"
return
End Sub
```

When you use EXIT SUB or EXIT FUNCTION, the compiler will create a jump to a label with the sub/function name, prefixed with two underscores.
For example your Sub routine is named Test(), and you use Exit Sub, a label will be created with the name __TEST:

See Also

DO 1243, WHILE 42, FOR 1260 , SUB 1545, FUNCTION 1545 , CONTINUE 1168, REDO 1422

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : exit.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: EXIT
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim B1 As Byte , A As Byte

B1 = 50                                                    'assign var
For A = 1 To 100                                           'for next
loop
   If A = B1 Then                                          'decision
      Exit For                                             'exit loop
   End If
Next
Print "Exit the FOR..NEXT when A was " ; A

A = 1
Do
```

```
  Incr A
  If A = 10 Then
     Exit Do
  End If
Loop
Print "Loop terminated"
End
```

7.49 FLIP

Action
Flips the bits in a byte.

Syntax
var = **FLIP**(s)

Remarks

Var	The variable that is assigned with the flipped byte S.
S	The source variable to flip.

The FLIP function can be useful in cases where you have reversed the data lines d0-d7.
It will reverse or mirror the bits

See also
NONE

Example
```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
$hwstack=32
$swstack = 16
$framesize=24

Dim B As Byte , V As Byte

For B = 1 To 20
    V = Flip(b)
    Print B ; " " ; Bin(b) ; " " ; Bin(v)
Next

End
```

OUTPUT
```
1 00000001 10000000
2 00000010 01000000
3 00000011 11000000
4 00000100 00100000
5 00000101 10100000
6 00000110 01100000
```

```
7 00000111 11100000
8 00001000 00010000
9 00001001 10010000
10 00001010 01010000
11 00001011 11010000
12 00001100 00110000
13 00001101 10110000
14 00001110 01110000
15 00001111 11110000
16 00010000 00001000
17 00010001 10001000
18 00010010 01001000
19 00010011 11001000
20 00010100 00101000
```

7.50 FOR-NEXT

Action

Execute a block of statements a number of times.

Syntax

FOR var = start **TO** end [**STEP** value]

Remarks

var	The variable counter to use
start	The starting value of the variable var
end	The ending value of the variable var
value	The value var is increased/decreased with each time NEXT is encountered.

- For incremental loops, you must use TO.
- For decremental loops, you must use a negative step size.
- You must end a FOR structure with the NEXT statement.
- The use of STEP is optional. By default, a value of 1 is used.

When you know in advance how many times a block of code must be executed, the FOR..NEXT loop is convenient to use.
You can exit a FOR .. NEXT loop with the EXIT FOR statement.

It is important that the if you use variables for START and END, that these are of the same data type. So for example:

```
Dim x, as byte, st as byte, ed as byte
FOR x = st TO ED ' this is ok since all variables are of the same data type

Dim x as Byte, st as Word, Ed as Long
FOR x = st TO ED ' this is NOT ok since all variables are of different data type.
```

The reason is that when the condition is evaluated, it will create a compare on 2 bytes, while you actually want to have a word since the end variable is a word.

A for next loop with an integer has an upper limit of 32766 and not 32767, the maximum value that fits into an integer.
This is done in order to save code space. Checking an overflow from 32767 to -32768 would cost extra code.

There are also other alternatives. You can use a Do.. Loop for example :

```
Dim Var As Byte
Do
  'code
   Incr Var
Loop Until Var = 10
```

There are various way to get the result you need.

See also

EXIT FOR 1257

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : for_next.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: FOR, NEXT
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte , B1 As Byte , C As Integer


For A = 1 To 10 Step 2
   Print "This is A " ; A
Next A

Print "Now lets count down"
For C = 10 To -5 Step -1
  Print "This is C " ; C
Next



Print "You can also nest FOR..NEXT statements."
For A = 1 To 10
  Print "This is A " ; A
  For B1 = 1 To 10
    Print "This is B1 " ; B1
  Next                                                     ' note that
```

```
you do not have to specify the parameter
Next A
End
```

7.51 GET

Action
Reads a byte from the hardware or software UART.
Reads data from a file opened in BINARY mode.

Syntax UART
GET #channel, var

Syntax DOS
GET #channel, var
GET #channel, var , [pos] [, length]

Remarks
GET in combination with the software/hardware UART reads one byte from the UART.
GET in combination with the AVR-DOS file system is very flexible and versatile. It works on files opened in BINARY mode and you can reads all data types.

#channel	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Var	The variable or variable array that will be assigned with the data from the file
Pos	This is an optional parameter that may be used to specify the position where the reading must start from. This must be a long variable.
Length	This is an optional parameter that may be used to specify how many bytes must be read from the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte will be read. When the variable is a word or integer, 2 bytes will be read. When the variable is a long or single, 4 bytes will be read. When the variable is a string, the number of bytes that will be read is equal to the dimensioned size of the string. DIM S as string * 10 , would read 10 bytes.

Note that when you specify the length for a string, the maximum length is 254. The maximum length for a non-string array is 65535.

In BASCOM-8051, GET was implemented to read only from the UART. While BASCOM-AVR supports GET for the UART, its primary purpose is to read from files with AVR-DOS. For the UART, GET is limited to read 1 byte, just like WAITKEY.

Partial Example :
```
GET #1 , var ,,2        ' read 2 bytes, start at current position
GET #1, var , PS        ' start at position stored in long PS
GET #1, var , PS, 2     ' start at position stored in long PS and read 2 bytes
```

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC

795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

current position	goto new position first
Byte:	
_FileGetRange_1 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_1 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Word/Integer:	
_FileGetRange_2 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_2 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Long/Single:	
_FileGetRange_4 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_4 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
String (<= 255 Bytes) with fixed length	
_FileGetRange_Bytes Input: r24: File number r20: Count of Bytes X: Pointer to variable T-Flag cleared	_FileGetRange_Bytes Input: r24: File number r20: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based)

	T-Flag Set
Array (> 255 Bytes) with fixed length	
_FileGetRange Input: r24: File number r20/21: Count of Bytes X: Pointer to variable T-Flag cleared	_FileGetRange Input: r24: File number r20/21: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set

Output from all kind of usage:
r25: Error Code
C-Flag on Error
X: requested info

Partial Example

```
'for the binary file demo we need some variables of different types
Dim B As Byte , W As Word , L As Long , Sn As Single , Ltemp As Long
Dim Stxt As String * 10
B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt = "test"

'open the file in BINARY mode
Open "test.biN"for Binary As #2
Put#2 , B ' write a byte
Put#2 , W ' write a word
Put#2 , L ' write a long
Ltemp = Loc(#2) + 1                                          ' get the
position of the next byte
Print Ltemp ; " LOC"                                         ' store the
location of the file pointer
Print Seek(#2) ; " = LOC+1"

Print Lof(#2) ; " length of file"
Print Fileattr(#2) ; " file mode"                            ' should be
32 for binary
Put #2 , Sn                                                  ' write a
single
Put #2 , Stxt                                                ' write a
string

Flush #2                                                     ' flush to
disk
Close #2

'now open the file again and write only the single
Open "test.bin" For Binary As #2
L = 1 'specify the file position
B = Seek(#2 , L)                                             ' reset is
the same as using SEEK #2,L
Get#2 , B ' get the byte
Get#2 , W ' get the word
Get#2 , L ' get the long
Get#2 , Sn ' get the single
```

```
Get#2 , Stxt ' get the string
Close #2
```

7.52 GETADC

Action
Retrieves the analog value from the specified channel.

Syntax
var = **GETADC**(channel [,offset])

Syntax Xmega
var = **GETADC**(ADC , channel [,MUX])

Syntax Xtiny
var = **GETADC**()
var = **GETADC**(channel)
var = **GETADC**(ADC , channel)

Remarks AVR

Var	The variable that is assigned with the A/D value. This should be a Word or other 16 bit variable.
Channel	The channel to measure. This is actual the MUX value that will be used. Most older chips with A/D converter only have 8 channels with singled ended input. Here you would use values from 0-7. Newer chips like the ATMEGA2560 have multiple modes. A MUX value of 0-7 would use single ended input mode and would read ADC0-ADC7. But a value from 8-15 would select differential mode using ADC0-ADC3 with different gain factors. Please have a look in the data sheet to see how the channel value translates into the mode and channel. It is different for most chips.
Offset	An optional numeric variable of constant that specifies gain or mode. This option has effect on newer AVR micro's only. The offset will be added to the channel value and inserted into the ADMUX register. This way you can control gain.

Remarks XMEGA

var	The variable that is assigned with the A/D value. This should be a Word or other 16 bit variable.
ADC	The ADC to use. This is either ADCA or ADCB.
Channel	The channel to use. There are 4 channels in the range from 0-3.
MUX	An optional numeric variable or constant that specifies the MUX value thus which input pin is used for the measurement. The MUX number is coded with negative and positive input pin info. The positive pins are have an offset of 8. So PIN0 in single ended mode would need a value of 8. When you do not supply the mux value, the value used by the CONFIG

	ADC command will be used. If you supply it, it will change the MUX register of the corresponding channel.

Remarks Xtiny

var	The variable that is assigned with the A/D value. This should be a Word or other 16 bit variable.
ADC	The ADC to use. This is either ADC0 or ADC1.
Channel	The channel to use. This value depends on the processor. Some channel values access the internal reference or temperature sensor. This value will set the MUX register. When there is no channel provided the current MUX setting will be used. When you define a constant named _adc_kelvin reading the internal temp sensor will return the result in Kelvin. The value of _adc_kelvin is not important. When you include this constant, the ADC routine will check if the internal temperature sensor is read. If so, the result is compensated with the temp gain and offset. To convert Kelvin to Celsius you can subtract 273.15 from the result.

Note:
It is the users responsibility to check the *Channel* values are in range.
Please check and consult your Microcontroller Datasheet.

The GETADC() function only will work on microprocessors that have an A/D converter.
The pins of the A/D converter input can be used for digital I/O too.
But it is important that no I/O switching is done while using the A/D converter.

NORMAL AVR
Make sure you turn on the AD converter with the START 1538 ADC statement or by setting the proper bit in the ADC configuration register.

Some micro's have more then 7 channels. This is supported as well. The ADCSRB register contains a bit named MUX5 that must be set when a channel higher then 7 is used. The compiler will handle this automatic. This is true for new chips like Mega1280, Mega2560 and probably other new chips with 100 pins.

An example on how to read singled ended input on a Mega1280:

```
W = Getadc(0, 32)  ' from data sheet :  100000 ADC8
W = Getadc(1, 32)   ' from data sheet :  100001 ADC9
```

This will read channel 0 and 1. The offset is 32 in order to use singled ended input.
ADC8 is portK.0

Without the offset, you need to provide the proper value for the channel.
So GetADC(0,32) would become : GetADC(32)
And GetADC(1,32) would become : GetADC(33)

GetADC() returns a word variable since the A/D converter data registers consist of 2 registers. The resolution depends on the chip.
The variable ADCD can be used to access the data register directly. The compiler will handle access to the byte registers automatically.

See also

CONFIG ADC 864 , CONFIG ADCA 867 , CONFIG ADC0 880

Example

```
'-----------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
or M163 micro
'micro                    : Mega163
'suited for demo          : yes
'commercial addon needed  : no
'use in simulator         : possible
' Getadc() will also work for other AVR chips that have an ADC converter
'-----------------------------------------------------------------------
---------
$regfile = "m163def.dat"                                    ' we use the
M163
$crystal = 4000000

$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               'default use
10 for the SW stack
$framesize = 40                                             'default use
40 for the frame space


'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc

'With STOP ADC, you can remove the power from the chip
'Stop Adc

Dim W As Word , Channel As Byte

Channel = 0
'now read A/D value from channel 0
Do
  W = Getadc(channel)
  Print "Channel " ; Channel ; " value " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
Loop
End

'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
'OFF      : AREF, internal reference turned off
'AVCC     : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin

'Using the additional param on chip that do not have the internal
```

reference will have no effect.

Example Xmega

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-ADC.bas
'  This sample demonstrates the Xmega128A1 ADC
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 64
$framesize = 64


'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

Print "ADC test"

'setup the ADC-A converter
Config Adca = Single , Convmode = Unsigned , Resolution = 12bit , Dma =
Off , Reference = Int1v , Event_mode = None , Prescaler = 32 , Ch0_gain
= 1 , Ch0_inp = Single_ended , Mux0 = &B000_00 _
 Ch1_gain = 1 , Ch1_inp = Single_ended , Mux1 = &B1_000 , Ch2_gain = 1 ,
Ch2_inp = Single_ended , Mux2 = &B10_000 , Ch3_gain = 1 , Ch3_inp =
Single_ended , Mux3 = &B11_000

Dim W As Word , I As Byte , Mux As Byte
Do
   Mux = I * 8            ' or you can use shift left,3 to get the
proper offset
   W = Getadc(adca , 0 , Mux)
'   W = Getadc(adca , 0)   'when not using the MUX parameter the last
value of the MUX will be used!
   ' use ADCA , use channel 0, and use the pinA.0-pinA.3
   Print "RES:" ; I ; "-" ; W
   Incr I
   If I > 3 Then I = 0
  Waitms 500
Loop Until Inkey(#1) = 27
```

Example Xtiny

```
'-----------------------------------------------------------------
----------------
'name                    : adc.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates ADC and DAC. Notice that
DAC is not available on all processors
'micro                   : xtiny816
'suited for demo         : no
```

```
'commercial addon needed   : yes
'-------------------------------------------------------------------
----------------
$regfile = "atXtiny816.dat"
$crystal = 20000000
$hwstack = 16
$swstack = 16
$framesize = 24

'set the system clock and prescaler
Config Sysclock = 20mhz , Prescale = 1

'configure the USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'configure the internal reference to be 1v1 for both the ADC and
the DAC
Config Vref = Dummy , Adc0 = 1v1 , Dac0 = 1v1

'configure the ADC0 to read the DAC
Config Adc0 = Single , Resolution = 10bit , Adc = Enabled ,
Reference = Internal , Prescaler = 32 , Sample_len = 1 ,
Sample_cap = Above_1v , Init_delay = 32 , Mux = Dac0

'configure the DAC. We do not output the signal on a port pin
otherwise out_enable would be required too
Config Dac0 = Enabled

'dimension a variable
Dim W As Word , B As Byte

Print "Test ADC"

'set the DAC to halve the output which would be halve of 1.1V
which is 0.55V
Dac0_data = 127

Do
  'when getadc() does not have parameters, it will use the
current mux setting
  'other options are : getadc(channel)  and getadc(adc0 | adc1 ,
channel)
   W = Getadc() : Print "W:" ; W
   'output should be 512
   Waitms 1000
   Incr B
Loop Until B = 3
```

```
'now read the internal temp sensor
Const _adc_kelvin = 1
Do
  W = Getadc(&H1e)                                          'get
internal temp sensor value
  Print W                                                   'this
is in KELVIN
  'to adjust to Celsius, sub 273.15
  Waitms 1000
Loop

End
```

7.53 GETATKBD

Action

Reads a key from a PC AT keyboard.

Syntax

var = **GETATKBD**()

Remarks

var	The variable that is assigned with the key read from the keyboard. It may be a byte or a string variable. When no key is pressed a 0 will be returned.

The GETAKBD() function needs 2 input pins and a translation table for the keys. You can read more about this at the CONFIG KEYBOARD 989 compiler directive.

The Getatkbd function will wait for a pressed key. When you want to escape from the waiting loop you can set the ERR bit from an interrupt routine for example.

Getatkbd is using 2 bits from register R6 : bit 4 and 5 are used to hold the shift and control key status.

AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

http://panda.cs.ndsu.nodak.edu/~achapwes

KEY	MAKE	BREAK		KEY	MAKE	BREAK		KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0, F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0, F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0, F0,7D

E	24	F0,24	\	5D	F0,5D	DELETE	E0,71	E0, F0,71
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	E0, F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0, F0,7A
H	33	F0,33	TAB	0D	F0,0D	U ARROW	E0,75	E0, F0,75
I	43	F0,43	CAPS	58	F0,58	L ARROW	E0,6B	E0, F0,6B
J	3B	F0,3B	L SHFT	12	FO,12	D ARROW	E0,72	E0, F0,72
K	42	F0,42	L CTRL	14	FO,14	R ARROW	E0,74	E0, F0,74
L	4B	F0,4B	L GUI	E0,1F	E0,F0,1F	NUM	77	F0,77
M	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	E0, F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	F0,7C
O	44	F0,44	R CTRL	E0,14	E0,F0,14	KP -	7B	F0,7B
P	4D	F0,4D	R GUI	E0,27	E0,F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	E0,F0,11	KP EN	E0,5A	E0, F0,5A
R	2D	F0,2D	APPS	E0,2F	E0,F0,2F	KP .	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
T	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	05	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	06	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	04	F0,04	KP 4	6B	F0,6B
X	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	03	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	01	F0,01	]	5B	F0,5B
3	26	F0,26	F10	09	F0,09	;	4C	F0,4C
4	25	F0,25	F11	78	F0,78	'	52	F0,52
5	2E	F0,2E	F12	07	F0,07	,	41	F0,41
6	36	F0,36	PRNT SCRN	E0,12 , E0,7C	E0,F0, 7C,E0, F0,12	.	49	F0,49
7	3D	F0,3D	SCROLL	7E	F0,7E	/	4A	F0,4A
8	3E	F0,3E	PAUSE	E1,14 ,77, E1, F0,14 , F0,77	-NONE-			

These are the usable scan codes from the keyboard. If you want to implement F1 ,

you look at the generated scan code : 05 hex. So in the table, at position 5+1=6, you write the value for F1.

In the sample program below, you can find the value 200. When you now press F1, the value form the table will be used so 200 will be returned.

See also

CONFIG KEYBOARD 989 , GETATKBDRAW 1274

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : getatkbd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : PC AT-KEYBOARD Sample
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "8535def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'For this example :
'connect PC AT keyboard clock to PIND.2 on the 8535
'connect PC AT keyboard data to PIND.4 on the 8535

'The GetATKBD() function does not use an interrupt.
'But it waits until a key was pressed!

'configure the pins to use for the clock and data
'can be any pin that can serve as an input
'Keydata is the label of the key translation table
Config Keyboard = Pind.2 , Data = Pind.4 , Keydata = Keydata

'Dim some used variables
Dim S As String * 12
Dim B As Byte

'In this example we use SERIAL(COM) INPUT redirection
$serialinput = Kbdinput

'Show the program is running
Print "hello"

Do
  'The following code is remarked but show how to use the GetATKBD()
function
  ' B = Getatkbd()     'get a byte and store it into byte variable
```

```
  'When no real key is pressed the result is 0
  'So test if the result was > 0
  ' If B > 0 Then
  '    Print B ; Chr(b)
  ' End If

  'The purpose of this sample was how to use a PC AT keyboard
  'The input that normally comes from the serial port is redirected to
the
  'external keyboard so you use it to type
  Input "Name " , S
  'and show the result
  Print S
  'now wait for the F1 key , we defined the number 200 for F1 in the
table
  Do
    B = Getatkbd()
  Loop Until B <> 0
  Print B
Loop
End

'Since we do a redirection we call the routine from the redirection
routine
'
Kbdinput:
'we come here when input is required from the COM port
'So we pass the key into R24 with the GetATkbd function
' We need some ASM code to save the registers used by the function
$asm
push r16            ; save used register
push r25
push r26
push r27

Kbdinput1:
rCall _getatkbd     ; call the function
tst r24             ; check for zero
breq Kbdinput1      ; yes so try again
pop r27             ; we got a valid key so restore registers
pop r26
pop r25
pop r16
$end Asm
'just return
Return

'The tricky part is that you MUST include a normal call to the routine
'otherwise you get an error
'This is no clean solution and will be changed
B = Getatkbd()

'This is the key translation table

Keydata:
'normal keys lower case
Data 0 , 0 , 0 , 0 , 0 , 200 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0
Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 ,
50 , 0
Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116 ,
114 , 53 , 0
Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 , 117
, 55 , 56 , 0
Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 ,
```

```
112 , 43 , 0
Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0
Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 ,
0 , 0

'shifted keys UPPER case
Data 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
Data 0 , 0 , 0 , 0 , 0 , 81 , 33 , 0 , 0 , 0 , 90 , 83 , 65 , 87 , 34 ,
0
Data 0 , 67 , 88 , 68 , 69 , 0 , 35 , 0 , 0 , 32 , 86 , 70 , 84 , 82 ,
37 , 0
Data 0 , 78 , 66 , 72 , 71 , 89 , 38 , 0 , 0 , 76 , 77 , 74 , 85 , 47 ,
40 , 0
Data 0 , 59 , 75 , 73 , 79 , 61 , 41 , 0 , 0 , 58 , 95 , 76 , 48 , 80 ,
63 , 0
Data 0 , 0 , 0 , 0 , 0 , 96 , 0 , 0 , 0 , 0 , 13 , 94 , 0 , 42 , 0 , 0
Data 0 , 62 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 ,
0 , 0
```

7.54 GETATKBDRAW

Action
Reads a key from a PC AT keyboard.

Syntax
var = **GETATKBDRAW**()

Remarks

var	The variable that is assigned with the key read from the keyboard. It may be a byte or a string variable. When no key is pressed a 0 will be returned.

The GETATKBDRAW() function needs 2 input pins and a translation table for the keys. You can read more about this at the CONFIG KEYBOARD 989 compiler directive.

The GetatkbdRAW function will return RAW data from a PS/2 keyboard or Mouse.

While GetatKBD is intended to wait for pressed keys, GetATkbdRAW just returns raw PS/2 data so you can use your own code to process the data.

See Also
GETATKBD 1270 , CONFIG KEYBOARD 989

Example
See GETATKBD.BAS

7.55 GETKBD

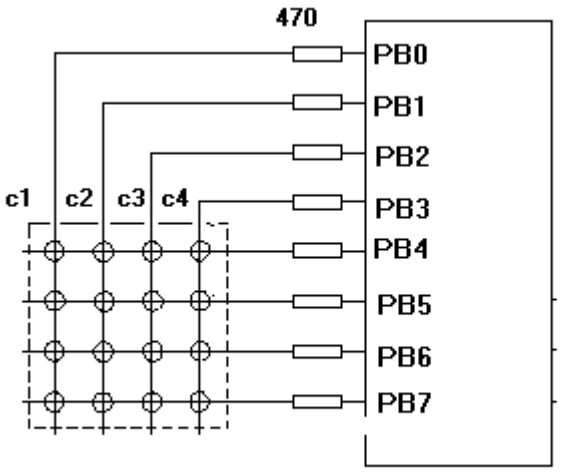
Action
Scans a 4x4 matrix keyboard and return the value of the key pressed.

Syntax
var = **GETKBD**()

Remarks

Var	The numeric variable that is assigned with the value read from the keyboard

The GETKBD() function can be attached to a port of the uP.
You can define the port with the CONFIG KBD statement.
A schematic for PORTB is shown below



Note that the port pins can be used for other tasks as well. But you might need to set the port direction of those pins after you have used getkbd(). For example the LCD pins are set to output at the start of your program. A call to getkbd() would set the pins to input.

By setting DDR.x register you can set the pins to the proper state again.
As an alternative you can use CONFIG PIN or CONFIG PORT.

When no key is pressed 16 will be returned.

When using the 2 additional rows, 24 will be returned when no key is pressed.

On the STK200 this might not work since other hardware is connected too that interferes.

You can use the Lookup() 1365 function to convert the byte into another value. This because the GetKBD() function does not return the same value as the key pressed. It will depend on which keyboard you use.

Sometimes it can happen that it looks like a key is pressed while you do not press a

key. This is caused by the scanning of the pins which happens at a very high frequency.

It will depend on the used keyboard. You can add series resistors with a value of 470-1K

The routine will wait for 100 mS by default after the code is retrieved. With CONFIG KBD you can set this delay.

See also

CONFIG KBD 988

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : getkbd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : GETKBD
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'specify which port must be used
'all 8 pins of the port are used
Config Kbd = Portb

'dimension a variable that receives the value of the pressed key
Dim B As Byte

'loop for ever
Do
  B = Getkbd()
  'look in the help file on how to connect the matrix keyboard
  'when you simulate the getkbd() it is important that you press/click
the keyboard button
  ' before running the getkbd() line !!!
  Print B
  'when no key is pressed 16 will be returned
  'use the Lookup() function to translate the value to another one
' this because the returned value does not match the number on the
keyboad
Loop
End
```

7.56 GETRC

Action
Retrieves the value of a resistor or a capacitor.

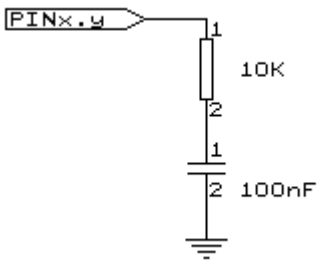
Syntax
var = **GETRC**(pin , number)

Remarks

Var	The word variable that is assigned with the value.
Pin	The PIN name for the R/C is connection.
Number	The port pin for the R/C is connection.

The name of the input port (PIND for example) must be passed even when all the other pins are configured for output. The pin number must also be passed. This may be a constant or a variable.

A circuit is shown below:



The capacitor is charged and the time it takes to discharge it is measured and stored in the variable. Now when you vary either the resistor or the capacitor, different values will be returned. This function is intended to return a relative position of a resistor wiper, not to return the value of the resistor. But with some calculations it can be retrieved.

The GETRC function passes the address of the PIN register to the _GETRC library code.
This will not work for PINF of the ATMEGA128. The PORTF, PINF, DDRF map is not continuous grouped together.
To solve this, you can use the $lib "getRc_m128_PINF.lib"
This lib is only for the M128/M64 PORTF, and when the compatibility fuse is not set to M103.

See also
NONE

Example
```
'-----------------------------------------------------------------------
-----------------
'name                    : getrc.bas
```

```
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates how to get the value of a
resistor
'micro                   : AT90S8535
'suited for demo         : yes
'commercial addon needed  : no
' The library also shows how to pass a variable for use with individual
port
' pins. This is only possible in the AVR architecture and not in the
8051
'-------------------------------------------------------------------------
-----------------

$regfile = "8535def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'The function works by charging a capacitor and uncharge it little by
little
'A word counter counts until the capacitor is uncharged.
'So the result is an indication of the position of a pot meter not the
actual
'resistor value

'This example used the 8535 and a 10K ohm variable resistor connected to
PIND.4
'The other side of the resistor is connected to a capacitor of 100nF.
'The other side of the capacitor is connected to ground.
'This is different than BASCOM-8051 GETRC! This because the architecture
is different.

'The result of getrc() is a word so DIM one
Dim W As Word
Do
  'the first parameter is the PIN register.
  'the second parameter is the pin number the resistor/capacitor is
connected to
  'it could also be a variable!
  W = Getrc(pind , 4)
  Print W
  Wait 1
Loop
```

7.57 GETRC5

Action

Retrieves the RC5 remote code from a IR transmitter.

Syntax

GETRC5(address, command)

Uses
TIMER0

Remarks

address	The RC5 address
command	The RC5 command.

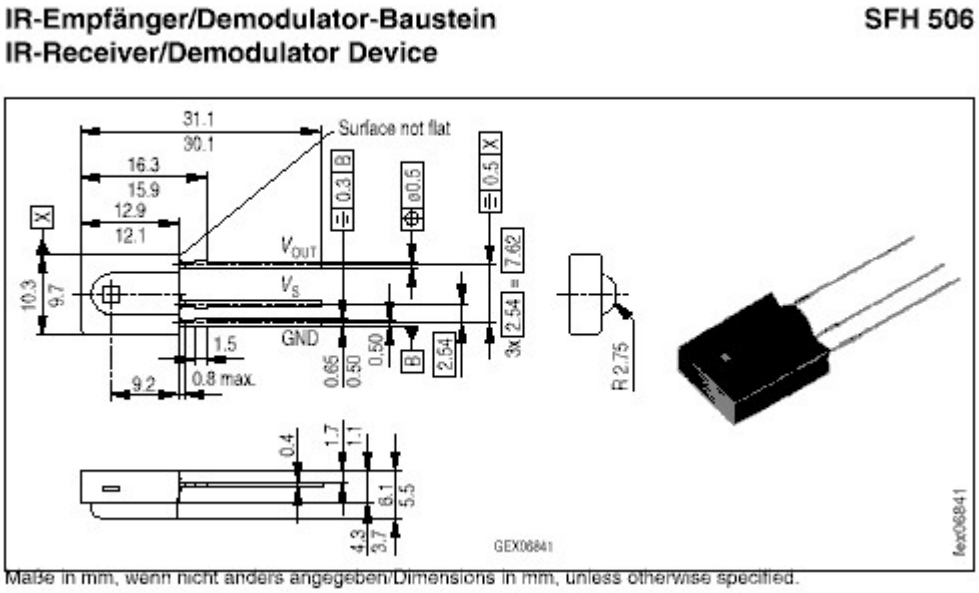
This statement is based on the AVR 410 application note. Since a timer is needed for accurate delays and background processing TIMER0 is used by this statement.

The interrupt of TIMER0 is also used by this statement.
TIMER0 can be used by your application since the values are preserved by the statement but a delay can occur. The interrupt can not be reused.

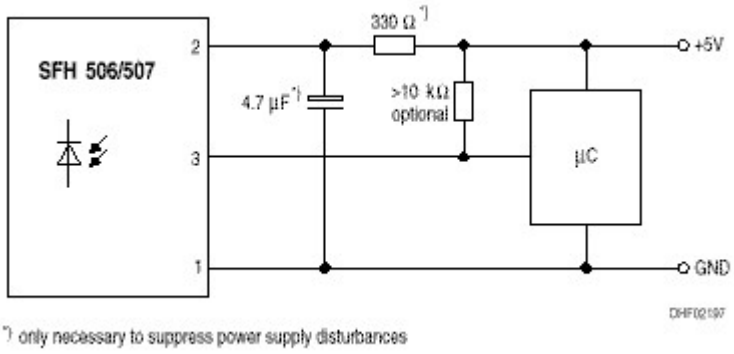
You may use any pin that can work as an input pin. Use the CONFIG RC5 statement to specify which pin is connected to the IR receiver.

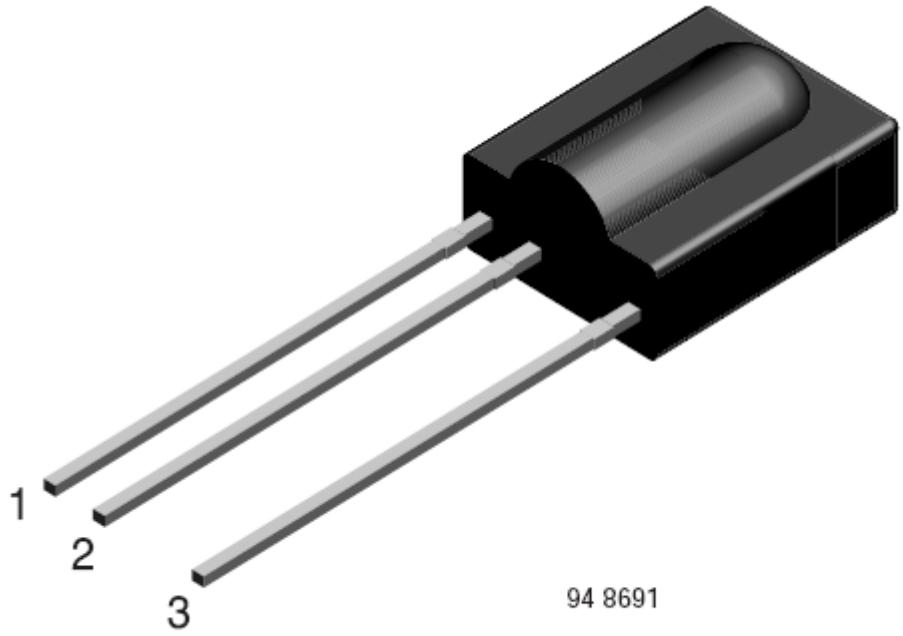
GETRC5 supports extended RC5 code reception.

The SFH506-36 is used from Siemens. Other types can be used as well. The TSOP1736 has been tested with success.
You can also use the pin compatible TSOP31236



For a good operation use the following values for the filter.





TSOP 312xx
1=GND, 2=VSS, 3=OUT

Most audio and video systems are equipped with an infra-red remote control.

The RC5 code is a 14-bit word bi-phase coded signal.
The two first bits are start bits, always having the value 1.
The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.
Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).

For extended RC5 code, the extended bit is bit 6 of the command.
The toggle bit is stored in bit 7 of the command.

Xmega
The Xmega will use timer TCC0 instead of TIMER0.

You MUST enable the low priority interrupts since TCC0 is used in this mode. You can do this with this command :
Config Priority = Static , Vector = Application , **Lo = Enabled**

Xtiny
The Xtiny will use a TCAx or TCBx timer.

Alternative Background decoding normal AVR
A special alternative library named RC5.LIB can be used to decode the RC5 signals on the background. The developing of this library was sponsored by Lumicoin.
See CONFIG RC5 1037 for more information.

Alternative Background decoding Xtiny
A special alternative library named RC5_BG_XTINY.lib can be used to decode the RC5 signals on the background. The library is automatically included when you select the BACKGROUND mode.
See CONFIG RC5 1037 for more information and an example.

See also
CONFIG RC5 1037 , RC5SEND 1405, RC6SEND 1411

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : rc5.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : based on Atmel AVR410 application note
'micro                    : 90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'use byte library for smaller code
```

```
$lib "mcsbyte.lbx"

'This example shows how to decode RC5 remote control signals
'with a SFH506-35 IR receiver.

'Connect to input to PIND.2 for this example
'The GETRC5 function uses TIMER0 and the TIMER0 interrupt.
'The TIMER0 settings are restored however so only the interrupt can not
'be used anymore for other tasks


'tell the compiler which pin we want to use for the receiver input

Config Rc5 = Pind.2

'the interrupt routine is inserted automatic but we need to make it
occur
'so enable the interrupts
Enable Interrupts

'reserve space for variables
Dim Address As Byte , Command As Byte
Print "Waiting for RC5..."

Do
  'now check if a key on the remote is pressed
  'Note that at startup all pins are set for INPUT
  'so we dont set the direction here
  'If the pins is used for other input just unremark the next line
  'Config Pind.2 = Input
  Getrc5(address , Command)

  'we check for the TV address and that is 0
  If Address = 0 Then
     'clear the toggle bit
     'the toggle bit toggles on each new received command
     'toggle bit is bit 7. Extended RC5 bit is in bit 6
     Command = Command And &B01111111
     Print Address ; "  " ; Command
  End If
Loop
End
```

Example XTINY

```
'-------------------------------------------------------------------
----------------
'name                     : avr128da28-getrc5.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates GETRC5 using time TCA0
and TCB0
'micro                    : avr128da28
'suited for demo          : no
'commercial addon needed  : yes
'-------------------------------------------------------------------
----------------

$regfile = "AVRX128da28.dat"
$crystal = 24000000
$hwstack = 40
```

```
$swstack = 40
$framesize = 40

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz

'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'we use the uart for some terminal output
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

Config Rc5 = Pind.1 , Mode = Normal , Timer = Tca0
'                              ^^^ use either a timer TCA0, TCA1
or a TCBx timer
'             ^^^^ define the pin that is connected to the output
of the IR received
'  also have a look at the background mode that runs completely
in the background!

'since the timer is used in interrupt mode you must enabl global
interrupts
Enable Interrupts

'reserve space for variables
Dim Address As Byte , Command As Byte

Print "Waiting for RC5..."

do
  'now check if a key on the remote is pressed
  'Note that at startup all pins are set for INPUT
  'so we dont set the direction here
  'If the pins is used for other input just unremark the next
line
  'Config Pind.2 = Input
   Getrc5(address , Command)
  'we check for the TV address and that is 0
   If Address = 0 Then
    'clear the toggle bit
    'the toggle bit toggles on each new received command
    'toggle bit is bit 7. Extended RC5 bit is in bit 6
      Command = Command And &B01111111
      Print Address ; "  " ; Command
   End If
   'waitms 500
loop
```

7.58 GETREG

Action
Reads a byte from an internal register.

Syntax
var = GETREG(Reg)

Remarks
Most AVR chips have 32 registers named R0-R31. The GetReg function will return the value of the specified register.

Reg	The register name : R0-R31 or a register definition.
Var	The name of a variable that will be assigned with the content of the register.

PEEK and POKE work with an address. And will return a HW register on the Xmega since Xmega has a different address map.
GetReg and SetReg will read/write registers on all AVR processors.

 In version 2078, all internal registers (R0-R31) are made available as normal BYTE variables. This means that you can simply assign or read a register from basic : Rx=value.
This is more convenient than using SETREG and GETREG.

See also
SETREG 1441, PEEK 1395 , POKE 1396

Example

7.59 GOSUB

Action
Branch to and execute subroutine.

Syntax
GOSUB label

Remarks

Label	The name of the label where to branch to.

With GOSUB, your program jumps to the specified label, and continues execution at that label.
When it encounters a RETURN statement, program execution will continue after the GOSUB statement.
A GOSUB can not pass parameters, all it does is calling a label, execute it and

returns.
So why use a GOSUB? Imagine you have a set of code you want to execute from different locations in your code.
While you can repeat the code, you can best write the code once, and call it using GOSUB.
Example :

```
if a = 1 Then
   Gosub ABC
end if
if b =1  then
  gosub ABC
End if
End

ABC:
  print "this is label ABC"
  a=a+1
RETURN
```

If A and B are both 1, the ABC label is called twice.
Do notice the END statement which will make sure that the code does not execute the ABC label without an actual GOSUB. You can test in the simulator, and see what happens in you remark the END statement.

 Instead of using GOSUB, it is better to use a SUB procedure with a CALL. A SUB module can have local variables and you can pass parameters.

See also

GOTO 1286 , CALL 806 , RETURN 1430

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : gosub.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: GOTO, GOSUB and RETURN
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Goto Continue
Print "This code will not be executed"
```

```
Continue:                                                    'end a label
with a colon
Print "We will start execution here"
Gosub Routine
Print "Back from Routine"
End


Routine:                                                     'start a
subroutine
  Print "This will be executed"
Return                                                       'return from
subroutine
```

7.60 GOTO

Action
Jump to the specified label or address.

Syntax
GOTO label

Remarks
Labels can be up to 32 characters long.
When you use duplicate labels, the compiler will give you a warning.

Valid labels
- A valid label ends with a colon (:)
- A valid label starts on the line.
- There is no space between the label name and the colon.

Label: is a valid label.
Label : is invalid

Since a colon is also used to separate multiple lines of code, the label must be the only code on the line.
For example :

print "abc" : print "klm" 'these lines are separated by a colon.
abc: : print "klm" 'this is invalid since the line starts with a label.

Besides using a label you can also specify an address. GOTO &H0000 would jump to the reset vector of the processor.
Because numeric addresses can be specified, it is advised to use non-numerical labels.

Notice that an address in the AVR is a WORD address. AVR instructions are 16 bit wide which means that for each instruction you need at least 2 bytes.
It is best to use label names.

See also

GOSUB 1284

Example

```
Dim A As Byte
Start:              ' a label must end with a colon
A = A + 1           ' increment a
If A < 10 Then      ' is it less than 10?
   Goto  Start      ' do it again
End If              ' close IF
Print   "Ready"     ' that is it
```

7.61 HIGH

Action

Retrieves the most significant byte of a variable.
Sets the most significant byte of a WORD variable.

Syntax

var = **HIGH**(s)
HIGH(s) = value

Remarks

Var	The variable that is assigned with the MSB of var S.
S	The source variable to get the MSB from when used as a function. The target variable to set the MSB when used in an assignment
value	The value to assign when used in an assignment.

When used in an assignment , only the second byte of the variable will be set. The intention purpose is to set the MSB of a WORD or INTEGER.
It will also work on a LONG or DWORD but it will set the second byte in memory which is not the MSB of a LONG/DWORD.
Also, this will work on arrays, but there is no type checking which means that you should not use this on a single BYTE since it could overwrite other memory.

In version 2083 the HIGH function can also be used to set the MSB of a variable. This for compatibility with BASCOM-8051.

See also

LOW 1367 , HIGHW 1288

Example

```
Dim I As Integer , Z As Byte
I = &H1001
Z = High(i)                                          ' is 10 hex
or 16 dec
End
```

7.62 HIGHW

Action
Retrieves the most significant word of a long variable.

Syntax
var = **HIGHW**(s)

Remarks

Var	The variable that is assigned with the MS word of var S.
S	The source variable to get the MSB from.

There is no LowW() function. This because when you assign a Long to a word or integer, only the lower part is assigned. For this reason you do not need a Loww() function. W=L will do the same.

See also
LOW 1367 , HIGH 1287

Example
```
Dim X As Word , L As Long
L = &H12345678
X = Highw(l)
Print Hex(x)
```

7.63 Encryption-Decryption

7.63.1 AESENCRYPT

Action
This statement of function uses the Xmega AES encryption engine to encrypt a block of data.

Syntax
AESENCRYPT key, var , size
targ = **AESENCRYPT (** key, var , size**)**

Remarks

key	The name of a label that contains 16 bytes of key data. Or an array holding 16 bytes of key data.
var	A variable or array containing the data to be encrypted. When you use the statement, this variable will contain the encrypted data after the conversion.
size	The number of bytes to encrypt. Encryption is done with blocks of 16 bytes. So the size should be a multiple of 16. If you supply only 14 bytes this is ok too, but the result will still be 16 bytes. It is important that your array is big enough to hold the result. Without the full 16 byte result, you can not decrypt the data.

targ	In case you use the function, this variable will hold the result.

This function only works for Xmega chips that have an AES encryption unit.
128 bit encryption is used.

You can either use a label with a fixed key, or use a variable.
You should use the same key data for encryption and decryption.

See also

$LOADER 667, $AESKEY 606, AESDECRYPT 1290, DESENCRYPT 1292, DESDECRYPT 1294, $XTEAKEY 715, XTEAENCODE 1297, XTEADECODE 1298

Example

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-AES.bas
'  This sample demonstrates the Xmega128A1 AES encryption/decryption
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 38400 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

'$external _aes_enc

Dim Key(16) As Byte                                         ' room for
key
Dim Ar(34) As Byte
Dim Arenc(34) As Byte
Dim J As Byte
Print "AES test"

Restore Keydata
For J = 1 To 16                                             ' load a key
to memory
   Read Key(j)
Next

'load some data
For J = 1 To 32                                             ' fill some
data to encrypt
  Ar(j) = J
Next


Aesencrypt Keydata , Ar(1) , 32
```

```
Print "Encrypted data"
For J = 1 To 32                                             ' fill some
data to encrypt
  Print Ar(j)
Next


Aesdecrypt Keydata , Ar(1) , 32
Print "Decrypted data"
For J = 1 To 32                                             ' fill some
data to encrypt
  Print Ar(j)
Next

Print "Encrypt function"
Arenc(1) = Aesencrypt(keydata , Ar(1) , 32)
For J = 1 To 32                                             ' fill some
data to encrypt
  Print Ar(j) ; "-" ; Arenc(j)
Next

Print "Decrypt function"
Ar(1) = Aesdecrypt(keydata , Arenc(1) , 32)

For J = 1 To 32
 Print J ; ">" ; Ar(j) ; "-" ; Arenc(j)
Next

End



Keydata:
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 ,
16
```

7.63.2 AESDECRYPT

Action

This statement of function uses the Xmega AES encryption engine to decrypt a block of data.

Syntax

AESDECRYPT key, var , size
targ = **AESDECRYPT (** key, var , size**)**

Remarks

key	The name of a label that contains 16 bytes of key data. Or an array holding 16 bytes of key data.
var	A variable or array containing the data to be encrypted. When you use the statement, this variable will contain the encrypted data after the conversion.
size	The number of bytes to encrypt. Encryption is done with blocks of 16 bytes. So the size should be a multiple of 16. If you supply only 14 bytes this is ok too, but the result will still be 16 bytes.  It is important that your array is big enough to hold the result.

	Without the full 16 byte result, you can not decrypt the data.
targ	In case you use the function, this variable will hold the result.

This function only works for Xmega chips that have an AES encryption unit.
128 bit encryption is used.

You can either use a label with a fixed key, or use a variable.
You should use the same key data for encryption and decryption.

See also

$LOADER 667, $AESKEY 606, AESENCRYPT 1288, DESENCRYPT 1292, DESDECRYPT 1294, $XTEAKEY 715, XTEAENCODE 1297, XTEADECODE 1298

Example

```
'-----------------------------------------------------------------
'                   (c) 1995-2025 MCS
'                       xm128-AES.bas
'  This sample demonstrates the Xmega128A1 AES encryption/decryption
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40


'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 38400 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

'$external _aes_enc

Dim Key(16) As Byte                                         ' room for
key
Dim Ar(34) As Byte
Dim Arenc(34) As Byte
Dim J As Byte
Print "AES test"

Restore Keydata
For J = 1 To 16                                             ' load a key
to memory
   Read Key(j)
Next

'load some data
For J = 1 To 32                                             ' fill some
data to encrypt
  Ar(j) = J
Next
```

```
Aesencrypt Keydata , Ar(1) , 32
Print "Encrypted data"
For J = 1 To 32                                                    ' fill some
data to encrypt
  Print Ar(j)
Next


Aesdecrypt Keydata , Ar(1) , 32
Print "Decrypted data"
For J = 1 To 32                                                    ' fill some
data to encrypt
  Print Ar(j)
Next

Print "Encrypt function"
Arenc(1) = Aesencrypt(keydata , Ar(1) , 32)
For J = 1 To 32                                                    ' fill some
data to encrypt
  Print Ar(j) ; "-" ; Arenc(j)
Next

Print "Decrypt function"
Ar(1) = Aesdecrypt(keydata , Arenc(1) , 32)

For J = 1 To 32
 Print J ; ">" ; Ar(j) ; "-" ; Arenc(j)
Next

End



Keydata:
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 ,
16
```

7.63.3 DESENCRYPT

Action

This statement of function uses the Xmega DES encryption engine to encrypt a block of data.

Syntax

targ = **DESENCRYPT (** key, var , size**)**

Remarks

key	The name of a label that contains 16 bytes of key data. Or an array holding 16 bytes of key data.
var	A variable or array containing the data to be encrypted.
size	The number of bytes to encrypt. Encryption is done with blocks of 16 bytes. So the size should be a multiple of 16. If you supply only 14 bytes this is ok too, but the result will still be 16 bytes. It is important that your array is big enough to hold the result. Without the full 16 byte result, you can not decrypt the data.
targ	An array of variable that will hold the result. Since the result will be at least 16 bytes long, this is only practical with arrays.

This function only works for Xmega chips that have an DES encryption unit.
Normal DES encryption is used. The DES encryption is faster than the AES but also weaker.

You can either use a label with a fixed key, or use a variable.
You should use the same key data for encryption and decryption.

See also

$LOADER 667, $AESKEY 606, AESENCRYPT 1288, AESDECRYPT 1290, DESDECRYPT 1294, $XTEAKEY 715, XTEAENCODE 1297, XTEADECODE 1298

Example

```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                       xm128-DES.bas
'  This sample demonstrates the Xmega128A3 DES
encryption/decryption
' Notice that you need to encrypt blocks with at least 8 bytes
'-----------------------------------------------------------------
-
$regfile = "xm128a3def.dat"
$crystal = 32000000
'32MHz
$hwstack = 128
$swstack = 128
$framesize = 128

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

'configure used UART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8
Open "com1:" For Binary As #1

Dim Key(8) As Byte                                          '
room for key
Dim Ar(34) As Byte
Dim Arenc(34) As Byte
Dim J As Byte

Print #1 , "DES test"

Restore Keydata
```

```
For J = 1 To 8                                                '
load a key to memory
   Read Key(j)
Next


'load some data
For J = 1 To 16                                               '
fill some data to encrypt
   Ar(j) = J
Next


Print #1 , "Encrypt function"
Arenc(1) = Desencrypt(key(1) , Ar(1) , 16)
'encrypt 16 bytes


For J = 1 To 16
   Print #1 , Ar(j) ; "-" ; Arenc(j)
'print result and original data
Next


Print #1 , "Decrypt function"
Ar(1) = Desdecrypt(keydata , Arenc(1) , 16)
'decrypt and return in ar()

For J = 1 To 16
   Print #1 , J ; ">" ; Ar(j) ; "-" ; Arenc(j)
'print index, decrypted data and encrypted data
Next

Do

Loop

End



Keydata:                                                      ' key
data can go into flash ROM or into sram
   Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
```

7.63.4 DESDECRYPT

Action

This statement of function uses the Xmega DES encryption engine to decrypt a block of data.

Syntax
targ = **DESDECRYPT (** key, var , size**)**

Remarks

key	The name of a label that contains 16 bytes of key data. Or an array holding 16 bytes of key data.
var	A variable or array containing the data to be decrypted.
size	The number of bytes to decrypt. Encryption is done with blocks of 16 bytes. So the size should be a multiple of 16. If you supply only 14 bytes this is ok too, but the result will still be 16 bytes. It is important that your array is big enough to hold the result. Without the full 16 byte result, you can not decrypt the data.
targ	An array of variable that will hold the result. Since the result will be at least 16 bytes long, this is only practical with arrays.

This function only works for Xmega chips that have an DES encryption unit.
Normal DES encryption is used. The DES encryption is faster than the AES but also weaker.

You can either use a label with a fixed key, or use a variable.
You should use the same key data for encryption and decryption.

See also
$LOADER 667, $AESKEY 606, AESENCRYPT 1288, AESDECRYPT 1290, DESENCRYPT 1292, $XTEAKEY 715, XTEAENCODE 1297, XTEADECODE 1298

Example
```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                       xm128-DES.bas
'  This sample demonstrates the Xmega128A3 DES
encryption/decryption
' Notice that you need to encrypt blocks with at least 8 bytes
'-----------------------------------------------------------------
-
$regfile = "xm128a3def.dat"
$crystal = 32000000
'32MHz
$hwstack = 128
$swstack = 128
$framesize = 128

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
```

```
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

'configure used UART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8
Open "com1:" For Binary As #1

Dim Key(8) As Byte                                          '
room for key
Dim Ar(34) As Byte
Dim Arenc(34) As Byte
Dim J As Byte

Print #1 , "DES test"

Restore Keydata
For J = 1 To 8                                              '
load a key to memory
   Read Key(j)
Next


'load some data
For J = 1 To 16                                             '
fill some data to encrypt
   Ar(j) = J
Next


Print #1 , "Encrypt function"
Arenc(1) = Desencrypt(key(1) , Ar(1) , 16)
'encrypt 16 bytes


For J = 1 To 16
   Print #1 , Ar(j) ; "-" ; Arenc(j)
'print result and original data
Next


Print #1 , "Decrypt function"
Ar(1) = Desdecrypt(keydata , Arenc(1) , 16)
'decrypt and return in ar()

For J = 1 To 16
   Print #1 , J ; ">" ; Ar(j) ; "-" ; Arenc(j)
'print index, decrypted data and encrypted data
Next
```

```
Do

Loop

End



Keydata:                                                    ' key
data can go into flash ROM or into sram
   Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
```

7.63.5 XTEAENCODE

Action

Encrypts a variable or array using the XTEA protocol.

Syntax

XTEAENCODE Msg , Key , size

Remarks

Msg	The variable to encrypt. Encryption is performed in blocks of 8 bytes. This means that you need to specify an array that has a minimal size of 8 bytes. For example, 2 Longs will be 8 bytes in size. After the encryption is performed, Msg will contain the encrypted data. The original data will be overwritten.
Key	The 128 bit key which is used to encrypt the message data. You need to pass this as an array of 16 bytes.
Size	The number of bytes to encrypt. This must be a multiple of 8.

The XTEA encryption/decryption is described well at http://en.wikipedia.org/wiki/XTEA
The XTEA is an enhanced version of the TEA encryption protocol.

The XTEA encoding/decoding routines have a small footprint. You could use the XTEADECODE in a bootloader and encrypt your firmware.

When you use other tools to encode your data, you will find differences because of memory order. You can use the xtea2.lib for using the same memory order.
Include it in your code like : $LIB "xtea2.lib"

See also

$LOADER 667, $AESKEY 606, AESENCRYPT 1288, AESDECRYPT 1290, DESENCRYPT 1292, DESDECRYPT 1294, $XTEAKEY 715, XTEADECODE 1298

Example

```
'-----------------------------------------------------------
'                       XTEA.BAS
```

```
' This sample demonstrates the XTEA encryption/decryption
' statements
'             (c) 1995-2025 MCS Electronics
'-------------------------------------------------------------
$regfile = "m88def.dat"
$hwstack = 40
$swstack = 32


'The XTEA encryption/decryption has a small footprint
'XTEA processes data in blocks of 8 bytes. So the minimum length of the
data is 8 bytes.
'A 128 bit key is used to encrypt/decrypt the data. You need to supply
this in an array of 8 bytes.

'Using the encoding on a string can cause problems when the data
contains a 0. This is the end of the string marker.

Dim Key(16) As Byte                                 ' 128 bit key
Dim Msg(32) As Byte                                 ' this need to be a
multiple of 8

Dim B As Byte                                       ' counter byte

For B = 1 To 16                                     ' create a simple key
and also fill the data
  Key(b) = B
  Msg(b) = B
Next


Xteaencode Msg(1) , Key(1) , 32                     ' encode the data

For B = 1 To 16
  Print Hex(msg(b)) ; " , " ;
Next
Print


Xteadecode Msg(1) , Key(1) , 32                     ' decode the data

For B = 1 To 16
  Print Hex(msg(b)) ; " , " ;
Next                                                ' it should print 1-16
now
Print

End
```

7.63.6 XTEADECODE

Action

Decrypts a variable or array using the XTEA protocol. This is a software implementation.

Syntax

XTEADECODE Msg , Key , size

Remarks

Msg	The variable to decrypt. Decryption is performed in blocks of 8 bytes. This means that you need to specify an array that has a minimal size of 8 bytes. For example, 2 Longs will be 8 bytes in size. After the decryption is performed, Msg will contain the original unencrypted data.
Key	The 128 bit key which is used to decrypt the message data. You need to pass this as an array of 16 bytes.
Size	The number of bytes to decrypt. This must be a multiple of 8.

The XTEA encryption/decryption is described well at http://en.wikipedia.org/wiki/XTEA
The XTEA is an enhanced version of the TEA encryption protocol.

The XTEA encoding/decoding routines have a small footprint. You could use the XTEADECODE in a bootloader and encrypt your firmware.

When you use other tools to encode your data, you will find differences because of memory order. You can use the xtea2.lib for using the same memory order.
Include it in your code like : $LIB "xtea2.lib"

See also

$LOADER 667, $AESKEY 606, AESENCRYPT 1288, AESDECRYPT 1290, DESENCRYPT 1292, DESDECRYPT 1294, $XTEAKEY 715, XTEAENCODE 1297

Example

```
'-----------------------------------------------------------
'                      XTEA.BAS
' This sample demonstrates the XTEA encryption/decryption
' statements
'             (c) 1995-2025 MCS Electronics
'-----------------------------------------------------------
$regfile = "m88def.dat"
$hwstack = 40
$swstack = 32


'The XTEA encryption/decryption has a small footprint
'XTEA processes data in blocks of 8 bytes. So the minimum length of the
data is 8 bytes.
'A 128 bit key is used to encrypt/decrypt the data. You need to supply
this in an array of 8 bytes.

'Using the encoding on a string can cause problems when the data
contains a 0. This is the end of the string marker.

Dim Key(16) As Byte                                 ' 128 bit key
Dim Msg(32) As Byte                                 ' this need to be a
multiple of 8

Dim B As Byte                                       ' counter byte

For B = 1 To 16                                     ' create a simple key
and also fill the data
  Key(b) = B
  Msg(b) = B
Next
```

```
Xteaencode Msg(1) , Key(1) , 32                 ' encode the data

For B = 1 To 16
  Print Hex(msg(b)) ; " , " ;
Next
Print


Xteadecode Msg(1) , Key(1) , 32                 ' decode the data

For B = 1 To 16
  Print Hex(msg(b)) ; " , " ;
Next                                            ' it should print 1-16
now
Print

End
```

7.64 I2C-TWI

7.64.1 I2CINIT

Action

Initializes the SCL and SDA pins.

Syntax

I2CINIT
I2CINIT twi
I2CINIT #const

Remarks

By default the SCL and SDA pins are in the right state when you reset the chip. Both the PORT and the DDR bits are set to 0 in that case.
When you need to change the DDR and/or PORT bits you can use I2CINIT to bring the pins in the proper state again.

For the XMEGA which has multiple TWI interfaces you can use a channel to specify the TWI interface otherwise the default TWIC will be used.

For normal AVR processors with multiple TWI interfaces you can specify the interface : TWI or TWI1.
When no parameter is provided, the first default TWI will be selected.

ASM

The I2C routines are located in i2c.lib. _i2c_init is called.

See also

I2CSEND 1305 , I2CSTART 1306 , I2CSTOP 1306 , I2CRBYTE 1306 , I2CWBYTE 1306 , I2C_TWI Library for using TWI 1753

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
I2cinit

Dim X As Byte , Slave As Byte
X = 0                                                    'reset
variable
Slave = &H40                                             'slave
address of a PCF 8574 I/O IC
I2creceive Slave , X                                     'get the
value
Print X                                                  'print it
```

Example XMEGA

```
Open "twic" For Binary As #4                             ' or use
TWID,TWIE oR TWIF
Config TwiC = 100000                                      'CONFIG TWI
will ENABLE the TWI master interface
I2cinit #4
```

Example Mega328PB

```
'-----------------------------------------------------------------------
----------
'name                     : m328pb.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates M328pb
'micro                    : Mega328pb
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
----------
$regfile = "m328pbdef.dat"
$crystal = 8000000
$baud = 19200
$hwstack = 40
$swstack = 40
$framesize = 40

' USART   TX    RX
'   0     D.1   D.0
'   1     B.3   B.4

' ISP programming
' MOSI-PB3     MISO-PB4   SCK-PB5

' TWI    SDA   SCL
'  0     C.4   C.5
'  1     E.0   E.1

'Configuration
```

```
Config Clockdiv = 1                                         'make sure
we get 8 Mhz from internal osc

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'we have 2 TWI interfaces
Config Scl = Portc.5                                        ' we need
to provide the SCL pin name
Config Sda = Portc.4                                        ' we need
to provide the SDA pin name

Config Sda1 = Porte.0                                       'use this
for the second TWI
Config Scl1 = Porte.1

Config Twi = 100000                                         'speed 100
KHz
Config Twi1 = 100000                                        'speed 100
KHz

'some constants for the signature row
Const Device_signature_byte1 = 0
Const Device_signature_byte2 = 2
Const Device_signature_byte3 = 4

Const Rc_oscillator_calibration = 1

Const Serial_number_byte0 = &H0E
Const Serial_number_byte1 = &H0F
Const Serial_number_byte2 = &H10
Const Serial_number_byte3 = &H11
Const Serial_number_byte4 = &H12
Const Serial_number_byte5 = &H13
Const Serial_number_byte6 = &H14
Const Serial_number_byte7 = &H15
Const Serial_number_byte8 = &H16
Const Serial_number_byte9 = &H17

$lib "I2C_TWI-MULTI.lib"                                    'important
for using 2 TWI interfaces

Dim _i2cchannel As Byte                                     ' you MUST
dim this variable yourself when using the above lib
Dim B As Byte                                               'just a
used byte

I2cinit                                                     'default
TWI init
I2cinit Twi1                                                'optional
specify TWI1 to init that interface
```

```
Open "com2:" For Binary As #2                               'create a
channel to reference the UART

'print the chip ID
Print "ID : " ; Hex(readsig(device_signature_byte1)) ; Hex(readsig(
device_signature_byte2)) ; Hex(readsig(device_signature_byte3))

'all I2C statements will work the same. All you need to do is to set
the _i2cchannel variable to 0 or 1
_i2cchannel = 1                                             'try the
second bus

Print "Scan start"
For B = 0 To 254 Step 2                                     'for all
odd addresses
  I2cstart
  I2cwbyte B                                                'send
address
  If Err = 0 Then                                           'we got an
ack
     Print "Slave at : " ; B ; " hex : " ; Hex(b) ; " bin : " ; Bin(b)
  End If
  I2cstop                                                   'free bus
Next


Do
   Print "COM1"
   Print #2 , "COM2"
   Waitms 1000
Loop
```

7.64.2 I2CRECEIVE

Action

Receives data from an I2C serial slave device.

Syntax

I2CRECEIVE slave, var
I2CRECEIVE slave, var , b2W, b2R

Syntax Xmega

I2CRECEIVE slave, var , **#**const
I2CRECEIVE slave, var , b2W, b2R , **#**const

Remarks

Slave	A byte, Word/Integer variable or constant with the slave address from the I2C-device. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a

	read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
Var	A byte or integer/word or numeric variable or array that will receive the information from the I2C-device. This same variable is used for sending the optional data as for receiving the data. This means that when you need to send/receive data, you first fill the variable with the data you will send. And when the statement ends, the variable will contain the data received. You should dimension the variable or array large enough to hold the data sent/received.
b2W	The number of bytes to write. When you use a value of 0, no data will be sent.
b2R	The number of bytes to receive. When you use a value of 0. no data will be received. But since this statement is to receive data, that would not make much sense.
#const	For the Xmega, a channel constant that was specified with OPEN.

You must specify the base address of the slave chip because the read/write bit is set/reset by the software.
When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

The I2CRECEIVE statement combines the i2cstart,i2cwbyte, i2crbyte and i2cstop statements.
For the xmega you can optional specify the channel. Without it, SPIC will be used.

ASM
The I2C routines are located in the i2c.lib/i2c.lbx files.

See also
I2CSEND 1305, I2CSTART 1306 , I2CSTOP 1306, I2CRBYTE 1306 , I2CWBYTE 1306

Example
```
Config Sda = Portb.5
Config Scl = Portb.7
Dim X As Byte , Slave As Byte
X = 0                                                    'reset
variable
Slave = &H40                                             'slave
address of a PCF 8574 I/O IC
I2creceive Slave , X                                     'get the
value
Print X                                                  'print it


Dim Buf(10)as Byte
Buf(1) = 1 : Buf(2) = 2
```

```
I2creceive Slave , Buf(1) , 2 , 1                          'send two
bytes buf(1) and buf(2) and receive one byte into buf(1)
Print Buf(1)                                               'print the
received byte
End
```

7.64.3 I2CSEND

Action
Send address and data to an I2C-device.

Syntax
I2CSEND slave, var
I2CSEND slave, var , bytes

Syntax Xmega
I2CSEND slave, var , **#**const
I2CSEND slave, var , bytes , **#**const

Remarks

Slave	The slave address off the I2C-device. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
Var	A byte, integer/word or numbers that holds the value, which will be, send to the I2C-device.
Bytes	The number of bytes to send.
#const	For the Xmega, a channel constant that was specified with OPEN.

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.
The I2CSEND statement combines the i2cstart,i2cwbyte and i2cstop statements.
For the xmega you can optional specify the channel. Without it, SPIC will be used.

The I2CSEND is a compound statement that will send :
- I2CSTART
- I2C slave address for writing
- I2C data
- I2CSTOP

ASM
The I2C routines are located in the i2c.lib/i2c.lbx files.

See also
I2CRECEIVE 1303 , I2CSTART 1306, I2CSTOP 1306 , I2CRBYTE 1306 , I2CWBYTE 1306

Example
```
Config Sda = Portb.5
Config Scl = Portb.7
Dim X As Byte , A As Byte , Bytes As Byte
X = 5                                                    'assign
variable to 5
Dim Ax(10)as Byte
Const Slave = &H40                                       'slave
address of a PCF 8574 I/O IC
I2csend Slave , X                                        'send the
value or


For A = 1 To 10
  Ax(a) = A                                              'Fill
dataspace
Next
Bytes = 10
I2csend Slave , Ax(1) , Bytes
End
```

7.64.4 I2START,I2CSTOP, I2CRBYTE, I2CWBYTE, I2CREPSTART

Action
I2CSTART generates an I2C start condition.
I2CREPSTART generates an I2C repeated start condition.
I2CSTOP generates an I2C stop condition.
I2CRBYTE receives one byte from an I2C-device.
I2CWBYTE sends one byte to an I2C-device.

Syntax
I2CSTART
I2CREPSTART
I2CSTOP
I2CRBYTE var, ack/nack
I2CWBYTE val

Syntax Xmega
I2CSTART #const
I2CREPSTART #const
I2CSTOP #const
I2CRBYTE var, ack/nack , **#**const
I2CWBYTE val , **#**const

Remarks

Var	A variable that receives the value from the I2C-device.
ack/nack	Specify ACK if there are more bytes to read.

	Specify NACK if it is the last byte to read.
Val	A variable or constant to write to the I2C-device.
#const	For the Xmega, a channel constant that was specified with OPEN.

These statements are provided as an addition to the I2CSEND and I2CRECEIVE statements.
While **I2csend** and **I2CRECEIVE** are well suited for most tasks, a slave chip might need a special sequence that is not possible with these I2C routines.
Using I2CSTART, I2CWBYTE, I2CRBYTE and I2CSTOP you can create any I2C sequence you need.

When an error occurs, the internal **Err** variable will return 1. Otherwise it will be set to 0.

The Xmega has multiple TWI interfaces. You can use a channel to specify the TWI interface.
When you do not use a channel the TWIC interface will be used.

 When using a repeated start, you must use **I2CREPSTART** on the XMega ! This is also true when using the i2cv2 1763.lib.

Normal AVR processors may use **either I2CSTART or I2CREPSTART**.

 For Xmega, the I2CSTART does not actually create a bus START signal. This because for Xmega the start is combined with the first data write (the address).
This means that ERR will always return 0 for Xmega I2CSTART. For this reason the bus scanner example checks ERR after the address write.

All I2C statements are master mode statements. They are stored in the i2c.lib. There is also an improved version of this library available named i2cv2.lib
Since the repeated start is not compatible with the one from i2c.lib, you need to specify yourself that you want to use the improved lib. See also I2CV2 1763.

I2CSTOP and Xmega

The I2CSTOP statement on the Xmega can be influenced by defining a constant in your code.
There are two optional constant you can define.

```
Const _TWI_STOP_1 = 1
or
Const _TWI_STOP_2 = 1
```

Notice that the value does not matter ! The library only checks if the constant exists.
Also notice that there are 2 different constants.
When not defining any of the above constants, the default will be used as it was in version 2079.
This default will send a stop, then checks if the bus is not in the owner state, and send new stop commands till it becomes in non-owner state.

Some slave chips choke on multiple stop commands. In such a case you can define the constant named **_TWI_STOP_2**
This will send a stop, and then keep checking till the bus is in non-owner state.

The last mode you get when defining a constant named **_TWI_STOP_1**
This will only send a stop without checking if the bus is non-owner. This can have advantages. But generating a new I2CSTART could fail since the bus is not in the right mode yet. You should check the ERR variable in such a case after the I2CSTART command.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files. There is also an alternative i2c_twi.lib for when using TWI, and an alternative soft lib named i2cv2.lib
For the XMega, the routines are located in the xmega.lib file.

See also

I2CSEND 1305 , I2CRECEIVE 1303 , I2CSTART 1306 , I2CSTOP 1306, I2CRBYTE 1306 , I2CWBYTE 1306, Using the I2C protocol 297, CONFIG TWI 1116 , I2CV2 1763

Example (using Software I2C Routines)

```
'-----------------------------------------------------------------------
------------------
'name                     : i2c.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: I2CSEND and I2CRECEIVE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 20                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Config Scl = Portb.4
Config Sda = Portb.5
I2cinit


Declare Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
Declare Sub Read_eeprom(byval Adres As Byte , Value As Byte)

Const Addressw = 174                                        'slave write
address
Const Addressr = 175                                        'slave read
address

Dim B1 As Byte , Adres As Byte , Value As Byte              'dim byte
```

```
Call Write_eeprom(1 , 3)                                    'write value
of three to address 1 of EEPROM


Call Read_eeprom(1 , Value) : Print Value                   'read it
back
Call Read_eeprom(5 , Value) : Print Value                   'again for
address 5


'-------- now write to a PCF8474 I/O expander -------
I2csend &H40 , 255                                          'all outputs
high
I2creceive &H40 , B1                                        'retrieve
input
Print "Received data " ; B1                                 'print it
End

Rem Note That The Slaveaddress Is Adjusted Automaticly With I2csend &
I2creceive
Rem This Means You Can Specify The Baseaddress Of The Chip.



'sample of writing a byte to EEPROM AT2404
Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
    I2cstart                                                'start
condition
    I2cwbyte Addressw                                       'slave
address
    I2cwbyte Adres                                          'asdress of
EEPROM
    I2cwbyte Value                                          'value to
write
    I2cstop                                                 'stop
condition
    Waitms 10                                               'wait for 10
milliseconds
End Sub


'sample of reading a byte from EEPROM AT2404
Sub Read_eeprom(byval Adres As Byte , Value As Byte)
   I2cstart                                                 'generate
start
   I2cwbyte Addressw                                        'slave
adsress
   I2cwbyte Adres                                           'address of
EEPROM
   I2cstart                                                 'repeated
start
   I2cwbyte Addressr                                        'slave
address (read)
   I2crbyte Value , Nack                                    'read byte
   I2cstop                                                  'generate
stop
End Sub


' when you want to control a chip with a larger memory like the 24c64 it
requires an additional byte
' to be sent (consult the datasheet):
' Wires from the I2C address that are not connected will default to 0 in
```

```
most cases!

'   I2cstart                                                    'start
condition
'   I2cwbyte &B1010_0000                                        'slave
address
'   I2cwbyte H                                                  'high
address
'   I2cwbyte L                                                  'low address
'   I2cwbyte Value                                              'value to
write
'   I2cstop                                                     'stop
condition
'   Waitms 10
```

Xmega Example

```
'-----------------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                       xm128-TWI.bas
'  This sample demonstrates the Xmega128A1 TWI
'-----------------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40

Dim S As String * 20

Config Osc = Enabled , 32mhzosc = Enabled
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Dim N As String * 16 , B As Byte
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Config Input1 = Cr , Echo = Crlf                                ' CR is used
for input, we echo back CR and LF

Open "COM1:" For Binary As #1
'       ^^^^ change from COM1-COM8

Print #1 , "Xmega revision:" ; Mcu_revid                        ' make sure
it is 7 or higher !!! lower revs have many flaws

Const Usechannel = 1


Dim B1 As Byte , B2 As Byte
Dim W As Word At B1 Overlay

Open "twic" For Binary As #4                                    ' Use TWI on
Port C
'you can also use TWIC, TWID, TWIE of TWIF
Config Twi = 100000                                             ' 100KHz


#if Usechannel = 1
   I2cinit #4
#else
  I2cinit
#endif
```

```
Do
  I2cstart
  Waitms 20
  I2cwbyte &H70                                          ' slave
address write
  Waitms 20
  I2cwbyte &B10101010                                    ' write
command
  Waitms 20
  I2cwbyte 2
  Waitms 20
  I2cstop
  Print "Error : " ; Err                                 ' show error
status

 'waitms 50
  Print "start"
  I2cstart
  Print "Error : " ; Err                                 ' show error
  I2cwbyte &H71
  Print "Error : " ; Err                                 ' show error
  I2crbyte B1 , Ack
  Print "Error : " ; Err                                 ' show error
  I2crbyte B2 , Nack
  Print "Error : " ; Err                                 ' show error
  I2cstop
  Print "received A/D : " ; W ; "-" ; B1 ; "-" ; B2
  Waitms 500                                             'wait a bit
Loop



Dim J As Byte , C As Byte , K As Byte
Dim Twi_start As Byte                                    ' you MUST
dim this variable since it is used by the lib

'determine if we have an i2c slave on the bus
For J = 0 To 200 Step 2
  Print J
  #if Usechannel = 1
    I2cstart #4
  #else
    I2cstart
  #endif

   I2cwbyte J
   If Err = 0 Then                                       ' no errors
      Print "FOUND : " ; Hex(j)
      'write some value to the pcf8574A
      #if Usechannel = 1
         I2cwbyte &B1100_0101 , #4
      #else
         I2cwbyte &B1100_0101
      #endif
      Print Err
      Exit For
   End If
   #if Usechannel = 1
       I2cstop #4
   #else
       I2cstop
   #endif
Next
```

```
#if Usechannel = 1
    I2cstop #4
#else
    I2cstop
#endif

#if Usechannel = 1
  I2cstart #4
  I2cwbyte &H71 , #4                                    ' read
address
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack , #4
  Print Bin(j) ; " err:" ; Err
  I2cstop #4
#else
  I2cstart
  I2cwbyte &H71                                         ' read
address
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack
  Print Bin(j) ; " err:" ; Err
  I2cstop
#endif

'try a transaction
#if Usechannel = 1
  I2csend &H70 , 255 , #4                               ' all 1
  Waitms 1000
  I2csend &H70 , 0 , #4                                 ' all 0
#else
  I2csend &H70 , 255
  Waitms 1000
  I2csend &H70 , 0
#endif
Print Err


'read transaction
Dim Var As Byte
Var = &B11111111
#if Usechannel = 1
  I2creceive &H70 , Var , 1 , 1 , #4                    ' send and
receive
  Print Bin(var) ; "-" ; Err
  I2creceive &H70 , Var , 0 , 1 , #4                    ' just
receive
  Print Bin(var) ; "-" ; Err
#else
  I2creceive &H70 , Var , 1 , 1                         ' send and
receive
  Print Bin(var) ; "-" ; Err
  I2creceive &H70 , Var , 0 , 1                         ' just
receive
  Print Bin(var) ; "-" ; Err
#endif

End
```

7.65 IDLE

Action
Put the processor into the idle mode.

Syntax
IDLE

Remarks
In the idle mode, the system clock is removed from the CPU but not from the interrupt logic, the serial port or the timers/counters.

The idle mode is terminated either when an interrupt is received(from the watchdog, timers, external level triggered or ADC) or upon system reset through the RESET pin.

Most new chips have many options for Power down/Idle. It is advised to consult the data sheet to see if a better mode is available.

 You should use the new CONFIG POWERMODE 1017 statement.

See also
POWERDOWN 1398 , POWERSAVE 1399 , POWER mode 1397

Example
```
IDLE
```

7.66 IF-THEN-ELSE-END IF

Action
Allows conditional execution or branching, based on the evaluation of a Boolean expression.

Syntax
IF expression **THEN**

[**ELSEIF** expression **THEN**]

[**ELSE**]

END IF

Remarks

Expression	Any expression that evaluates to true or false.

The one line version of IF can be used :
IF expression THEN statement [ELSE statement]
The use of [ELSE] is optional.

Tests like IF THEN can also be used with bits and bit indexes.

```
IF var.bit = 1 THEN
        ^--- bit is a variable or numeric constant in the range from 0-255
```

You can use OR or AND to test on multiple conditions. The conditions are evaluated from left to right.

```
IF A=1 OR A=2 OR A=3 OR B>10 THEN
IF A=1 AND A>3 THEN
```

```
Dim Var As Byte, Idx As Byte
Var = 255
Idx = 1
If Var.idx = 1 Then
   Print "Bit 1 is 1"
EndIf
```

See also

ELSE 1248

Example

```
Dim A As Integer
A = 10
If A = 10 Then                                          'test
expression
  Print "This part is executed."                       'this will
be printed
Else
  Print "This will never be executed."                 'this not
End If
If A = 10 Then Print "New in BASCOM"
If A = 10 Then Goto Label1 Else print "A<>10"
Label1:

Rem The following example shows enhanced use of IF THEN
If A.15 = 1 Then                                        'test for
bit
  Print "BIT 15 IS SET"
EndIf
Rem the following example shows the 1 line use of IF THEN [ELSE]
If A.15 = 0 Then Print "BIT 15 is cleared" Else Print "BIT 15 is set"
```

7.67 INCR

Action

Increments a variable by one.

Syntax

INCR var

Remarks

Var	Any numeric variable.

See also
DECR 1214

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : incr.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: INCR
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte

A = 5                                                      'assign
value to a
Incr A                                                     'inc (by
one)
Print A                                                    'print it
End
```

7.68 INP

Action
Returns a byte read from a hardware port or any internal or external memory location.

Syntax
var = **INP**(address)

Remarks

var	Numeric variable that receives the value.
address	The address where to read the value from. (0- &HFFFF) For Xmega which supports huge memory, the address is in range from 0-&HFFFFFF.

The PEEK() function will read only the lowest 32 memory locations (registers).
The INP() function can read from any memory location since the AVR has a linear memory model.

When you want to read from XRAM memory you must enable external memory access in the Compiler Chip Options 143.

See also

OUT 1394, PEEK 1395, POKE 1396, SETREG 1441, GETREG 1284

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : peek.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates PEEk, POKE, CPEEK, INP and OUT
'micro                    : Mega162
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m162def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31                                             'only 32
registers in AVR
  B1 = Peek(i)                                              'get byte
from internal memory
  Print Hex(b1) ; "  ";
  'Poke I , 1                            'write a value into memory
Next
Print                                                       'new line
'be careful when writing into internal memory !!

'now dump a part ofthe code-memory(program)
For I = 0 To 255
  B1 = Cpeek(i)                                             'get byte
from internal memory
  Print Hex(b1) ; "  ";
Next
'note that you can not write into codememory!!

Out &H8000 , 1                                              'write 1
into XRAM at address 8000
B1 = Inp(&H8000)                                            'return
value from XRAM
Print B1
End
```

7.69 LCD Commands

7.69.1 BOX

Action
Create a filled box on a graphical display.

Syntax
BOX (x1,y1) - (x2,y2) , color

Remarks

x1	The left corner position of the box
y1	The top position of the box
x2	The right corner position of the box
y2	The bottom position of the box
color	The color to use to fill the box

On COLOR displays, the box will be filled with the specified color.
On B&W displays, the box will not be filled. Only the box is drawn in the specified color.
On B&W displays you can use the BOXFILL statement to create a solid box.

See also
LINE 1344, CIRCLE 1319 , BOXFILL 1319

ASM
NONE

Example
```
'
------------------------------------------------------------------------
----------------
' The support for this display has been made possible by Peter Küsters
from (c) Display3000
' You can buy the displays from Display3000 or MCS Electronics
'
------------------------------------------------------------------------
----------------'
'
$lib "lcd-pcf8833.lbx"                                      'special
color display support

$regfile = "m88def.dat"                                     'ATMega 8,
change if using different processors
$crystal = 8000000                                          '8 MHz

'First we define that we use a graphic LCD
Config Graphlcd = Color , Controlport = Portc , Cs = 1 , Rs = 0 , Scl =
3 , Sda = 2
```

```
'here we define the colors

Const Blue = &B00000011                                    ''predefined
contants are making programming easier
Const Yellow = &B11111100
Const Red = &B11100000
Const Green = &B00011100
Const Black = &B00000000
Const White = &B11111111
Const Brightgreen = &B00111110
Const Darkgreen = &B00010100
Const Darkred = &B10100000
Const Darkblue = &B00000010
Const Brightblue = &B00011111
Const Orange = &B11111000


'clear the display
Cls

'create a cross
Line(0 , 0) -(130 , 130) , Blue
Line(130 , 0) -(0 , 130) , Red

Waitms 1000

'show an RLE encoded picture
Showpic 0 , 0 , Plaatje
Showpic 40 , 40 , Plaatje

Waitms 1000

'select a font
Setfont Color16x16
'and show some text
Lcdat 100 , 0 , "12345678" , Blue , Yellow


Waitms 1000
Circle(30 , 30) , 10 , Blue

Waitms 1000
'make a box
Box(10 , 30) -(60 , 100) , Red

'set some pixels
Pset 32 , 110 , Black
Pset 38 , 110 , Black
Pset 35 , 112 , Black

End


Plaatje:
$bgf "a.bgc"

$include "color.font"
$include "color16x16.font"
```

7.69.2 BOXFILL

Action
Create a filled box on a graphical display.

Syntax
BOXFILL (x1,y1) - (x2,y2) , color

Remarks

x1	The left corner position of the box
y1	The top position of the box
x2	The right corner position of the box
y2	The bottom position of the box
color	The color to use to fill the box

The BOXFILL command will draw a number of lines which will appear as a filled box.

See also
LINE 1344, CIRCLE 1319 , BOX 1317

ASM
NONE

Example
```
'create a bargraph effect
Boxfill(0 , 0) -(60 , 10) , 1
Boxfill(2 , 2) -(40 , 8) , 0
```

7.69.3 CIRCLE

Action
Draws a circle on a graphic display.

Syntax
CIRCLE(x0,y0) , radius, color

Remarks

X0	Starting horizontal location of the line.
Y0	Starting vertical location of the line.
Radius	Radius of the circle
Color	Color of the circle

See Also
LINE 1344

Example

```
'-------------------------------------------------------------------------
------------------
'name                     : t6963_240_128.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : T6963C graphic display support demo 240 *
128
'micro                    : Mega8535
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
------------------

$regfile = "m8535.dat"                                      ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'-----------------------------------------------------------------
'                    (c) 1995-2025 MCS Electronics
'                 T6963C graphic display support demo 240 * 128
'-----------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin                   connected to
' 1        GND             GND
'2        GND             GND
'3        +5V             +5V
'4        -9V             -9V potmeter
'5        /WR             PORTC.0
'6        /RD             PORTC.1
'7        /CE             PORTC.2
'8        C/D             PORTC.3
'9        NC              not conneted
'10       RESET           PORTC.4
'11-18    D0-D7            PA
'19       FS              PORTC.5
'20       NC              not connected

'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
```

```
Cls
'Other options are :
' CLS TEXT   to clear only the text display
' CLS GRAPH  to clear only the graphical part

Cursor Off

Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30


Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "12345678901234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"

Wait 2

Cls Text


'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255                                  ' diagonal
line
Line(0 , 127) -(239 , 0) , 255                                  ' diagonal
line
Line(0 , 0) -(240 , 0) , 255                                    ' horizontal
upper line
Line(0 , 127) -(239 , 127) , 255                                'horizontal
lower line
Line(0 , 0) -(0 , 127) , 255                                    ' vertical
left line
Line(239 , 0) -(239 , 127) , 255                                ' vertical
right line


Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
   Pset X , 20 , 255                                            ' set the
pixel
Next

For X = 0 To 140
   Pset X , 127 , 255                                           ' set the
pixel
Next

Wait 2

'circle time
'circle(X,Y), radius, color
'X,y is the middle of the circle,color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
  Circle(20 , 20) , X , 255                                     ' show
```

```
circle
  Wait 1
  Circle(20 , 20) , X , 0                                    'remove
circle
  Wait 1
Next

Wait 2

For X = 1 To 10
  Circle(20 , 20) , X , 255                                  ' show
circle
  Waitms 200
Next
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Test:
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje                                     ' show 2
since we have a big display
Wait 2
Cls Text                                                     ' clear the
text
End

'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

7.69.4 CLS

Action

Clear the LCD display and set the cursor to home.

Syntax

CLS

Syntax for graphical LCD

CLS
CLS TEXT
CLS GRAPH
CLS Y, X1 , X2 [, CHAR]

Remarks

Clearing the LCD display does not clear the CG-RAM in which the custom characters are stored.
For graphical LCD displays CLS will clear both the text and the graphical display.
The EADOG128 and KS108 support the option to clear a portion of a line. Depending on the used graphic chip, this option might be added to other graphical LCD lib's too.
Graphical displays coordinates start with 1. To clear the entire first line you need to

code : CLS 1,1,128
This will clear the first line, from the starting position X1(1) to the ending position (X2). You may specify an optional character to use. By default 0 is used. When you have inverse text, you need to use 255.

See also

$LCD 657 , $LCDRS 662 , LCD 1335 , SHIFTLCD 1354 , SHIFTCURSOR 1353 , SHIFTLCD 1354 , INITLCD 1334

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
SHIFTCURSOR, HOME
'                           CURSOR, DISPLAY
'micro                    : Mega8515
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8515.dat"                                      ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation


'note : tested in PIN mode with 4-bit

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector

Rem with the config lcdpin statement you can override the compiler
settings

Dim A As Byte
Config Lcd = 16 * 2                                         'configure
```

```
lcd screen

'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                     'clear the
LCD display
Lcd "Hello world."                                      'display
this at the top line
Wait 1
Lowerline                                               'select the
lower line
Wait 1
Lcd "Shift this."                                       'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                       'shift the
text to the right
   Wait 1                                               'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                        'shift the
text to the left
   Wait 1                                               'wait a
moment
Next

Locate 2 , 1                                            'set cursor
position
Lcd "*"                                                 'display
this
Wait 1                                                  'wait a
moment

Shiftcursor Right                                       'shift the
cursor
Lcd "@"                                                 'display
this
Wait 1                                                  'wait a
moment

Home Upper                                              'select line
1 and return home
Lcd "Replaced."                                         'replace the
text
Wait 1                                                  'wait a
moment

Cursor Off Noblink                                      'hide cursor
Wait 1                                                  'wait a
moment
Cursor On Blink                                         'show cursor
Wait 1                                                  'wait a
moment
Display Off                                             'turn
```

```
display off
Wait 1                                                          'wait a
moment
Display On                                                      'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                      'goto home
on line three
Home Fourth
Home F                                                          'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                             'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                             'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                      'value into
ACC
!rCall _write_lcd                                               'put it on
LCD
End
```

7.69.5 CURSOR

Action

Set the LCD Cursor State.

Syntax

CURSOR ON / OFF , BLINK / NOBLINK

Remarks

You can use both the ON or OFF and BLINK or NOBLINK parameters.
At power up the cursor state is ON and NOBLINK.
To get the proper value in all cases it is best to specify both parameters.

In 1995 when the LCD display support was created the processors had little pins. And the WR pin was not used and connected to ground.
But because of this there was no way to read data from the display. And since both parameters were optional, the state of the cursor was maintained internally by the compiler. In some cases this can give problems, especially when sub procedures are

called in various order.
That is why it is best to enter both parameters when you use the CURSOR statement.

See also

DISPLAY 1329 , LCD 1335 , SHIFTLCD 1354 , SHIFTCURSOR 1353

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : lcd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
SHIFTCURSOR, HOME
'                           CURSOR, DISPLAY
'micro                    : Mega8515
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m8515.dat"                                      ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation


'note : tested in PIN mode with 4-bit

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector

Rem with the config lcdpin statement you can override the compiler
settings


Dim A As Byte
Config Lcd = 16 * 2                                         'configure
lcd screen
```

```
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                     'clear the
LCD display
Lcd "Hello world."                                      'display
this at the top line
Wait 1
Lowerline                                               'select the
lower line
Wait 1
Lcd "Shift this."                                       'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                       'shift the
text to the right
   Wait 1                                               'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                        'shift the
text to the left
   Wait 1                                               'wait a
moment
Next

Locate 2 , 1                                            'set cursor
position
Lcd "*"                                                 'display
this
Wait 1                                                  'wait a
moment

Shiftcursor Right                                       'shift the
cursor
Lcd "@"                                                 'display
this
Wait 1                                                  'wait a
moment

Home Upper                                              'select line
1 and return home
Lcd "Replaced."                                         'replace the
text
Wait 1                                                  'wait a
moment

Cursor Off Noblink                                      'hide cursor
Wait 1                                                  'wait a
moment
Cursor On Blink                                         'show cursor
Wait 1                                                  'wait a
moment
Display Off                                             'turn
display off
Wait 1                                                  'wait a
```

```
moment
Display On                                                    'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                    'goto home
on line three
Home Fourth
Home F                                                        'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                           'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                           'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                    'value into
ACC
!rCall _write_lcd                                             'put it on
LCD
End
```

7.69.6 DEFLCDCHAR

Action

Define a custom LCD character.

Syntax

DEFLCDCHAR char,r1,r2,r3,r4,r5,r6,r7,r8

Remarks

char	Constant representing the character (0-7).
r1-r8	The row values for the character.

You can use the LCD designer 131 to build the characters.

- It is important that a CLS follows the DEFLCDCHAR statement(s).
So make sure you use the DEFLCDCHAR before your CLS statement.
- When using INITLCD make sure this is called before DEFLCDCHAR since it will reset the LCD controller.

Special characters can be printed with the Chr 827() function.

LCD Text displays have a 64 byte memory that can be used to show your own custom characters. Each character uses 8 bytes as the character is an array from 8x8 pixels. You can create a maximum of 8 characters this way. Or better said : you can show a maximum of 8 custom characters at the same time. You can redefine characters in your program but with the previous mentioned restriction.
A custom character can be used to show characters that are not available in the LCD font table. For example a Û.
You can also use custom characters to create a bar graph or a music note.

Note:
You cannot use Chr(0)-**Deflcdchar** 0 in any with any String Variables/Arrays, Chr(0) will be interpreted as a String terminator
and not as Custom Character for **Deflcdchar** 0 (**Deflcdchar** from 1 to 7 is fine).

See also
Tools LCD designer 131 , LCD 1335 , CLS 1322 , CURSOR 1325 , DISPLAY 1329 , LOCATE 1347

Partial Example
```
Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                          'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                          'print the
special character
```

7.69.7 DISPLAY

Action
Turn LCD display ON or OFF.

Syntax
DISPLAY ON | OFF
DISPLAY ON | OFF , CURSOR | NOCURSOR , BLINK | NOBLINK

Remarks
The display is turned on at power up.
When you use DISPLAY with a single parameter, the compiler will maintain a variable to hold the status of the display. In some cases this can lead to an unexpected result. This depends on the order of how the commands are called.
If you experience this problem, you can use the alternative syntax which demands that all 3 parameters are specified.
This does not use any state variable and will update the LCD command register.
The second syntax is advised to be used.

See also

LCD 1335

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
SHIFTCURSOR, HOME
'                           CURSOR, DISPLAY
'micro                    : Mega8515
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8515.dat"                                      ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation


'note : tested in PIN mode with 4-bit

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector

Rem with the config lcdpin statement you can override the compiler
settings


Dim A As Byte
Config Lcd = 16 * 2                                         'configure
lcd screen


'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines
```

```
'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                     'clear the
LCD display
Lcd "Hello world."                                      'display
this at the top line
Wait 1
Lowerline                                               'select the
lower line
Wait 1
Lcd "Shift this."                                       'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                       'shift the
text to the right
   Wait 1                                               'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                        'shift the
text to the left
   Wait 1                                               'wait a
moment
Next

Locate 2 , 1                                            'set cursor
position
Lcd "*"                                                 'display
this
Wait 1                                                  'wait a
moment

Shiftcursor Right                                       'shift the
cursor
Lcd "@"                                                 'display
this
Wait 1                                                  'wait a
moment

Home Upper                                              'select line
1 and return home
Lcd "Replaced."                                         'replace the
text
Wait 1                                                  'wait a
moment

Cursor Off Noblink                                      'hide cursor
Wait 1                                                  'wait a
moment
Cursor On Blink                                         'show cursor
Wait 1                                                  'wait a
moment
Display Off                                             'turn
display off
Wait 1                                                  'wait a
moment
Display On                                              'turn
display on
'-----------------NEW support for 4-line LCD------
```

```
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                    'goto home
on line three
Home Fourth
Home F                                                        'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                           'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                           'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                    'value into
ACC
!rCall _write_lcd                                             'put it on
LCD
End
```

7.69.8 FOURTHLINE

Action

Set LCD cursor to the start of the fourth line.

Syntax

FOURTHLINE

Remarks

Only valid for LCD displays with 4 lines.

See also

HOME 1334 , UPPERLINE 1357 , LOWERLINE 1347 , THIRDLINE 1357, LOCATE 1347

Example

```
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                    'goto home
on line three
```

```
Home Fourth
Home F                                    'first   letter   also   works
```

7.69.9 GLCDCMD

Action
Sends a command byte to the SED graphical LCD display.

Syntax
GLCDCMD byte [,chip]

Remarks

byte	A variable or numeric constant to send to the display.
chip	An optional numeric variable or constant in the range from **1-2** which indicates which graphic chip CE line need to be selected. The routine _selchip1 or _selchip2 is called.

With GLCDCMD you can write command bytes to the display. This is convenient to control the display when there is no specific statement available.

You need to include the glibSED library with :
$LIB "glibsed.lbx"

See also
CONFIG GRAPHLCD 992 , LCDAT 1339, GLCDDATA 1333

Example
NONE

7.69.10 GLCDDATA

Action
Sends a data byte to the SED graphical LCD display.

Syntax
GLCDDATA byte [,chip]

Remarks

byte	A variable or numeric constant to send to the display.
chip	An optional numeric variable or constant in the range from **1-2** which indicates which graphic chip CE line need to be selected. The routine _selchip1 or _selchip2 is called.

With GLCDDATA you can write data bytes to the display. This is convenient to control the display when there is no specific statement available.
You need to include the glibSED library with :

$LIB "glibsed.lbx"

See also
CONFIG GRAPHLCD 992 , LCDAT 1339, GLCDCMD 1333

Example
NONE

7.69.11 HOME

Action
Place the cursor at the specified line at location 1.

Syntax
HOME UPPER | LOWER | THIRD | FOURTH

Remarks
If only HOME is used than the cursor will be set to the upper line.
You may also specify the first letter of the line like: HOME U

See also
CLS 1322 , LOCATE 1347

For a complete example see LCD 1335

Partial Example
```
Locate 2 , 1                                    'set cursor
position
Lcd "*"                                         'display this
Home Upper                                      'select line
1 and return home
```

7.69.12 INITLCD

Action
Initializes the LCD display.

Syntax
INITLCD

Remarks
The LCD display is initialized automatic at start up when LCD statements are used by your code.
This is done by a call to _LCD_INIT.
If you include the INITLCD statement in your code, the automatic call is disabled and the _LCD_INIT is called at the place in your code where you put the INITLCD

statement. (initlcd is translated into a call to _init_lcd).

Why is this useful?
- In an environments with static electricity, the display can give strange output. You can initialize the display then once in a while. When the display is initialized, the display content is cleared also.
- The LCD routines depend on the fact that the WR pin of the LCD is connected to ground. But when you connect it to a port pin, you must first set the logic level to 0 and after that you can initialize the display by using INITLCD
- Xmega chips need a stable oscillator. This is done with some CONFIG statements. The INITLCD should be placed after these commands. And since the Xmega by default has a slow internal oscillator, without using INITLCD at the proper location, your application would start slow. See the explanation below.
- So in short you have more control when the LCD is initialized.

 The CONFIG LCDPIN 1001 has an option to use the WR pin, and use the busy flag of the display. If you have enough pins, this is the best mode.

 The **XMEGA** has a built in internal oscillator that runs at a relative slow speed. If your code sets the speed to 32 MHz and you also include the $crystal=32000000 directive, you will notice a delay in the start of the code. This is caused by the fact that the delay routines are calculated with the 32 Mhz frequency, but the actual oscillator speed is 1 or 2 MHz.
There are 2 solutions possible.
- you can use $crystal=1000000 and then after you have set up the clock speed with CONFIG OSC, you can use another $CRYSTAL directive with the new speed.
- you use $INITMICRO and put the CONFIG OSC in the _INIT_MICRO code. This will ensure that the micro will run at the specified speed early as possible.

ASM
The generated ASM code :
Rcall _Init_LCD

See also
LCD 657 , CONFIG LCDPIN 1001

Example
NONE

7.69.13 LCD

Action
Send constant or variable to LCD display.

Syntax
LCD x

Remarks

X	Variable or constant to display.

More variables can be displayed separated by the **;** -sign

```
LCD a ; b1 ; "constant"
```

The LCD statement behaves just like the PRINT 1501 statement. So SPC 1508() can be used too.
The only difference with PRINT is that no CR+LF is added when you send data to the LCD.

See also

$LCD 657 , $LCDRS 662 , CONFIG LCD 992 , SPC 1508 , CLS 1322 , INITLCD 1334 , SHIFTLCD 1354 , SHIFTCURSOR 1353 , CURSOR 1325 , LCDCMD 1341, LCDDATA 1341

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
SHIFTCURSOR, HOME
'                           CURSOR, DISPLAY
'micro                    : Mega8515
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8515.dat"                                      ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation


'note : tested in PIN mode with 4-bit

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector
```

```
Rem with the config lcdpin statement you can override the compiler
settings


Dim A As Byte
Config Lcd = 16 * 2                                         'configure
lcd screen


'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it aint need the port pins !

Cls                                                         'clear the
LCD display
Lcd "Hello world."                                          'display
this at the top line
Wait 1
Lowerline                                                   'select the
lower line
Wait 1
Lcd "Shift this."                                           'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                           'shift the
text to the right
   Wait 1                                                   'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                            'shift the
text to the left
   Wait 1                                                   'wait a
moment
Next

Locate 2 , 1                                                'set cursor
position
Lcd "*"                                                     'display
this
Wait 1                                                      'wait a
moment

Shiftcursor Right                                           'shift the
cursor
Lcd "@"                                                     'display
this
Wait 1                                                      'wait a
moment

Home Upper                                                  'select line
1 and return home
Lcd "Replaced."                                             'replace the
text
Wait 1                                                      'wait a
moment
```

```
Cursor Off Noblink                                          'hide cursor
Wait 1                                                      'wait a
moment
Cursor On Blink                                             'show cursor
Wait 1                                                      'wait a
moment
Display Off                                                 'turn
display off
Wait 1                                                      'wait a
moment
Display On                                                  'turn
display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                  'goto home
on line three
Home Fourth
Home F                                                      'first
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228       '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240       '
replace ? with number (0-7)
Cls                                                         'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)                                         'print the
special character

'----------------- Now use an internal routine ------------
_temp1 = 1                                                  'value into
ACC
!rCall _write_lcd                                           'put it on
LCD
REM BETTER USE LCDCMD
End
```

7.69.14 LCDAUTODIM

Action

Dims the 20x4vfd LCD.

Syntax

LCDAUTODIM x

Remarks

X	A variable or constant in the range from 0-54

	0 will turn auto dim off. A value between 1-54 dims the brightness after the given number of seconds. The value is stored permanent.

This statement works only with the 20x4vfd display from "Electronic Design Bitzer" Available in the MCS Shop.

See also
NONE

Example
NONE

7.69.15 LCDAT

Action
Send constant or variable to a SED or other graphical display.

Syntax
LCDAT y , x , var [, inv]
LCDAT y , x , var [, FG, BG]

Remarks

X	X location. In the range from 0-63. The SED displays columns are 1 pixel width. Other displays might have a bigger range such as 132 or 255.
Y	Y location. The row in pixels. The maximum value depends on the display. The minimum value also depends on the used display. Most displays have minimum value of 0. KS108 has a minimum value of 1.
Var	The constant or variable to display
inv	Optional number. Value 0 will show the data normal. Any other value will invert the data.
	For COLOR DISPLAYS
FG	Foreground color
BG	Background color

You need to include the glibSED library with :
$LIB "glibsed.lbx"

Other libraries must be included with a different directive.

See also
CONFIG GRAPHLCD 992 , SETFONT 1351, GLCDCMD 1333, GLCDDATA 1333

Example
```
'-------------------------------------------------------------------
-----------------
```

```
'name                     : sed1520.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates the SED1520 based graphical
display support
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 7372800                                         ' used
crystal frequency
$baud = 115200                                             ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'I used a Staver to test

'some routines to control the display are in the glcdSED.lib file
'IMPORTANT : since the SED1520 uses 2 chips, the columns are split into
2 of 60.
'This means that data after column 60 will not print correct. You need
to locate the data on the second halve
'For example when you want to display a line of text that is more then 8
chars long, (8x8=64) , byte 8 will not draw correctly
'Frankly i find the KS0108 displays a much better choice.

$lib "glcdSED1520.lbx"

'First we define that we use a graphic LCD

Config Graphlcd = 120 * 64sed , Dataport = Porta , Controlport = Portd ,
Ce = 5 , Ce2 = 7 , Cd = 3 , Rd = 4


'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE =CS  Chip Enable/ Chip select
'CE2= Chip select / chip enable of chip 2
'CD=A0   Data direction
'RD=Read

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'clear the screen
Cls
Wait 2
'specify the font we want to use
Setfont Font8x8

'You can use locate but the columns have a range from 1-132

'When you want to show somthing on the LCD, use the LDAT command
'LCDAT Y , COL, value
```

```
Lcdat 1 , 1 , "1231231"
Lcdat 3 , 80 , "11"
'lcdat accepts an additional param for inversing the text
'lcdat 1,1,"123" , 1   ' will inverse the text

Wait 2
Line(0 , 0) -(30 , 30) , 1
Wait 2

Showpic 0 , 0 , Plaatje                                    'show a
comnpressed picture
End                                                        'end program


'we need to include the font files
$include "font8x8.font"
'$include "font16x16.font"


Plaatje:
'include the picture data
$bgf "smile.bgf"
```

7.69.16 LCDCMD

Action
Send a byte in command mode to a Text LCD display.

Syntax
LCDCMD byte

Remarks
To send data to an LCD display you need to use the LCD statement. If you have the need to call the internal LCD routine which sends a byte in command mode, you can use the LCDCMD statement. The byte can be a variable or numeric constant.

See also
LCD 1335 , LCDDATA 1341

Example

```
Lcdcmd 10                                  ' will call _lcd_control
Lcddata 65                                 ' will call _write_lcd and
send ASCII 65 (A)
```

7.69.17 LCDDATA

Action
Send a byte in data mode to a Text LCD display.

Syntax
LCDDATA byte

Remarks
To send data to an LCD display you need to use the LCD statement. If you have the need to call the internal LCD routine which sends a byte in data mode, you can use the LCDDATA statement. The byte can be a variable or numeric constant.

See also
LCD 1335 , LCDCMD 1341

Example
```
Lcdcmd 10                                   ' will call _lcd_control
Lcddata 65                                  ' will call _write_lcd and
send ASCII 65 (A)
```

7.69.18 LCDCONTRAST

Action
Set the contrast of a TEXT LCD.

Syntax
LCDCONTRAST x

Remarks

X	A variable or constant in the range from 0-3.

Some LCD text displays support changing the contrast. Noritake displays have this option for example.

See also
LCD 1335, LCDFONT 1342

Example
NONE

7.69.19 LCDFONT

Action
Selects the font of the TEXT LCD.

Syntax
LCDFONT x

Remarks

X	A variable or constant in the range from 0-3.

Most text LCD displays have one or more built in font tables. By default font 0 is selected.
The LCDFONT statement allows you to chose another font.

See also

LCD 1335 , INITLCD 1334 , LCDCMD 1341, LCDDATA 1341

Example

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
$hwstack=32
$swstack = 16
$framesize=24



$lib "lcd4_anypin_oled_RS0010.lib"                          'override
default lib with OLED lib

'Config Lcd Sets The Portpins Of The Lcd
Config Lcdpin = Pin , Db4 = Portb.2 , Db5 = Portb.3 , Db6 = Portb.4 ,
Db7 = Portb.5 , E = Portb.1 , Rs = Portb.0
Config Lcd = 16x2                                           '16*2 type
LCD screen

Dim V As Byte

Cls
Lcd "ABC" ; Chr(253)
Lowerline
Lcd "test"
Const Test = " this is a test"                              ' Just  A
Test

Lcdfont 0                                                   'select
first font

Cls
Dim X As Byte , Y As Byte
X = &B1000_0000 + 0
Lcdcmd &B0001_1111                                          'gmode
Lcdcmd X                                                    'X  (0-99)
Lcdcmd &B0100_0000                                          'Y  (0-1)

'send data
For V = 1 To 80
   Lcddata &B10101010
   Waitms 100
```

```
Next
End
```

7.69.20 LINE

Action
Draws a line on a graphic display.

Syntax
LINE(x0,y0) - (x1,y1), color

Remarks

X0	Starting horizontal location of the line.
Y0	Starting vertical location of the line.
X1	Horizontal end location of the line
Y1	Vertical end location of the line.
color	The color to use. Use 0 or a non zero value.

See Also
LINE 1344 , CONFIG GRAPHLCD 964 , BOX 1317 , BOXFILL 1319

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : t6963_240_128.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : T6963C graphic display support demo 240 *
128
'micro                    : Mega8535
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8535.dat"                                      ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'-------------------------------------------------------------------
'                    (c) 1995-2025 MCS Electronics
'                 T6963C graphic display support demo 240 * 128
'-------------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin                  connected to
```

```
' 1        GND             GND
'2        GND             GND
'3        +5V             +5V
'4        -9V             -9V potmeter
'5        /WR             PORTC.0
'6        /RD             PORTC.1
'7        /CE             PORTC.2
'8        C/D             PORTC.3
'9        NC              not conneted
'10       RESET           PORTC.4
'11-18    D0-D7            PA
'19       FS              PORTC.5
'20       NC              not connected

'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT   to clear only the text display
' CLS GRAPH  to clear only the graphical part

Cursor Off

Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30


Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "12345678901234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"

Wait 2

Cls Text


'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255                                  ' diagonal
line
Line(0 , 127) -(239 , 0) , 255                                  ' diagonal
line
Line(0 , 0) -(240 , 0) , 255                                    ' horizontal
upper line
```

```
Line(0 , 127) -(239 , 127) , 255                                    'horizontal
lower line
Line(0 , 0) -(0 , 127) , 255                                        ' vertical
left line
Line(239 , 0) -(239 , 127) , 255                                    ' vertical
right line


Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
   Pset X , 20 , 255                                                ' set the
pixel
Next

For X = 0 To 140
   Pset X , 127 , 255                                               ' set the
pixel
Next

Wait 2

'circle time
'circle(X,Y), radius, color
'X,y is the middle of the circle,color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
  Circle(20 , 20) , X , 255                                         ' show
circle
  Wait 1
  Circle(20 , 20) , X , 0                                           'remove
circle
  Wait 1
Next

Wait 2

For X = 1 To 10
  Circle(20 , 20) , X , 255                                         ' show
circle
  Waitms 200
Next
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Test:
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje                                            ' show 2
since we have a big display
Wait 2
Cls Text                                                            ' clear the
text
End

'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

7.69.21 LOCATE

Action
Moves the LCD cursor to the specified position.

Syntax
LOCATE y , x

Remarks

X	Constant or variable with the position. (1-64*)
Y	Constant or variable with the line (1 - 4*)

* Depending on the used display

See also
CONFIG LCD 992 , LCD 1335 , HOME 1334 , CLS 1322

Partial Example
```
LCD "Hello"
Locate 1,10
LCD "*"
```

7.69.22 LOWERLINE

Action
Reset the LCD cursor to the lower line.

Syntax
LOWERLINE

Remarks
NONE

See also
UPPERLINE 1357 , THIRDLINE 1357 , FOURTHLINE 1332 , HOME 1334

Partial Example
```
Lcd "Test"
Lowerline
Lcd "Hello"
End
```

7.69.23 PSET

Action
Sets or resets a single pixel.

Syntax
PSET X , Y, value

Remarks

X	The X location of the pixel. In range from 0-239.
Y	The Y location of the pixel. In range from 0-63.
value	The value for the pixel. 0 will clear the pixel. 1 Will set the pixel.

The PSET is handy to create a simple data logger or oscilloscope.

See also
SHOWPIC 1355 , CONFIG GRAPHLCD 964 , LINE 1344

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : t6963_240_128.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : T6963C graphic display support demo 240 *
128
'micro                    : Mega8535
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8535.dat"                                      ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'---------------------------------------------------------------------
'                    (c) 1995-2025 MCS Electronics
'                 T6963C graphic display support demo 240 * 128
'---------------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin                 connected to
'1        GND            GND
'2        GND            GND
'3        +5V            +5V
'4        -9V            -9V potmeter
'5        /WR            PORTC.0
```

```
'6        /RD           PORTC.1
'7        /CE           PORTC.2
'8        C/D           PORTC.3
'9        NC            not conneted
'10       RESET         PORTC.4
'11-18    D0-D7         PA
'19       FS            PORTC.5
'20       NC            not connected

'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT   to clear only the text display
' CLS GRAPH  to clear only the graphical part

Cursor Off

Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30


Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "12345678901234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"

Wait 2

Cls Text


'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255                                  ' diagonal
line
Line(0 , 127) -(239 , 0) , 255                                  ' diagonal
line
Line(0 , 0) -(240 , 0) , 255                                    ' horizontal
upper line
Line(0 , 127) -(239 , 127) , 255                                'horizontal
lower line
Line(0 , 0) -(0 , 127) , 255                                    ' vertical
left line
Line(239 , 0) -(239 , 127) , 255                                ' vertical
```

```
right line


Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
   Pset X , 20 , 255                                          ' set the
pixel
Next

For X = 0 To 140
   Pset X , 127 , 255                                         ' set the
pixel
Next

Wait 2

'circle time
'circle(X,Y), radius, color
'X,y is the middle of the circle,color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
  Circle(20 , 20) , X , 255                                   ' show
circle
  Wait 1
  Circle(20 , 20) , X , 0                                     'remove
circle
  Wait 1
Next

Wait 2

For X = 1 To 10
  Circle(20 , 20) , X , 255                                   ' show
circle
  Waitms 200
Next
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Test:
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje                                      ' show 2
since we have a big display
Wait 2
Cls Text                                                      ' clear the
text
End

'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"
'You could insert other picture data here
```

7.69.24 RGB8TO16

Action

This function converts an RGB8 byte value into an RGB16 word value.

Syntax
var **= RGB8TO16(**bOld**)**

Remarks

var	The word value that is assigned with the RGB16 value of bOld.
bOld	The byte that contains the RGB8 value.

There are many different graphical LCD displays and most new displays can display in color. There are 8 bit and 16 bit displays. And beside the data bus width displays have different color resolution.
While high resolution is nice, it also means you need more data to display a pixel. The RGB8TO16() function converts an 8 bit RGB value into a 16 bit RGB value. This way you can use the bascom created BGC files.

See also
NONE

Example
```
NONE
```

7.69.25 SETFONT

Action
Sets the current font which can be used on some graphical displays.

Syntax
SETFONT font

Remarks

font	The name of the font that need to be used with LCDAT statements.

Since SED-based displays do not have their own font generator, you need to define your own fonts. You can create and modify your own fonts with the FontEditor Plugin.

SETFONT will set an internal used data pointer to the location in memory where you font is stored. The name you specify is the same name you use to define the font.

You need to include the used fonts with the $include directive:

$INCLUDE "font8x8.font"

The order of the font files is not important. The location in your source is however important.
The $INCLUDE statement will include binary data and this may not be accessed by the flow of your program.
When your program flow enters into font code, unpredictable results will occur.
So it is best to place the $INCLUDE files at the end of your program behind the END statement.

You need to include the glibSED library with :
$LIB "glibsed.lbx"
While original written for the SED1521, fonts are supported on a number of displays now including color displays.

See also

CONFIG GRAPHLCD 992 , LCDAT 1339, GLCDCMD 1333, GLCDDATA 1333

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : sed1520.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates the SED1520 based graphical
display support
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 7372800                                          ' used
crystal frequency
$baud = 115200                                              ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'I used a Staver to test

'some routines to control the display are in the glcdSED.lib file
'IMPORTANT : since the SED1520 uses 2 chips, the columns are split into
2 of 60.
'This means that data after column 60 will not print correct. You need
to locate the data on the second halve
'For example when you want to display a line of text that is more then 8
chars long, (8x8=64) , byte 8 will not draw correctly
'Frankly i find the KS0108 displays a much better choice.

$lib "glcdSED1520.lbx"

'First we define that we use a graphic LCD

Config Graphlcd = 120 * 64sed , Dataport = Porta , Controlport = Portd ,
Ce = 5 , Ce2 = 7 , Cd = 3 , Rd = 4


'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE =CS  Chip Enable/ Chip select
'CE2= Chip select / chip enable of chip 2
'CD=A0   Data direction
```

```
'RD=Read

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'clear the screen
Cls
Wait 2
'specify the font we want to use
Setfont Font8x8

'You can use locate but the columns have a range from 1-132

'When you want to show somthing on the LCD, use the LDAT command
'LCDAT Y , COL, value
Lcdat 1 , 1 , "1231231"
Lcdat 3 , 80 , "11"
'lcdat accepts an additional param for inversing the text
'lcdat 1,1,"123" , 1  ' will inverse the text

Wait 2
Line(0 , 0) -(30 , 30) , 1
Wait 2

Showpic 0 , 0 , Plaatje                                    'show a
comnpressed picture
End                                                        'end program

'we need to include the font files
$include "font8x8.font"
'$include "font16x16.font"


Plaatje:
'include the picture data
$bgf "smile.bgf"
```

7.69.26 SHIFTCURSOR

Action

Shift the cursor of the LCD display left or right by one position.

Syntax

SHIFTCURSOR LEFT | RIGHT

See also

SHIFTLCD 1354

Partial Example

```
LCD "Hello"
SHIFTCURSOR LEFT
End
```

7.69.27 SHIFTLCD

Action
Shift the LCD display left or right by one position.

Syntax
SHIFTLCD LEFT / RIGHT

Remarks
NONE

See also
SHIFTCURSOR 1353 , SHIFTCURSOR 1353 , INITLCD 1334 , CURSOR 1325

Partial Example
```
Cls                                                    'clear the
LCD display
Lcd "Hello world."                                     'display
this at the top line
Wait 1
Lowerline                                              'select the
lower line
Wait 1
Lcd "Shift this."                                      'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right                                      'shift the
text to the right
   Wait 1                                              'wait a
moment
Next

For A = 1 To 10
   Shiftlcd Left                                       'shift the
text to the left
   Wait 1                                              'wait a
moment
Next

Locate 2 , 1                                           'set cursor
position
Lcd "*"                                                'display
this
Wait 1                                                 'wait a
moment

Shiftcursor Right                                      'shift the
cursor
Lcd "@"                                                'display
this
```

7.69.28 SHOWPIC

Action
Shows a BGF file on the graphic display

Syntax
SHOWPIC x, y , label

Remarks
Showpic can display a converted BMP file. The BMP must be converted into a BGF file with the Tools Graphic Converter 133.

The X and Y parameters specify where the picture must be displayed. X and Y must be 0 or a multiple of 8. The picture height and width must also be a multiple of 8.

The label tells the compiler where the graphic data is located. It points to a label where you put the graphic data with the $BGF directive.

You can store multiple pictures when you use multiple labels and $BGF directives,

Note that the BGF files are RLE encoded to save code space.

See also
PSET 1348 , $BGF 609 , CONFIG GRAPHLCD 964 , LINE 1344 , CIRCLE 1319 , SHOWPICE 1355

Example
See $BGF 609 example

7.69.29 SHOWPICE

Action
Shows a BGF file stored in EEPROM on the graphic display

Syntax
SHOWPICE x, y , label

Remarks
Showpice can display a converted BMP file that is stored in the EEPROM of the micro processor. The BMP must be converted into a BGF file with the Tools Graphic Converter 133.

The X and Y parameters specify where the picture must be displayed. X and Y must be 0 or a multiple of 8. The picture height and width must also be a multiple of 8.

The label tells the compiler where the graphic data is located. It points to a label where you put the graphic data with the $BGF directive.
You can store multiple pictures when you use multiple labels and $BGF directives,

Note that the BGF files are RLE encoded to save code space.

See also

PSET 1348 , $BGF 609 , CONFIG GRAPHLCD 964 , LINE 1344 , SHOWPIC 1355 , CIRCLE 1319

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : showpice.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates showing a picture from EEPROM
'micro                    : AT90S8535
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "8535def.dat"                                    ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is th e portname that is connected to the data lines of
the LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

'we will load the picture data into EEPROM so we specify $EEPROM
'the data must be specified before the showpicE statement.
$eeprom
Plaatje:
'the $BGF directive will load the data into the EEPROM or FLASH
depending on the $EEPROM or $DATA directive
$bgf "mcs.bgf"
'switch back to normal DATA (flash) mode
$data

'Clear the screen will both clear text and graph display
Cls
'showpicE is used to show a picture from EEPROM
'showpic must be used when the data is located in Flash
Showpice 0 , 0 , Plaatje
End
```

7.69.30 THIRDLINE

Action
Reset LCD cursor to the third line.

Syntax
THIRDLINE

Remarks
NONE

See also
UPPERLINE 1357 , LOWERLINE 1347 , FOURTHLINE 1332

Example
```
Dim A As Byte
A = 255
Cls
Lcd A
Thirdline
Lcd A
Upperline
End
```

7.69.31 UPPERLINE

Action
Reset LCD cursor to the upper line.

Syntax
UPPERLINE

Remarks
Optional you can also use the LOCATE statement.

See also
LOWERLINE 1347 , THIRDLINE 1357 , FOURTHLINE 1332 , LCD 1335, CLS 1322 , LOCATE 1347

Example
```
Dim A As Byte
A = 255
Cls
Lcd A
Thirdline
Lcd A
Upperline
End
```

7.70 LOAD

Action
Load specified TIMER with a reload value.

Syntax
LOAD TIMER , value

Remarks

TIMER	TIMER0 , TIMER1 or TIMER2(or valid timer name)
Value	The variable or value to load.

The TIMER0 does not have a reload mode. But when you want the timer to generate an interrupt after 10 ticks for example, you can use the LOAD statement.

It will do the calculation : (256-value)

So LOAD TIMER0, 10 will load the TIMER0 with a value of 246 so that it will overflow after 10 ticks.
TIMER1 is a 16 bit counter so it will be loaded with the value of 65536-value.

See Also
NONE

Example
NONE

7.71 LOADADR

Action
Loads the address of a variable into a register pair.

Syntax
LOADADR var , reg

Remarks

var	A variable which address must be loaded into the register pair X, Y or Z.
reg	The register X, Y or Z.

The LOADADR statement serves as an assembly helper routine.

Example
```
Dim S As String * 12
Dim A As Byte

$ASM
loadadr S , X     ; load address into R26 and R27
```

```
ld _temp1, X      ; load value of location R26/R27 into R24(_temp1)
$END ASM
```

7.72 LOADLABEL

Action
Assigns a word variable with the address of a label.

Syntax
Var = **LOADLABEL**(label)

Remarks

var	The variable that is assigned with the address of the label.
lbl	The name of the label

In some cases you might need to know the address of a point in your program. To perform a Cpeek() for example.
You can place a label at that point and use LoadLabel to assign the address of the label to a variable.

When you assign a DWORD variable, the 24 bit address will be loaded into the variable.

If you use Loadlabel on an EEPROM label (a label used in the $EEPROM data area) , these labels must precede the Loadlabel function.

This would be ok :

```
$eeprom     ' eeprom image
label1:
data 1,2,3,4,5
label2:
data 6,7,8,9,10
$data       ' back to normal mode

dim w as word
w=loadlabel(label2)
```

This code will work since the loadlabel is used after the EEPROM data labels.

7.73 LOADWORDADR

Action
Loads the Z-register and sets RAMPZ if available.

Syntax
LOADWORDADR label

Remarks

label	The name of the label which address will be loaded into R30-R31 which form the Z-register.

The code that will be generated :
```
LDI R30,Low(label * 2)
LDI R31,High(label * 2)
LDI R24,1 or CLR R24
STS RAMPZ, R24
```

As the AVR uses a word address, to find a byte address we multiply the address with 2. RAMPZ forms together with pointer **Z** an address register. As the LS bit of Z is used to identify the lower or the upper BYTE of the address, it is extended with the RAMPZ to address more then 15 bits. For example the Mega128 has 128KB of space and needs the RAMPZ register set to the right value in order to address the upper or lower 64KB of space.

See also

LOADLABEL 1359, LOADADR 1358 , LOOKUP 1365

Example

```
LOADWORDADR label
```

7.74 LOCAL

Action

Dimensions a variable LOCAL to the function or sub program.

Syntax

LOCAL var As Type

Remarks

Var	The name of the variable
Type	The data type of the variable.

There can be only LOCAL variables of the type BYTE, INTEGER, WORD, DWORD, LONG, SINGLE, DOUBLE or STRING.

A LOCAL variable is a temporary variable that is stored on the frame.
When the SUB or FUNCTION is terminated, the memory will be released back to the system.
A Sub/Function is full reentrant which means that it can be called recursively.
Because of this, local memory is dynamic and not static as global variables.
BIT variables are not possible because they are GLOBAL to the system.

The AT , ERAM, SRAM, XRAM directives can not be used with a local DIM statement.
Also local arrays are not possible.

Notice that a LOCAL variable is not initialized. It will contain a value that will depend on the value of the FRAME data. So you can not assume the variable is 0. If you like it to be 0, you need to assign it.

A normal DIM-med variable is also not initialized to 0. The reason all variables are 0 (and strings are ""), is that the RAM memory is cleared. With the $NORAMCLEAR 686 option you can turn this behaviour off.
So to conclude, a LOCAL variable will behave the same as a normal variable with the $NORAMCLEAR option enabled.

While it would be simple to initialize the LOCAL variables to 0, in most/all cases, you will assign a value to it anyway, so it would be a waste of code space.

See also
DIM 1228

ASM
NONE

Example
```
'-------------------------------------------------------------------------
------------------
'name                     : declare.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrate using declare
'micro                    : Mega48
'suited for demo          : yes
'commercial add on needed  : no
' Note that the usage of SUBS works different in BASCOM-8051
'-------------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space


' First the SUB programs must be declared

'Try a SUB without parameters
Declare Sub Test2()

'SUB with variable that can not be changed(A) and
'a variable that can be changed(B1), by the sub program
'When BYVAL is specified, the value is passed to the subprogram
'When BYREF is specified or nothing is specified, the address is passed
to
'the subprogram

Declare Sub Test(byval A As Byte , B1 As Byte)
Declare Sub Testarray(byval A As Byte , B1 As Byte)
'All variable types that can be passed
```

```
'Notice that BIT variables can not be passed.
'BIT variables are GLOBAL to the application
Declare Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S
As String)

'passing string arrays needs a different syntax because the length of
the strings must be passed by the compiler
'the empty () indicated that an array will be passed
Declare Sub Teststr(b As Byte , Dl() As String)

Dim Bb As Byte , I As Integer , W As Word , L As Long , S As String * 10
      'dim used variables
Dim Ar(10) As Byte
Dim Sar(10) As String * 8                                   'strng array

For Bb = 1 To 10
  Sar(bb) = Str(bb)                                         'fill the
array
Next
Bb = 1
'now call the sub and notice that we always must pass the first address
with index 1
Call Teststr(bb , Sar(1))


Call Test2                                                  'call sub
Test2                                                       'or use
without CALL
'Note that when calling a sub without the statement CALL, the enclosing
parentheses must be left out
Bb = 1
Call Test(1 , Bb)                                          'call sub
with parameters
Print Bb                                                    'print value
that is changed

'now test all the variable types
Call Testvar(bb , I , W , L , S )
Print Bb ; I ; W ; L ; S

'now pass an array
'note that it must be passed by reference
Testarray 2 , Ar(1)
Print "ar(1) = " ; Ar(1)
Print "ar(3) = " ; Ar(3)

$notypecheck                                                ' turn off
type checking
Testvar Bb , I , I , I , S
'you can turn off type checking when you want to pass a block of memory
$typecheck                                                  'turn it
back on
End

'End your code with the subprograms
'Note that the same variables and names must be used as the declared
ones

Sub Test(byval A As Byte , B1 As Byte)                      'start sub
    Print A ; " " ; B1                                      'print
passed variables
    B1 = 3                                                  'change
value
    'You can change A, but since a copy is passed to the SUB,
```

```
    'the change will not reflect to the calling variable
End Sub

Sub Test2                                                    'sub without
parameters
    Print "No parameters"
End Sub


Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As
String)
    Local X As Byte
    X = 5                                                    'assign
local
    B = X
    I = -1
    W = 40000
    L = 20000
    S = "test"
End Sub

Sub Testarray(byval A As Byte , B1 As Byte)                  'start sub
    Print A ; " " ; B1                                       'print
passed variables
    B1 = 3                                                   'change
value of element with index 1
    B1(1) = 3                                                'specify the
index which does the same as the line above
    B1(3) = 3                                                'modify
other element of array
    'You can change A, but since a copy is passed to the SUB,
    'the change will not reflect to the calling variable
End Sub

'notice the empty() to indicate that a string array is passed
Sub Teststr(b As Byte , Dl() As String)
  Dl(b) = Dl(b) + "add"
End Sub
```

7.75 LOOKDOWN

Action

Returns the index of a series of data.

Syntax

var = **LOOKDOWN**(value, label, entries)

Remarks

Var	The returned index value
Value	The value to search for
Label	The label where the data starts
entries	The number of entries that must be searched

When you want to look in BYTE series the VALUE variable must be dimensioned as a BYTE. When you want to look in INTEGER or WORD series the VALUE variable must be dimensioned as an INTEGER.

The LookDown function is the counterpart of the LookUp function.
Lookdown will search the data for a value and will return the index when the value is found. It will return –1 when the data is not found.

Lookdown() supports byte, integer, dword and long data types.

See also

LOOKUPSTR 1366 , LOOKUP 1365

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : lookdown.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : LOOKDOWN
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim Idx As Integer , Search As Byte , Entries As Byte

'we want to search for the value 3
Search = 3
'there are 5 entries in the table
Entries = 5

'lookup and return the index
Idx = Lookdown(search , Label , Entries)
Print Idx

Search = 1
Idx = Lookdown(search , Label , Entries)
Print Idx


Search = 100
Idx = Lookdown(search , Label , Entries)
Print Idx                                                  ' return -1
if not found


'looking for integer or word data requires that the search variable is
'of the type integer !
Dim Isearch As Integer
```

```
Isearch = 400
Idx = Lookdown(isearch , Label2 , Entries)
Print Idx                                                    ' return 3
End

Label:
Data 1 , 2 , 3 , 4 , 5

Label2:
Data 1000% , 200% , 400% , 300%
```

7.76 LOOKUP

Action
Returns a value from a data table based on the index.

Syntax
var = **LOOKUP**(value, label)

Remarks

Var	The returned value
Value	A value with the index of the table
Label	The label where the data starts. You may also use a variable that holds the address of a label. This way you can pass data to a sub module. When processors are used with multiple 64KB pages, the page RAMPZ will be set as well.

The maximum index value to use is 65535. The first entry will return a value of 0.
All items in the data table must be of the same data type. So you can not mix bytes and singles for example.
The data type of the return value must match the data type of the items in the table.

So this is **wrong** :

dim x as single
x=lookup(2,Dta)
dta:
data 1,2,3 **'data does not match the used single in lookup**

See also
LOOKUPSTR 1366 , DATA 1177 , LOOKDOWN 1363 , LOADWORDADR 1359

Example
```
$regfile = "m48def.dat"                                      ' specify
the used micro
$crystal = 4000000                                           ' used
crystal frequency
$baud = 19200                                                ' use baud
rate
```

```
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Dim B1 As Byte , I As Integer
B1 = Lookup(2 , Dta)
Print B1                                                    ' Prints 3
(zero based)

I = Lookup(0 , Dta2)                                        ' print 1000
Print I
End


Dta:
Data 1 , 2 , 3 , 4 , 5
Dta2:
Data 1000% , 2000%
```

7.77 LOOKUPSTR

Action

Returns a string from a table.

Syntax

var = **LOOKUPSTR**(index, label)

Remarks

Var	The string returned
Index	A value with the index of the table. The index is zero-based. That is, 0 will return the first element of the table. The maximum value is 65535.
Label	The label where the data starts. A variable with the address is accepted as well.

See also

LOOKUP 1365 , LOOKDOWN 1363 , DATA 1177

Example

```
$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space
```

```
Dim S As String * 8 , Idx As Byte
Idx = 0 : S = Lookupstr(idx , Sdata)
Print S                                                    'will print
'This'
End

Sdata:
Data "This" , "is" , "a test"
```

7.78 LOW

Action

Retrieves the least significant byte of a variable.
Sets the least significant byte of a variable

Syntax

var = **LOW**(s)
LOW(s) = value

Remarks

Var	The variable that is assigned with the LSB of var S.
S	The source variable to get the LSB from when used as a function The target variable to set the LSB of when used in an assignment.
value	The value to assign to the LSB when used as a statement

You can also assign a byte to retrieve the LSB of a Word or Long.
For example :
B = L , where B is a byte and L is a Long.

In version 2083 the LOW function can also be used to set the LSB of a variable. This for compatibility with BASCOM-8051.

See also

HIGH 1287 , HIGHW 1288

Example

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space
```

```
Dim I As Integer , Z As Byte
I = &H1001
Z = Low(i)                                                    ' is 1
End
```

7.79 MACRO

Action
This statement allow you to define a Macro.

Syntax
MACRO name
macrodef
END MACRO

Remarks

name	The name of the macro. Each macro need to have a unique name.
macrodef	The code you want to have inserted when you use the macro.

Macro's must be defined before they can be used. When a macro is defined but not used in your code, it will not be compiled. You can use $INCLUDE to include a large number of macro's.

When the compiler encounters the name of a defined macro, it will insert the defined code at that place. While it looks similar to a sub routine, there are differences. A sub routine for example is called and has a RETURN(RET).

See also
SUB 1545 , GOSUB 1284

Example
```
Macro Usb_reset_data_toggle
    Ueconx.rstdt = 1
End Macro

Macro Usb_disable_stall_handshake
 Ueconx.stallrqc = 1
End Macro

Macro Set_power_down_mode
 Smcr = 0
 Smcr = Bits(se , Sm1)
 sleep
End Macro

Usb_reset_data_toggle    ' this will insert UECONRX.RSTD=1
Set_power_down_mode      ' this will insert the following code
 Smcr = 0
 Smcr = Bits(se , Sm1)
 sleep
```

7.80 MAKEBCD

Action
Convert a variable into its BCD value.

Syntax
var1 = **MAKEBCD**(var2)

Remarks

var1	Variable that will be assigned with the converted value.
Var2	Variable that holds the decimal value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from decimal to BCD.
For printing the BCD value of a variable, you can use the BCD() function which converts a BCD number into a BCD string.

See also
MAKEDEC 1369 , BCD 822 , MAKEINT 1370

Example
```
Dim A As Byte
A = 65
Lcd A
Lowerline
Lcd Bcd(a)
A = Makebcd(a)
Lcd " " ; A
End
```

7.81 MAKEDEC

Action
Convert a BCD byte or Integer/Word variable to its DECIMAL value.

Syntax
var1 = **MAKEDEC**(var2)

Remarks

var1	Variable that will be assigned with the converted value.
var2	Variable that holds the BCD value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from BCD to decimal.

See also
MAKEBCD 1369 , MAKEBCD 1369, MAKEINT 1370

Example
```
Dim A As Byte
A = 65
Print A
Print Bcd(a)
A = Makedec(a)
Print Spc(3) ; A
End
```

7.82 MAKEINT

Action
Compact two bytes into a word or integer.

Syntax
varn = **MAKEINT**(LSB , MSB)

Remarks

Varn	Variable that will be assigned with the converted value.
LSB	Variable or constant with the LS Byte.
MSB	Variable or constant with the MS Byte.

The equivalent code is:
varn = (256 * MSB) + LSB

See also
LOW 1367 , HIGH 1287 , MAKEBCD 1369 , MAKEDEC 1369

Example
```
Dim A As Integer , I As Integer
A = 2
I = Makeint(a , 1)                                          'I = (1 *
256) + 2 = 258
End
```

7.83 MAX

Action
Returns the maximum value of a byte or word array.

Syntax
var1 = **MAX**(var2)
MAX(ar(1), m ,idx)

Remarks

var1	Variable that will be assigned with the maximum value.

var2	The first address of the array.
	The MAX statement can return the index too
Ar(1)	Starting element to get the maximum value and index of.
M	Returns the maximum value of the array.
Idx	Return the index of the array that contains the maximum value. Returns 0 if there is no maximum value.

The MIN() and MAX() functions work on BYTE and WORD arrays only.

See also

MIN 1375

Example

```
'--------------------------------------------------------------------
------------------
'name                     : minmax.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : show the MIN and MAX functions
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space


' These functions only works on BYTE and WORD arrays at the moment !!!!!

'Dim some variables
Dim Wb As Byte , B As Byte
Dim W(10) As Word                                          ' or use a
BYTE array

'fill the word array with values from 1 to 10
For B = 1 To 10
  W(b) = B
Next

Print "Max number " ; Max(w(1))
Print "Min number " ; Min(w(1))

Dim Idx As Word , M1 As Word
Min(w(1) , M1 , Idx)
Print "Min number " ; M1 ; " index " ; Idx
```

```
Max(w(1) , M1 , Idx)
Print "Max number " ; M1 ; " index " ; Idx
End
```

7.84 MEMCOPY

Action
Copies a block of memory

Syntax
bts **= MEMCOPY**(source, target , bytes [, option])

Remarks

bts	The total number of bytes copied. This must be an integer or word variable.
source	The first address of the source variable that will be copied.
target	The first address of the target variable that will be copied to.
bytes	The number of bytes to copy from "source" to "target" The range is from 1-65535.  There is not check for 0 bytes to copy. When using a variable make sure that it is not zero, since the effect will be that &HFFFF bytes will be copied.
option	An optional numeric constant with one of the following values : 1 - only the source address will be increased after each copied byte 2 - only the target address will be increased after each copied byte 3 - both the source and target address will be increased after each copied byte

By default, option 3 is used as this will copy a block of memory from one memory location to another location. But it also possible to fill an entire array of memory block with the value of 1 memory location. For example to clear a whole block or preset it with a value.
And with option 2, you can for example get a number of samples from a register like PINB and store it into an array.

MEMCOPY checks the size of the target variable and it will not overwrite data if the number of bytes is greater than the size of the target data. For example :

```
Dim tar(4) as byte, sar(8) as byte
MEMCOPY sar(1), tar(1),8
```

Even while 8 bytes are specified, the data size for tar() is 4 and thus only 4 bytes will be copied.

When you use MEMCOPY Inside a sub routine/function with passed parameters, there is no way to check the target size.
In this case, there is no check on the target size and the number of specified bytes will be moved, no matter the target data size.
This is potential unsafe when you specify too many bytes since other memory could be overwritten.

MEMCOPY could be used to clear an array quickly.

See also
MEMFILL 1374

ASM
NONE

Example
```
'-----------------------------------------------------------------------
'name                        : MEMCOPY.BAS
'copyright                   : (c) 1995-2025, MCS Electronics
'purpose                     : show memory copy function
'suited for demo             : yes
'commercial addon needed     : no
'use in simulator            : possible
'-----------------------------------------------------------------------
$regfile = "m88def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 16                                              ' default
use 10 for the SW stack
$framesize = 40

Dim Ars(10) As Byte                                        'source
bytes
Dim Art(10) As Byte                                        'target
bytes
Dim J As Byte                                              'index
For J = 1 To 10                                            'fill array
  Ars(j) = J
Next

J = Memcopy(ars(1) , Art(1) , 4)                           'copy 4
bytes

Print J ; " bytes copied"
For J = 1 To 10
  Print Art(j)
Next

J = Memcopy(ars(1) , Art(1) , 10 , 2)                      'assign them
all with element 1

Print J ; " bytes copied"
For J = 1 To 10
  Print Art(j)
Next


Dim W As Word , L As Long
W = 65511
J = Memcopy(w , L , 2)                                     'copy 2
bytes from word to long
End
```

7.85 MEMFILL

Action
Fills a block of memory with a given value

Syntax
MEMFILL source, bytes, value

Remarks

source	The first address of the source variable that will be filled. This can be a normal numeric variable or an array like ar(1)
bytes	The number of bytes to fill. The range is from 1-65535.  There is not check for 0 bytes to copy. When using a variable make sure that it is not zero, since the effect will be that &HFFFF bytes will be filled.
value	This is a byte or numeric constant with the ASCII value to use for the memory filling. To clear an array, use 0.

MEMFILL intended use is to clear an array or to fill an array quickly.

To clear an array use a value of 0.

See also
MEMCOPY 1372

ASM
CALLS _MEM_FILL in mcs.lib

Example
```
$regfile = "m1280def.dat"
$crystal = 8000000
$hwstack = 64
$swstack = 64
$framesize = 64
$baud = 19200

Config Base = 0
'array start at 0

Dim Ar(100) As Byte , X As Byte

Print "MEMFILL test"

'fill array/memory with ASCII A
```

```
Memfill Ar(1) , 10 , 65                                    'skip
the first entry so it remains 0

Do
   Print X ; "->" ; Ar(x)
   Incr X
Loop Until X = 10

End
```

7.86 MIN

Action
Returns the minimum value of a byte or word array.

Syntax
var1 = **MIN**(var2)
MIN(ar(1), m , idx)

Remarks

var1	Variable that will be assigned with the minimum value.
var2	The first address of the array.
	The MIN statement can return the index too
Ar(1)	Starting element to get the minimum value and index of
M	Returns the minimum value of the array
Idx	Return the index of the array that contains the minimum value. Returns 0 if there is no minimum value.

The MIN() ans MAX() functions work on BYTE and WORD arrays only.

See also
MAX 1370

Example
```
'-------------------------------------------------------------------
------------------
'name                     : minmax.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : show the MIN and MAX functions
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
```

```
crystal frequency
$baud = 19200                                                ' use baud
rate
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space


' These functions only works on BYTE and WORD arrays at the moment !!!!!

'Dim some variables
Dim Wb As Byte , B As Byte
Dim W(10) As Word                                            ' or use a
BYTE array

'fill the word array with values from 1 to 10
For B = 1 To 10
  W(b) = B
Next

Print "Max number " ; Max(w(1))
Print "Min number " ; Min(w(1))

Dim Idx As Word , M1 As Word
Min(w(1) , M1 , Idx)
Print "Min number " ; M1 ; " index " ; Idx

Max(w(1) , M1 , Idx)
Print "Max number " ; M1 ; " index " ; Idx
End
```

7.87 MOD

Action

Calculates the remainder of a division.

Syntax

var1 = var2 **MOD** var3

Remarks

var1	Variable that will be assigned with the modules of var2 and var3.
var2	A numeric variable to take the modules from
var3	The modulus

The MOD operation is similar to the division operation(/). But while a division returns the number of times a number can be divided, the MOD returns the remainder.

For example : 21 MOD 3 will result in 0 since 7x3=21. There will be no remainder. But 22 MOD 3 will result in 1 since 22-(7x3)=1

In BASCOM, the variable you assign determines which kind of math will be used. When you have 2 word variables you want to get the modulus from, you have to assign a word variable too.

When you assign a byte, byte math will be used.

Floating Point

When using singles or doubles, the MOD uses this equivalent code :
Dim A as single, B as single, c as single, d as single
a = 13 : b = 2.7 'sample
c = a MOD b
d = a - FIX(a / b) * b

See also

Language Fundamentals 560

Example

```
Dim L As Long , L2 As Long
For L = 1 To 1000
   L2 = L Mod 100
   If L2 = 0 Then                                          ' multiple
of 100
      Print L
   End If
Next
```

7.88 NBITS

Action

Set all except the specified bits to 1.

Syntax

Var = **NBITS**(b1 [,bn])

Remarks

Var	The BYTE/PORT variable that is assigned with the constant.
B1 , bn	A list of bit numbers that NOT must be set to 1.

While it is simple to assign a value to a byte, and there is special Boolean notation **&B** for assigning bits, the Bits() and NBits() function makes it simple to assign a few bits.

B = &B01111101 : how many zero's are there?
This would make it more readable: B = NBits(1, 7)
You can read from the code that bit 1 and bit 7 are NOT set to 1.
It does not save code space as the effect is the same.

The NBITS() function will set all bits to 1 except for the specified bits.
It can only be used on bytes and port registers.
Valid bits are in range from 0 to 7.

See Also

BITS 804

Example

```
'-----------------------------------------------------------------------
---------
'name                     : bits-nbits.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo for Bits() AND Nbits()
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'use in simulator         : possible
'-----------------------------------------------------------------------
---------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim B As Byte

'while you can use &B notation for setting bits, like B = &B1000_0111
'there is also an alternative by specifying the bits to set
B = Bits(0 , 1 , 2 , 7)                                    'set only
bit 0,1,2 and 7
Print B

'and while bits() will set all bits specified to 1, there is also Nbits
()
'the N is for NOT. Nbits(1,2) means, set all bits except 1 and 2
B = Nbits(7)                                               'do not set
bit 7
Print B
End
```

7.89 NOP

Action

This statement does noting.

Syntax

NOP

Remarks

The NOP statement will create 1 NOP assembly instruction. A NOP takes 1 machine cycle and can be used to create a small delay.
For example, at a processor clock of 1 MHz, one NOP will take exact 1 uS to execute.
You can use the ASM NOP by using : ! NOP in your code, but since using NOP is popular amongst many programmers, we introduced it as a BASCOM BASIC statement as well.

See also
BREAK 805

Example
```
NOP
```

7.90 ON INTERRUPT

Action
Execute subroutine when the specified interrupt occurs.

Syntax
ON interrupt label [NOSAVE|SAVE|SAVEALL]

Remarks

Interrupt	INT0, INT1, INT2, INT3, INT4,INT5, TIMER0 ,TIMER1, TIMER2, ADC , EEPROM , CAPTURE1, COMPARE1A, COMPARE1B,COMPARE1. Or you can use the AVR name convention: OC2 , OVF2, ICP1, OC1A, OC1B, OVF1, OVF0, SPI, URXC, UDRE, UTXC, ADCC, ERDY and ACI. The available interrupts depend on the processor.
Label	The label to jump to if the interrupt occurs. When using a label, you need to use a RETURN to resume the main program. The label may also be a sub routine. When using a sub routine, the sub routine needs to end with END SUB like any normal sub routine. This sub routine may not have parameters. So either a label must be uses like : ISR_INT0: Or you define a sub routine : Declare Sub MyISR_Int0() ON INT0 MyISR_Int0 SAVEALL
NOSAVE	When you specify NOSAVE, **no** registers are saved and restored in the interrupt routine. So when you use this option make sure to save and restore all used registers. When you omit NOSAVE all used registers will be saved. These are SREG , R31 to R16 and R11 to R0 with exception of R6,R8 and R9 . R12 – R15 are not saved. When you use floating point math in the ISR (not recommended) you must save and restore R12-R15 yourself in the ISR. My_Isr: Push R12 ' save registers Push R13

	Push R14 Push R15 Single = single + 1 ' we use FP Pop R15 ' restore registers Pop R14 Pop R13 Pop R12 RETURN  When the AVR has extended IO-space (for example ATMega48, 88 or 168, see datasheet at the end: Registersummary), the compiler uses **R23** for a number of operations. So Push and Pop R23 as well when using the NOSAVE-option when using these AVR's with extended IO-space.
SAVE	This is the default and is the same as when no parameter is provided. The most common used registers, SREG, and RAMPZ are saved and restored. Saved : SREG , R31 to R16 and R11 to R0 with exception of R6,R8 and R9. If RAMPZ exists, it will be saved as well.
SAVEALL	This will save all registers that SAVE will save, but it will also save R12-R15. You should use this option when using floating point math in the ISR.

When using a label you must return from the interrupt routine with the RETURN 1430 statement.

The first RETURN statement that is encountered that is outside a condition will generate a RETI instruction. You may have only one such RETURN statement in your interrupt routine because the compiler restores the registers and generates a RETI instruction when it encounters a RETURN statement in the ISR. All other RETURN statements are converted to a RET instruction.

While the label is supported because the old GW-BASIC supported it, it is best to use a Sub routine which you can end with End Sub.

The possible interrupt names can be looked up in the selected microprocessor register file. 2313def.dat for example shows that for the compare interrupt the name is COMPARE1. (look at the bottom of the file)

Using the editor, type ON (SPACE) and press CTRL+SPACE key to get a pop up list with possible interrupt sources.

What are interrupts good for?

An interrupt will halt your program and will jump to a specific part of your program. You can make a DO .. LOOP and poll the status of a pin for example to execute some code when the input on a pin changes.

But with an interrupt you can perform other tasks and when then pin input changes a special part of your program will be executed. When you use INPUT "Name ", v for example to get a user name via the RS-232 interface it will wait until a RETURN is

received(a byte with value 13, not the RETURN statement !).
When you have an interrupt routine and the interrupt occurs it will branch to the interrupt code and will execute the interrupt code. When it is finished it will return to the Input statement, waiting until a RETURN is entered(a byte with the return value 13).

Maybe a better example is writing a clock program. You could update a variable in your program that updates a second counter. But a better way is to use a TIMER interrupt and update a seconds variable in the TIMER interrupt handler.

There are multiple interrupt sources and it depends on the used chip/processor which are available.

To allow the use of interrupts you must set the global interrupt switch with an ENABLE INTERRUPTS statement. This only allows that interrupts can be used. You must also set the individual interrupt switches on!

ENABLE TIMER0 for example allows the TIMER0 interrupt to occur.

With the DISABLE statement you turn off the switches.

When the processor must handle an interrupt it will branch to an address at the start of flash memory. These addresses can be found in the DAT files.

The compiler normally generates a RETI instruction at these addresses so that in the event that an interrupt occurs, it will return immediately.

When you use the ON ... LABEL statement, the compiler will generate code that jumps to the specified label. The SREG and other registers are saved at the LABEL location and when the RETURN is found the compiler restores the registers and generates the RETI so that the program will continue where it was at the time the interrupt occurred.

When an interrupt is serviced no other interrupts can occur because the processor(not the compiler) will disable all interrupts by clearing the master interrupt enable bit. When the interrupt is serviced the interrupt is also cleared so that it can occur again when the conditions are met that sets the interrupt.

It is not possible to give interrupts a priority. The interrupt with the lowest address has the highest interrupt!

Finally some tips :

* when you use a timer interrupt that occurs each 10 uS for example, be sure that the interrupt code can execute in 10 uS. Otherwise you would loose time.

* it is best to set just a simple flag in the interrupt routine and to determine it's status in the main program. This allows you to use the NOSAVE option that saves stack space and program space. You only have to Save and Restore R24 and SREG in that case.

* Since you can not PUSH a hardware register, you need to load it first:

```
PUSH R24 ; since we are going to use R24 we better save it

IN r24, SREG ; get content of SREG into R24
PUSH R24 ; we can save a register
```

```
;here goes your asm code
POP R24 ;get content of SREG

OUT SREG, R24 ; save into SREG
POP R24 ; get r24 back
```

* When you call user functions or sub routines which passes variables from your interrupt, you need to enable frame protection. Use $frameprotect 637=1 to activate this protection.

 Unlike the ON VALUE statement, the ON INTERRUPT does not accept GOTO or GOSUB. The GOSUB/GOSUB tells the compiler that ON VALUE is used rather than ON INTERRUPT. Since interrupt sources are constants with an address, the compiler is happy to accept ON INT0 GOSUB which will do something entirely different than you expect.

See Also

On VALUE 1383 , ENABLE 1250, DISABLE 1240

Partial Example using label

```
Enable Interrupts
Enable Int0                                    'enable the
interrupt
On Int0 Label2 Nosave                          'jump to
label2 on INT0
Do'endless loop
  nop
Loop
End

Label2:
Dim A As Byte
If A > 1 Then
  Return                                       'generates a
RET because it is inside a condition
End If
Return                                         'generates a
RETI because it is the first RETURN
Return                                         'generates a
RET because it is the second RETURN
```

Partial Example using Sub

```
Declare Sub Label2()
Dim A As Byte
Enable Interrupts
Enable Int0                                    'enable the
interrupt
On Int0 Label2 Nosave                          'jump to
label2 on INT0
Do'endless loop
  nop
Loop
End

Sub Label2()
```

```
  If A > 1 Then
     exit sub
  Else
      gosub test
  End If
  exit sub
Test:
  print "test"
Return
End Sub                                                    'generates a
RETI
```

As you can see, using a Sub is more flexible because you can include local routines using a label/return.

7.91 ON VALUE

Action
Branch to one of several specified labels, depending on the value of a variable.

Syntax
ON var GOTO|GOSUB label1 [, label2] [,CHECK]

Remarks

Var	The numeric variable to test. This can also be a SFR such as PORTB.
label1, label2	The labels to jump to depending on the value of var.
CHECK	An optional check for the number of provided labels. When used, the maximum number of labels is 255. The check will insert code to jump over the address jump block. This will limit the number of entries.

Note that the index value is zero based. So when var is 0, the first specified label is jumped/branched.
It is important that each possible value has an associated label.

You must specify if you jump to the label or that you call the the label.
Use GOTO to jump to the label. Program flow will continue at that label.
Use GOSUB to call the label. The label must have a matching RETURN. Optional you can call a sub routine but it may not have parameters.

When there are not enough labels, the stack will get corrupted. For example :
ON value GOTO label1, label2

When the variable value has a value of two (2), there is no associated label.

You can use the optional CHECK so the compiler will check the value against the number of provided labels. When there are not enough labels for the value, there will be no GOTO or GOSUB and the next line will be executed.

See Also
ON INTERRUPT 1379 , GOTO 1286 , GOSUB 1284

ASM

The following code will be generated for a non-MEGA micro with ON value GOTO.

```
Ldi R26,$60       ; load address of variable
Ldi R27,$00 ; load constant in register
Ld R24,X
Clr R25

Ldi R30, Low(ON_1_ * 1)     ; load Z with address of the label
Ldi R31, High(ON_1_ * 1)

Add zl,r24       ; add value to Z
Adc zh,r25

Ijmp       ; jump to address stored in Z

ON_1_:

Rjmp lbl1       ; jump table
Rjmp lbl2
Rjmp lbl3
```

The following code will be generated for a non-MEGA micro with ON value GOSUB.

```
;##### On X Gosub L1 , L2
Ldi R30,Low(ON_1_EXIT * 1)
Ldi R31,High(ON_1_EXIT * 1)
Push R30 ;push return address
Push R31
Ldi R30,Low(ON_1_ * 1)      ;load table address
Ldi R31,High(ON_1_ * 1)
Ldi R26,$60
Ld R24,X
Clr R25

Add zl,r24 ; add to address of jump table
Adc zh,r25
Ijmp       ; jump !!!

ON_1_:
Rjmp L1
Rjmp L2
ON_1_EXIT:
```

As you can see a jump is used to call the routine. Therefore the return address is first saved on the stack.

Example 1

```
'-----------------------------------------------------------------------
-----------------
'name                     : ongosub.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : ON .. GOSUB/GOTO
'micro                    : Mega48
'suited for demo          : yes
```

```
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte
Input "Enter value 0-2 " , A                               'ask for
input
Rem Note That The Starting Value Begins With 0
On A Gosub L0 , L1 , L2
Print "Returned"

If Portb < 2 Then                                          'you can
also use the portvalue
  On Portb Goto G0 , G1
End If
End_prog:
End


L0:
  Print "0 entered"
Return

L1:
  Print "1 entered"
Return

L2:
  Print "2 entered"
Return

G0:
  Print "P1 = 0"
  Goto End_prog

G1:
  Print "P1 = 1"
  Goto End_prog
```

Example 2

This sample use call/sub instead of labels

```
'-------------------------------------------------------------------------
-----------------
'name                     : ongosub.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : ON .. GOSUB/GOTO
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
```

```
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Declare Sub  L0()
Declare Sub  L1()
Declare Sub  L2()

Dim A As Byte
Input "Enter value 0-2 " , A                               'ask for
input
Rem Note That The Starting Value Begins With 0
On A Gosub L0 , L1 , L2
Print "Returned"

If Portb < 2 Then                                          'you can
also use the portvalue
  On Portb Goto G0 , G1
End If
End_prog:
End


Sub L0()
  Print "0 entered"
End Sub

Sub L1()
  Print "1 entered"
End Sub

Sub L2()
  Print "2 entered"
End Sub

G0:
  Print "P1 = 0"
  Goto End_prog

G1:
  Print "P1 = 1"
  Goto End_prog
```

7.92 OPEN

Action

Opens a device.

Syntax

OPEN "device" for MODE As #channel
OPEN file FOR MODE as #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits. The format for COM1 and COM2 is : COM1: or COM2: There is no speed/baud rate parameter since the default baud rate will be used which is specified with $BAUD or $BAUD1 The format for the software UART is: COMpin:speed,8,N,stopbits[, INVERTED] Where pin is the name of the PORT-pin. Speed must be specified and stop bits can be 1 or 2. 7 bit data or 8 bit data may be used. For parity N, O or E can be used. An optional parameter ,INVERTED can be specified to use inverted RS-232. Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232. For the AVR-DOS file system, Device can also be a string or filename constant like "readme.txt" or sFileName For the Xmega, you can also open SPIC, SPID, SPIE and SPIF for SPI communication. Or for TWI you can use TWIC, TWID, TWIE or TWIF.
MODE	You can use BINARY or RANDOM for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT. For the AVR-DOS file system, MODE may be INPUT, OUTPUT, APPEND or BINARY.
Channel	The number of the channel to open. Must be a positive constant >0. For the AVR-DOS file system, the channel may be a positive constant or a numeric variable. Note that the AVD-DOS file system uses real file handles. The software UART does not use real file handles. For the Xmega UART, you may use a variable that starts with BUART. This need to be a numeric variable like a byte. Using a variable allows you to use the UART dynamic.

UART

The statements that support the device are PRINT 1501 , INPUT 1493 , INPUTHEX 1496 , INKEY 1492 and WAITKEY 1511

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

In DOS the #number is a DOS file number that is passed to low level routines. In BASCOM the channel number is only used to identify the channel but there are no file handles. So opening a channel, will not use a channel. Closing a channel is not needed for UARTS. When you do so, it is ignored. If you OPEN the channel again, you will get an error message.
So use OPEN in the begin of your program, and if you use CLOSE, use it at the end of your program.

What is the difference?

In VB you can close the channel in a subroutine like this:

```
OPEN "com1:" for binary as #1
Call test
Close #1
End

Sub test
  Print #1, "test"
End Sub
```

This will work since the file number is a real variable in the OS.
In BASCOM it will not work : the CLOSE must come after the last I/O statement:

```
OPEN "com1:" for binary as #1
Call test
End

Sub test
  Print #1, "test"
End Sub
Close #1
```

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

AVR-DOS

The AVR-DOS file system uses real file handles. This means that the CLOSE statement can be used at any place in your program just as with VB.

There are a few file modes, all inherited from VB/QB. They work exactly the same.

File mode	Description
OUTPUT	Use OUTPUT to create a file, and to write ASCII data to the file. A readme.

	txt file on your PC is an example of an ASCII file. ASCII files have a trailing CR+LF for each line you print. The PRINT statement is used in combination with OUTPUT mode.
INPUT	This mode is intended to OPEN an ASCII file and to read data only. You can not write data in this mode. The file need to exist, and must contain ASCII data. LINEINPUT can be used to read data from the file.
APPEND	APPEND mode is used on ASCII files and will not erase the file, but will append data to the end of the file. This is useful when you want to log data to a file. Opening in OUTPUT mode would erase the file if it existed. When a file does not exist yet, it will be created. This is not the case in QB/VB.
BINARY	In BINARY mode you have full read and write access to all data in the file. You can open a text file to get binary access, or you can open a binary file such as an image file. GET and PUT can be used with binary files.

 The following information from the author is for advanced users only.

GET/PUT is not supposed to work with INPUT/OUTPUT due to the rules in VB/QBASIC.
In the file CONFIG_AVR-DOS.bas (nearly at the of the file) you will find the constants
' permission Masks for file access routine regarding to the file open mode
Const cFileWrite_Mode = &B00101010 ' Binary, Append, Output
Const cFileRead_Mode = &B00100001 ' Binary, Input
Const cFileSeekSet_Mode = &B00100000 ' Binary
Const cFileInputLine = &B00100001 ' Binary, Input
Const cFilePut_Mode = &B00100000 ' Binary
Const cFileGet_Mode = &B00100000 , Binary

Where you can control, which routines can used in each file open mode. There you can see, that in standard usage GET and PUT is only allowed in BINARY.
Some time ago I wrote the Bootloader with AVR-DOS and I had the problem to keep Flash usage as low as possible. In the Bootloader I had to work with GET to read in the bytes, because the content is no ASCII text. On the other side, if you open a file in INPUT mode, you need less code. So I tested to open the File in input mode and allow to use GET in Input Mode.

I changed:
Const cFileGet_Mode = &B00100001
So GET can work in INPUT too in the BOOTLOADER.

If you switch in the constants cFileGet_Mode the last 0 to a 1, you can use GET in INPUT Open mode to. With the bootloader.bas I changed the Config_AVR-DOS.bas too. With this changed Config_AVR-DOS.bas GET can used in INPUT, with the standard CONFIG_AVR-DOS not.
This change makes no problem in code, but I think this is only something for experienced AVR-DOS user.
Whether he can use GET in INPUT mode depends only on this last bit in the constant cFileGET_Mode in the file Config_AVR-DOS.bas. This bit controls, what can be used in INPUT mode.

Xmega-SPI

The Xmega has 4 SPI interfaces. The channel is used to communicate with the different devices.

And just as with the Xmega UART, you can use the SPI dynamic. When the channel variable starts with BSPI, you can pass a variable channel.
An example you will find at CONFIG SPIx 1068
You can OPEN a SPI device only in BINARY mode.

Xmega-TWI

The Xmega has 4 TWI interfaces. The channel is used to communicate with the different devices.
You can OPEN a TWI device only in BINARY mode. Only constants are allowed for the channel.

See also

CLOSE 848 , CRYSTAL 1176, PRINT 1501, LINE INPUT 794 , LOC 795 , LOF 796 , EOF 785

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : open.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates software UART
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 10000000                                         ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


Dim B As Byte

'Optional you can fine tune the calculated bit delay
'Why would you want to do that?
'Because chips that have an internal oscillator may not
'run at the speed specified. This depends on the voltage, temp etc.
'You can either change $CRYSTAL or you can use
'BAUD #1,9610

'In this example file we use the DT006 from www.simmstick.com
'This allows easy testing with the existing serial port
'The MAX232 is fitted for this example.
'Because we use the hardware UART pins we MAY NOT use the hardware UART
'The hardware UART is used when you use PRINT, INPUT or other related
statements
```

```
'We will use the software UART.
Waitms 100

'open channel for output
Open "comd.1:19200,8,n,1" For Output As #1
Print #1 , "serial output"


'Now open a pin for input
Open "comd.0:19200,8,n,1" For Input As #2
'since there is no relation between the input and output pin
'there is NO ECHO while keys are typed
Print #1 , "Number"
'get a number
Input #2 , B
'print the number
Print #1 , B

'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
   'store in byte
   B = Inkey(#2)
   'when the value > 0 we got something
   If B > 0 Then
      Print #1 , Chr(b)                                    'print the
character
   End If
Loop Until B = 27


Close #2
Close #1


'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B


'When you dont want to use a level inverter such as the MAX-232
'You can specify ,INVERTED :
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

Example XMEGA TWI

```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-TWI.bas
'  This sample demonstrates the Xmega128A1 TWI
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40
```

```
Dim S As String * 20

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Dim N As String * 16 , B As Byte
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Config Input1 = Cr , Echo = Crlf                              ' CR is used
for input, we echo back CR and LF

Open "COM1:" For Binary As #1
'       ^^^^ change from COM1-COM8

Print #1 , "Xmega revision:" ; Mcu_revid                      ' make sure
it is 7 or higher !!! lower revs have many flaws

Const Usechannel = 1


Dim B1 As Byte , B2 As Byte
Dim W As Word At B1 Overlay




Open "twic" For Binary As #4                                  ' or use
TWID,TWIE oR TWIF
Config Twi = 100000                                           'CONFIG TWI
will ENABLE the TWI master interface
'you can also use TWIC, TWID, TWIE of TWIF

#if Usechannel = 1
   I2cinit #4
#else
  I2cinit
#endif


Do
  I2cstart
  Waitms 20
  I2cwbyte &H70                                               ' slave
address write
  Waitms 20
  I2cwbyte &B10101010                                         ' write
command
  Waitms 20
  I2cwbyte 2
  Waitms 20
  I2cstop
  Print "Error : " ; Err                                      ' show error
status

 'waitms 50
  Print "start"
  I2cstart
  Print "Error : " ; Err                                      ' show error
  I2cwbyte &H71
```

```
  Print "Error : " ; Err                                    ' show error
  I2crbyte B1 , Ack
  Print "Error : " ; Err                                    ' show error
  I2crbyte B2 , Nack
  Print "Error : " ; Err                                    ' show error
  I2cstop
  Print "received A/D : " ; W ; "-" ; B1 ; "-" ; B2
  Waitms 500                                                'wait a bit
Loop



Dim J As Byte , C As Byte , K As Byte
Dim Twi_start As Byte                                       ' you MUST
dim this variable since it is used by the lib

'determine if we have an i2c slave on the bus
For J = 0 To 200 Step 2
  Print J
  #if Usechannel = 1
    I2cstart #4
  #else
    I2cstart
  #endif

   I2cwbyte J
   If Err = 0 Then                                          ' no errors
      Print "FOUND : " ; Hex(j)
      'write some value to the pcf8574A
      #if Usechannel = 1
         I2cwbyte &B1100_0101 , #4
      #else
         I2cwbyte &B1100_0101
      #endif
      Print Err
      Exit For
   End If
   #if Usechannel = 1
       I2cstop #4
   #else
       I2cstop
   #endif
Next
#if Usechannel = 1
    I2cstop #4
#else
    I2cstop
#endif

#if Usechannel = 1
  I2cstart #4
  I2cwbyte &H71 , #4                                        'read
address
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack , #4
  Print Bin(j) ; " err:" ; Err
  I2cstop #4
#else
  I2cstart
  I2cwbyte &H71                                             'read
```

```
address
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack
  Print Bin(j) ; " err:" ; Err
  I2cstop
#endif

'try a transaction
#if Usechannel = 1
  I2csend &H70 , 255 , #4                                    ' all 1
  Waitms 1000
  I2csend &H70 , 0 , #4                                      'all 0
#else
  I2csend &H70 , 255
  Waitms 1000
  I2csend &H70 , 0
#endif
Print Err


'read transaction
Dim Var As Byte
Var = &B11111111
#if Usechannel = 1
  I2creceive &H70 , Var , 1 , 1 , #4                         ' send and
receive
  Print Bin(var) ; "-" ; Err
  I2creceive &H70 , Var , 0 , 1 , #4                         ' just
receive
  Print Bin(var) ; "-" ; Err
#else
  I2creceive &H70 , Var , 1 , 1                              ' send and
receive
  Print Bin(var) ; "-" ; Err
  I2creceive &H70 , Var , 0 , 1                              ' just
receive
  Print Bin(var) ; "-" ; Err
#endif

End
```

7.93 OUT

Action

Sends a byte to a hardware port or internal or external memory address.

Syntax

OUT address, value

Remarks

Address	The address where to send the byte to in the range of 0-FFFF hex. For Xmega which supports huge memory, the address is in range from 0-&HFFFFFF.
Value	The variable or value to output.

The OUT statement can write a value to any AVR memory location.

It is advised to use Words for the address. An integer might have a negative value and will write of course to a word address. So it will be 32767 higher as supposed. This because an integer has it's most significant bit set when it is negative.

 To write to XRAM locations you must enable the External RAM access in the Compiler Chip Options 143.

You do not need to use OUT when setting a port variable. Port variables and other registers of the micro can be set like this : PORTB = value , where PORTB is the name of the register.

 Take special care when using register variables. The address-part of the OUT statement, expects a numeric variable or constant. When you use a hardware register like for example PORTB, what will happen is that the value of PORTB will be used. Just as when you use a variable, it will use the variable value.
So when the goal is to just write to a hardware register, you need to use the normal assignment : PORTB=3

See also
INP 1315 , PEEK 1395 , POKE 1396 , SETREG 1441, GETREG 1284

Example
```
Out &H8000 , 1 'send 1 to the databus(d0-d7) at hex address 8000
End
```

7.94 PEEK

Action
Returns the content of a register.

Syntax
var = **PEEK**(address)

Remarks

Var	Numeric variable that is assigned with the content of the memory location address
Address	Numeric variable or constant with the address location.(0-31)

Peek() will read the content of a register.
Inp() can read any memory location

See also
POKE 1396 , CPEEK 1173 , INP 1315 , OUT 1394 , SETREG 1441, GETREG 1284

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : peek.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates PEEk, POKE, CPEEK, INP and OUT
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m162def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31                                             'only 32
registers in AVR
  B1 = Peek(i)                                              'get byte
from internal memory
  Print Hex(b1) ; "  ";
  'Poke I , 1                            'write a value into memory
Next
Print                                                       'new line
'be careful when writing into internal memory !!

'now dump a part ofthe code-memory(program)
For I = 0 To 255
  B1 = Cpeek(i)                                             'get byte
from internal memory
  Print Hex(b1) ; "  ";
Next
'note that you can not write into codememory!!

Out &H8000 , 1                                              'write 1
into XRAM at address 8000
B1 = Inp(&H8000)                                            'return
value from XRAM
Print B1
End
```

7.95 POKE

Action

Write a byte to an internal register.

Syntax

POKE address , value

Remarks

Address	Numeric variable with the address of the memory location to set. (0-31)
Value	Value to assign. (0-255)

See also

PEEK 1395 , CPEEK 1173 , INP 1315 , OUT 1394, SETREG 1441, GETREG 1284

Example

```
Poke 1 , 1 'write 1 to R1
End
```

7.96 POPALL

Action

Restores all registers that might be used by BASCOM.

Syntax

POPALL

Remarks

When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background.

That is why Pushall saves all used registers and POPALL restores all registers.

The SREG register is also saved/restored. The SREG register contains the processor flags and it is important to save these.
If the micro has a RAMPZ register, the RAMPZ register is saved/restored also. RAMPZ is used to address multiple pages in flash and SRAM memory.

See also

PUSHALL 1402

7.97 POWER MODE

Action

Put the micro processor in one of the supported power reserving modes.

Syntax

POWER mode

Remarks

The mode depends on the micro processor.

Some valid options are :
- IDLE
- POWERDOWN
- STANDBY
- ADCNOISE
- POWERSAVE

So for standby you would use : POWER STANDBY
It is also possible to use POWERDOWN, IDLE or POWERSAVE. These modes were/are supported by most processors. It is recommended to use the new POWER command because it allows to use more modes.

POWER has nothing to do with the POWER 758() function.

 THIS STATEMENT IS NOT RECOMMENDED. Please use CONFIG POWERMODE 1017 instead.

See also
IDLE 1313, POWERDOWN 1398 , POWERSAVE 1399

Example
```
POWER IDLE
```

7.98 POWERDOWN

Action
Put processor into power down mode.

Syntax
```
POWERDOWN
```

Remarks
In the power down mode, the external oscillator is stopped. The user can use the WATCHDOG to power up the processor when the watchdog timeout expires. Other possibilities to wake up the processor is to give an external reset or to generate an external level triggered interrupt.

 You should use the new CONFIG POWERMODE 1017 statement.

See also
IDLE 1313 , POWERSAVE 1399 , POWER mode 1397

Example
Powerdown

7.99 POWERSAVE

Action
Put processor into power save mode.

Syntax
POWERSAVE

Remarks
The POWERSAVE mode is only available in the 8535, Mega8, Mega163.

Most new chips have many options for Power down/Idle. It is advised to consult the data sheet to see if a better mode is available.

 You should use the new CONFIG POWERMODE 1017 statement.

See also
IDLE 1313, POWERDOWN 1398 , POWER mode 1397

Example
Powersave

7.100 PS2MOUSEXY

Action
Sends mouse movement and button information to the PC.

Syntax
PS2MOUSEXY X , Y, button

Remarks

X	The X-movement relative to the current position. The range is –255 to 255.
Y	The Y-movement relative to the current position. The range is –255 to 255.
Button	A variable or constant that represents the button state. 0 – no buttons pressed 1- left button pressed 2- right button pressed 4- middle button pressed You can combine these values by adding them. For example, 6 would emulate that the right and middle buttons are pressed. To send a mouse click, you need to send two ps2mouseXY statements. The first must indicate that the button is pressed, and the second must

	release the button. Ps2mouseXY 0,0,1 ' left mouse pressed PsmouseXY 0,0,0 ' left mouse released

The SENDSCAN statement could also be used.

See also
SENDSCAN 1441, CONFIG PS2EMU 1030

7.101 PULSEIN

Action
Returns the number of units between two occurrences of an edge of a pulse.

Syntax
PULSEIN var , PINX , PIN , STATE

Remarks

var	A word variable that is assigned with the result.
PINX	A PIN register like PIND
PIN	The pin number(0-7) to get the pulse time of.
STATE	May be 0 or 1. 0 means sample 0 to 1 transition. 1 means sample 1 to 0 transition.

ERR variable will be set to 1 in case of a time out. A time out will occur after 65535 unit counts. With 10 uS units this will be after 655.35 mS.

You can add a bitwait 803 statement to be sure that the PULSEIN statement will wait for the start condition. But when using the BITWAIT statement and the start condition will never occur, your program will stay in a loop.

The PULSIN statement will wait for the specified edge.

When state 0 is used, the routine will wait until the level on the specified input pin is 0. Then a counter is started and stopped until the input level gets 1.

No hardware timer is used. A 16 bit counter is used. It will increase in 10 uS units. But this depends on the XTAL. You can change the library routine to adjust the units.

PULSEIN.LIB
The full version includes a lib named pulsein.lib. It overloads the pulsein statement. This special lib allows to set a custom timeout and delay.
You need to add the following to your code :

```
  const cPulseIn_Timeout = 0     'This is the default timeout value. When you increase the value you will get a shorter time out period.
  dim bPulseIn_Delay as byte :  bPulseIn_Delay = 10   'For 10 uS units , the default is 1
```

```
$lib "pulsein.lib"  'include the lib to overload the function
```

See also
PULSEOUT 1401

ASM
The following ASM routine is called from mcs.lib
_pulse_in (calls _adjust_pin)

On entry ZL points to the PINx register , R16 holds the state, R24 holds the pin number to sample.
On return XL + XH hold the 16 bit value.

Example
```
Dim w As Word
pulsein w , PIND , 1 , 0 'detect time from 0 to 1
print w
End
```

7.102 PULSEOUT

Action
Generates a pulse on a pin of a PORT of specified period in 1uS units for 4 MHz.

Syntax
PULSEOUT PORT , PIN , PERIOD

Remarks

PORT	Name of the PORT. PORTB for example
PIN	Variable or constant with the pin number (0-7).
PERIOD	Number of periods the pulse will last. The periods are in uS when an XTAL of 4 MHz is used.

The pulse is generated by toggling the pin twice, thus the initial state of the pin determines the polarity.
The PIN must be configured as an output pin before this statement can be used.

See also
PULSEIN 1400

Example
```
Dim A As Byte
Config Portb = Output                                    'PORTB all
output pins
Portb = 0                                                'all pins 0
Do
  For A = 0 To 7
   Pulseout Portb , A , 60000                            'generate
```

```
pulse
   Waitms 250                                                    'wait a bit
  Next
Loop                                                             'loop for
ever
```

7.103 PUSHALL

Action
Saves all registers that might be used by BASCOM.

Syntax
PUSHALL

Remarks
When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background.

That is why Pushall saves all used registers. Use POPALL to restore the registers.

The saved registers are : R0-R5, R7,R10,R11 and R16-R31

The SREG register is also saved. The SREG register contains the processor flags and it is important to save these.
If the micro has a RAMPZ register, the RAMPZ register is saved too. RAMPZ is used to address multiple pages in flash and SRAM memory.
In order to save SREG a register is required. R24 was used for that till version 2086. In version 2087 R25 is used. This because R24 is used to pass data and the value would be altered by the operation. Of course an additional push/pop could be used but this take time.
So when it is essential that after PUSHALL statement all registers have the same value, you must save R25 yourself using this code :
!push R25
PUSHAL
!pop R25

See also
POPALL 1397

7.104 PUT

Action
Writes a byte to the hardware or software UART.
Writes data to a file opened in BINARY mode.

Syntax
PUT #channel, var
PUT #channel, var ,[pos] [,length]

Remarks

PUT in combination with the software/hardware UART is provided for compatibility with BASCOM-8051. It writes one byte

PUT in combination with the AVR-DOS file system is very flexible and versatile. It works on files opened in BINARY mode and you can write all data types.

#channel	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Var	The variable or variable array that will be written to the file
Pos	This is an optional parameter that may be used to specify the position where the data must be written. This must be a long variable.
Length	This is an optional parameter that may be used to specify how many bytes must be written to the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte will be written. When the variable is a word or integer, 2 bytes will be written. When the variable is a long or single, 4 bytes will be written. When the variable is a string, the number of bytes that will be written is equal to the dimensioned size of the string. DIM S as string * 10 , would write 10 bytes.

Note that when you specify the length for a string, the maximum length is 255. The maximum length for a non-string array is 65535.

Example

```
PUT #1, var
PUT #1, var , , 2  ' write 2 bytes at default position
PUT #1, var ,PS, 2  ' write 2 bytes at location storied in variable PS
```

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262, FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493, AVR-DOS File system 1794

ASM

current position	Goto new position first
Byte:	
_FilePutRange_1 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_1 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Word/Integer:	
_FilePutRange_2 Input: r24: File number X: Pointer to variable	_FilePutRange_2 Input: r24: File number X: Pointer to variable

T-Flag cleared	r16-19 (A): New position (1-based) T-Flag Set
Long/Single:	
_FilePutRange_4 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_4 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
String (<= 255 Bytes) with fixed length	
_FilePutRange_Bytes Input: r24: File number r20: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange_Bytes Input: r24: File number r20: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Array (> 255 Bytes) with fixed length	
_FilePutRange Input: r24: File number r20/21: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange Input: r24: File number r20/21: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set

Output from all kind of usage:
r25: Error Code
C-Flag on Error

Example

```
'for the binary file demo we need some variables of different types
Dim B As Byte, W As Word, L As Long, Sn As Single, Ltemp As Long
Dim Stxt As String* 10
B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt ="test"

'open the file in BINARY mode
Open "test.biN" For Binary As#2
Put#2 , B           ' write a byte
Put#2 , W           ' write a word
Put#2 , L           ' write a long
Ltemp =Loc(#2)+ 1 ' get the position of the next byte
Print Ltemp ;" LOC"' store the location of the file pointer
Print Seek(#2);" = LOC+1"

Print Lof(#2);" length of file"
Print Fileattr(#2);" file mode"' should be 32 for binary
Put#2 , Sn          ' write a single
Put#2 , Stxt        ' write a string

Flush#2             ' flush to disk
Close#2

'now open the file again and write only the single
Open "test.bin" For Binary As #2
L = 1 'specify the file position
```

```
B =Seek(#2 , L)    ' reset is the same as using SEEK #2,L
Get#2 , B          ' get the byte
Get#2 , W          ' get the word
Get#2 , L          ' get the long
Get#2 , Sn         ' get the single
Get#2 , Stxt       ' get the string
Close#2
```

7.105 RC5SEND

Action
Sends RC5 remote code.

Syntax
RC5SEND togglebit, address, command

Uses
TIMER1 or TCAx on Xtiny

Remarks

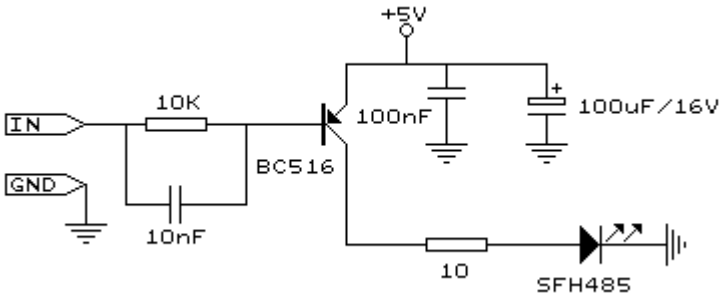
Togglebit	Make the toggle bit 0 or 32 to set the toggle bit
Address	The RC5 address
Command	The RC5 command.

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.
Look in a data sheet for the proper pin when used with a different chip.

Most audio and video systems are equipped with an infra-red remote control.
The RC5 code is a 14-bit word bi-phase coded signal.
The two first bits are start bits, always having the value 1.
The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.
Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).
An IR booster circuit is shown below:



XTINY

For XTINY platform timer TCA0 or TCA1 can be used. You can chose the WO0, WO1 or WO2 pin.
You must use the CONFIG RC5SEND directive to set up RC5SEND.

See also

CONFIG RC5 1037 , GETRC5 1278 , RC6SEND 1411

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : sendrc5.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : code based on application note from Ger
Langezaal
'micro                    : AT90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'   +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC5SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3

Dim Togbit As Byte , Command As Byte , Address As Byte

Command = 12                                                ' power on
off
Togbit = 0                                                  ' make it 0
or 32 to set the toggle bit
Address = 0
Do
   Waitms 500
```

```
   Rc5send Togbit , Address , Command
   'or use the extended RC5 send code. You can not use both
   'make sure that the MS bit is set to 1, so you need to send
   '&B10000000 this is the minimal requirement
   '&B11000000 this is the normal RC5 mode
   '&B10100000 here the toggle bit is set
   ' Rc5sendext &B11000000 , Address , Command
Loop
End
```

Example XTINY

```
'-------------------------------------------------------------------
----------------
'name                       :
avrx128da28-rc5-background-send-receive.bas
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : demonstrates GETRC5 in background
mode using timer TCB1
'                             and RC5 transmission using TCA0
'micro                      : avr128da28
'suited for demo            : no
'commercial addon needed    : yes
'-------------------------------------------------------------------
----------------

$regfile = "AVRX128da28.dat"

$crystal = 24000000
$hwstack = 40
$swstack = 40
$framesize = 64

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz

'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'set up the COM por/USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

'setup RC5 receive and specify pin to use, and the timer which
should be a timer type TCB !!!
Config Rc5 = Pind.1 , Mode = Background , Timer = Tcb1
Config Event_system = Dummy , Ch2 = Pd1 , Evsys_usertcb1capt =
Ch2
'it is very important that you also configure the event system
'you need to chose a channel that has access to the PIN used for
the RC5 input in this case PIND.1
'this pin will create an event when it changes. And you need to
```

```
connect the TCB CAPTURE user to this channel
'In this example we use TCB1 thus EVSYS_USER will be
evsys_userTCB1CAPT, and since PD1 (pin D.1) is connected to
'channel 2, we also need to select channel 2.
'Now there is a path from PIN2.1 to TCB0, capture event
'The compiler could have created this link too but then it is not
clear to the user that this event channel is used
'for this reason you need to configure it manual

'for the RC5 transmission we need a TCA0 WOx pin. This pin is
used in output mode.
'we will use WO2 which is connected to PA2. you could use config
port_mux to chose an altenative pin
'the IR diode anode is connected to the power(vcc) and the
cathode is connected to a 220 ohm resistor
' the other end of the resistor is connected to the WO2 pin,
porta.2 in this case
Config Pina.2 = Output : Porta.2 = 1                          'set
the port direction and also set the pin high

'we need this new command to select the timer (tca0 or when
available tca1) and the WO pin
'the timer is operated in frequency generation mode, 36 KHz for
RC5
Config Rc5send = Tca0 , Wo = Wo2

Dim Bcmd As Byte
Enable Interrupts
'since interrupts are used we must enable the global interrupt
switch

Print "RC5 test,REV:" ; Hex(syscfg_revid)

do
  Incr Bcmd
  Rc5send 0 , 0 , Bcmd                                        'send
RC5 code
  Waitms 500                                                  'wait
500 msec

  'this is the background mode part that receives from the IR led
or a remote control
  If _rc5_bits.4 = 1 Then                                     'this
variable is automatically created
     _rc5_bits.4 = 0
     Print "Address: " ; Rc5_address                         'auto
created variable
     Print "Command: " ; Rc5_command                         'auto
created variable
```

```
  End If
loop

End
```

7.106 RC5SENDEXT

Action
Sends extended RC5 remote code.

Syntax
RC5SENDEXT togglebit, address, command

Uses
TIMER1

Remarks

Togglebit	Make the toggle bit 0 or 32 to set the toggle bit
Address	The RC5 address
Command	The RC5 command.

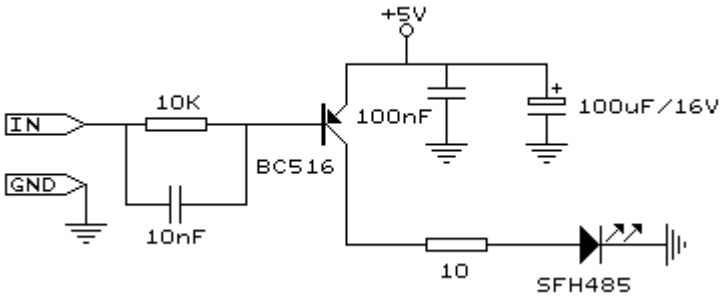
Normal RC5 code uses 2 leading bits with the value '1'. After that the toggle bit follows.
With extended RC5, the second bit is used to select the bank. When you make it 1 (the default and normal RC5) the RC5 code is compatible. When you make it 0, you select bank 0 and thus use extended RC5 code.

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.
Look in a data sheet for the proper pin when used with a different chip.

Most audio and video systems are equipped with an infra-red remote control.
The RC5 code is a 14-bit word bi-phase coded signal.
The two first bits are start bits, always having the value 1.
The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.
Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).
An IR booster circuit is shown below:



See also

CONFIG RC5 1037 , GETRC5 1278 , RC6SEND 1411

Example

```
'-----------------------------------------------------------------------
------------------
'name                       : sendrc5.bas
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : code based on application note from Ger
Langezaal
'micro                      : AT90S2313
'suited for demo            : yes
'commercial addon needed    : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'   +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC5SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3

Dim Togbit As Byte , Command As Byte , Address As Byte

Command = 12                                                ' power on
off
Togbit = 0                                                  ' make it 0
or 32 to set the toggle bit
Address = 0
Do
   Waitms 500
    ' Rc5send Togbit , Address , Command
   'or use the extended RC5 send code. You can not use both
   'make sure that the MS bit is set to 1, so you need to send
   '&B10000000 this is the minimal requirement
   '&B11000000 this is the normal RC5 mode
```

```
    '&B10100000 here the toggle bit is set
    Rc5sendExt &B11000000 , Address , Command
Loop
End
```

7.107 RC6SEND

Action
Sends RC6 remote code.

Syntax
RC6SEND togglebit, address, command

Uses
TIMER1

Remarks

Togglebit	Make the toggle bit 0 or 1 to set the toggle bit
Address	The RC6 address
Command	The RC6 command.

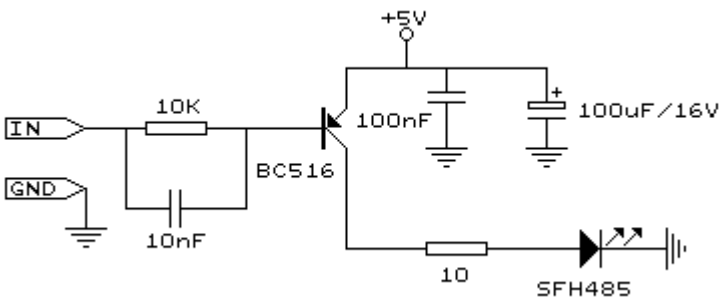
The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.
Look in a data sheet for the proper pin when used with a different chip.

Most audio and video systems are equipped with an infrared remote control.
The RC6 code is a 16-bit word bi-phase coded signal.
The header is 20 bits long including the toggle bits.
Eight system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is eight bits long, allowing up to 256 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).

An IR booster circuit is shown below:



Device	Address
TV	0
VCR	5
SAT	8
DVD	4

This is not a complete list.

Command	Value	Command	Value
Key 0	0	Balance right	26
Key 1	1	Balance left	27
Key 2-9	2-9	Channel search+	30
Previous program	10	Channel search -	31
Standby	12	Next	32
Mute/un-mute	13	Previous	33
Personal preference	14	External 1	56
Display	15	External 2	57
Volume up	16	TXT submode	60
Volume down	17	Standby	61
Brightness up	18	Menu on	84
Brightness down	19	Menu off	85
Saturation up	20	Help	129
Saturation down	21	Zoom -	246
Bass up	22	Zoom +	247
Bass down	23		
Treble up	24		
Treble down	25		

This list is by far not complete.
Since there is little info about RC6 on the net available, use code at your own risk!

See also

CONFIG RC5 1037 , GETRC5 1278 , RC5SEND 1405

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : sendrc6.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : code based on application note from Ger
Langezaal
'micro                    : AT90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
```

```
$baud = 19200                                                ' use baud
rate
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

'   +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC6SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3

Dim Togbit As Byte , Command As Byte , Address As Byte

'this controls the TV but you could use rc6send to make your DVD region
free as well :-)
'Just search the net for the codes you need to send. Do not ask me for
info please.
Command = 32                                                 ' channel
next
Togbit = 0                                                   ' make it 0
or 32 to set the toggle bit
Address = 0
Do
   Waitms 500
   Rc6send Togbit , Address , Command
Loop
End
```

7.108 READ

Action

Reads those values and assigns them to variables.

Syntax

READ var

Remarks

Var	Variable that is assigned data value.

It is best to place the DATA 1177 lines at the end of your program.

It is important that the variable is of the same type as the stored data.

See also

DATA 1177 , RESTORE 1429

Example

```
'-------------------------------------------------------------------
-----------------
'name                    : readdata.bas
'copyright               : (c) 1995-2025, MCS Electronics
```

```
'purpose                  : demo : READ,RESTORE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                          ' specify
the used micro
$crystal = 4000000                                               ' used
crystal frequency
$baud = 19200                                                    ' use baud
rate
$hwstack = 32                                                    ' default
use 32 for the hardware stack
$swstack = 10                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space

Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long
Restore Dta1                                                     'point to
stored data
For Count = 1 To 3                                               'for number
of data items
   Read B1 : Print Count ; "  " ; B1
Next

Restore Dta2                                                     'point to
stored data
For Count = 1 To 2                                               'for number
of data items
   Read A : Print Count ; "  " ; A
Next

Restore Dta3
Read S : Print S
Read S : Print S


Restore Dta4
Read L : Print L                                                 'long type


'demonstration of readlabel
Dim W As Iram Word At 8 Overlay                                  ' location
is used by restore pointer
'note that W does not use any RAM it is an overlayed pointer to the data
pointer
W = Loadlabel(dta1)                                              ' loadlabel
expects the labelname
Read B1
Print B1
End


Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%

Dta3:
```

```
Data "Hello" , "World"
'Note that integer values (>255 or <0) must end with the %-sign
'also note that the data type must match the variable type that is
'used for the READ statement

Dta4:
Data 123456789&
'Note that LONG values must end with the &-sign
'Also note that the data type must match the variable type that is used
'for the READ statement
```

7.109 READEEPROM

Action

Reads the content from the DATA EEPROM and stores it into a variable.

Syntax

READEEPROM var , address

Remarks

Var	The name of the variable that must be stored
Address	The address in the EEPROM where the data must be read from.

This statement is provided for backwards compatibility with BASCOM-8051.
You can also use the ERAM variable instead of READEEPROM :

```
Dim V as Eram Byte 'store in EEPROM
Dim B As Byte  'normal variable
B = 10
V = B   'store variable in EEPROM
B = V   'read from EEPROM
```

When you use the assignment version, the data types must be equal!
According to a data sheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset so don't use it.

You may also use ERAM variables as indexes. Like :
Dim ar(10) as Eram Byte

When you omit the address label in consecutive reads, you must use a new READEEPROM statement. It will not work in a loop:

```
Readeeprom B , Label1
Print B

Do
  Readeeprom B
  Print B Loop
Until B = 5
```

This will not work since there is no pointer maintained. The way it will work :

```
ReadEEprom B , Label1 ' specify label
ReadEEPROM B ' read next address in EEPROM
ReadEEPROM B ' read next address in EEPROM
```

OR

```
Dim Next_Read as Integer
Dim In_byte as Byte
Dim Eerom_position as Integer

Eerom_position = 20                          ' Set the start read point in eerom
For Next_Read = 1 To 5 Step 1                ' Set up the bytes to be read from
eeprom
Readeeprom In_byte , eeprom_position          ' Use a variable as the pointer to
eeprom location
Call another_sub_routine                     '
 Incr Chr_pos_font                           ' Now set pointer for next eeprom data
byte
Next
```

In the XMEGA, you need to set the mode to mapped : CONFIG EEPROM 952 = MAPPED.

See also

WRITEEEPROM 1611 , $EEPROM 631

ASM

NONE

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : eeprom2.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows how to use labels with READEEPROM
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'first dimension a variable
Dim B As Byte
Dim Yes As String * 1

'Usage for readeeprom and writeeprom :
```

```
'readeeprom var, address

'A new option is to use a label for the address of the data
'Since this data is in an external file and not in the code the eeprom
data
'should be specified first. This in contrast with the normal DATA lines
which must
'be placed at the end of your program!!

'first tell the compiler that we are using EEPROM to store the DATA
$eeprom

'the generated EEP file is a binary file.
'Use $EEPROMHEX to create an Intel Hex file usable with AVR Studio.
'$eepromhex

'specify a label
Label1:
Data 1 , 2 , 3 , 4 , 5
Label2:
Data 10 , 20 , 30 , 40 , 50

'Switch back to normal data lines in case they are used
$data

'All the code above does not generate real object code
'It only creates a file with the EEP extension

'Use the new label option
Readeeprom B , Label1
Print B                                                    'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B                                                    'prints 2

Readeeprom B , Label2
Print B                                                    'prints 10
Readeeprom B
Print B                                                    'prints 20

'And it works for writing too :
'but since the programming can interfere we add a stop here
Input "Ready?" , Yes
B = 100
Writeeeprom B , Label1
B = 101
Writeeeprom B

'read it back
Readeeprom B , Label1
Print B                                                    'prints 100
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B                                                    'prints 101
End
```

7.110 READHITAG

Action

Read HITAG RFID transponder serial number.

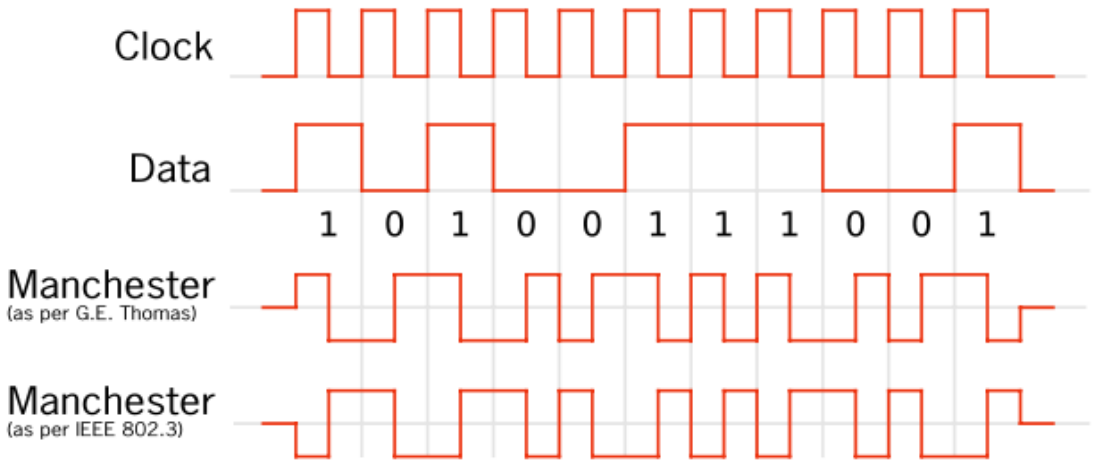
Syntax
result = **READHITAG**(var)

Remarks

result	A numeric variable that will be 0 if no serial number was read from the transponder. It will return 1 if a valid number was read.

RFID is used for entrance systems, anti theft, and many other applications where a wireless chip is an advantage over the conventional magnetic strip and chip-card. The HITAG series from Philips(NXP) is one of the oldest and best available. The HTRC110 chip is a simple to use chip that can read and write transponders. Each transponder chip has a 5 byte(40 bits) unique serial number.
The only disadvantage of the HTRC110 is that you need to sign an NDA in order to get the important documents and 8051 example code.

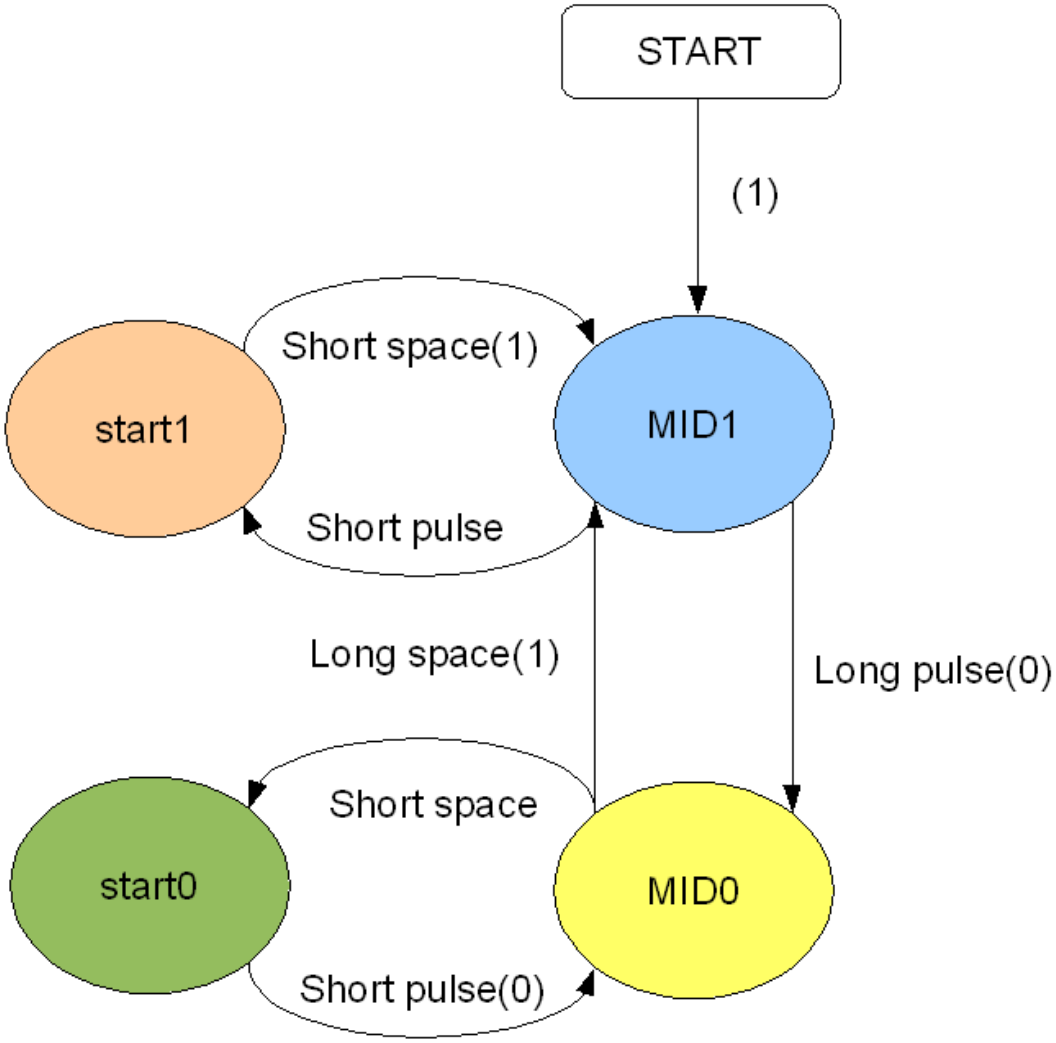
When the transponder is held before the coil of the receiver, the bits stream will be modulated with the bit values. Just like RC5, HITAG is using Manchester encoding. This is a simple and reliable method used in transmission systems.
Manchester encoding is explained very well at the Wiki Manchester page.



The image above is copied from the Wiki.

There are 2 methods to decode the bits. You can detect the edges of the bits and sample on 3/4 of the bit time.
Another way is to use a state machine. The state machine will check the length between the edges of the pulse. It will start with the assumption that there is a (1). Then it will enter the MID1 state. If the next pulse is a long pulse, we have received a (0). When it received a short pulse, we enter the start1 state. Now we need to receive a short space which indicated a (1), otherwise we have an invalid state. When we are in the MID0 state, we may receive a long space(1) or a short space. All others pulses are invalid and lead to a restart of the pulse state(START).

Have a look at the image above. Then see how it really works. We start with assuming a (1). We then receive a long pulse so we receive a (0). Next we receive a long space which is a (1). And again a long pulse which is a (0) again. Then we get a short space and we are in start1 state. We get a short pulse which is a (0) and we are back in MID0 state. The long space will be a (1) and we are in MID1 state again. etc.etc. When ever we receive a pulse or space which is not defined we reset the pulse state machine.



At 125 KHz, the bit time is 512 uS. A short pulse we define as halve a bit time which is 256 uS.
We use a 1/4 of the bit time as an offset since the pulses are not always exactly precise.
So a short bit is 128-384(256-128 - 256+128) uS. And a long bit is 384-640 uS (512-128 - 512+128).
We use TIMER0 which is an 8 bit timer available in all AVR's to determine the time.
Since most micro's have an 8 MHz internal clock, we run the program in 8 MHz. It depends on the pre scaler value of the timer, which value are used to determine the length between the edges.
You can use 64 or 256. The generated constants are : _TAG_MIN_SHORT, _TAG_MAX_SHORT , _TAG_MIN_LONG and _TAG_MAX_LONG.

We need an interrupt to detect when an edge is received. We can use the INTx for this and configure the pin to interrupt when a logic level changes. Or we can use the PIN interrupt so we can use more pins.
The sample contains both methods.
It is important that the ReadHitag() functions needs a variable that can store 5 bytes. This would be an array.
And you need to check the _TAG constants above so that they do not exceed 255.

When you set up the interrupt, you can also use it for other tasks if needed. You only

need to call the **_checkhitag** routine in the subroutine. And you need to make sure that the additional code you write does not take up too much time.

When you use the PCINT interrupt it is important to realize that other pins must be masked off. The PCMSK register may have only 1 bit enabled. Otherwise there is no way to determine which pin was changed.

EM4095

The EM4095 is similar to the HTRC110. The advantage of the EM4095 is that it has a synchronized clock and needs no setup and less pins.
The EM4095 library uses the same method as the RC5 decoding : the bit is sampled on 3/4 of the bit length. The parity handling is the same. The EM4095 decoding routine is smaller then the HTRC110 decoding library.
A reference design for the EM4095 will be available from MCS.

See also

READMAGCARD 1420 , CONFIG HITAG 970

Example

See CONFIG HITAG 970 for 2 examples.

7.111 READMAGCARD

Action

Read data from a magnetic card.

Syntax

READMAGCARD var , count , coding

Remarks

Var	A byte array the receives the data.
Count	A byte variable that returns the number of bytes read.
coding	A numeric constant that specifies if 5 or 7 bit coding is used. Valid values are 5 and 7.

There can be 3 tracks on a magnetic card.
Track 1 stores the data in 7 bit including the parity bit. This is handy to store alpha numeric data.
On track 2 and 3 the data is stored with 5 bit coding.

The ReadMagCard routine works with ISO7811-2 5 and 7 bit decoding.

The returned numbers for 5 bit coding are:

Returned number	ISO characterT
0	0
1	1
2	2
3	3

4	4
5	5
6	6
7	7
8	8
9	9
10	hardware control
11	start byte
12	hardware control
13	separator
14	hardware control
15	stop byte

Example

```
'-----------------------------------------------------------------------
------------------
'name                       : magcard.bas
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : show you how to read data from a magnetic
card
'micro                      : Mega48
'suited for demo            : yes
'commercial addon needed    : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'[reserve some space]
Dim Ar(100) As Byte , B As Byte , A As Byte

'the magnetic card reader has 5 wires
'red      - connect to +5V
'black    - connect to GND
'yellow   - Card inserted signal CS
'green    - clock
'blue     - data

'You can find out for your reader which wires you have to use by
connecting +5V
'And moving the card through the reader. CS gets low, the clock gives a
clock pulse of equal pulses
'and the data varies
'I have little knowledge about these cards and please dont contact me
about magnectic readers
'It is important however that you pull the card from the right direction
as I was doing it wrong for
```

```
'some time :-)
'On the DT006 remove all the jumpers that are connected to the LEDs

'[We use ALIAS to specify the pins and PIN register]
_mport Alias Pinb                                           'all pins
are connected to PINB
_mdata Alias 0                                              'data line
(blue) PORTB.0
_mcs Alias 1                                                'CS line
(yellow) PORTB.1
_mclock Alias 2                                             'clock line
(green) PORTB.2

Config Portb = Input                                        'we only
need bit 0,1 and 2 for input
Portb = 255                                                 'make them
high

Do
  Print "Insert magnetic card"                              'print a
message
  Readmagcard Ar(1) , B , 5                                 'read the
data
  Print B ; " bytes received"
  For A = 1 To B
    Print Ar(a);                                            'print the
bytes
  Next
  Print
Loop

'By specifying 7 instead of 5 you can read 7 bit data
```

7.112 REDO

Action

The REDO statement will restart code inside a loop.

Syntax

REDO

Remarks

REDO must be used inside a DO-LOOP, WHILE-WEND or FOR-NEXT loop.
The code jump is always inside the current loop.
Some times you want to repeat some code inside a loop. You can solve this with a GOTO and a label but use of GOTO creates hard to understand code.

DO-LOOP

DO
REDO_WILL_JUMP_TO_THIS_POINT
  some code here
  some code here
LOOP

WHILE-WEND

```
WHILE <CONDIITON>
REDO_WILL_JUMP_TO_THIS_POINT
  some code here
  some code here
WEND
```

FOR-NEXT

```
FOR VAR=START TO END
REDO_WILL_JUMP_TO_THIS_POINT
  some code here
  some code here
NEXT
```

See also

EXIT 1257 , CONTINUE 1168

Example

```
'-----------------------------------------------------------------------
---------------------------------------
'                                    REDO and CONTINUE example
'
'-----------------------------------------------------------------------
---------------------------------------
$regfile = "m128def.dat"
$hwstack = 32
$swstack = 16
$FrameSize = 24


dim b as byte
const test = 0

#if test = 0
   for b = 1 to 10
'when REDO is used, the code will continue here
      print b
      if b = 3 then
         continue                                                ' when b
becomes 3,  the code will continue at the NEXT statement
      end if
      if b = 9 then exit for
      if b = 8 then
         redo                                                 ' when b
becomes 8, the code will continue after the FOR statement, it will not
increase the variable B !
      'so in this example the loop will be forever
      end if
      print b
    'code continues here when CONTINUE is used
   next
```

```
#elseif test = 1
   b = 0
   do
      incr b
      if b = 2 then
         continue
      elseif b = 3 then
         redo
      end if
   loop until b > 5

#elseif test = 2
   b = 0
   while b < 5
      incr b
      if b = 2 then
         continue
      elseif b = 3 then
         redo
      end if
   wend
#endif
end
```

7.113 READSIG

Action

This function reads a byte from the signature area.

Syntax

var **= READSIG(**offset**)**

Remarks

Var	A byte that is assigned with the signature byte.
Offset	A byte variable or constant with an offset to the signature.

The Xmega has a number of signature bytes that are important.
For example the ADC is calibrated in the factory and the calibration data need to be loaded into the ADC registers in order to achieve 12 bit resolution.

The following offset table is copied from the Xmega128A1 definition file. It should be the same for all other Xmega chips but it is best to check it.

```
Const NVM_PROD_SIGNATURES_RCOSC2M_offset = &H00       ' RCOSC 2MHz Calibration Valu
Const NVM_PROD_SIGNATURES_RCOSC32K_offset =&H02       ' RCOSC 32kHz Calibration Val
Const NVM_PROD_SIGNATURES_RCOSC32M_offset = &H03      ' RCOSC 32MHz Calibration Val
Const NVM_PROD_SIGNATURES_LOTNUM0_offset = &H08       ' Lot Number Byte 0, ASCII
Const NVM_PROD_SIGNATURES_LOTNUM1_offset = &H09       ' Lot Number Byte 1, ASCII
Const NVM_PROD_SIGNATURES_LOTNUM2_offset = &H0A       ' Lot Number Byte 2, ASCII
Const NVM_PROD_SIGNATURES_LOTNUM3_offset = &H0B       ' Lot Number Byte 3, ASCII
Const NVM_PROD_SIGNATURES_LOTNUM4_offset = &H0C       ' Lot Number Byte 4, ASCII
Const NVM_PROD_SIGNATURES_LOTNUM5_offset = &H0D       ' Lot Number Byte 5, ASCII
Const NVM_PROD_SIGNATURES_WAFNUM_offset = &H10        ' Wafer Number
```

```
Const NVM_PROD_SIGNATURES_COORDX0_offset = &H12       ' Wafer Coordinate X Byte 0
Const NVM_PROD_SIGNATURES_COORDX1_offset = &H13       ' Wafer Coordinate X Byte 1
Const NVM_PROD_SIGNATURES_COORDY0_offset = &H14       ' Wafer Coordinate Y Byte 0
Const NVM_PROD_SIGNATURES_COORDY1_offset = &H15       ' Wafer Coordinate Y Byte 1
Const NVM_PROD_SIGNATURES_ADCACAL0_offset = &H20      ' ADCA Calibration Byte 0
Const NVM_PROD_SIGNATURES_ADCACAL1_offset = &H21      ' ADCA Calibration Byte 1
Const NVM_PROD_SIGNATURES_ADCBCAL0_offset = &H24      ' ADCB Calibration Byte 0
Const NVM_PROD_SIGNATURES_ADCBCAL1_offset = &H25      ' ADCB Calibration Byte 1
Const NVM_PROD_SIGNATURES_TEMPSENSE0_offset = &H2E    ' Temperature Sensor Calibrat
Const NVM_PROD_SIGNATURES_TEMPSENSE1_offset = &H2F    ' Temperature Sensor Calibrat
Const NVM_PROD_SIGNATURES_DACAOFFCAL_offset = &H30    ' DACA Calibration Byte 0
Const NVM_PROD_SIGNATURES_DACACAINCAL_offset = &H31   ' DACA Calibration Byte 1
Const NVM_PROD_SIGNATURES_DACBOFFCAL_offset = &H32    ' DACB Calibration Byte 0
Const NVM_PROD_SIGNATURES_DACBGAINCAL_offset = &H33   ' DACB Calibration Byte 1
```

While the XMEGA was the first processor with support of reading the signature row, all new UPDI AVR chips also have this functionality. And some plain AVR processors like the PB series (atmega328PB)
While in UPDI chips (Xtiny, megaX, AVRX) all these signature registers can be accessed by a register, the ReadSig() function can also read the register contents. ReadSig() was added for compatibility.

See also

READUSERSIG 1427

Example XMEGA

```
'-----------------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                      xm128-readsig.bas
'  This sample demonstrates how to read signature bytes
'-----------------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40

'include the following lib and code, the routines will be replaced since
they are a workaround

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8


Dim Offset As Byte , J As Byte

For J = 0 To 32
    Offset = Readsig(j) : Print J ; " - " ; Offset
Next
End
```

Example MEGA328

```
'-----------------------------------------------------------------------
----------
'name                     : m328pb.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates M328pb
'micro                    : Mega328pb
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
----------
$regfile = "m328pbdef.dat"
$crystal = 8000000
$baud = 19200
$hwstack = 40
$swstack = 40
$framesize = 40

'USART   TX    RX
'  0     D.1   D.0
'  1     B.3   B.4

'ISP programming
'MOSI-PB3     MISO-PB4   SCK-PB5
Config Clockdiv = 1
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Const Device_signature_byte1 = 0
Const Device_signature_byte2 = 2
Const Device_signature_byte3 = 4
Const Rc_oscillator_calibration = 1
Const Serial_number_byte0 = &H0E
Const Serial_number_byte1 = &H0F
Const Serial_number_byte2 = &H10
Const Serial_number_byte3 = &H11
Const Serial_number_byte4 = &H12
Const Serial_number_byte5 = &H13
Const Serial_number_byte6 = &H14
Const Serial_number_byte7 = &H15
Const Serial_number_byte8 = &H16
Const Serial_number_byte9 = &H17


Print "ID : " ; Hex(readsig(device_signature_byte1)) ; Hex(readsig(
device_signature_byte2)) ; Hex(readsig(device_signature_byte3))
```

7.114 READUSERSIG

Action
This function reads data from the User Signature Area available in the UPDI platform.

Syntax
targ = ReadUserSig(address [, bytes])

Remarks

targ	A variable that is assigned with the data byte(s)
address	The address where reading must start. This must be in the range from 0 up to the available data length - 1. So when the user signature is 32 bytes, the range is from 0-31.
bytes	The number of bytes to read. This is an optional variable. When not used the target variable data type is used to determine how many bytes must be read.

The User Signature Data area can be written by the UPDI. See also the $USER directive.

See also
READSIG 1424, $USER

Example

7.115 REM

Action
Instruct the compiler that comment will follow.

Syntax
REM or '

Remarks
You can and should comment your program for clarity and your later sanity.
You can use REM or **'** followed by your comment.
All statements after REM or ' are treated as comments so you cannot use statements on the same line after a REM statement.

Block comments can be used too:

'(start block comment
print "This will not be compiled
') end block comment

Example
```
Rem TEST.BAS version 1.00

Print A ' " this is comment : PRINT " Hello "

           ^ - - - This Will Not Be Executed!
```

7.116 RESET

Action
Reset a bit to zero.

Syntax
RESET bit
RESET var.x
RESET var
RESET micro
RESET watchdog

Remarks

Bit	Bit or Boolean variable.
Var	A byte, integer, word, long or dword variable.
X	Bit of variable to clear. Valid values are : 0-7 (byte, registers), 0-15 (Integer/Word) and (0-31) for a Long/ Dword
watchdog	This will reset the Watchdog timer.
micro	This will reset the processor. Depending on the AVR platform different methods will be used: - XTINY/MEGAX/AVR : Reset controller is used - XMEGA : Reset controller is used - AVR : a jump to address 0 is used. This will not reset hardware registers and is not a real reset.

You can also use the constants from the definition file to set or reset a bit.
RESET PORTB.PB7 'will reset pin 7 of portB. This because PB7 is a defined constant in the definition file.

When the bit is not specified, bit 0 will be cleared.

See also
SET 1438 , TOGGLE 1595

Example
SEE SET 1438

7.117 RESTORE

Action
Allows READ to reread values in specified DATA statements by setting data pointer to beginning of data statement.

Syntax
RESTORE label

Remarks

label	The label of a DATA statement.

See also
DATA 1177 , READ 1413 , LOOKUP 1365

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : readdata.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : READ,RESTORE
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long
Restore Dta1                                               'point to
stored data
For Count = 1 To 3                                         'for number
of data items
   Read B1 : Print Count ; "  " ; B1
Next

Restore Dta2                                               'point to
stored data
For Count = 1 To 2                                         'for number
of data items
   Read A : Print Count ; "  " ; A
Next
```

```
Restore Dta3
Read S : Print S
Read S : Print S


Restore Dta4
Read L : Print L                                                    'long type


'demonstration of readlabel
Dim W As Iram Word At 8 Overlay                                     ' location
is used by restore pointer
'note that W does not use any RAM it is an overlayed pointer to the data
pointer
W = Loadlabel(dta1)                                                 ' loadlabel
expects the labelname
Read B1
Print B1
End


Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%

Dta3:
Data "Hello" , "World"
'Note that integer values (>255 or <0) must end with the %-sign
'also note that the data type must match the variable type that is
'used for the READ statement

Dta4:
Data 123456789&
'Note that LONG values must end with the &-sign
'Also note that the data type must match the variable type that is used
'for the READ statement
```

7.118 RETURN

Action
Return from a subroutine.

Syntax
RETURN

Remarks
Subroutines must be ended with a related RETURN statement.
Interrupt subroutines must also be terminated with the Return statement.

See also
GOSUB 1284

Example
```
'-----------------------------------------------------------------------
```

```
------------------
'name                     : gosub.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: GOTO, GOSUB and RETURN
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Goto Continue
Print "This code will not be executed"

Continue:                                                  'end a label
with a colon
Print "We will start execution here"
Gosub Routine
Print "Back from Routine"
End

Routine:                                                   'start a
subroutine
  Print "This will be executed"
Return                                                     'return from
subroutine
```

7.119 RND

Action

Returns a random number.

Syntax

var = **RND**(limit)

Remarks

Limit	Word that limits the returned random number.
Var	The variable that is assigned with the random number.

The RND() function returns an Integer/Word and needs an internal storage of 2 bytes. (___RSEED). Each new call to Rnd() will give a new positive random number.

 Notice that it is a software based generated number. And each time you will restart your program the same sequence will be created.

You can use a different SEED value by dimensioning and assigning ___RSEED yourself:
Dim ___rseed as word : ___rseed = 10234
Dim I as word : I = rnd(10)

When your application uses a timer you can assign ___RSEED with the timer value. This will give a better random number.

See also
CONFIG RND 1044

Example
```
'-------------------------------------------------------------------------
------------------
'name                     : rnd.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : RND() function
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim I As Word                                              ' dim
variable
Do
  I = Rnd(40)                                              'get random
number (0-39)
  Print I                                                  'print the
value
  Wait 1                                                   'wait 1
second
Loop                                                       'for ever
End
```

7.120 ROTATE

Action
Rotate all bits one place to the left or right.

Syntax
ROTATE var , LEFT/RIGHT[, shifts]

Remarks

Var	Byte, Integer/Word or Long variable.
Shifts	The number of shifts to perform.

The ROTATE statement rotates all the bits in the variable to the left or right. All bits are preserved so no bits will be shifted out of the variable.
This means that after rotating a byte variable with a value of 1, eight times the variable will be unchanged.
When you want to shift out the MS bit or LS bit, use the SHIFT statement.

See also

SHIFT 1447 , SHIFTIN 1449 , SHIFTOUT 1453

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : rotate.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : example for ROTATE and SHIFT statement
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'dimension some variables
Dim B As Byte , I As Integer , L As Long

'the shift statement shift all the bits in a variable one
'place to the left or right
'An optional paramater can be provided for the number of shifts.
'When shifting out then number 128 in a byte, the result will be 0
'because the MS bit is shifted out

B = 1
Shift B , Left
Print B
'B should be 2 now

B = 128
Shift B , Left
Print B
'B should be 0 now

'The ROTATE statement preserves all the bits
```

```
'so for a byte when set to 128, after a ROTATE, LEFT , the value will
'be 1

'Now lets make a nice walking light
'First we use PORTB as an output
Config Portb = Output
'Assign value to portb
Portb = 1
Do
   For I = 1 To 8
     Rotate Portb , Left
     'wait for 1 second
     Wait 1
   Next
   'and rotate the bit back to the right
   For I = 1 To 8
     Rotate Portb , Right
     Wait 1
   Next
Loop
End
```

7.121 SEEK

Action

Function: Returns the position of the next Byte to be read or written
Statement: Sets the position of the next Byte to be read or written

Syntax

Function: NextReadWrite = **SEEK** (#bFileNumber)
Statement: **SEEk** #bFileNumber, NewPos

Remarks

bFileNumber	A byte holding the File number, which identifies a previous opened file
NextReadWrite	A Long Variable, which is assigned with the Position of the next Byte to be read or written (1-based). In case of an error, 0 is returned.
NewPos	A Long variable that holds the new position the file pointer must be set to.

This function returns the position of the next Byte to be read or written.
Check DOS-Error in variable gbDOSError in case of an error, when the function returns a zero.

SEEK only works on files opened in BINARY mode. The SEEK() function returns 1 for an opened file since this is the start of the file. Once you write data to the file, SEEK() will return the updated location.

The statement also returns an error in the gbDOSerror variable in the event that an error occurs.
You can for example not set the file position behind the file.

In VB the file is filled with 0 bytes when you set the file pointer behind the size of the file. For embedded systems this does not seem a good idea.

Seek and Loc seems to do the same function, but take care : the seek function will

return the position of the next read/write, while the Loc function returns the position of the last read/write. You may say that Seek = Loc+1.

 In QB/VB you can use seek to make the file bigger. When a file is 100 bytes long, setting the file pointer to 200 will increase the file with 0 bytes. By design this is not the case in AVR-DOS.

See also

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , FILELEN 788 , WRITE 801 , INPUT 1493

ASM

Function Calls	_FileSeek	
Input	r24: filenumber	X: Pointer to Long-variable, which gets the result
Output	r25: Errorcode	C-Flag: Set on Error

Statement Calls	_FileSeekSet	
Input	r24: filenumber	X: Pointer to Long-variable with the position
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

```
Open "test.biN"for Binary As #2
Put#2 , B                                                    ' write a
byte
Put#2 , W                                                    ' write a
word
Put#2 , L                                                    ' write a
long
Ltemp = Loc(#2) + 1                                          ' get the
position of the next byte
Print Ltemp ; " LOC"                                         ' store the
location of the file pointer
Print Seek(#2) ; " = LOC+1"
Close #2


'now open the file again and write only the single
Open "test.bin" For Binary As #2
Seek#2 , Ltemp                                               ' set the
filepointer
Sn = 1.23                                                    ' change the
single value so we can check it better
Put #2 , Sn,1                                                ' specify
the file position
Close #2
```

Example2

```
'-----------------------------------------------------------------
--------------
'                          simulate-AVR-DOS.bas
'  simulate AVR-DOS using virtual XRAM drive
'
'-----------------------------------------------------------------
--------------
$regfile = "M128def.dat"
$crystal = 16000000
' Adjust HW Stack, Soft-Stack and Frame size to 128 minimum
each!!!
$hwstack = 128 : $swstack = 128 : $framesize = 128
$xramsize = &H10000
'specify 64KB of XRAM for the file system
$sim                                                        'for
simulation only !
$baud = 19200

Config Clock = Soft
Enable Interrupts
Config Date = Mdy , Separator = Dot

Dim Btemp1 As Byte , Battr1 As Byte , Battr2 As Byte
$include "Config_XRAMDrive.bas"                             '
Does drive init too
$include "Config_AVR-DOS.BAS"

Print "Wait for Drive"
If Gbdriveerror = 0 Then
   Print "Init File System ... ";
   Btemp1 = Initfilesystem(1)                               '
Partition 1
                                          ' use 0 for drive
without Master boot record
   If Btemp1 <> 0 Then
      Print "Error: " ; Btemp1 ; " at Init file system"
   Else
      Print " OK"
      Print "Filesystem: " ; Gbfilesystem
      Print "FAT Start Sector: " ; Glfatfirstsector
      Print "Root Start Sector: " ; Glrootfirstsector
      Print "Data First Sector: " ; Gldatafirstsector
      Print "Max. Cluster Nummber: " ; Glmaxclusternumber
      Print "Sectors per Cluster: " ; Gbsectorspercluster
      Print "Root Entries: " ; Gwrootentries
      Print "Sectors per FAT: " ; Glsectorsperfat
```

```
      Print "Number of FATs: " ; Gbnumberoffats
   End If
Else
   Print "Error during Drive Init: " ; Gbdriveerror
End If

Dim Lpos As Long
Open "test.bin" For Binary As #11
Print Seek(#11)                                          ' 1
Put #11 , Btemp1
Print Seek(#11)                                          ' 2
Lpos = Lof(#11) + 1                                      '
1+1=2
Seek #11 , Lpos
Put #11 , Btemp1
Print Seek(#11)                                          '3
Close #11

End
```

7.122 SELECT-CASE-END SELECT

Action

Executes one of several statement blocks depending on the value of an expression.

Syntax

SELECT CASE var
CASE test1 : statements
[**CASE** test2 : statements]
CASE ELSE : statements
END SELECT

Remarks

Var	Variable to test the value of
Test1	Value to test for.
Test2	Value to test for.

You can test for conditions to like:

```
CASE IS > 2 :
```

Another option is to test for a range :

```
CASE 2 TO 5 :
```

See also

IF THEN 1313

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : case.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates SELECT CASE statement
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space


Dim I As Byte                                              'dim
variable
Dim S As String * 5 , Z As String * 5

Do

  Input "Enter value (0-255) " , I
  Select Case I
    Case 1 : Print "1"
    Case 2 : Print "2"
    Case 3 To 5 : Print "3-5"
    Case Is >= 10 : Print ">= 10"
    Case Else : Print "Not in Case statement"
  End Select
Loop
End

'note that a Boolean expression like > 3 must be preceded
'by the IS keyword
```

7.123 SET

Action

Set a bit to the value one.

Syntax

SET bit
SET var.x
SET var

Remarks

Bit	Bit or Boolean variable.
Var	A byte, integer, word or long variable.
X	Bit of variable to set. Valid values are : 0-7 (byte, registers), 0-15 (Integer/Word) and (0-31) for a Long

When the bit is not specified, bit 0 will be set.
Also notice that the bit range is 0-255. Using a larger value on a variable will overwrite a different variable !
When you need an array of say 128 bits you can use code like this : dim ar(32) as long
You can index these variables like : SET ar(1).127 , in this case you write only to the memory of the intended variable.

See also

RESET 1428 , TOGGLE 1595

Example

```
'-----------------------------------------------------------------------
---------
'name                     : boolean.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: AND, OR, XOR, NOT, BIT, SET, RESET and
MOD
'suited for demo          : yes
'commercial add on needed : no
'use in simulator         : possible
'-----------------------------------------------------------------------
---------
'This very same program example can be used in the Help-files for
'      AND, OR, XOR, NOT, BIT, SET, RESET and MOD


$baud = 19200
$crystal = 16000000
$regfile = "m32def.dat"

$hwstack = 40
$swstack = 20
$framesize = 20

Dim A As Byte , B1 As Byte , C As Byte
Dim Aa As Bit , I As Integer

A = 5 : B1 = 3                                          ' assign
values
C = A And B1                                            ' and a with
b
Print "A And B1 = " ; C                                 ' print it:
result = 1

C = A Or B1
Print "A Or B1 = " ; C                                  ' print it:
result = 7

C = A Xor B1
```

```
Print "A Xor B1 = " ; C                                    ' print it:
result = 6

A = 1
C = Not A
Print "c = Not A " ; C                                     ' print it:
result = 254
C = C Mod 10
Print "C Mod 10 = " ; C                                    ' print it:
result = 4


If Portb.1 = 1 Then
  Print "Bit set"
Else
  Print "Bit not set"
End If                                                     'result =
Bit not set

Aa = 1                                                     'use this or
..
Set Aa                                                     'use the set
statement
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                     'result =
Bit set (aa=1)

Aa = 0                                                     'now try 0
Reset Aa                                                   'or use
reset
If Aa = 1 Then
  Print "Bit set (aa=1)"
Else
  Print "Bit not set(aa=0)"
End If                                                     'result =
Bit not set(aa=0)

C = 8                                                      'assign
variable to &B0000_1000
Set C                                                      'use the set
statement without specifying the bit
Print C                                                    'print it:
result = 9 ; bit0 has been set

B1 = 255                                                   'assign
variable
Reset B1.0                                                 'reset bit 0
of a byte variable
Print B1                                                   'print it:
result = 254 = &B11111110

B1 = 8                                                     'assign
variable to &B00001000
Set B1.0                                                   'set it
Print B1                                                   'print it:
result = 9 = &B00001001
End
```

7.124 SETREG

Action
Writes a byte value to an internal register.

Syntax
SETREG Reg , value

Remarks
Most AVR chips have 32 registers named R0-R31. Registers R16-R31 can be assigned directly. Register R0-R15 do not accept this.
In some cases you might want to write to the internal registers. While you can include some ASM code directly, you can also use the BASIC SETREG statment.

Reg	The register name : R0-R31 or a register definition.
Value	A constant or byte value to assign to the register.

PEEK and POKE work with an address. And will return a HW register on the Xmega since Xmega has a different address map.
GetReg and SetReg will read/write registers on all AVR processors.

Internally the compiler will use R24 if you write a constant to register R0-R15 :
For example :
Setreg R0 , 1

Compiles into:
```
Ldi R24,$01
Mov R0, R24
```

Setreg R31 , 1
Compiles into:
```
Ldi R31,$01
```

 In version 2078, all internal registers (R0-R31) are made available as normal BYTE variables. This means that you can simply assign or read a register from basic : Rx=value.
This is more convenient than using SETREG and GETREG.

See also
GETREG 1284 , PEEK 1395 , POKE 1396

Example
```
Setreg R16,&HFF
```

7.125 SENDSCAN

Action
Sends scan codes to the PC.

Syntax
SENDSCAN label

Remarks

Label	The name of the label that contains the scan codes.

The SENDSCAN statement can send multiple scan codes to the PC.
The label is used to specify the start of the scan codes. The first byte specifies the number of bytes that follow.

The following table lists all mouse scan codes.

Emulated Action	Data sent to host
Move up one	08,00,01
Move down one	28,00,FF
Move right one	08,01,00
Move left one	18,FF,00
Press left button	09,00,00
Release left button	08,00,00
Press middle button	0C,00,00
Release middle button	08,00,00
Press right button	0A,00,00
Release right button	08,00,00

To emulate a left mouse click, the data line would look like this:

```
DATA 6 , &H09, &H00, &H00, &H08 , &H00, &H00
      ^ send 6 bytes
            ^ left click
                              ^ release
```

See also
PS2MOUSEXY 1399 , CONFIG PS2EMU 1030

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : ps2_emul.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : PS2 Mouse emulator
'micro                    : 90S2313
'suited for demo          : NO, commercial addon needed
'commercial addon needed  : yes
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
```

```
$hwstack = 32                                                   ' default
use 32 for the hardware stack
$swstack = 10                                                   ' default
use 10 for the SW stack
$framesize = 40                                                 ' default
use 40 for the frame space

$lib "mcsbyteint.lbx"                                           ' use
optional lib since we use only bytes

'configure PS2 pins
Config Ps2emu = Int1 , Data = Pind.3 , Clock = Pinb.0
'                 ^------------------------ used interrupt
'                           ^------------ pin connected to DATA
'                                     ^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin


Waitms 500                                                      ' optional
delay

Enable Interrupts                                               ' you need
to turn on interrupts yourself since an INT is used

Print "Press u,d,l,r,b, or t"
Dim Key As Byte
Do
    Key = Waitkey()                                             ' get key
from terminal
    Select Case Key
      Case "u" : Ps2mousexy 0 , 10 , 0                          ' up
      Case "d" : Ps2mousexy 0 , -10 , 0                         ' down
      Case "l" : Ps2mousexy -10 , 0 , 0                         ' left
      Case "r" : Ps2mousexy 10 , 0 , 0                          ' right
      Case "b" : Ps2mousexy 0 , 0 , 1                           ' left
button  pressed
                 Ps2mousexy 0 , 0 , 0                           ' left
button released
      Case "t" : Sendscan Mouseup                               ' send a
scan code
      Case Else
    End Select
Loop


Mouseup:
Data 3 , &H08 , &H00 , &H01                                     ' mouse up
by 1 unit
```

7.126 SENDSCANKBD

Action

Sends keyboard scan codes to the PC.

Syntax

SENDSCANKBD label | var

Remarks

Label	The name of the label that contains the scan codes.
var	The byte variable that will be sent to the PC.

The SENDSCANKBD statement can send multiple scan codes to the PC.
The label is used to specify the start of the scan codes. The first byte specifies the number of bytes that follow.

You can also send the content of a variable. This way you can send dynamic information.
You need to make sure you send the make and break codes.

The following tables lists all scan codes.

AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

http://panda.cs.ndsu.nodak.edu/~achapwes

KEY	MAKE	BREAK		KEY	MAKE	BREAK		KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0, F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0, F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0, F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0, F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0, F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73

Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1, F0,14, F0,77	-NONE-				

ACPI Scan Codes

Key	Make Code	Break Code
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

Windows Multimedia Scan Codes

Key	Make Code	Break Code
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48

Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favorites	E0, 18	E0, F0, 18

To emulate volume up, the data line would look like this:

DATA 5 , &HE0, &H32, &HE0, &HF0 , &H32
^ send 5 bytes
^ volume up

See also

CONFIG ATEMU

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : ps2_kbdemul.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : PS2 AT Keyboard emulator
'micro                    : 90S2313
'suited for demo          : no, ADD ON NEEDED
'commercial addon needed  : yes
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

$lib "mcsbyteint.lbx"                                       ' use
optional lib since we use only bytes

'configure PS2 AT pins
Enable Interrupts                                           ' you need
to turn on interrupts yourself since an INT is used
Config Atemu = Int1 , Data = Pind.3 , Clock = Pinb.0
'                  ^------------------------ used interrupt
'                               ^----------- pin connected to DATA
'                                         ^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin
```

```
Waitms 500                                                    ' optional
delay

'rcall _AT_KBD_INIT
Print "Press t for test, and set focus to the editor window"
Dim Key2 As Byte , Key As Byte
Do
    Key2 = Waitkey()                                          ' get key
from terminal
    Select Case Key2
      Case "t" :
      Waitms 1500
      Sendscankbd Mark                                        ' send a
scan code
      Case Else
    End Select
Loop
Print Hex(key)

Mark:                                                         ' send mark
Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 , &H2D ,
 &H42 , &HF0 , &H42
'    ^ send 12 bytes
'           m                   a                   r
 k
```

7.127 SHIFT

Action

Shift all bits one place to the left or right.

Syntax

SHIFT var , LEFT/RIGHT[, shifts] [,SIGNED]

Remarks

Var	Byte, Integer, Word, Long, Dword or Single variable.
Shifts	The number of shifts to perform.
signed	An option that only works with right shifts. It will preserve the sign bit which otherwise would be cleared by the first shift.

The SHIFT statement rotates all the bits in the variable to the left or right.

When shifting LEFT the most significant bit, will be shifted out of the variable. The LS bit becomes zero. Shifting a variable to the left, multiplies the variable with a value of two.

When shifting to the RIGHT, the least significant bit will be shifted out of the variable. The MS bit becomes zero. Shifting a variable to the right, divides the variable by two. Use the SIGNED parameter to preserve the sign.

A Shift performs faster than a multiplication or division.

See also

ROTATE 1432 , SHIFTIN 1449 , SHIFTOUT 1453

Example

```
'-------------------------------------------------------------------------
------------------
'name                     : shift.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : example for SHIFTIN and SHIFTOUT statement
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim L As Long

Clock Alias Portb.0
Output Alias Portb.1
Sin Alias Pinb.2                                           'watch the
PIN instead of PORT

'shiftout pinout,pinclock, var,parameter [,bits , delay]
' value for parameter :
'  0 - MSB first ,clock low
'  1 - MSB first,clock high
'  2 - LSB first,clock low
'  3 - LSB first,clock high
'The bits is a new option to indicate the number of bits to shift out
'For a byte you should specify 1-8 , for an integer 1-16 and for a long
1-32
'The delay is an optional delay is uS and when used, the bits parameter
must
'be specified too!

'Now shift out 9 most significant bits of the LONG variable L
Shiftout Output , Clock , L , 0 , 9



'shiftin pinin,pinclock,var,parameter [,bits ,delay]
'  0 - MSB first ,clock low  (4)
'  1 - MSB first,clock high  (5)
'  2 - LSB first,clock low   (6)
'  3 - LSB first,clock high  (7)

'To use an external clock, add 4 to the parameter
'The shiftin also has a new optional parameter to specify the number of
bits

'The bits is a new option to indicate the number of bits to shift out
'For a byte you should specify 1-8 , for an integer 1-16 and for a long
```

```
1-32
'The delay is an optional delay is uS and when used, the bits parameter
must
'be specified too!


'Shift in 9 bits into a long
Shiftin Sin , Clock , L , 0 , 9
'use shift to shift the bits to the right place in the long
Shift L , Right , 23
End
```

7.128 SHIFTIN

Action

Shifts a bit stream into a variable.

Syntax

SHIFTIN pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as an input.PINB.2 for example
Pclock	The port pin which generates the clock.
Var	The variable that is assigned. The existing value is not preserved. For example when you shiftin 3 bits, the whole byte will be replaced with the 3 bits. See CONFIG SHIFTIN for other SHIFTIN behaviour.
Option	Option can be : 0 – MSB shifted in first when clock goes low 1 – MSB shifted in first when clock goes high 2 – LSB shifted in first when clock goes low 3 – LSB shifted in first when clock goes high Adding 4 to the parameter indicates that an external clock signal is used for the clock. In this case the clock will not be generated. So using 4 will be the same a 0 (MSB shifted in first when clock goes low) but the clock must be generated by an external signal. 4 – MSB shifted in first when clock goes high with ext. clock 5 – MSB shifted in first when clock goes low with ext. clock 6 – LSB shifted in first when clock goes high with ext. clock 7 – LSB shifted in first when clock goes low with ext. clock
Bits	Optional number of bits to shift in. Maximum 255. The number of bits is automatic loaded depending on the used variable. For a long for example which is 4 bytes long, 32 will be loaded.
Delay	Optional delay in uS.

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.
When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur. For a Long and Single 32 shifts will occur.

The SHIFTIN routine can be used to interface with all kind of chips.

The PIN is normally connected with the output of chip that will send information.

The PCLOCK pin can be used to clock the bits as a master, that is the clock pulses will be generated. Or it can sample a pin that generates these pulses.

The VARIABLE is a normal BASIC variable. And may be of any type except for BIT. The data read from the chip is stored in this variable.

The OPTIONS is a constant that specifies the direction of the bits. The chip that outputs the data may send the LS bit first or the MS bit first. It also controls on which edge of the clock signal the data must be stored.

When you add 4 to the constant you tell the compiler that the clock signal is not generated but that there is an external clock signal.
The number of bits may be specified. You may omit this info. In that case the number of bits of the element data type will be used.
The DELAY normally consists of 2 NOP instructions. When the clock is too fast you can specify a delay time(in uS).

SHIFTIN with option NEW

The new option CONFIG SHIFTIN 1060=NEW , will change the behaviour of the SHIFTIN statement.
When using this option, it will work for all SHIFTIN statements. The SHIFTIN will work more like the normal SHIFT statement. Bits are shifted from left to right or right to left.

The new SHIFTIN can preserve the value/bits when shifting in bits.
For example when the value of a word is &B101 and you shift in 3 bits with value &B111, the resulting value will be &B101**111**. When you **not** want to preserve the value, you can add a value of **8** to the parameter. When you add a value of **16**, the value will also not be preserved, but then the value will be cleared initially. You would only need this when shifting in less 8 bits then the size of the variable.
Another important difference is that the new SHIFTIN can only SHIFTIN a maximum of 8 bytes. For quick operation, register R16-R23 are used. You may specify the number of bits to shiftin. This may be a variable too. When you shiftin a value into a Word, the number of bits is automatic loaded with 16. This is true for all numeric data types.

Some of the code is stored in the MCS library. While this reduces code when SHIFTIN is used multiple times, it has the drawback that the code is written for 8 bytes and thus is not optimal for shifting in less bytes.
You can choose to generate a part of the library code instead. Add a value of 32 to the parameter to do so.
Another new option is not to set the initial pin state for the clock and input pin. By default the clock pin is made an input or output, depending on the external clock option. And the clock is set to an initial state when no external clock is used.
When you want to use shiftin after a shiftout, you might not want the level to change. In this case, add 64 to the parameter.

Pin	The port pin which serves as an input.PINB.2 for example
Pclock	The port pin which generates the clock. An external signal can also be used for the clock. In that case, the pin is used in input mode.
Var	The variable that is assigned. The existing value is preserved. With some additional constants which you can add to the option parameter, you can influence the behaviour : - 8 - Do NOT preserve the value. This saves code.

	-16 - Do not preserve value, but clear the value before shifting in the bits
Option	A constant which can be one of the following values : 0 – MS bit shifted in first when clock goes low 1 – MS bit shifted in first when clock goes high 2 – LS bit shifted in first when clock goes low 3 – LS bit shifted in first when clock goes high Adding 4 to the parameter indicates that an external clock signal is used for the clock. In this case the clock will not be generated. So using 4 will be the same a 0 (MSB shifted in first when clock goes low) but the clock must be generated by an external signal. 4 – MSB shifted in first when clock goes high with ext. clock 5 – MSB shifted in first when clock goes low with ext. clock 6 – LSB shifted in first when clock goes high with ext. clock 7 – LSB shifted in first when clock goes low with ext. clock Add a value of 8 to the option, so the existing variable will not be preserved. Add a value of 16 to the option to clear the variable first. Add a value of 32 to the option to generate code instead of using the lib code. Add a value of 64 to the option when you do not want the clock and input pin data direction and state want to be set. For example, when using SHIFTIN after a SHIFTOUT statement. Example : Shiftin Pind.3 , Portd.4 , W , **2 + 32 + 16** , 3
Bits	Optional number of bits to shift in. Maximum 64. The number of bits is automatic loaded depending on the used variable. For a long for example which is 4 bytes long, 32 will be loaded. You can use a constant or variable.
Delay	Optional delay in uS. When not specified, 2 nops are used. The delay is intended to slow down the clock frequency.

The initial state for the clock depends on the option. For option 1 and 3, it will be low. For option 0 and 2 it will be high.
Thus for example option 2 will set the clock pin high. Then the clock is brought low and the data is sampled/stored. After this the clock is made high again. This means when ready, the clock pin will be in the same state as the initial state.

See also

SHIFTOUT 1453 , SHIFT 1447

Example

```
'-------------------------------------------------------------------------
-----------------
'name                          : shift.bas
'copyright                     : (c) 1995-2025, MCS Electronics
'purpose                       : example for SHIFTIN and SHIFTOUT statement
'micro                         : Mega48
```

```
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                          ' specify
the used micro
$crystal = 4000000                                               ' used
crystal frequency
$baud = 19200                                                    ' use baud
rate
$hwstack = 32                                                    ' default
use 32 for the hardware stack
$swstack = 10                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space

Dim L As Long

clock Alias Portb.0
Output Alias Portb.1
sinp Alias Pinb.2                                                 'watch the
PIN instead of PORT

'shiftout pinout,pinclock, var,parameter [,bits , delay]
' value for parameter :
'  0 - MSB first ,clock low
'  1 - MSB first,clock high
'  2 - LSB first,clock low
'  3 - LSB first,clock high
'The bits is a new option to indicate the number of bits to shift out
'For a byte you should specify 1-8 , for an integer 1-16 and for a long
1-32
'The delay is an optional delay is uS and when used, the bits parameter
must
'be specified too!

'Now shift out 9 most significant bits of the LONG variable L
Shiftout Output , Clock , L , 0 , 9



'shiftin pinin,pinclock,var,parameter [,bits ,delay]
'  0 - MSB first ,clock low  (4)
'  1 - MSB first,clock high  (5)
'  2 - LSB first,clock low   (6)
'  3 - LSB first,clock high  (7)

'To use an external clock, add 4 to the parameter
'The shiftin also has a new optional parameter to specify the number of
bits

'The bits is a new option to indicate the number of bits to shift out
'For a byte you should specify 1-8 , for an integer 1-16 and for a long
1-32
'The delay is an optional delay is uS and when used, the bits parameter
must
'be specified too!


'Shift in 9 bits into a long
Shiftin Sinp , Clock , L , 0 , 9
'use shift to shift the bits to the right place in the long
```

```
Shift L , Right , 23
End
```

7.129 SHIFTOUT

Action
Shifts a bit stream out of a variable into a port pin .

Syntax
SHIFTOUT pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as a data output.
Pclock	The port pin which generates the clock.
Var	The variable that is shifted out.
Option	Option can be : 0 – MSB shifted out first when clock goes low 1 – MSB shifted out first when clock goes high 2 – LSB shifted out first when clock goes low 3 – LSB shifted out first when clock goes high
Bits	Optional number of bits to shift out.
Delay	Optional delay in uS. When you specify the delay, the number of bits must also be specified. When the default must be used you can also use NULL for the number of bits.

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.
When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur. For a Long and Single 32 shifts will occur.

The SHIFTIN routine can be used to interface with all kind of chips.

The PIN is normally connected with the input of a chip that will receive information.

The PCLOCK pin is used to clock the bits out of the chip.

The VARIABLE is a normal BASIC variable. And may be of any type except for BIT. The data that is stored in the variable is sent with PIN.

The OPTIONS is a constant that specifies the direction of the bits. The chip that reads the data may want the LS bit first or the MS bit first. It also controls on which edge of the clock signal the data is sent to PIN.

The number of bits may be specified. You may omit this info. In that case the number of bits of the element data type will be used.

The DELAY normally consists of 2 NOP instructions. When the clock is too fast you can specify a delay time(in uS).

 The clock pin is brought to a initial level before the shifts take place. For mode 0, it is made 1. This way, the first clock can go from 1 to 0. And back to 1. You could see this as another clock cycle. So check if you use the proper mode. Or put the clock pin

in the right state before you use SHIFT.

See also
SHIFTIN 1449 , SHIFT 1447

Example
See SHIFTIN 1449 sample

7.130 SONYSEND

Action
Sends Sony remote IR code.

Syntax
SONYSEND address [, bits]

Uses
TIMER1

Remarks

Address	The address of the Sony device.
bits	This is an optional parameter. When used, it must be 12, 15 or 20. Also, when you use this option, the address variable must be of the type LONG.

SONY CD Infrared Remote Control codes (RM-DX55)

Function	Hex	Bin
Power	A91	1010 1001 0001
Play	4D1	0100 1101 0001
Stop	1D1	0001 1101 0001
Pause	9D1	1001 1101 0001
Continue	B91	1011 1001 0001
Shuffle	AD1	1010 1101 0001
Program	F91	1111 1001 0001
Disc	531	0101 0011 0001
1	011	0000 0001 0001
2	811	1000 0001 0001
3	411	0100 0001 0001
4	C11	1100 0001 0001
5	211	0010 0001 0001
6	A11	1010 0001 0001
7	611	0110 0001 0001
8	E11	1110 0001 0001
9	111	0001 0001 0001

0	051	0000 0101 0001
>10	E51	1110 0101 0001
enter	D11	1101 0001 0001
clear	F11	1111 0001 0001
repeat	351	0011 0101 0001
disc -	BD1	1011 1101 0001
disc +	H7D1	0111 1101 0001
|<<	0D1	0000 1101 0001
>>|	8D1	1000 1101 0001
<<	CD1	1100 1101 0001
>>	2D1	0010 1101 0001
SONY Cassette	RM-J901)	
Deck A		
stop	1C1	0001 1100 0001
play >	4C1	0100 1100 0001
play <	EC1	1110 1100 0001
>>	2C1	0010 1100 0001
<<	CC1	1100 1100 0001
record	6C1	0110 1100 0001
pause	9C1	1001 1100 0001
Dec B		
stop	18E	0001 1000 1110
play >	58E	0101 1000 1110
play <	04E	0000 0100 1110
>>	38E	0011 1000 1110
<<	D8E	1101 1000 1110
record	78E	0111 1000 1110
pause	98E	1001 1000 1110

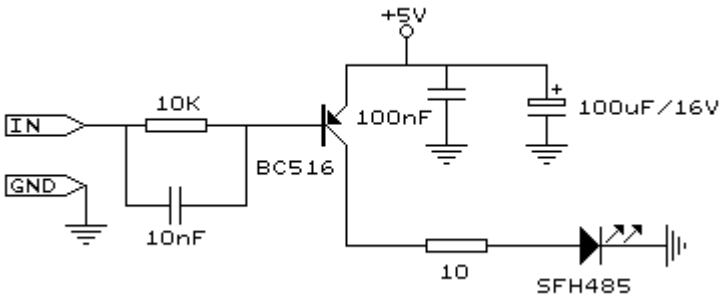
---[SONY TV Infrared Remote Control codes (RM-694)]-------------------------

```
program + = &H090 : 0000 1001 0000
program - = &H890 : 1000 1001 0000
volume + = &H490 : 0100 1001 0000
volume - = &HC90 : 1100 1001 0000
power = &HA90 : 1010 1001 0000
sound on/off = &H290 : 0010 1001 0000
1 = &H010 : 0000 0001 0000
2 = &H810 : 1000 0001 0000
3 = &H410 : 0100 0001 0000
4 = &HC10 : 1100 0001 0000
5 = &H210 : 0010 0001 0000
6 = &HA10 : 1010 0001 0000
7 = &H610 : 0110 0001 0000
8 = &HE10 : 1110 0001 0000
9 = &H110 : 0001 0001 0000
0 = &H910 : 1001 0001 0000
-/-- = &HB90 : 1011 1001 0000
```

For more SONY Remote Control info:
http://www.fet.uni-hannover.de/purnhage/

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.
Look in a data sheet for the proper pin when used with a different chip.

An IR booster circuit is shown below:



When sending hex, prefix with **&H**. When sending binary data, prefix with **&B**.

```
Sonysend &HA90
Sonysend &B010011010001
```

See also

CONFIG RC5 1037 , GETRC5 1278 , RC5SEND 1405 , RC6SEND 1411

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : sonysend.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : code based on application note from Ger
Langezaal
'micro                    : AT90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


'   +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC5SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3

Do
```

```
   Waitms 500
   Sonysend &HA90
Loop
End
```

7.131 SIZEOF

Action
This function returns the size of a variable.

Syntax
size = Sizeof(var)

Remarks
The SizeOf function returns the size in memory of a variable. It does not matter where the variable is stored : sram, xram or eeprom.
The functions accepts normal variables and indexed variables. When using an index, note that the total size is returned of the array. No matter which index you specify.

Since variables do not store run time information this function only works at compile time. So you can not use this function in a dynamic way
Variables use the following amount of memory.

Variable	Size in Bytes
Bit	1
Byte	1
Integer	2
Word	2
Dword	4
Long	4
Single	4
Double	8
String	length + 1
String * 1	2
String * 10	11

A bit is 1/8 of a byte. So there will fit 8 bits into a byte. The function will return the minimum length.
A string always uses an additional byte since strings are null terminated. Which means a null marks the end of a string.

See also
VARPTR 1604

Example
```
'-----------------------------------------------------------------
---------------------
'                              sizeof.bas
'                      (c)1995-2025, MCS Electronics
```

```
'  demo of sizeof() function
'
'-----------------------------------------------------------------
----------------------
$regfile = "atXtiny816.dat"
$hwstack = 32
$swstack = 32
$FrameSize=24

'set the base for arrays to 0
Config Base = 0

Dim S As String * 10
Dim B As Byte
Dim I As Integer
Dim L As Long
Dim Sng As Single
Dim D As Double

Dim Bar(3) As Byte
Dim Sar(4) As String * 5

Const X3 = Sizeof(sar(0))

Print X3
Print Sizeof(s)
Print Sizeof(b)
Print Sizeof(i)
Print Sizeof(l)
Print Sizeof(sng)
Print Sizeof(d)
Print Sizeof(bar(_base))
Print Sizeof(sar(_base))
B = Sizeof(sar(_base))

End
```

7.132 SORT

Action

Sorts an array in ascending order.

Syntax

SORT array() [,elements]

Remarks

array()	The first element of the array to sort.

elements	The number of elements to sort. This is an optional value. By default all elements will be sorted.

Sorting is implemented for BYTE, WORD, INTEGER, LONG and DWORD arrays.
The routines are located in mcs.lib.

See also

The SAMPLES folder contains a user contributed insertion sort algorithm sample. (insertionsort.bas)

Example

```
'-----------------------------------------------------------------------
--------
'                          SORT.BAS
'                 (c) 1995-2025 , MCS Electronics
' This demo demonstrates the SORT statement. It will sort an array
'-----------------------------------------------------------------------
-------
$regfile = "m88def.dat"
$crystal = 8000000
$hwstack = 16
$swstack = 8
$framesize = 30

'Dim some arrays
Dim B(10) As Byte , I(10) As Integer , W(10) As Word
Dim J As Byte

'point to data
Restore Arraydata

'read the data
For J = 1 To 10
   Read B(j)
Next
'read the words
For J = 1 To 10
   Read W(j)
Next
'read the integers
For J = 1 To 10
   Read I(j)
Next

'now sort the arrays
Sort B(1) , 10                                              ' 10
elements
Sort W(1)                                                   ' all
elements
Sort I(1)

'and show the result
For J = 1 To 10
   Print J ; "  " ; B(j) ; " " ; W(j) ; " " ; I(j)
Next
End
```

```
Arraydata:
Data 1 , 4 , 8 , 9 , 2 , 5 , 3 , 7 , 6 , 4
Data 1000% , 101% , 1% , 400% , 30000% , 20000% , 15000% , 0% , 999% ,
111%
Data -1000% , 101% , -1% , 400% , 30000% , 2000% , -15000% , 0% , 999% ,
111%
```

7.133 SOUND

Action

Sends pulses to a port pin.

Syntax

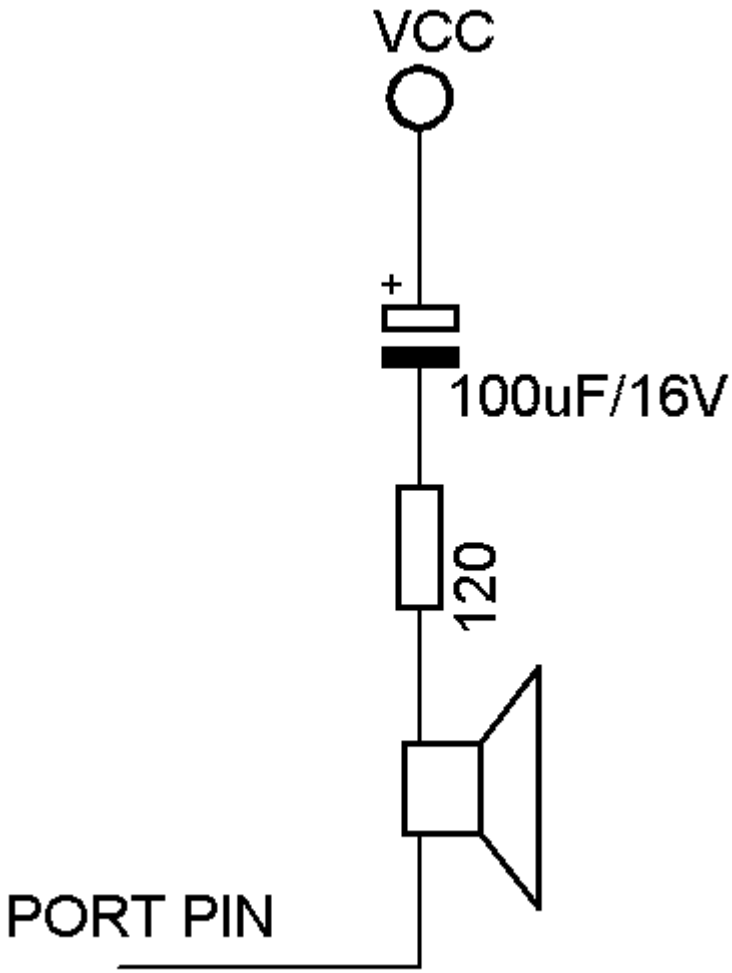
SOUND pin, duration, pulses

Remarks

Pin	Any I/O pin such as PORTB.0 etc.
Duration	The number of pulses to send. Byte, integer/word or constant.
Pulses	The time the pin is pulled low and high. This is the value for a loop counter.

When you connect a speaker or a buzzer to a port pin (see hardware) , you can use the SOUND statement to generate some tones.
The port pin is switched high and low for pulses times.
This loop is executed duration times.

The SOUND statement is not intended to generate accurate frequencies. Use a TIMER to do that.



See also
NONE

Example
```
'--------------------------------------------------------------------
------------------
'name                     : sound.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo : SOUND
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
```

```
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Dim Pulses As Word , Periods As Word
Pulses = 65535 : Periods = 10000                             'set
variables
Speaker Alias Portb.1                                        'define port
pin

Sound Speaker , Pulses , Periods                             'make some
noice
'note that pulses and periods must have a high value for high XTALS
'sound is only intended to make some noise!

'pulses  range from 1-65535
'periods range from 1-65535
End
```

7.134 RAINBOW

7.134.1 RB_SELECTCHANNEL

Action

Selects the active channel.

Syntax

RB_SELECTCHANNEL channel

Remarks

channel	A numeric variable or constant that specifies the active channel. The range is from 0-7

This statement will set the active channel and initializes the output pin. The channel is a numeric variable in the range from 0-7.
All rainbow commands will work on the active channel. This means that you need to use RB_SelectChannel at least once.
You should not specify undefined channels. Channels are defined with CONFIG RAINBOW 1033
It is NOT required to use RB_SELECTCHANNEL when the channel remains the same. So use it once and only when the channel changes.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

See RB_CHANGEPIN 1465

7.134.2 RB_SETCOLOR

Action
Set the color of a LED.

Syntax
RB_SETCOLOR LEDnr , color()

Remarks

LEDnr	A word variable or numeric constant which defines the index of the LED. This should be a valid index for the active channel. When the current channel has 8 leds defined with CONFIG RAINBOW, a valid number would be in the range from 0-7. Leds start counting at 0. This is independent of the option base !
color()	A byte array with a minimum length of 3 that holds the RGB information. A LONG or DWORD can be used as well.

The color information is set in memory. To update the color of the LED, use RB_SEND

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
```
'-----------------------------------------------------------------
---------------
'                   rainbow_ws2812_KnightriderDual.bas
'                      based on sample from Galahat
'-----------------------------------------------------------------
---------------
$Regfile = "m88pdef.dat"
$Crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 32

Config RAINBOW=1, RB0_LEN=8, RB0_PORT=PORTB,rb0_pin=0
'                                                  ^ connected
to pin 0
'                                        ^------------ connected
to portB
'                           ^------------------------- 8 leds on
stripe
```

```
'                ^----------------------------------- 1 channel


'Global Color-variables
Dim Color(3) as Byte
R alias Color(_base) : G alias Color(_base + 1) : B alias Color(
_base + 2)

'CONST
const numLeds=8

'----[MAIN]-----------------------------------------------------
---------------
Dim n as Byte

RB_SelectChannel 0           ' select first channel
R = 50 : G = 0 : B = 100     ' define a color
RB_SetColor 0 , color(1)     ' update led on the left
RB_SetColor 7 , color(1)     ' update led on the right
RB_Send

Do
   For n = 1 to Numleds/2 - 1
      rb_Shiftright 0 , Numleds/2  'shift to the right
      rb_Shiftleft 4 , Numleds/2   'shift to the left all leds
except the last one
      Waitms 100
      RB_Send
   Next
   For n = 1 to Numleds/2 - 1
      rb_Shiftleft 0 , Numleds/2   'shift to the left all leds
except the last one
      rb_Shiftright 4 , Numleds/2  'shift to the right
      Waitms 100
      RB_Send
   Next
   waitms 500                    'wait a bit
Loop
```

7.134.3 RB_SEND

Action

Transmits the channel data to the defined port pin.

Syntax

RB_SEND

Remarks

The WS2812 will latch the received information. You only need to use RB_SEND when you want to send new color information.
Some statements and functions will call RB_SEND internally.

The following table shows which statements update the LED at once

STATEMENT	UPDATE LED
RB_ADDCOLOR 1468	-
RB_ANDCOLOR 1468	-
RB_ORCOLOR 1470	-
RB_SUBCOLOR 1470	-
RB_CLEARSTRIPE 1471	YES
RB_CLEARCOLORS 1472	-
RB_FILL 1472	YES
RB_FILLCOLORS 1473	-
RB_FILLSTRIPE 1475	YES
RB_SELECTCHANNEL 1462	-
RB_SEND 1464	YES
RB_SETCOLOR 1463	-
RB_SWAPCOLOR 1475	-
RB_ROTATELEFT 1476	-
RB_ROTATERIGHT 1477	-
RB_SHIFTLEFT 1478	-
RB_SHIFTRIGHT 1479	-
RB_CHANGEPIN 1465	-
RB_SETTABLECOLOR 1479	-
RB_GETCOLOR 1483	-
RB_LOOKUPCOLOR 1484	-
RB_COPY 1486	-
RB_COLOR 1486	-

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
See RB_CHANGEPIN 1465

7.134.4 RB_CHANGEPIN

Action
Changes the defined output pin at run time

Syntax
RB_CHANGEPIN Port , Pin

Remarks

Port	A numeric variable or constant with the I/O address of the port. Notice that this is an absolute memory address. For ports in the normal IO range, you need to add a value of &H20 to the address. Example : Const nprt=varptr(portb) + &H20 Rb_ChangePIN nprt, 1
Led	A numeric variable or constant with the pin number in the range from 0-7

When you want to use multiple stripes with the same color, it would require CONFIG RAINBOW to set up all these stripes.
But each configured pin will use memory for the RGB information. When you change the pin at run time, you will use the color information of one stripe.

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
```
'-----------------------------------------------------------------
---------------
'                    rainbow_ws2812_Demo.bas
'-----------------------------------------------------------------
---------------
$Regfile = "m88pdef.dat"
$Crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 32
Config RAINBOW=1, RB0_LEN=8, RB0_PORT=PORTB,rb0_pin=0
'                                                    ^ connected
to pin 0
'                                          ^------------ connected
to portB
'                           ^-------------------------- 8 leds on
stripe
'               ^------------------------------------ 1 channel


'Global Color-variables
```

```
Dim Color(3) as Byte
R alias Color(_base) : G alias Color(_base + 1) : B alias Color(
_base + 2)

'CONST
const numLeds=8

'----[MAIN]----------------------------------------------------
---------------
Dim n as Byte, state as Byte, tel as Byte
state=0 : tel=0

RB_SelectChannel 0          ' select first channel
R = 50 : G = 0 : B = 100    ' define a color
RB_SetColor 0 , color(1)    ' update led on the left
RB_SetColor 7 , color(1)    ' update led on the right
RB_Send

Do
   For n = 1 to Numleds/2 - 1
      rb_Shiftright 0 , Numleds/2  'shift to the right
      rb_Shiftleft 4 , Numleds/2   'shift to the left all leds
except the last one
      Waitms 100
      RB_Send
   Next
   For n = 1 to Numleds/2 - 1
      rb_Shiftleft 0 , Numleds/2   'shift to the left all leds
except the last one
      rb_Shiftright 4 , Numleds/2  'shift to the right
      Waitms 100
      RB_Send
   Next
   'waitms 500                    'wait a bit
   select case state
      case 0 : r=r+5 : Rb_AddColor 0, color(1) : rb_send: tel=tel
+1
      case 1:  g=g+5 : Rb_subColor 0, color(1) : rb_send:tel=tel+
1
      case 2:  b=b+5 : Rb_orColor 0, color(1) :  rb_send: tel=tel
+1
      case 3:  Rb_ClearStripe : tel=4
      case 4:  rb_send : tel=5
      case 5:  Rb_Fill color(1) : tel=5
      case 6:  const nprt=varptr(portb) + &H20 :  Rb_ChangePIN
nprt, 1
      case else
        state=0
   end select
```

```
   if tel>=2 then
      state=state+1 : tel=0
   end if
Loop
```

7.134.5 RB_ADDCOLOR

Action
Adds specified color info to the specified LED in memory

Syntax
RB_ADDCOLOR Led , Color

Remarks

Color	Color is a byte array or variable that contains color information.
Led	The index of the LED number. First LED is 0.

The operation is performed on the memory. When the R, G or B exceeds 255, the value is limited to 255. You need to use RB_SEND so that the LED reflects the new color information.

See also
CONFIG RAINBOW 1033 , RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
See RB_CHANGEPIN 1465

7.134.6 RB_ANDCOLOR

Action
Ands specified color info to the specified LED in memory

Syntax
RB_ANDCOLOR Led , Color

Remarks

Color	Color is a byte array or variable that contains color information.
Led	The index of the LED number. First LED is 0.

The operation is performed on the memory. An AND operation can clear part of a

color. You need to use RB_SEND so that the LED reflects the new color information.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

```
'-----------------------------------------------------------------
---------------
'                    rainbow_ws2812_Demo_Softblink.bas
'This demo show RB_OrColor and RB_AndColor which can be used
'for a flashing LED with a fade effect.
'-----------------------------------------------------------------
---------------
$Regfile = "m88pdef.dat"
$Crystal=8000000
$hwstack=32
$swstack=16
$framesize=32
Config RAINBOW=1, RB0_LEN=8, RB0_PORT=PORTB,rb0_pin=0
'                                                  ^ connected
to pin 0
'                                       ^------------ connected
to portB
'                           ^-------------------------- 8 leds on
stripe
'               ^------------------------------------ 1 channel

Const Numled=8
Dim MASK as Dword
Dim Fade as Byte

'----[MAIN]------------------------------------------------------
---------------
RB_SelectChannel 0          ' select first channel

Do
   For Fade = 0 to 7
      Waitms 20
      Shift MASK , left
      Incr MASK
      RB_ORColor 0 , MASK
      RB_Send
```

```
   Next
   For Fade = 0 to 7
      Waitms 20
      Shift MASK , right
      RB_ANDColor 0 , MASK
      RB_Send
   Next
Loop
```

7.134.7 RB_ORCOLOR

Action
Ors specified color info to the specified LED in memory

Syntax
RB_ORCOLOR Led , Color

Remarks

Color	Color is a byte array or variable that contains color information.
Led	The index of the LED number. First LED is 0.

The operation is performed on the memory. An OR operation can set part of a color. You need to use RB_SEND so that the LED reflects the new color information.

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
See RB_ANDCOLOR 1468

7.134.8 RB_SUBCOLOR

Action
Subtracts specified color info to the specified LED in memory

Syntax
RB_SUBCOLOR Led , Color

Remarks

Color	Color is a byte array or variable that contains color information.
Led	The index of the LED number. First LED is 0.

The operation is performed on the memory. When the R, G or B go below 0, the value is limited to 0. You need to use RB_SEND so that the LED reflects the new color information.

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
See RB_CHANGEPIN 1465

7.134.9 RB_CLEARSTRIPE

Action
Turns off all LEDs of the active channel

Syntax
RB_CLEARSTRIPE

Remarks
The LEDS are all turned off. The information in memory is NOT changed. RB_SEND does not need to be used. In fact, using RB_SEND would send the data from memory again.

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
See RB_CHANGEPIN 1465

7.134.10 RB_CLEARCOLORS

Action
Clears all color info in memory of the active channel

Syntax
RB_CLEARCOLORS

Remarks
All color info of the active channel is cleared. The LEDS keep their color until an RB_SEND is used.

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

7.134.11 RB_FILL

Action
Fills the memory of the active channel with a color and updates the LED's.

Syntax
RB_FILL Color

Remarks

Color	Color is a byte array or variable that contains color information.

All LED's of the active channel will be set to the specified color in memory. This statement will also update the LED's so it is not needed to use RB_SEND.
This statement is similar to RB_CLEARCOLORS except that you can provide a color and that it is not required to use RB_SEND

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

.

7.134.11 RB_FILLCOLORS

Action

Fills the entire memory of the active channel with a specified color

Syntax

RB_FILLCOLORS Color

Remarks

Color	Color is a byte array or variable that contains color information.

The entire memory of the active channel is filled with the specified color.
This statement will not update the LED's. This means that you need to use RB_SEND to update the LED's. Or use RB_FILL which will update the LED's as well.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

```
'-----------------------------------------------------------------
---------------
'                   rainbow_ws2812_Levelmeter.bas
'
' This example demonstrate the switching between two
Rainbow-Stripes while simulating
' a simple kind of an stereo levelmeter and the use of some
RB_statements.
'
'-----------------------------------------------------------------
---------------
$Regfile = "m88pdef.dat"
$Crystal=8000000
$hwstack=40
$swstack=16
$framesize=32
```

```
Config RAINBOW= 2, RB0_LEN=8, RB0_PORT=PORTB,rb0_pin=0  , RB1_LEN
=8, RB1_PORT=PORTB,rb1_pin=1
Dim n as Byte
Dim Color as DWord
Dim CH as Byte
Dim LEFT_Level as Byte ,  Left_Level_OLD as Byte
Dim Right_Level as Byte , Right_Level_OLD as Byte
Const Channels = 2
Const Backcolor = &H000005

'----[MAIN]-----------------------------------------------------
---------------
Color = Backcolor
For ch = 0 to Channels -1
   Rb_SelectChannel Ch
   RB_Fillcolors Color
   Rb_SetTableColor 0,0
   RB_send
Next
Do
   incr n:  n = n and &H30   'n counts from 0 to 63
   If n = 0 then Gosub Get_Level   'Read signal
   'Switch channel
   toggle Ch
   Rb_SelectChannel Ch
   Waitms 40
   If ch = 0 then  'Channel 0
      If left_level_old <  left_level then
         incr Left_level_old
      ElseIf Left_level_old > Left_level then
         Decr Left_level_old
      End if
      RB_Fillcolors Color
      Rb_SetTableColor Left_level_old ,0
   Else  'Channel 1
      If right_level_old <  right_level then
         incr right_level_old
      ElseIf right_level_old > right_level then
         Decr right_level_old
      End if
      RB_Fillcolors Color
      Rb_SetTableColor right_level_old ,0
   end if
   RB_Send
Loop

Get_Level:
```

```
   Left_Level  = rnd(7)
   Right_Level = rnd(7)
Return

Rainbow_Colors:
   Data 100,50,0      'orange
```

7.134.13 RB_FILLSTRIPE

Action
Set all LED's of the active channel to the specified color. This statement will not change the memory.

Syntax
RB_FILLSTRIPE Color

Remarks

Color	Color is a byte array or variable that contains color information.

All LED's of the active channel will be set to the specified color. This statement will not change the memory, just the LED's.

See also
CONFIG RAINBOW 1033, RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471, RB_CLEARCOLORS 1472, RB_FILL 1472, RB_FILLCOLORS 1473, RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463, RB_SWAPCOLOR 1475, RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479, RB_CHANGEPIN 1465, RB_SETTABLECOLOR 1479, RB_GETCOLOR 1483, RB_LOOKUPCOLOR 1484, RB_COPY 1486, RB_COLOR 1484

Example

7.134.14 RB_SWAPCOLOR

Action
Exchange color between to LED's of the active channel. This statement will only change the memory.

Syntax
RB_SWAPCOLOR Led1 , Led2

Remarks

Led1 , Led2	The index of the LED of the active channel.

This statement will swap the color info of the specified LED's.
So after the execution of the statement, LED1 becomes the color of LED2 and LED2 becomes the color of LED1.
This statement operates on the memory, it will not update the LED's.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

7.134.1 RB_ROTATELEFT

Action

Rotate all LED's of the active channel to the left

Syntax

RB_ROTATELEFT Index , Width

Remarks

Index	A numeric variable or constant that specifies at which position the rotation should start. The first LED has index value 0.
Width	The number of LED's that ROTATE, starting at Index. Width should at least be 1.

This statement will rotate the memory to the left by one position. Width specifies how many LED's , index inclusive, will take part in the rotation.
Imagine 4 chairs with people on it. When they all stand up and go one place to the left, the person most left will have no chair to sit on. He will take the free chair on the right.

There is also a similar operation named SHIFT. When you SHIFT, information is lost : the person that has no chair on his left will leave the room and there will be 1 empty chair.

Since you can specify both the index and the width, rotation is very flexible : you can rotate all leds, or just a part of them.

The table below demonstrates a number of operation on a LED stipe of 4 LED's.

LED0	LED1	LED2	LED3	OPERATION/RESULT
				OPERATION : RB_ROTATELEFT 0,4

				RESULT
				OPERATION : RB_ROTATELEFT 0,2
				RESULT
				OPERATION : RB_ROTATELEFT 0,4
				RESULT
				OPERATION : RB_ROTATELEFT 1,2
				RESULT

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

7.134.10 RB_ROTATERIGHT

Action

Rotate all LED's of the active channel to the right

Syntax

RB_ROTATERIGHT Index , Width

Remarks

Index	A numeric variable or constant that specifies at which position the rotation should start.
Width	The number of LED's that ROTATE, starting at Index. Width should at least be 1.

This statement will rotate the memory to the right by one place. Width specifies how many LED's , index inclusive, will take part in the rotation.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 ,

RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

7.134.17 RB_SHIFTLEFT

Action
Shift all LED's of the active channel to the left

Syntax
RB_SHIFTLEFT Index , Width

Remarks

Index	A numeric variable or constant that specifies at which position the shift should start.
Width	The number of LED's that SHIFT, starting at Index. Width should at least be 1.

This statement will shift the memory to the left by one position. Width specifies how many LED's , index inclusive, will take part in the shift operation.
When you shift information to the LEFT, the RIGHT-most LED will loose it's color information since it had no LED with data at the right.
Imagine 4 chairs with people on it. When they all stand up and go one place to the left, the person most left will have no chair to sit on. The chair on the right will be empty.

The table below demonstrates a number of operation on a LED stipe of 4 LED's.

LED0	LED1	LED2	LED3	OPERATION/RESULT
				OPERATION : RB_SHIFTLEFT 0,4
				RESULT
				OPERATION : RB_SHIFTLEFT 0,2
				RESULT
				OPERATION : RB_SHIFTLEFT 0,4
				RESULT
				OPERATION : RB_SHIFTLEFT 1,2
				

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 ,

RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
See RB_CHANGEPIN 1465

7.134.18 RB_SHIFTRIGHT

Action
Shift all LED's of the active channel to the right

Syntax
RB_SHIFTRIGHT Index , Width

Remarks

Index	A numeric variable or constant that specifies at which position the shift should start.
Width	The number of LED's that SHIFT, starting at Index. Width should at least be 1.

This statement will shift the memory to the right by one position. Width specifies how many LED's , index inclusive, will take part in the shift operation
When you shift information to the RIGHT, the LEFT-most LED will loose it's color information.
Imagine 4 chairs with people on it. When they all stand up and go one place to the right, the person most right will have no chair to sit on. The chair on the left will become empty.

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example
See RB_CHANGEPIN 1465

7.134.19 RB_SETTABLECOLOR

Action
Set the color of a LED using a lookup table.

Syntax

RB_SETTABLECOLOR LED , Index [, Label]

Remarks

LED	The index of the LED of the active channel which color need to be changed. The first LED number is 0.
Index	A byte variable or constant that holds the index of the table. The table need to be identified by a label. This is either a user defined label, or a label named RAINBOW_COLORS The table has the R, G, B format. Example: Rainbow_Colors: ' R , G , B index Data &HFF , &H00 , &H00 'Red 0 Data &H00 , &HFF , &H00 'Green 1 Data &H00 , &H00 , &HFF 'Blue 2 Data &HFF , &HA5 , &H00 'Orange 3 Data &HFF , &HFF , &H00 'Yellow 4 Data &HFF , &H69 , &HB4 'HotPink 5
Label	The label name of the table. This is an optional value. If the label name is not specified, the name RAINBOW_COLORS will be used.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_GETCOLOR 1483 , RB_LOOKUPCOLOR 1484 , RB_COPY 1486 , RB_COLOR 1484

Example

```
'-----------------------------------------------------------------
---------------
'                   rainbow_ws2812_Trafficlights.bas
'
' This example  simulates two simple Trafficlights.
' It shows how switch between two Stripes with just one defined
Rainbow.
' The active output gets changed by the RB_ChangePin statement.
' Thus the use of memory is small.
'
'-----------------------------------------------------------------
---------------
'(
Following situation:
The one way route from the Weststreet to Nothstreet and vice
```

```
versa is the main route
and cars on these roads have priority.The corresponding light
will show green light normally.
In our simple world, every five seconds a car wants to drive from
the Eaststreet to the Northstreet.
Thus, the trafficflow from the main street has to stop, to let
the cars pass.
'               Northstreet
'
'                |     |
'                |     |
'                |     | ooo
'          ------'     '------
'                              EastStreet
'           --->         ------
WestStreet              |
'          -----------/
'               ooo
'
'
')
$Regfile = "m88pdef.dat"
$Crystal=8000000
$hwstack=40
$swstack=16
$framesize=32
'We use just one Channel for both Trafficlights, cause LED
stripes are static
Config RAINBOW= 1, RB0_LEN=3, RB0_PORT=PORTB,rb0_pin=0
Rb_SelectChannel 0   'we use the defined Channel

'Port+Pin combinations, formed to a word
Const MainStreet_0 = (((varptr(portb) + &H20) *256) OR PB0)
Const EastStreet_1 = (((varptr(portb) + &H20) *256) OR PB1)

'----[MAIN]-----------------------------------------------------
---------------
Dim PortPin as Word
Dim Street as Byte  'selects the current PortPin cofiguration
Const Mainstreet = 0
Const Eaststreet = 1
'Index for LED and colors also
Const Red = 0
Const Yellow = 1
Const Green = 2
Gosub inital_state
Do
   Gosub Wait_for_car
   'Trafficlight turns to Red
```

```
   Street =  Mainstreet
   Gosub Turn_to_Red
   'Trafficlight turns to green
   Street = Eaststreet
   Gosub Turn_to_green
   Gosub Wait_for_car 'let some cars passing
   Gosub Turn_to_red
   'Mainstreet becomes green
   Street =  Mainstreet
   Gosub Turn_to_green
Loop

Wait_for_car:
   Wait 5
Return

Turn_to_Green:
   Gosub Change_Port_Pin
   RB_SettableColor Yellow,Yellow,Light    'load and set color
from table
   RB_Send                                 'refresh stripe
   Wait 1
   RB_clearcolors                          'clear colors in
memory
   RB_SettableColor green,green,Light      'load and set color
from table
   RB_Send                                 'refresh stripe
   Wait 2
Return

Turn_to_red:
   Gosub Change_Port_Pin
   RB_clearcolors                          'clear colors in
memory
   RB_SettableColor Yellow,Yellow,Light    'load and set color
from table
   RB_Send                                 'refresh stripe
   Wait 3
   RB_clearcolors                          'clear colors in
memory
   RB_SettableColor red,red,Light          'load and set color
from table
   RB_Send                                 'refresh stripe
   Wait 2
Return

Inital_State:
'select Mainstreet, green
   Street = Mainstreet
```

```
   Gosub Change_Port_Pin
   RB_clearcolors
   RB_SettableColor green,green,Light
   RB_Send
'select Eaststreet, red
   Street = Eaststreet
   Gosub Change_Port_Pin
   RB_clearcolors
   RB_SettableColor Red,Red,Light
   RB_Send
Return

Change_PORT_PIN:
   PortPin = Lookup(Street,PortPin_Tbl)    'get PortPin comination
   RB_ChangePin High(PortPin),PortPin      'use PortPin
Return

PortPin_Tbl:
   Data MainStreet_0%
   Data EastStreet_1%

Light:
   Data 150,0,0       'Red
   Data 100,50,0      'Yello
   Data 0,150,0       'Green
```

7.134.2 RB_GETCOLOR

Action

Returns the RGB color information of a LED of the active channel.

Syntax

Color = **RB_GETCOLOR(** LED **)**

Remarks

Color	Color is a byte array or variable that contains color information.
LED	The index of the LED number. First LED is 0.

See also

CONFIG RAINBOW 1033, RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471, RB_CLEARCOLORS 1472, RB_FILL 1472, RB_FILLCOLORS 1473, RB_FILLSTRIPE 1475, RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463, RB_SWAPCOLOR 1475, RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479, RB_CHANGEPIN 1465, RB_SETTABLECOLOR 1479, RB_LOOKUPCOLOR 1484, RB_COPY 1486, RB_COLOR 1484

Example

7.134.21 RB_LOOKUPCOLOR

Action
Returns the RGB color information from a data table using an index.

Syntax
Color = **RB_LOOKUPCOLOR(** Index [, Label] **)**

Remarks

Index	A byte variable or constant that holds the index of the table. The table need to be identified by a label. This is either a user defined label, or a label named RAINBOW_COLORS The table has the R, G, B format. Example: Rainbow_Colors: ' R , G , B index Data &HFF , &H00 , &H00 'Red 0 Data &H00 , &HFF , &H00 'Green 1 Data &H00 , &H00 , &HFF 'Blue 2 Data &HFF , &HA5 , &H00 'Orange 3 Data &HFF , &HFF , &H00 'Yellow 4 Data &HFF , &H69 , &HB4 'HotPink 5
Label	The label name of the table. This is an optional value. If the label name is not specified, the name RAINBOW_COLORS will be used.

RB_LOOKUP is a help function. It does not work on the memory or LED's. I just returns a color from a data table using a lookup value.

See also
CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_GETCOLOR 1483 , RB_COPY 1486 , RB_COLOR 1484

Example

7.134.22 RB_COLOR

Action
Color multiple LED's according to the bit pattern of a mask.

Syntax

RB_COLOR LED_start , Mask, Color1 **[**,Color2**]**

Remarks

LED_start	The index of the LED number. First LED is 0.
Mask	Bitmask of 8 LED's. A set bit(1) will color a LED with **COLOR1**, according to its bit position + LED_start. A zero-bit turns a LED off, or optionally colors an LED with COLOR2.
Color1	A byte array with a minimum length of 3 that holds the RGB information. A LONG or DWORD can be used as well.
Color2	This is an optional parameter. A byte array with a minimum length of 3 that holds the RGB information. A LONG or DWORD can be used as well.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_LOOKUPCOLOR 1484 RB_COPY 1486

Example

```
'=======================================
'RB_COLOR test
'=======================================
$Regfile = "m32adef.dat"
$Crystal = 16000000
$hwstack = 40
$swstack = 32
$framesize = 32
Config Rainbow = 2 , Rb0_len = 8 , Rb0_port = Porta , Rb0_pin =
0 , Rb1_len = 8 , Rb1_port = Porta , Rb1_pin = 1
Dim Color1 as Dword
Dim Color2 as Dword
Const Red = &H000010
Const Blue = &H100000

Color1 = Red
Color2 = Blue
Rb_selectchannel 0
Do
   Rb_color 0 , &H88 , Color1
   Rb_send
   Wait 1
```

```
    Rb_color 0 , &H88 , Color2 , Color1
    Rb_send
    Wait 1
Loop
End
```

7.134.23 RB_COPY

Action

Copy whole stripes or any parts of it, to another stripes memory space at any position.

Syntax

RB_COPY Source , SourceStart, Target, TargetStart, Count

Remarks

Source	The index of the source stripe.
Source Start	The position to start the copy from
Target	The index of the target stripe. This can be the same stripe or another one.
TargetStart	The position to start the copy to.
Count	The number of bytes to copy.

This routine offers a faster track to copy a whole bunch of color data , if necessary.

See also

CONFIG RAINBOW 1033 , RB_ADDCOLOR 1468, RB_ANDCOLOR 1468, RB_ORCOLOR 1470, RB_SUBCOLOR 1470, RB_CLEARSTRIPE 1471 , RB_CLEARCOLORS 1472 , RB_FILL 1472 , RB_FILLCOLORS 1473 , RB_FILLSTRIPE 1475 , RB_SELECTCHANNEL 1462, RB_SEND 1464, RB_SETCOLOR 1463 , RB_SWAPCOLOR 1475 , RB_ROTATELEFT 1476, RB_ROTATERIGHT 1477, RB_SHIFTLEFT 1478, RB_SHIFTRIGHT 1479 , RB_CHANGEPIN 1465 , RB_SETTABLECOLOR 1479 , RB_LOOKUPCOLOR 1484 , RB_COLOR 1484

Example

```
$regfile = "m32adef.dat"
$crystal = 16000000
$hwstack = 40
$swstack = 16
$framesize = 32

'(
RB_Copy Rainbow0_ , 5 ,Rainbow1_ ,0   , 3
'                                        ^---- count of leds to
copy
'                                    ^---------- Led, start index
```

```
of target
'                              ^------------------ target stripe or
array
'                        ^------------------------ LED, start index
of source
'                ^------------------------------- source stripe or
array
')

Config Rainbow = 2 , Rb0_len = 8 , Rb0_port = Porta , Rb0_pin =
0 , _
                      Rb1_len = 8 , Rb1_port = Porta , Rb1_pin = 1

Dim Color as Dword

Const Red = &H000010
Const Green = &H001000
Const Blue = &H100000
Const Yellow = &H001010

'color the first 4 LED
Rb_selectchannel 0
Color = Green
Rb_setcolor 0 , Color

Color = Red
RB_SetColor 1 , Color

Color = Blue
RB_SetColor 2 , Color

Color = Yellow
RB_SetColor 3 , Color
RB_Send
wait 1
' copy LED 0 to 3 of stripe 0 to pos 4 to 7
Rb_copy Rainbow0_ , 0 , Rainbow0_ , 4 , 4
Rb_send
Wait 1

' copy whole stripe0 to stripe1
RB_Copy Rainbow0_ , 0 , Rainbow1_ , 0 , 8
RB_Clearcolors
RB_SelectChannel(0) : RB_Send
RB_SelectChannel(1) : RB_Send
Wait 1
'copy LED1  of stripe1 to LED7 of stripe0
RB_Copy Rainbow1_ , 1 , Rainbow0_ , 7 , 1
RB_SelectChannel(0) : RB_Send
```

```
End
```

7.135 Serial Data RS232-RS485

7.135.1 BAUD

Action
Changes the baud rate for the hardware or software UART.

Syntax
BAUD = const

Syntax Software UART
BAUD #x , const

Remarks

X	The channel number of the software UART.
Const	A numeric constant for the baud rate that you want to use.

 Do not confuse the BAUD statement with the $BAUD 607 compiler directive.

And do not confuse $CRYSTAL 625 and CRYSTAL 1176

$BAUD overrides the compiler setting for the baud rate while BAUD will change the current baud rate.
So $BAUD is a global project setting in your source code while BAUD will change the baud rate during run time.
You could use BAUD to change the baud rate during run time after the user changes a setting.

BAUD = ... will work on the hardware UART.

BAUD #x, yyyy will work on the software or HW UART. The specified channel must be the same as used with the OPEN statement.
When you use a software UART and change the baud rate at run time using BAUD, you must set the baud rate after the OPEN statements as well.
When you do not use BAUD, there is no need to set it. So for example :

```
Open "COMC.1:9600,8,N,1" For Output As #1
print #1 , "this is a test 9600"   'no need for BAUD since one baud rate is used
```

But when BAUD is changed :

```
Open "COMC.1:9600,8,N,1" For Output As #1
baud #1 , 9600  'we need to set it since we change baud at run time
print #1 , "this is a test 9600"

baud #1 , 115200
print #1 , "this is a test 115200"
```

 Variables are not supported. Only constants.

See also
$CRYSTAL 625 , $BAUD 607 , BAUD1 1489

ASM
NONE

Example
```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Print "Hello"

'Now change the baud rate in a program
Baud = 9600
Print "Did you change the terminal emulator baud rate too?"
End
```

7.135.2 BAUD1-BAUDx

Action
Changes the baud rate for the specified hardware UART.

Syntax
BAUD = var
BAUD1 = var
BAUD2 = var
BAUD3 = var

Syntax Xmega
BAUD = var
BAUD1 = var
BAUD2 = var
BAUD3 = var
BAUD4 = var
BAUD5 = var
BAUD6 = var
BAUD7 = var

Xmega Syntax
BAUDx = constant

Remarks

Var	The baud rate that you want to use.
baud	COM1, USART0, xmega and normal AVR

baud1	COM2, USART1, xmega and normal AVR
baud2	COM2, USART2, xmega and normal AVR
baud3	COM3, USART3, xmega and normal AVR
baud4	COM4, USART4, xmega
baud5	COM5, USART5, xmega
baud6	COM6, USART6, xmega
baud7	COM7, USART7, xmega

Do not confuse the BAUD1 statement with the $BAUD1 compiler directive.

And do not confuse $CRYSTAL 625 and CRYSTAL 1176

$BAUD1 overrides the compiler setting for the baud rate while BAUD1 will change the current baud rate.
BAUD1 = ... will work on the hardware UART.
BAUD**n** = ... will work on the specified hardware UART.

mega

For the mega, the X represents the UART number. BAUD means, the first UART which you refer to with OPEN as COM1, BAUD1 the second UART, and BAUD3 is the last UART. A channel number is not supported.
You need to use a constant for the baud rate. Variables are not supported.

Xmega

For the xmega, the X represents the UART number. BAUD means, the first UART which you refer to with OPEN as COM1, BAUD1 the second UART, and BAUD7 is the last UART. A channel number is not supported.
For the Xmega you need to use a constant for the baud rate. Variables are not supported.

See also

$CRYSTAL 625 , $BAUD 607 , $BAUD1 608 , BAUD 1488, CONFIG COMx 913

ASM

NONE

Example

```
'-----------------------------------------------------------------------
--------
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : Mega162
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates BAUD1 directive and BAUD1
statement

'-----------------------------------------------------------------------
--------
$regfile = "M162def.dat"
$baud1 = 2400
$crystal= 14000000 ' 14 MHz crystal
```

```
Open "COM2:" For BINARY As #1

Print #1 , "Hello"
'Now change the baud rate in a program
Baud1 = 9600                                                    '
Print #1 , "Did you change the terminal emulator baud rate too?"
Close #1
End
```

7.135.3 BUFSPACE

Action

Returns the amount of free space of a serial buffer.

Syntax

Var = **BufSpace**(n)

Remarks

Var	A word or integer variable that is assigned with the free buffer space.
N	A constant in the range from 0-15. Odd numbers are for the INPUT buffers. Even numbers are for the OUTPUT buffers. A value of 0 : output buffer USART0 (first UART) A value of 1 : input buffer USART0 (first UART) A value of 2 : output buffer USART1 (second UART) A value of 3 : input buffer USART1 (second UART) A value of 4 : output buffer USART2 A value of 5 : input buffer USART2 A value of 6 : output buffer USART3 A value of 7 : input buffer USART3 A value of 8 : output buffer USART4 A value of 9 : input buffer USART4 A value of 10 : output buffer USART5 A value of 11 : input buffer USART5 A value of 12 : output buffer USART6 A value of 13 : input buffer USART6 A value of 14 : output buffer USART7 A value of 15 : input buffer USART7 The function will only work when the processor has the chosen UART and when it has been setup using CONFIG SERIAL.

While serial buffers are great because you do not have to wait/block the processor, the buffer can become full when the micro has no time to empty the buffer. With the bufspace() function you can determine if there is still room in the buffer.

See Also
CONFIG SERIAL 1057 , CLEAR 846

Example
```
'-----------------------------------------------------
NONE
```

7.135.4 INKEY

Action
Returns the ASCII value of the first character in the serial input buffer.

Syntax
var = **INKEY**()
var = **INKEY**(#channel)

Remarks

Var	Byte, Integer, Word, Long or String variable.
Channel	A constant number that identifies the opened channel if software UART mode

If there is no character waiting, a zero will be returned.
Use the IsCharWaiting() function to check if there is a byte waiting.

The INKEY routine can be used when you have a RS-232 interface on your uP.
The RS-232 interface can be connected to a comport of your computer.

As zero(0) will be returned when no character is waiting, the usage is limited when the value of 0 is used in the serial transmission. You can not make a difference between a byte with the value 0 and the case where no data is available.
In that case you can use IsCharwaiting to deterimine if there is a byte waiting.

See also
WAITKEY 1511 , ISCHARWAITING 1498 , $TIMEOUT 708

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : inkey.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: INKEY , WAITKEY
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
```

```
crystal frequency
$baud = 19200                                                  ' use baud
rate
$hwstack = 32                                                  ' default
use 32 for the hardware stack
$swstack = 10                                                  ' default
use 10 for the SW stack
$framesize = 40                                                ' default
use 40 for the frame space

Dim A As Byte , S As String * 2
Do
   A = Inkey()                                                 'get ascii
value from serial port
   's = Inkey()
   If A > 0 Then                                               'we got
something
      Print "ASCII code " ; A ; " from serial"
   End If
Loop Until A = 27                                              'until ESC
is pressed

A = Waitkey()                                                  'wait for a
key
's = waitkey()
Print Chr(a)

'wait until ESC is pressed
Do
Loop Until Inkey() = 27

'When you need to receive binary data and the bibary value 0 ,
'you can use the IScharwaiting() function.
'This will return 1 when there is a char waiting and 0 if there is no
char waiting.
'You can get the char with inkey or waitkey then.
End
```

7.135.5 INPUT

Action

Allows input from the keyboard, file or SPI during program execution.

Syntax

INPUT [" prompt"] , var[, varn]
INPUT #ch, var[, varn]

Syntax SPI

INPUT #ch, var [;bts] [, varn [;bts]]

Remarks

Prompt	An optional string constant printed before the prompt character.
Var,varn	A variable to accept the input value or a string.
Ch	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
bts	An optional number of byes to read. Only for SPI.

The INPUT routine can be used when you have an RS-232 interface on your uP.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator and the keyboard as an input device.
You can also use the built-in terminal emulator.

For usage with the AVR-DOS file system, you can read variables from an opened file.
Since these variables are stored in ASCII format, the data is converted to the proper format automatically.
When you use INPUT with a file, the prompt is not supported.

When $BIGSTRINGS 611 is used you can read read up to 65535 bytes.

Difference with VB

In VB you can specify **&H** with INPUT so VB will recognize that a hexadecimal string is being used.
BASCOM implements a new statement : INPUTHEX.

Xmega-SPI

When receiving data from the SPI interface, you need to activate the SS pin. Some chips might need an active low, others might need an active high. This will depends on the slave chip.
When you use the SS=AUTO option, the level of SS will be changed automatic. Thus SS is made low, then the data bytes are received, and finally , SS is made high again.

Receiving data works by sending a data byte and returning the data that is shifted out. The data that will be sent is a 0. You can alter this in the library, _inputspivar routine.
You can not sent constants using the INPUT with SPI. So INPUT #10, "SPI", var is not supported.
INPUT used with SPI will not wait for a return either. It will wait for the number of bytes that fits into the variable. See CONFIG SPIx 1068 for an example.

Number of Bytes

The compiler will receive 1 byte for a variable which was dimensioned as a BYTE.
It will receive 2 bytes for a WORD/INTEGER, 4 bytes for a LONG/SINGLE and 8 bytes for a DOUBLE.
As with all routines in BASCOM, the least significant Byte will be received first.

If you specify an array, one element will be received.

SPI

With an optional parameter you can provide how many bytes must be received. You must use a semicolon (;) to specify this parameter. This because the comma (,) is used to receive multiple variables.

```
Dim Tmparray(5) As Byte  , Spi_send_byte As Byte , W as Word
Input #12 , Spi_receive_byte ; 1                ' READ 1 byte
Input #12 , Tmparray(1) ; 1 , Tmparray(2) ; B   ' read 1 byte and 'b' bytes startin
```

The optional parameter is only supported for the SPI channel. When required with

serial data, you can also use INPUTBIN.

See also

INPUTHEX 1496 , PRINT 1501 , ECHO 1247 , WRITE 801 , INPUTBIN 1497

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : input.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: INPUT, INPUTHEX
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15

Input "Use this to ask a question " , V
Input B1                                                   'leave out
for no question

Input "Enter integer " , C
Print C


Inputhex "Enter hex number (4 bytes) " , C
Print C
Inputhex "Enter hex byte (2 bytes) " , D
Print D

Input "More variables " , C , D
Print C ; " " ; D

Input C Noecho                                             'supress
echo

Input "Enter your name " , S
Print "Hello " ; S

Input S Noecho                                             'without
echo
```

```
Print S
End
```

7.135.6 INPUTHEX

Action
Allows hexadecimal input from the keyboard during program execution.

Syntax
INPUTHEX [" prompt"] , var[, varn]

Remarks

prompt	An optional string constant printed before the prompt character.
Var,varn	A numeric variable to accept the input value.

The INPUTHEX routine can be used when you have a RS-232 interface on your uP. The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator and the keyboard as input device.
You can also use the build in terminal emulator.
The input entered may be in lower or upper case (0-9 and A-F)

If var is a byte then the input can be maximum 2 characters long.
If var is an integer/word then the input can be maximum 4 characters long.
If var is a long then the input can be maximum 8 characters long.

In VB you can specify **&H** with INPUT so VB will recognize that a hexadecimal string is being used.
BASCOM implements a new statement: INPUTHEX. This is only to save code as otherwise also code would be needed for decimal conversion.

See also
INPUT 1493 , ECHO 1247 , INPUTBIN 1497

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : input.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: INPUT, INPUTHEX
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
```

```
use 32 for the hardware stack
$swstack = 10                                                    ' default
use 10 for the SW stack
$framesize = 40                                                  ' default
use 40 for the frame space

Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15

Input "Use this to ask a question " , V
Input B1                                                         'leave out
for no question

Input "Enter integer " , C
Print C


Inputhex "Enter hex number (4 bytes) " , C
Print C
Inputhex "Enter hex byte (2 bytes) " , D
Print D

Input "More variables " , C , D
Print C ; " " ; D

Input C Noecho                                                   'supress
echo

Input "Enter your name " , S
Print "Hello " ; S

Input S Noecho                                                   'without
echo
Print S
End
```

7.135.7 INPUTBIN

Action

Read binary data from the serial port.

Syntax

INPUTBIN var1 [;bts] [,var2]
INPUTBIN #channel , var1 [,var2]

Remarks

var1	The variable that is assigned with the characters from the serial port.
var2	An optional second (or more) variable that is assigned with the data from the serial input stream.
bts	Optional numeric variable that specifies how many bytes must be read. This optional variable must be placed after a semi colon delimiter (;)

The channel need to be used in combination with OPEN and the optional CLOSE.

The number of bytes to read depends on the variable you use.

When you use a byte variable, 1 character is read from the serial port.
An integer will wait for 2 characters and an array will wait until the whole array is filled.

Note that the INPUTBIN statement doesn't wait for a CRLF but just for the number of bytes.

You may also specify an additional numeric parameter that specifies how many bytes will be read. This is convenient when you are filling an array.

```
Inputbin ar(1) , 4 ' will fill 4 bytes starting at index 1.
```

In version 2083 the INPUTBIN statement is enhanced with an option to specify the number of bytes to read using a variable.
In earlier versions only a constant could be used. To keep code compatible, use a semi colon followed by a variable to specify how many bytes must be read.

```
Inputbin ar(1) , bts ' will fill the number of bytes equal with the value of bts
```

See also

PRINTBIN 1504 , CONFIG INPUTBIN 984

Example

```
Dim A As Byte , C As Integer
Inputbin A , C 'wait for 3 characters and fill 2 variables
End
```

7.135.8 ISCHARWAITING

Action

Returns one(1) when a character is waiting in the hardware UART buffer.

Syntax

var = **ISCHARWAITING**()
var = **ISCHARWAITING**(#channel)

Remarks

Var	Byte, Integer, Word or Long variable.
Channel	A constant number that identifies the opened channel.

If there is no character waiting, a zero will be returned.
If there is a character waiting, a one (1) will be returned.
The character is not retrieved or altered by the function.

While the Inkey() will get the character from the HW UART when there is a character in the buffer, it will return a zero when the character is zero. This makes it unusable to work with binary data that might contain the value 0.

With IsCharWaiting() you can first check for the presence of a character and when the function returns 1, you can retrieve the character with Inkey or Waitkey.

IsCharWaiting can NOT be used with a software uart (SW-UART). This because a SW-

UART does not buffer the data it receives or sends.

See also
WAITKEY 1511 , INKEY 1492 , $TIMEOUT 708

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte , S As String * 2
Do
   A = Ischarwaiting()
   If A = 1 Then                                           'we got
something
      A = Waitkey()                                        'get it
      Print "ASCII code " ; A ; " from serial"
   End If
Loop Until A = 27                                          'until ESC
is pressed
```

7.135.9 MAKEMODBUS

Action
Creates a MODBUS master/client frame.

Syntax
PRINT [#x,] **MAKEMODBUS**(slave, function, address, varbts)

Remarks

slave	The slave to address. This is a variable or constant with a valid MODBUS slave to address.
function	The function number. This must be a constant. At the moment the following functions are supported : • 01 : read coils • 02 : read discrete inputs • 03 : read register(s) • 04 : read input registers • 06 : write single register • 16 : write multiple registers
address	The starting address of the register
varbts	For a function that sends data like function 6 and 16, this must be a

	variable. For function 06 which can only write a single register, this can be a byte or integer or word. For function 16 it may be a long, single or double. For function 6 and 16 the address of the variable is passed to the function. For function 1,2,3 and 4 you may also specify the number of bytes to receive. Or you can use a variable. When you specify a byte, a word will be used anyway since a word (2 bytes) is the minimum in MODBUS protocol. But when sending data, you can send content of a byte. For the MSB the value 0 will be sent in that case. With : CONFIG MODBUS = VAR you can change the varbts mode. In VAR mode, you have to pass the number of bytes in the variable.

The MAKEMODBUS function need to be used in combination with the PRINT statement. It can only be used with the hardware UART(1-4).
The MODBUS protocol is an industry standard. The protocol can be used with RS-232, RS-485 or TCP/IP or CAN.
The current BASCOM implementation only works with RS-232 or RS485.
In MODBUS we use client/master and server/slave. You may see it as a web server and a web browser. The web server is the client/slave that reacts on the master/web browser.
A slave will only respond when it is addressed. All other slaves just keep listening till they are addressed.
An addressed slave will process the data and send a response.
In MODBUS the data is sent with MSB first and LSB last. The special CRC16 checksum is sent LSB first and MSB last.
When multiple registers are sent with function 16, the data is split up into words, and for each word, the MSB-LSB order is used.
For example a LONG is 4 bytes. LSB, NSB1, NSB2, MSB. It would be sent as : NSB1, LSB, MSB, NSB2.
In order to use the MODBUS functionality, you need to include the MODBUS.LBX with the $LIB directive.

See also

PRINT 1501 , CONFIG MODBUS 1005

Example

```
'-----------------------------------------------------------------------------
'name                     : rs485-modbus-master.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo file for MAKEMODBUS
'micro                    : Mega162
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------

$regfile = "m162def.dat"                                    ' specify the used micr
$crystal = 8000000
$baud = 19200                                               ' use baud rate
$hwstack = 42                                               ' default use 42 for th
$swstack = 40                                               ' default use 40 for th
```

```
$framesize = 40                                                 ' default use 40 for th

$lib "modbus.lbx"                                               ' specify the additiona
Config Print1 = Portb.1 , Mode = Set                            ' specify RS-485 and di


Rs485dir Alias Portb.1                                          'make an alias
Config Rs485dir = Output                                        'set direction register
Rs485dir = 0                                                    ' set the pin to 0 for

Portc.0 = 1                                                     '  a pin is used with a

'The circuit from the help is used. See Using MAX485
'          TX    RX
' COM0    PD.1   PD.0   rs232 used for debugging
' COM1    PB.3   PB.2   rs485 used for MODBUS halve duplex
'            PB.1       data direction rs485


'configure the first UART for RS232
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,

'configure the second UAR for RS485/MODBUS. Make sure all slaves/servers use the sa
Config Com2 = 9600 , Synchrone = 0 , Parity = Even , Stopbits = 1 , Databits = 8 ,


'use OPEN/CLOSE for using the second UART
Open "COM2:" For Binary As #1

'dimension some variables
Dim B As Byte
Dim W As Word
Dim L As Long

W = &H4567                                                      'assign a value
L = &H12345678                                                  'assign a value


Print "RS-485 MODBUS master"
Do
   If Pinc.0 = 0 Then                                           ' test switch
      Waitms 500                                                ' delay
      Print "send request to slave/server"
     ' Send one of the following three messages
     ' Print #1 , Makemodbus(2 , 3 , 8 , 2);                    ' slave 2, function 3,
     ' Print #1 , Makemodbus(2 , 6 , 8 , W);                    ' slave 2, function 6,
      Print #1 , Makemodbus(2 , 16 , 8 , L);                    ' slave 2, function 16,
   End If
   If Ischarwaiting(#1) <> 0 Then                               'was something returned
       B = Waitkey(#1)                                          'then get it
       Print Hex(b) ; ",";                                      'print the info
   End If
Loop

End
```

7.135.1 PRINT

Action

Send output to the UART.
Writes a string to a file.
Writes data to a device.

Syntax
PRINT [#channel ,] var ; " constant"

Remarks

Var	The variable or constant to print.

You can use a semicolon (**;**) to print multiple variables or constants after each other. When you end a line with a semicolon, no linefeed and carriage return will be added.

The PRINT routine can be used when you have a RS-232 interface on your processor. The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator as an output device.
You can also use the build in terminal emulator.
When using RS-485 you can use CONFIG PRINT to set up a pin for the direction.
When printing arrays, you can only print one element at the time. When you need to print the content of a complete array, you need to use PRINTBIN 1504.

PRINT will automatic convert numeric variables into the string representation.
This means that when you have a byte variable named B with the value of 123, the numeric variable is converted into a string "123" and then printed.
In this case, print will print 3 characters or bytes. When you want to print the byte you can use the chr() function : print chr(b);
This will send just one byte to the UART.

You can connect the processors UART (TX/RX pins) to a MAX232, an FTDI232RL, a Bluetooth module or a GPS modem. Always check the logic level vcc of the UART and the device you connect to. Connecting 5V devices to a 3v3 device might damage the 3v3 device.
A serial port can be used to update firmware with a so called boot loader.

AVR-DOS
The AVR-DOS file system also supports PRINT. But in that case, only strings can be written to disk.
When you need to print to the second hardware UART, or to a software UART, you need to specify a channel : PRINT **#1,** "test"
The channel must be opened first before you can print to it. Look at OPEN 1386 and CLOSE 848 for more details about the optional channel. For the first hardware UART, there is no need to use channels. The default for PRINT without a channel specifier, is the first UART.
So : *PRINT " test"* will always use the first hardware UART.

Xmega-SPI
When sending data to the SPI interface, you need to activate the SS pin. Some chips might need an active low, others might need an active high. This will depends on the slave chip.
When you use the SS=AUTO option, the level of SS will be changed automatic. Thus SS is made low, then the data bytes are sent, and finally , SS is made high again.

For SPI, no CRLF will be sent. Thus a trailing **;** is not needed.

SPI Number of Bytes

The compiler will send 1 byte for variable which was dimensioned as a BYTE.
It will send 2 bytes for a WORD/INTEGER, 4 bytes for a LONG/SINGLE and 8 bytes for a DOUBLE.
As with all routines in BASCOM, the least significant Byte will be send first.

When you send a numeric constant, the binary value will be sent : 123 will be send a 1 byte with the value of 123.

If you send an array element, one element will be send.
With an optional parameter you can provide how many bytes must be sent. You must use a comma (**,**) to specify this parameter. This because the semi colon (;) is used to send multiple variables.

 The delimiter for sending multiple variables is a semi colon (;) while INPUT uses the comma (,) to separate multiple variables.

Sample

```
Dim Tmparray(5) As Byte, Spi_send_byte As Byte, W as Word
Config Spid = Hard, Master = Yes, Mode = 0, Clockdiv = Clk32, Data_order = Msb , Ss
Open "SPID" For Binary As #12
Print #12, Spi_send_byte; W          ' send ONE BYTE and 2 bytes of W
Print #12, Tmparray(1) , 2           ' send 2 bytes of tmparray, starting at element
Print #12, Tmparray(1)               ' send 1 byte
Print #12, Tmparray(3) , 2           ' send 2 bytes starting at index 3
Print #12, 123; 1000; Tmparray(1), B' send byte with value 123, 2 bytes with value
```

See also

INPUT 1493, OPEN 1386 , CLOSE 848 , SPC 1508 , PRINTBIN 1504 , HEX 833, BIN 824

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : print.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: PRINT, HEX
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space
```

```
Dim A As Byte , B1 As Byte , C As Integer , S As String * 4
A = 1
Print "print variable a " ; A
Print                                                    ' new line
Print "Text to print."                                   ' constant
to print

B1 = 10
Print Hex(b1)                                            ' print in
hexa notation
C = &HA000                                               ' assign
value to c%
Print Hex(c)                                             ' print in
hex notation
Print C                                                  ' print in
decimal notation

C = -32000
Print C
Print Hex(c)
Rem Note That Integers Range From -32767 To 32768

Print "You can also use multiple" _
; "lines using _"
Print "use it for long lines"
'From version 1.11.6.4 :
A = &B1010_0111
Print Bin(a)
S = "1001"
A = Binval(s)
Print A                                                  '9 dec
End
```

7.135.1 PRINTBIN

Action

Print binary content of a variable to the serial port.

Syntax

PRINTBIN var [; varn] [;varn [,bytes]]
PRINTBIN #channel, var [; varn] [;varn [,bytes]]

Remarks

Var	The variable which value is send to the serial port.
varn	Optional variables to send.
bytes	The number of bytes to send

The channel is optional and intended to be used with the OPEN 1386 statement.

PRINTBIN is equivalent to `PRINT CHR(byteVar);`
Notice that the PRINT line is ending with ; to suppress the CR+LF to be send.
When you use a Long for example, 4 bytes are printed.

Multiple variables may be sent. They must be separated by the **;** sign.
Just like PRINT the ; is used to separate the data. While INPUT/INPUTBIN uses a comma (,) to separate the data.

The number of bytes to send can be specified by an additional **numeric** constant. This is convenient when sending the content of an array.

```
Printbin ar(1) ; 3 ' will send 3 bytes from array ar(1) starting at index element 1.
Printbin ar(1) ; 2 ; ar(2) ; 4  ' will send 2 bytes from array ar1() starting at index 1, then 4 bytes from array ar() starting at index 2.
```

When you use `Printbin ar(1)` the whole array will be printed assuming that CONFIG BASE=1.
When you need to print the content of a big array(array with more then 255 elements) or with a data size that exceeds 255 bytes, you need to use the **CONFIG PRINTBIN** option.

Variable number of bytes

Since version 2082 you can use a variable to specify the number of bytes to send. In order to keep the syntax compatible with older compilers, you must use a comma followed by the number of bytes. The number of bytes can be either a numeric constant or a numerical integer value.

Printbin Z , 1 ; Ar(1 , 1) , Q
In this example we sent 1 byte of variable Z , followed by Q bytes from variable ar(). The number will depend on the value of the variable Q.

Another example:
Dim Array(10) As Byte, Bytes_to_send As Byte

Bytes_to_send = 8
Printbin Array(1) , Bytes_to_send ' this will send 8 bytes
Bytes_to_send = 6
Printbin Array(1) , Bytes_to_send ' this will send 6 bytes, you should use comma if number of bytes is specified with variable

RS-485

When the CONFIG PRINT 1028 option is used for RS-485, the direction pin will be used by PRINTBIN as well.
When RS-485 is used, the following will happen :
- the direction pin is toggled
- all variables are transmitted
- a check if performed to ensure the last byte is transmitted
- the direction pin is toggled again

See also

INPUTBIN 1497 , CONFIG PRINTBIN 1029 , CONFIG PRINT 1028, CONFIG INPUTBIN 984

Example

```
Dim A(10) As Byte, C As Byte
For C = 1 To 10
   A(c)= c ' fill array
Next
Printbin A(1)  'print content of a(1). Note that the whole array will be sent!
```

End

7.135.1 SERIN

Action
Reads serial data from a dynamic software UART.

Syntax
SERIN var , bts , port , pin, baud , parity , dbits , sbits

Remarks
While the OPEN and CLOSE statements can be used for software UARTS, they do not permit to use the same pin for input and output. The settings used when opened the communication channel can also not be changed at run time.

The SERIN and SEROUT statements are dynamic software UART routines to perform input and output. You can use them on the same pin for example send some data with SEROUT and get back an answer using SERIN.

Since the SERIN and SEROUT routines can use any pin and can use different parameter values, the code size of these routines is larger.

Parameter	Description
Var	A variable that will be assigned with the received data.
Bts	The number of bytes to receive. String variables will wait for a return (ASCII 13). There is no check if the variable you assign is big enough to hold the result.
Port	The name of the port to use. For example: portA.
Pin	The pin number you want to use of the port. This must be in the range from 0-7.
Baud	The baud rate you want to use. For example 19200.
Parity	A number that codes the parity. 0= NONE, 1 = EVEN, 2 = ODD
Dbits	The number of data bits. Use 7 or 8.
Sbits	The number of stop bits. 1 to 2.

The use of SERIN will create an internal variable named ___SER_BAUD. This is a LONG variable. It is important that you specify the correct crystal value with $CRYSTAL so the correct calculation can be made for the specified baud rate.

Note that ___SER_BAUD will not hold the passed baud rate but will hold the bit delay used internal.

Since the SW UART is dynamic you can change all the parameters at run time. For example you can store the baud rate in a variable and pass this variable to the SERIN routine.

Your code could change the baud rate under user control this way.

It is important to realize that software timing is used for the bit timing. Any interrupt that occurs during SERIN or SEROUT will delay the transmission. Disable interrupts while you use SERIN or SEROUT.

ASM

The routine called is named _serin and is stored in mcs.lib
For Xmega it is located in Xmega.lib and Xtiny in Xtiny.lib
For the baud rate calculation, _calc_baud is called.

See also

SEROUT 1509

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : serin_out.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstration of DYNAMIC software UART
'micro                    : AT90S2313
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'tip : Also look at OPEN and CLOSE

'some variables we will use
Dim S As String * 10
Dim Mybaud As Long
'when you pass the baud rate with a variable, make sure you dimesion it
as a LONG

Mybaud = 19200
Do
  'first get some data
  Serin S , 0 , PORTD , 0 , Mybaud , 0 , 8 , 1
  'now send it
  Serout S , 0 , PORTD , 1 , Mybaud , 0 , 8 , 1
  '                                       ^ 1 stop bit
  '                                    ^---- 8 data bits
  '                                 ^------ even parity (0=N, 1 = E, 2=O)
  '                           ^-------------- baud rate
  '                      ^-------------------- pin number
  '                 ^----------------------- port so PORTA.0 and PORTA.1
are used
  '            ^--------------------------- for strings pass 0
  '       ^------------------------------- variable
  Wait 1
Loop
End

'because the baud rate is passed with a variable in this example, you
```

```
could change it under user control
'for example check some DIP switches and change the variable mybaud
```

7.135.13 SPC

Action
Prints the number of specified spaces.

Syntax
PRINT **SPC**(x)
LCD **SPC**(x)

Remarks

X	The number of spaces to print.

Using 0 for x will result in a string of 255 bytes because there is no check for a zero length assign.

SPC can be used with LCD 657 too.

The difference with the SPACE function is that SPACE returns a number of spaces while SPC() can only be used with printing. Using SPACE() with printing is also possible but it will use a temporary buffer while SPC does not use a temporary buffer.

See also
SPACE 1533

Example
```
'-----------------------------------------------------------------------
--------
'copyright                : (c) 1995-2025, MCS Electronics
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'purpose                  : demonstrates DEG2RAD function

'-----------------------------------------------------------------------
--------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 40                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Dim S As String * 15 , Z As String * 15
Print "{" ; Spc(5) ; "}"                                    '{ }
Lcd "{" ; Spc(5) ; "}"                                      '{ }
```

7.135.14 SEROUT

Action
Sends serial data through a dynamic software UART.

Syntax
SEROUT var , bts , port , pin, baud , parity , dbits , sbits [,INVERTED]

Remarks
While the OPEN and CLOSE statements can be used for software UARTS, they do not permit to use the same pin for input and output. The settings used when opening the communication channel can also not be changed at run time.

The SERIN and SEROUT statements are dynamic software UART routines to perform input and output. You can use them on the same pin for example to send some data with SEROUT and get back an answer using SERIN.

Since the SERIN and SEROUT routines can use any pin and can use different parameter values, the code size of these routines is larger.

Parameter	Description
Var	A variable which content is send through the UART. A constant can NOT be used.
Bts	The number of bytes to send. For strings you can specify 0. In that case the whole string will be sent.
Port	The name of the port to use. For example : portA.
Pin	The pin number you want to use of the port. This must be in the range from 0-7.
Baud	The baud rate you want to use. For example 19200.
Parity	A number that codes the parity. 0= NONE, 1 = EVEN, 2 = ODD
Dbits	The number of data bits. Use 7 or 8.
Sbits	The number of stop bits. 1 to 2.
INVERTED	This is an optional parameter. When set, the signal will be inverted.

The use of SEROUT will create an internal variable named ___SER_BAUD. This is a LONG variable. It is important that you specify the correct crystal value with $CRYSTAL so the correct calculation can be made for the specified baud rate.

Note that ___SER_BAUD will not hold the passed baud rate but will hold the bit delay which is used internal.

Since the SW UART is dynamic you can change all the parameters at run time. For example you can store the baud rate in a variable and pass this variable to the SEROUT routine.

Your code could change the baud rate under user control this way.

It is important to realize that software timing is used for the bit timing. Any interrupt that occurs during SERIN or SEROUT will delay the transmission. Disable interrupts while you use SERIN or SEROUT.

 SEROUT can be used in PORT and open collector TRI-state mode.

PORT mode means that the defined PORT PIN will be set to output mode, and that the pin output will be switched between 0 and 1.
The disadvantage of this mode is that you can not connect multiple outputs together. (never connect 2 outputs together).
For this reason there is also the TRI state/open collector mode.

By default TRI-state mode will be used. This mode requires an external pull up resistor on the Xmega/Xtiny. For the normal AVR this external pull up resistor is optional.
Since the port architecture differs for all platforms there are different implementations to create the bit stream in open collector mode.
The normal AVR has a pull up resistor that can be activated by writing 1 to a port.
The Xmega has a special register to activate the pull up resistor. And the Xtiny has a special register as well to activate the pull up resistor. For all platforms, a zero bit is created by setting the data direction register to output mode and clear the output pin.
To create a one, the normal AVR is set to input mode and the pull up is activated by writing a one to data direction.
For the Xmega the code is similar but more code is required for the pinctrl register since each pin has its own register. The pull up mode is the wired and mode.

In Open Collector mode you can connect several AVR chip pin and poll the 'bus' with the SERIN statement.
When you want to use the pins in PORT OUTPUT mode, the pins can not be tied together.
Define a constant named **SEROUT_EXTPULL** with a value of 1 for the TRI-STATE open collector mode.
Define a constant named **SEROUT_EXTPULL** with a value of 0 to work in PORT mode.
When you do not define a constant the SEROUT_EXTPULL will be created automatically with a value of 1.

 The mode you chose is fixed and global for all SEROUT statements. You can not switch between SEROUT_EXTPULL value in your code dynamically.

ASM
The routine called is named _serout and is stored in mcs.lib. An overloaded version is placed in xmega.lib and xtiny.lib
For the baud rate calculation, _calc_baud is called.

See also
SERIN

Example
```
'-----------------------------------------------------------------------
-----------------
'name                     : serin_out.bas
```

```
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstration of DYNAMIC software UART
'micro                   : AT90S2313
'suited for demo         : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'tip : Also look at OPEN and CLOSE

'some variables we will use
Dim S As String * 10
Dim Mybaud As Long
'when you pass the baud rate with a variable, make sure you dimesion it
as a LONG

Mybaud = 19200
Do
  'first get some data
  Serin S , 0 , PORTD , 0 , Mybaud , 0 , 8 , 1
  'now send it
  Serout S , 0 , PORTD , 1 , Mybaud , 0 , 8 , 1
  '                                       ^ 1 stop bit
  '                                    ^---- 8 data bits
  '                                 ^------ even parity (0=N, 1 = E, 2=O)
  '                           ^------------- baud rate
  '                     ^-------------------- pin number
  '                ^----------------------- port so PORTA.0 and PORTA.1
are used
  '           ^--------------------------- for strings pass 0
  '      ^-------------------------------- variable
  Wait 1
Loop
End

'because the baud rate is passed with a variable in this example, you
could change it under user control
'for example check some DIP switches and change the variable mybaud
```

7.135.1 WAITKEY

Action

Wait until a character is received.

Syntax

var = **WAITKEY**()
var = **WAITKEY**(#channel)

Remarks

var	Variable that receives the ASCII value of the serial buffer. Can be a numeric variable or a string variable.
#channel	The channel used for the software UART.

While Inkey() returns a character from the serial buffer too, INKEY() continues when there is no character. Waitkey() waits until there is a character received. This blocks your program.

See also

INKEY 1492 , ISCHARWAITING 1498 , $TIMEOUT 708

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : inkey.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: INKEY , WAITKEY
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte , S As String * 2
Do
   A = Inkey()                                             'get ascii
value from serial port
   's = Inkey()
   If A > 0 Then                                           'we got
something
      Print "ASCII code " ; A ; " from serial"
   End If
Loop Until A = 27                                          'until ESC
is pressed

A = Waitkey()                                              'wait for a
key
's = waitkey()
Print Chr(a)

'wait until ESC is pressed
Do
Loop Until Inkey() = 27
```

```
'When you need to receive binary data and the bibary value 0 ,
'you can use the IScharwaiting() function.
'This will return 1 when there is a char waiting and 0 if there is no
char waiting.
'You can get the char with inkey or waitkey then.
End
```

7.136 SPI

7.136.1 SPIIN

Action
Reads a value from the SPI-bus.

Syntax
SPIIN var, bytes

Syntax SPI1
SPI1IN var, bytes

Remarks

Var	The variable which receives the value read from the SPI-bus.
Bytes	The number of bytes to read. The maximum is 255.

In order to be able to read data from the SPI slave, the master need to send some data first. The master will send the value 0.
SPI is a 16 bit shift register. Thus writing 1 byte will cause 1 byte to be clocked out of the device which the SPIIN will read.

SPIIN always work on the first SPI interface (SPI0)
SPI1IN works on the second SPI interface (SPI1)

See also
SPIOUT 1518, SPIINIT 1514, CONFIG SPI 1061 , SPIMOVE 1515 , SPI1 1519

Example
```
'-----------------------------------------------------------------------
------------------
'name                     : spi.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo :SPI
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
```

```
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Dim B As Byte
Dim A(10) As Byte

Spiinit
B = 5
Spiout A(1) , B

Spiin A(1) , B

A(1) = Spimove(a(2))
End
```

7.136.2 SPIINIT

Action
Initiate the SPI pins.

Syntax
SPIINIT

Syntax SPI1
SPI1INIT

Remarks
After the configuration of the SPI pins, you must initialize the SPI pins to set them for the right data direction. When the pins are not used by other hardware/software, you only need to use SPIINIT once.
When the SPI bus is used in master mode, the MOSI, CLOCK and SS pins will be set to output.
When the SPI bus is used in slave mode, the MISO is set to output mode.

If you need to change the logic levels of the SPI pins, you need to disable the SPI. You can do this by setting the SPE bit to 0 in SPCR.

When other routines change the state of the SPI pins, use SPIINIT again before using SPIIN and SPIOUT.

SPIINIT is only required for normal AVR. Xmega and Xtiny do not require this statement.

SPIINIT always work on the first SPI interface (SPI0)
SPI1INIT works on the second SPI interface (SPI1)

See also
SPIIN 1513 , SPIOUT 1518, config spi 1061 , SPI1 1519

ASM
Calls _init_spi

Example
See SPIIN 1513

7.136.3 SPIMOVE

Action
Sends and receives a value or a variable to the SPI-bus.

Syntax
var = **SPIMOVE**(source [,count])

Syntax Xmega
var = **SPIMOVE**(source ,count , channel)

Syntax SPI1
var = **SPI1MOVE**(source [,count])

Remarks

Var	The variable that is assigned with the received byte(s) from the SPI-bus.
Source	The variable or constant whose content must be send to the SPI-bus.
Count	Optional byte value which specifies how many bytes need to be moved. Notice that for Xmega this parameter is not optional but mandatory.
Channel	For Xmega only : the channel number or channel variable

SPIMOVE always work on the first SPI interface (SPI0)
SPI1MOVE works on the second SPI interface (SPI1)

See also
SPIIN 1513 , SPIINIT 1514 , CONFIG SPI 1061 , SPI1 1519

Example
```
Config Spi = Soft , Din = Pinb.0 , Dout = Portb.1 , Ss = Portb.2 , Clock
= Portb.3

Spiinit

Dim a(10) as Byte , X As Byte
```

```
Spiout A(1) , 5                                                    'send 5
bytes
Spiout X , 1                                                       'send 1 byte
A(1) = Spimove(5)                                                  ' move 5 to
SPI and store result in a(1)
A(1) = Spimove(a(2),4)                                             ' move 4
bytes from a(2) to a(1)
End
```

Example Xmega

```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                       xm128A1_SPI_MOVE.bas
'  This sample demonstrates the Xmega128A1 SPI master mode
SPIMOVE
'-----------------------------------------------------------------
-

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40

Config Osc = Enabled , 32mhzosc = Enabled
Config Sysclock = 32mhz                                            '-->
32MHz


Config Com1 = 57600 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8
Waitms 2
Open "COM1:" For Binary As #1
Print #1 ,
Print #1 , "------------SPI MASTER-Slave Test----------------"

' Master = ATXMEGA128A1 running at 3.3 Volt
' Slave = ATMEGA328P running at 3.3 Volt

'We use Port E for SPI
'Ddre = &B1011_0000
'Bit7 = SCK = Output  ------> SCK ATMEGA328P    (PinB.5)
'Bit6 = MISO = Input  ------> MISO ATMEGA328P   (PinB.4)
'Bit5 = MOSI = Output ------> MOSI ATMEGA328P   (PinB.3)
'Bit4 = SS = Output   ------> SS ATMEGA328P     (PinB.2)
Slave_select Alias Porte.4
Set Slave_select

Dim Switch_bit As Bit
```

```
Switch Alias Pine.0                                                    '
Switch connected to GND
Config Xpin = Porte.0 , Outpull = Pullup



Dim Bspivar As Byte
Dim Spi_send_byte As Byte
Dim Spi_receive_byte As Byte
Dim Ar(4) As Byte


'SPI, Master|Slave , MODE, clock division
Config Spie = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk32 ,
Data_order = Msb , Ss = Auto
'SS = Auto set the Slave Select (SS) automatically before a print
#X or input #X command  (including initialization of the pin)
'Master SPI clock = 1MHz
Open "SPIE" For Binary As #12


Config Debounce = 50

Do

   Debounce Switch , 0 , Switch_sub , Sub
'Switch Debouncing

   If Switch_bit = 1 Then
'When Switch pressed
      Reset Switch_bit

      Incr Spi_send_byte
      Print "Spi_send_byte = " ; Spi_send_byte

   'SEND TO SLAVE
      Print #12 , Spi_send_byte
'SEND ONE BYTE TO SLAVE

      Waitms 3

  'READ FROM SLAVE
      Input #12 , Spi_receive_byte
'READ ONE BYTE FROM SLAVE

      Print #1 , "Spi_receive_byte = " ; Spi_receive_byte

      'Lets move some bytes
```

```
      Ar(1) = Spimove(ar(1) , 4 , #12)
   End If


Loop



End                                                    'end
program

'there is NO CLOSE for SPI


Switch_sub:
   Set Switch_bit
Return
```

7.136.4 SPIOUT

Action
Sends a value of a variable to the SPI-bus.

Syntax
SPIOUT var , bytes

Syntax SPI1
SPI1OUT var , bytes

Remarks

var	The variable whose content must be send to the SPI-bus.
bytes	The number of bytes to send. Maximum value is 255.

When SPI is used in HW(hardware) mode, there might be a small delay/pause after each byte that is sent. This is caused by the SPI hardware and the speed of the bus. After a byte is transmitted, SPSR bit 7 is checked. This bit 7 indicates that the SPI is ready for sending a new byte.
SPIOUT will always work on the first SPI interface (SPI0).
SPI1OUT will work on the second SPI interface (SPI1)

See also
SPIIN 1513 , SPIINIT 1514 , CONFIG SPI 1061 , SPIMOVE 1515 , SPI1 1519

Example
```
Dim A(10) As Byte
Config Spi = Soft , Din =Pinb.0 , Dout =Portb.1 , Ss =Portb.2 , Clock =
Portb.3
Spiinit
```

```
Spiout A(1), 4 'write 4 bytes a(1), a(2) , a(3) and a(4)
End
```

7.136.5 SPI1INIT, SPI1IN, SPI1OUT, SPI1MOVE

Some of the new MEGA processors like ATMEGA328**PB** have a second SPI bus. This is not a USART that can work in SPI mode but a full SPI bus.

In order to use the second SPI which is named SPI1, you have to add a '1' to the SPI commands :

CONFIG SPI1 1061
SPI1INIT 1514
SPI1IN 1513
SPI1OUT 1518
SPI1MOVE 1515

The statements above link to the description of the SPI statements (SPI0).

```
'in this demo we only use the second SPI interface
Config Spi1 = Hard , Interrupt = Off , Data_order = Msb , Master = Yes
, Polarity = Low , Phase = 0 , Clockrate = 128

'second SPI
Spi1init
B = 5
Spi1out A(1) , B
Spi1in A(1) , B
A(1) = Spi1move(a(2))
```

Some XTINY processors also have a second SPI bus. They also support the BASCOM SPI commands.

7.137 STRINGS

7.137.1 CHARPOS

Action

Returns the position of a single character in a string.

Syntax

pos = **CHARPOS**(string , search [,start [,SAFE]])

Remarks

Pos	Numeric variable that will be assigned with the position of the sub string in the string. Returns 0 when the sub string is not found.
String	The string to search.
Search	The search string. This can be a numeric variable too. For example a byte. When a string is used, only the first character will be used for the search.
Offset	An optional start position where the searching must start.

SAFE	If you specify an offset, Charpos will check if the offset is not located after the string. For example , when the string is "abc" and you specify an offset of 10, it will be located after the string. The SAFE option is default. When you specify SPEED, the compiler will add the offset without checking. This will result in shorter and quicker code.

No constant can be used for string it must be a string variable.

 The search is sensitive to case.
CHARPOS supports $BIGSTRINGS 611

See also

SPLIT 1534 , INSTR 1526 , REPLACECHARS 1530 , DELCHAR 1520 , INSERTCHAR 1522 , DELCHARS 1521

Example

```
'-----------------------------------------------------------------------
--------
'                       charpos.bas
'               (c) 1995-2025  MCS Electronics
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
'-----------------------------------------------------------------------
--------
Dim S As String * 20
Dim Bpos As Byte
Dim Z As String * 1

Z = "*"
Do
  Input "S:" , S
  Bpos = Charpos(s , Z)
  Print Bpos
Loop Until S = ""


Do
  Input "S:" , S
  Bpos = Charpos(s , "A")                                    ' notice
charpos is sensitive to case
  Print Bpos
Loop
```

7.137.2 DELCHAR

Action

Delete one character from a string.

Syntax

DELCHAR string, pos

Remarks

string	The string where the character is removed from.
pos	The position where the character must be removed from. A value of 1 would remove the first character.

Do not confuse with the DELCHAR**S** statement which removes all characters based on a character value.

The DELCHAR removes one character from a string based on an index.
DELCHAR supports $BIGSTRINGS 611

See also
DELCHARS 1521 , INSERTCHAR 1522 , INSTR 1526 , MID 1530 , CHARPOS 1519 , REPLACECHARS 1530

Example
```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                   del_insert_chars.bas
'  This sample demonstrates the delchar, delchars and insertchar
statements
'-----------------------------------------------------------------
$regfile="m88def.dat"
$crystal = 8000000
$hwstack = 40
$swstack = 40
$framesize = 40

dim s as string * 30
s = "This is a test string" ' create a string
delchar s, 1                ' remove the first char
print s                     ' print it

insertchar s,1, "t"         ' put a small t back
print s

delchars s,"s"              ' remove all s
print s
end
```

7.137.3 DELCHARS

Action
Delete all character from a string matching the provided character value.

Syntax
DELCHARS string, value

Remarks
string	The string where the characters are removed from.
value	The value of the character which must be removed from the string. You can use "A" to remove all capital A characters. Or you can pass a byte with the value of 65 to remove all characters with ASCII value 65 (A)

Do not confuse with the DELCHAR statement which removes one character based on an index value.

DELCHARS removes ALL characters from a string matching value.
DELCHARS also works with $BIGSTRINGS 611

See also
DELCHAR 1520 , INSERTCHAR 1522 , INSTR 1526 , MID 1530 , CHARPOS 1519 , REPLACECHARS 1530

Example
```
'-----------------------------------------------------------------
'                  (c) 1995-2025, MCS
'                   del_insert_chars.bas
'  This sample demonstrates the delchar, delchars and insertchar
statements
'-----------------------------------------------------------------
$regfile="m88def.dat"
$crystal = 8000000
$hwstack = 40
$swstack = 40
$framesize = 40

dim s as string * 30
s = "This is a test string" ' create a string
delchar s, 1                ' remove the first char
print s                     ' print it

insertchar s,1, "t"         ' put a small t back
print s

delchars s,"s"              ' remove all s
print s
end
```

7.137.4 INSERTCHAR

Action
Inserts one character into a string.

Syntax
INSERTCHAR string, pos, char

Remarks

string	The string where the character is inserted to.
pos	The position where the character is inserted to. A value of 1 would make the character the first character of the string.
char	A byte or string or string constant with the character that need to be inserted. For example you can use "A" to insert an "A", or use a byte with the value 65 to insert an "A". Or use a string. In case of a string, only the first character will be used.

INSERTCHAR supports $BIGSTRINGS 611

See also

DELCHAR 1520 , DELCHARS 1521 , INSTR 1526 , MID 1530 , CHARPOS 1519 , REPLACECHARS 1530

Example

```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                    del_insert_chars.bas
'  This sample demonstrates the delchar, delchars and insertchar
statements
'-----------------------------------------------------------------
-
$regfile="m88def.dat"
$crystal = 8000000
$hwstack = 40
$swstack = 40
$framesize = 40

dim s as string * 30
s = "This is a test string" ' create a string
delchar s, 1                ' remove the first char
print s                     ' print it

insertchar s,1, "t"         ' put a small t back
print s

delchars s,"s"              ' remove all s
print s
end
```

Example

```
'-----------------------------------------------------------------
----------------
'name                    : str-test.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : demonstrates some string routines
'micro                   : mega4809
'suited for demo         : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------
$regfile = "mx4809.dat"
$crystal = 20000000
$hwstack = 40
$swstack = 40
$framesize = 40

Config Sysclock = 16_20mhz , Prescale = 1                    'set
clock freq
```

```
Dim S1 As String * 10
Dim S2 As String * 10
Dim S4 As String * 80

S1 = "0123456789"
S2 = "abcdefghij"
Mid(s1 , 3 , 2) = "##"
'replace
Mid(s1 , 13 , 2) = "**"                                    'try
to do at an illegal position
Mid(s1 , 0 , 2) = "**"                                     'try
to do at an illegal position

S1 = "0123456789"
S2 = "abcdefghij"

Mid(s1 , 3 ) = "#"
'replace
Mid(s1 , 13 ) = "*"
'invalid

S1 = "---"
Mid(s1 , 1) = "ABC"

S4 = "abcdefghijklm"
Insertchar S4 , 0 , "*"
Insertchar S4 , 1 , "*"
Insertchar S4 , 20 , "*"

Delchar S4 , 0
Delchar S4 , 1
Delchar S4 , 20

End
```

7.137.5 JOIN

Action

The JOIN function returns a string from a string array

Syntax

target = **JOIN**(source(start) ,elements [,glue])

Remarks

target	The string that is assigned. You need to make sure that this string is dimensioned large enough to hold the content.
source	The source string array

start	The starting position within the string array
elements	The number of elements to process
glue	This is an optional byte which you can use to glue the elements together. For example a space, or a dot

The SPLIT 1534() function can split up a string into elements. The JOIN() function does the exact opposite : it creates a string out of a string array.

See also

SPLIT 1534

Example

```
'-----------------------------------------------------------------
---------------
'name                        : join.bas
'copyright                   : (c) 1995-2025, MCS Electronics
'purpose                     : demonstrates JOIN function
'micro                       : M88
'suited for demo             : no
'commercial addon needed     : no
'-----------------------------------------------------------------
---------------

$regfile = "m88def.dat"

$crystal = 8000000
$hwstack = 40
$swstack = 40
$framesize = 40


Config Com1 = 115200 , Parity = None , Databits = 8 , Stopbits =
1


Dim Ar(10) As String * 20
Dim S1 , S2 As String * 80
Dim Cnt As Byte

S1 = "this.is.a.test"
Cnt = Split(s1 , Ar(1) , ".")

S2 = Join(ar(1) , 3)
Print S2

S2 = Join(ar(1) , 3 , ".")
Print S2
```

```
End
```

7.137.6 INSTR

Action
Returns the position of a sub string in a string.

Syntax
var = **INSTR**(start , string , substr)
var = **INSTR**(string , substr)

Remarks

Var	Numeric variable that will be assigned with the position of the sub string in the string. Returns 0 when the sub string is not found. When used with $BIGSTRINGS, the target variable should be a word instead of a byte.
Start	An optional numeric parameter that can be assigned with the first position where must be searched in the string. By default (when not used) the whole string is searched starting from position 1.
String	The string to search.
Substr	The search string.

No constant can be used for *string* it must be a string variable.
Only *substr* can be either a string or a constant.

INSTR supports $BIGSTRINGS 611

See also
SPLIT 1534 , CHARPOS 1519

Example
```
'-------------------------------------------------------------------------
-----------------
'name                     : instr.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : INSTR function demo
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
```

```
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

'dimension variables
Dim Pos As Byte
Dim S As String * 8 , Z As String * 8

'assign string to search
S = "abcdeab"                                                ' Z = "ab"

'assign search string
Z = "ab"

'return first position in pos
Pos = Instr(s , Z)
'must return 1

'now start searching in the string at location 2
Pos = Instr(2 , S , Z)
'must return 6


Pos = Instr(s , "xx")
'xx  is not in the string so return 0
End
```

7.137.7 LCASE

Action

Converts a string in to all lower case characters.

Syntax

Target = **LCASE**(source)

Remarks

Target	The string that is assigned with the lower case string of string target.
Source	The source string.

LCASE supports $BIGSTRINGS 611

See also

UCASE 1537

ASM

The following ASM routines are called from MCS.LIB : _LCASE
The generated ASM code : (can be different depending on the micro used)

```
;##### Z = Lcase(s)
Ldi R30,$60
Ldi R31,$00 ; load constant in register
Ldi R26,$6D
Rcall _Lcase
```

Example

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space


Dim S As String * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

7.137.8 LEFT

Action

Return the specified number of leftmost characters in a string.

Syntax

var = **LEFT**(var1 , n)

Remarks

Var	The string that is assigned.
Var1	The source string.
n	The number of characters to get from the source string.

LEFT supports $BIGSTRINGS 611

See also

RIGHT 1531 , MID 1530

Partial Example

```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Z = Left(s , 5)
Print Z                                                    'ABCDE
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

7.137.9 LEN

Action
Returns the length of a string.

Syntax
var = **LEN**(string)

Remarks

var	A numeric variable that is assigned with the length of string.
string	The string to calculate the length of.

Strings can be maximum 254 bytes long.
When using $BIGSTRINGS, the string can be as long as 65534 bytes. This depends on the available memory.
LEN supports $BIGSTRINGS 611

See Also
VAL 839

Partial Example
```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Print Len(s)
```

7.137.10 LTRIM

Action
Returns a copy of a string with leading blanks removed

Syntax
var = **LTRIM**(org)

Remarks

Var	String that receives the result.
Org	The string to remove the leading spaces from

LTRIM supports $BIGSTRINGS 611

See also
RTRIM 1532 , TRIM 1536

ASM
NONE

Partial Example
```
Dim S As String * 6
```

```
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

7.137.11 MID

Action

The MID function returns part of a string (a sub string).
The MID statement replaces part of a string variable with another string.

Syntax

var = **MID**(var1 ,st [, l])
MID(var ,st [, l]) = var1

Remarks

var	The string that is assigned.
Var1	The source string.
st	The starting position.
l	The number of characters to get/set.

Both MID statement and MID function support $BIGSTRINGS 611.
In version 2085 the code has been rewritten to be more efficient and safe. The safety comes with a small penalty.
When you provide an offset the old code would simply add this offset to the string address. This will work out fine except when the string is smaller than specified.

See also

LEFT 1528 , RIGHT 1531

Example

```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Z = Left(s , 5)
Print Z                                                    'ABCDE
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

7.137.12 REPLACECHARS

Action

Replace all occurrences of a character in a string by a different character.

Syntax

REPLACECHARS string , old,new

Remarks

string	A string variable.
old	A character or byte with the ASCII value of the character to search for.
new	A character of byte with the ASCII value with the new value.

When we have a string with a content of : "abcdefabc" and we want to replace the "a" by an "A" we can use :
Replacechars string , "a" , "A"

All occurrences are replaced.

REPLACECHARS supports $BIGSTRINGS 611

See also

INSTR 1526 , MID 1530 , CHARPOS 1519 , DELCHAR 1520 , INSERTCHAR 1522 , DELCHARS 1521

Example

```
$regfile = "m644def.DAT"
$hwstack = 24                                              'default use
32 for the hw stack
$swstack = 24                                              ' default
use 10 for the SW stack
$framesize = 24                                            ' default
use 40 for the frame

Dim Textout As String * 22
Dim Var As String * 1

Textout = "abcdefabdef"
Replacechars Textout , "a" , "A"
Print Textout

Var = "e"
Replacechars Textout , Var , "A"
Print Textout

End
```

7.137.13 RIGHT

Action

Return a specified number of rightmost characters in a string.

Syntax

var = **RIGHT**(var1 ,n)

Remarks

var	The string that is assigned.
Var1	The source string.
st	The number of bytes to copy from the right of the string.

RIGHT supports $BIGSTRINGS 611

See also
LEFT 1528 , MID 1530

Example
```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Z = Left(s , 5)
Print Z                                                    'ABCDE
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

7.137.14 QUOTE

Action
The Quote function will return a string surrounded by quotes.

Syntax
var = **QUOTE**(Source)

Remarks

Var	A string variable that is assigned with the quoted string of variable source.
Source	The string or string constant to be quoted.

The Quote() function can be used in HTML web server pages.

QUOTE supports $BIGSTRINGS 611

See also
NONE

Example
```
Dim S as String * 20
S = "test"
S = Quote(s)
Print S  ' would print "test"
End
```

7.137.15 RTRIM

Action
Returns a copy of a string with trailing blanks removed

Syntax
var = **RTRIM**(org)

Remarks

var	String that is assigned with the result.
org	The string to remove the trailing spaces from

RTRIM supports $BIGSTRINGS 611

See also

TRIM 1536 , LTRIM 1529

ASM

NONE

Example

```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

7.137.16 SPACE

Action

Returns a string that consists of spaces.

Syntax

var = **SPACE**(x)

Remarks

X	The number of spaces. This must be a value > 0
Var	The string that is assigned.

In version 2085 the passed value is tested for 0. Since zero is not allowed, the resulting string will be unaltered when using 0.
SPACE supports $BIGSTRINGS 611

See also

STRING 1536 , SPC 1508

Example

```
'-------------------------------------------------------------------------
--------
'copyright               : (c) 1995-2025, MCS Electronics
'micro                   : Mega48
'suited for demo         : yes
'commercial addon needed  : no
'purpose                 : demonstrates DEG2RAD function
```

```
'-------------------------------------------------------------------------
--------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 40                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space


Dim S As String * 15 , Z As String * 15
S = Space(5)
Print " {" ; S ; " }"                                      '{ }

Dim A As Byte
A = 3
S = Space(a)
End
```

7.137.17 SPLIT

Action

Split a string into a number of array elements.

Syntax

count = **SPLIT** (source, array(idx), search)

Remarks

count	The number of elements that SPLIT() returned. When the array is not big enough to fill the array, this will be the maximum size of the array. So make sure the array is big enough to hold the results.
source	The source string or string constant to search for.
array(idx)	The index of the first element of the array that will be filled. Notice that arrays are global.
search	The character to search for. This can be a string or string constant or a byte with the ASCII value.

When you use the serial port to receive data, in some cases you need to process the data in parts.
For example when you need to split an IP number as "123.45.24.12" you could use INSTR() or you can use SPLIT().
You must DIM the array yourself. The content of the array will be overwritten.

It is also important to know that the individual elements of the array need to be big enough to store the string part.
For example when the array has 5 elements and each element may be 10 characters long, a string that is 11 bytes long will not fit. Another element will be used in that case to store the additional info.

The SPLIT function takes care not to overwrite other memory. So when you split "1.2.2.2.2.2.2.3.3.3" into an array of 3 elements, you will loose the data.
If empty data is encountered, an empty element will be created. Thus "1,2,3,,5" will create 5 elements. Element 4 will be empty.

See also

INSTR 1526 , CHARPOS 1519 , JOIN 1524

Example

```
'-----------------------------------------------------------
'                        mega48.bas
'                     mega48 sample file
'                  (c) 1995-2025, MCS Electronics
'-----------------------------------------------------------
$regfile = "m48def.dat"
$crystal = 8000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As String * 80
Dim Ar(5) As String * 10
Dim Bcount As Byte

'The split function can split a string or string constant into elements
'It returns the number of elements
'You need to take care that there are enough elements and that each
element is big enough
'to hold the result
'When a result does not fit into 1 element it will be put into the next
element
'The memory is protected against overwriting.

S = "this is a test"

Bcount = Split( "this is a test" , Ar(1) , " ")
'bcount will get the number of filled elements
'ar(1) is the starting address to use
'" " means that we check for a space

'When you use "  aa"   , the first element will contain a space
Bcount = Split( "thiscannotfit! into the element" , Ar(1) , " ")

Dim J As Byte
For J = 1 To Bcount
  Print Ar(j)
Next

'this demonstrates that your memory is safe and will not be overwritten
when there are too many string parts
Bcount = Split( "do not overflow the array please" , Ar(1) , " ")

For J = 1 To Bcount
  Print Ar(j)
Next
End
```

7.137.18 STRING

Action
Returns a string consisting of m repetitions of the character with ASCII Code n.

Syntax
var = **STRING**(m ,n)

Remarks

Var	The string that is assigned.
N	The ASCII-code that is assigned to the string.
M	The number of characters to assign. This must be a number > 0

Since a string is terminated by a 0 byte, you can't use 0 for n. That is you could but the result would be an empty string.
Using 0 for x will not alter the resulting string.

STRING supports $BIGSTRINGS 611

See also
SPACE 1533

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 40                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space


Dim S As String * 15
S = String(5 , 65)
Print S                                                    'AAAAA
End
```

7.137.19 TRIM

Action
Returns a copy of a string with leading and trailing blanks removed

Syntax
var = **TRIM**(org)

Remarks

Var	String that receives the result.
Org	The string to remove the spaces from

TRIM is the same as an LTRIM() and RTRIM() call. It will remove the spaces on the left and right side of the string.
TRIM supports $BIGSTRINGS 611

See also
RTRIM 1532 , LTRIM 1529

Partial Example
```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

7.137.2 UCASE

Action
Converts a string in to all upper case characters.

Syntax
Target = **UCASE**(source)

Remarks
Target	The string that is assigned with the upper case string of string target.
Source	The source string.

UCASE supports $BIGSTRINGS 611

See also
LCASE 1527

ASM
The following ASM routines are called from MCS.LIB : _UCASE
X must point to the target string, Z must point to the source string.
The generated ASM code : (can be different depending on the micro used)
```
;##### Z = Ucase(s)
Ldi R30,$60
Ldi R31,$00 ; load constant in register
Ldi R26,$6D
Rcall _Ucase
```

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
```

```
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


Dim S As String * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

7.138 START

Action

Start the specified hardware source.

Syntax

START device [, cfg]

Remarks

Device	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power), ADC(A/D converter power) or DAC(D/A converter).
XMEGA	For the Xmega you can also specify : DACA or DACB for the Digital/ Analog Converters A and B. ADCA and ADCB for the A/D converters. For the timers you can use TCC0, TCC1, TCD0, TCD1, TCE0, TCE1, TCF0 and TCF1. To start a DMA soft transfer, you can use DMACH**0**, DMACH**1**, DMACH**2** or DMACH**3**. The transfer starts after the DMA channel is ready. For Xmega with Enhanced DMA, use **E**DMACH**0**, **E**DMACH**1**, **E**DMACH**2** and **E**DMACH**3**.
cfg	The optional cfg is only used for the TIMER when the optional CONFIGURATION is used. If **CONFIG TIMERx = option , CONFIGURATION=mysetting** was used, you would specify START TIMERx, mysetting.

When you configure a timer (CONFIG TIMER), the TIMER is started automatically when a pre-scaler value is provided.
When you want to halt the timer you can use the STOP TIMER statement. To start the timer after it has been stopped, you can use the START TIMER statement. The START TIMER statement will only work correctly when you have selected a clock source or pre-scaler value with the CONFIG TIMER statement.

When you stored settings using the option CONFIGURATION=setting , then you can specify which configuration the timer must use by providing the setting name as a parameter : START TIMER1 , mysetting

When a timer is used in interrupt mode, it must be running otherwise the interrupt will never occur.
TIMER0 and COUNTER0 are the same device. And so are TIMER1 and COUNTER1.

The AC, ADC and DAC parameters will switch power to the device and thus enabling it to work.
The WATCHDOG parameter will activate the Watchdog.

See also

STOP 1544

Example

```
'-----------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
or M163 micro
'micro                    : Mega163
'suited for demo          : yes
'commercial addon needed  : no
'use in simulator         : possible
' Getadc() will also work for other AVR chips that have an ADC converter
'-----------------------------------------------------------------------
---------
$regfile = "m163def.dat"                                    ' we use the
M163
$crystal = 4000000

$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space


'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc

'With STOP ADC, you can remove the power from the chip
'Stop Adc

Dim W As Word , Channel As Byte

Channel = 0
'now read A/D value from channel 0
Do
```

```
  W = Getadc(channel)
  Print "Channel " ; Channel ; " value " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
Loop
End

'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
'OFF      : AREF, internal reference turned off
'AVCC     : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin

'Using the additional param on chip that do not have the internal
reference will have no effect.
```

7.139 STCHECK

Action

Calls a routine to check for various stack overflows. This routine is intended for debug purposes.

Syntax

STCHECK

Remarks

The different stack spaces used by BASCOM-AVR lead to lots of questions about them. The STCHECK routine can help to determine if the stack size are trashed by your program. The program STACK.BAS is used to explain the different settings.

Note that STCHECK should be removed form your final program. That is once you tested your program and found out is works fine, you can remove the call to STCHECK since it costs time and code space.

The settings used are :

```
Hwstack = 8
Softstack = 2
Framesize = 14
```

Below is a part of the memory of the 90S2313 used for the example:

```
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
FR FR FR FR FR FR FR FR
FR FR FR FR FR FR YY YY SP SP SP SP SP SP SP SP
```

Since the last memory in SRAM is DF, the hardware stack is occupied by D8-DF(8 bytes)
When a call is made or a push is used the data is saved at the position the hardware stack pointer is pointing to. After this the stack pointer is decreased.
A call uses 2 bytes so SP will be SP-2. (DF-2) =DD
When 8 bytes are stored the SP will point to D7. Another call or push will thus destroy memory position D7 which is occupied by the soft stack.

The soft stack begins directly after the hardware stack and is also growing down.

The Y pointer(r28+r29) is used to point to this data.

Since the Y pointer is decreased first and then the data is saved, the pointer must point at start up to a position higher. That is D8, the end of the hardware space.

St -y,r24 will point to D8-1=D7 and will store R24 at location D7.
Since 2 bytes were allocated in this example we use D7 and D6 to store the data.
When the pointer is at D6 and another St -y,r24 is used, it will write to position D5 which is the end of the frame space that is used as temporarily memory.

The frame starts at C8 and ends at D5. Writing beyond will overwrite the soft stack.
And when there is no soft stack needed, it will overwrite the hardware stack space.
The map above shows FR(frame), YY(soft stack data) and SP(hardware stack space)

How to determine the right values?

The stack check routine can be used to determine if there is an overflow.

It will check :
-if SP is below it's size. In this case below D8.
-if YY is below it's size in this case when it is D5
-if the frame is above its size in this case D6

When is YY(soft stack) used? When you use a LOCAL variable inside a SUB or function. Each local variable will use 2 bytes.
When you pass variables to user Subroutines or functions it uses 2 bytes for each parameter.
`call mysub(x,y)` will use 2 * 2 = 4 bytes.
`local z as byte` ' will use another 2 bytes

This space is freed when the routine ends.
But when you call another sub inside the sub, you need more space.

```
sub mysub(x as byte,y as byte)
     call testsub(r as byte) ' we must add another 2 bytes
```

When you use empty(no params) call like :

`call mytest()` , No space is used.

When do you need frame space?
When ever you use a num<>string conversion routine like:

Print b (where b is a byte variable)

Bytes will use 4 bytes max (123+0)
Integer will use 7 bytes max (-12345+0)c
Longs will use 16 bytes max
And the single will use 24 bytes max

When you add strings and use the original the value must be remembered by the compiler.

Consider this :
```
s$ = "abcd" + s$
```

Here you give s$ a new value. But you append the original value so the original value must be remembered until the operation has completed. This copy is stored in the frame too.

So when string s$ was dimmed with a length of 20, you need a frame space of 20+1 (null byte)

When you pass a variable by VALUE (BYVAL) then you actually pass a copy of the variable.
When you pass a byte, 1 byte of frame space is used, a long will take 4 bytes.
When you use a LOCAL LONG, you also need 4 bytes of frame space to store the local long.

The frame space is reused and so is the soft stack space and hardware stack space.
So the hard part is to determine the right sizes!

The stack check routine must be called inside the deepest nested sub or function.

```
Gosub test

test:
   gosub test1
return


test1:
' this is the deepest level so check the stack here
   stcheck
return
```

Stcheck will use 1 variable named ERROR. You must dimension it yourself.

```
Dim Error As Byte
```

Error will be set to :

1: if hardware stack grows down into the soft stack space
2: if the soft stack space grows down into the frame space
3: if the frame space grows up into the soft stack space.

The last 2 errors are not necessarily bad when you consider that when the soft stack is not used for passing data, it may be used by the frame space to store data.
Confusing right.?

 It is advised to use the simpler DBG/$DBG method. This requires that you can simulate your program.

ASM

Routines called by STCHECK :
_StackCheck : uses R24 and R25 but these are saved and restored.

Because the call uses 2 bytes of hardware stack space and the saving of R24 and R25

also costs 2 bytes, it uses 4 more bytes of hardware stack space than your final program would do that f course does not need to use STCHECK.

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : stack.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows how to check for the stack sizes
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 8                                               ' default
use 32 for the hardware stack
$swstack = 2                                               ' default
use 10 for the SW stack
$framesize = 14                                            ' default
use 40 for the frame space
'settings must be :

'HW Stack : 8
'Soft Stack : 2
'Frame size : 14

'note that the called routine (_STACKCHECK) will use 4 bytes
'ofhardware stack space
'So when your program works, you may subtract the 4 bytes of the needed
hardware stack size
'in your final program that does not include the STCHECK

'testmode =0  will work
'testmode =1 will use too much hardware stack
'testmode =2 will use too much soft stack space
'testmode =3 will use too much frame space
Const Testmode = 0
'compile and test the program with testmode from 0-3


'you need to dim the ERROR byte !!
Dim Error As Byte


#if Testmode = 2
   Declare Sub Pass(z As Long , Byval K As Long)
#else
   Declare Sub Pass()
#endif

Dim I As Long
I = 2
Print I
'call the sub in your code at the deepest level
'normally within a function or sub
```

```
#if Testmode = 2
   Call Pass(i , 1)
#else
   Call Pass()
#endif
End

#if Testmode = 2
   Sub Pass(z As Long , Byval K As Long)
#else
  Sub Pass()
#endif
    #if Testmode = 3
       Local S As String * 13
    #else
       Local S As String * 8
    #endif

    Print I
    Gosub Test
End Sub


Test:
#if Testmode = 1
  push r0 ; eat some hardware stack space
  push r1
  push r2
#endif

  ' *** here we call the routine ***
  Stcheck
  ' *** when error <>0 then there is a problem ***
#if Testmode = 1
  pop r2
  pop r1
  pop r0
#endif
Return
```

7.140 STOP

Action

Stop the specified device. Or stop the program

Syntax

STOP device
STOP

Remarks

Device	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power) , ADC(A/D converter power) or DAC(D/A converter)
XMEGA	For the Xmega you can also specify : DACA or DACB for the Digital/ Analog Converters A and B.

The single STOP statement will end your program by generating a never ending loop. When END is used it will have the same effect but in addition it will disable all interrupts.

The STOP statement with one of the above parameters will stop the specified device.

TIMER0 and COUNTER0 are the same device.
The AC and ADC parameters will switch power off the device to disable it and thus save power.

See also
START 1538 , END 1256

Example
See START 1538 example

7.141 SUB

Action
Defines a Sub procedure.

Syntax
SUB Name[(var1 , …)]

Remarks

Name	Name of the sub procedure, can be any non-reserved word.
var1	The name of the optional parameter(s).

You must end each subroutine with the END SUB statement.
You can copy the DECLARE SUB line and remove the DECLARE statement. This ensures that you have the right parameters.

You can also use CONFIG SUBMODE=NEW and only write the implementation. In that case you do not need to write the DECLARATION.

See Also
FUNCTION 1215 , CALL 806 , CONFIG SUBMODE 1079 , EXIT 1257

See the DECLARE SUB 1221 topic for more details.

7.142 SWAP

Action
Exchange two variables of the same type.
Exchange a nibble or 2 bytes

Syntax

SWAP var1, var2
SWAP var3

Remarks

var1	A variable of type bit, byte, integer, word, long or string.
var2	A variable of the same type as var1.
var3	A byte, integer,word,long or dword

After the swap, var1 will hold the value of var2 and var2 will hold the value of var1.

When using swap with a single variable it need to be a byte, integer/word or long/ dword variable.
In version 2084 you can also swap a dword or long.

When using swap on a byte, the nibbles will be swapped.
Example :
byte=&B1100_0001 : swap byte : byte will become : &B0001_1100

When using swap on a single integer or word, the 2 bytes will be swapped so the LSB becomes the MSB and the MSB becomes the LSB.
When using swap on a single dword or long, the 4 bytes will be swapped the following way :
LSB NSB1 NSB2 MSB will become : MSB NSB2 NSB1 LSB

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : swap.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: SWAP
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte , B1 As Byte
Dim Bbit1 As Bit , Bbit2 As Bit
Dim S1 As String * 10 , S2 As String * 10



S1 = "AAA" : S2 = "BBB"
Swap S1 , S2
```

```
A = 5 : B1 = 10                                          'assign some
vars
Print A ; "   " ; B1                                     'print them

Swap A , B1                                              'swap them
Print A ; "   " ; B1                                     'print is
again

Set Bbit1
Swap Bbit1 , Bbit2
Print Bbit1 ; Bbit2
End
```

7.143 TCP/IP

TCP/IP

7.143.1 BASE64DEC

Action

Converts Base-64 data into the original data.

Syntax

Result = **BASE64DEC**(source)
array = **BASE64DEC**(source, elements)

Remarks

Result	A string variable that is assigned with the un-coded string.
Source	The source string that is coded with base-64.
array	A byte array that is assigned with the un-coded strings
elements	The number of elements in the resulting array

Base-64 is not an encryption protocol. It sends data in 7-bit ASCII data format. MIME, web servers, and other Internet servers and clients use Base-64 coding.

The provided Base64Dec() function is a decoding function. It was written to add authentication to the web server sample.
When the web server asks for authentication, the client will send the user and password unencrypted, but base-64 coded to the web server.
Base-64 coded strings are always in pairs of 4 bytes. These 4 bytes represent 3 bytes.

Because strings can not contain a 0 byte, there is an alternative syntax. Instead of a string you assign a byte array.
The byte variable ELEMENTS is assigned with the number of elements filled with data.

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , BASE64ENC 1549 , URL2IP 1592

Example

```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space
$lib "tcpip.lbx"
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As String * 15 , Z As String * 15

S = "bWFyazptYXJr"
Z = Base64dec(s)
Print Z                                                    'mark:mark
End
```

Example 2

```
$regfile = "m32U4def.dat"
$crystal = 16000000
$baud = 19200

Dim S As String * 80 , Z As String * 80 , B As Byte , J As Byte
Dim Ar(81) As Byte At S Overlay

S = "This is a test"
'while we load the array with string data, we could load it with any
data that can contain a 0.

B = Len(s) + 1                                              'get the
length and the 0 byte
Z = Base64enc(ar(1) , B)                                    'use an
array
Print Z

Ar(1) = Base64dec(z , B)                                    'now B will
hold the number of elements

Print B

'Another example
Ar(1) = 0 : Ar(2) = 1 : Ar(3) = 2
Z = Base64enc(ar(1) , 3)                                    'use an
array
Print Z


Ar(1) = Base64dec(z , B)                                    'now B will
hold the number of elements
```

```
For J = 1 To B
    Print Ar(j)
Next
End
```

7.143.2 BASE64ENC

Action
Converts a string into the Base-64 representation.

Syntax
Result = **BASE64ENC**(source)
Result = **BASE64ENC**(array, length)

Remarks

Result	A string variable that is assigned with the base64 coded string.
Source	The source string that must be coded.
array	The first element of a byte array.
length	The number of elements to convert. Maximum 255.

Base-64 is not an encryption protocol. It sends data in 7-bit ASCII data format. MIME, web servers, and other Internet servers and clients use Base-64 coding.

The provided Base64Enc() function is an encoding function. You need it when you want to send attachments with POP3 for example.
The target string will use 1 additional byte for every 3 bytes. This means that the target string is ca. 33 % longer than the source string.
So make sure the target string is dimensioned longer then the original string.

Because strings can not contain a 0 byte, there is an alternative syntax. Instead of a string you pass the address of a byte array that contains the data you want to convert.
Because there is no end of string marker, you must provide the number of elements to convert.

See also
CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , BASE64DEC 1547, URL2IP 1592

Example
```
$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
```

```
$framesize = 40                                                 ' default
use 40 for the frame space
$lib "tcpip.lbx"
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


Dim S As String * 15 , Z As String * 15

S = "bWFyazptYXJr"
Z = Base64dec(s)
Print Z                                                         'mark:mark
s = Base64Enc(z)
Print s
End
```

Example 2

```
$regfile = "m32U4def.dat"
$crystal = 16000000
$baud = 19200

Dim S As String * 80 , Z As String * 80 , B As Byte , J As Byte
Dim Ar(81) As Byte At S Overlay

S = "This is a test"
'while we load the array with string data, we could load it with any
data that can contain a 0.

B = Len(s) + 1                                                  'get the
length and the 0 byte
Z = Base64enc(ar(1) , B)                                        'use an
array
Print Z

Ar(1) = Base64dec(z , B)                                        'now B will
hold the number of elements

Print B

'Another example
Ar(1) = 0 : Ar(2) = 1 : Ar(3) = 2
Z = Base64enc(ar(1) , 3)                                        'use an
array
Print Z


Ar(1) = Base64dec(z , B)                                        'now B will
hold the number of elements
For J = 1 To B
    Print Ar(j)
Next
End
```

7.143.3 GETDSTIP

Action
Returns the IP address of the peer.

Syntax
Result = **GETDSTIP**(socket)

Remarks

Result	A LONG variable that will be assigned with the IP address of the peer or destination IP address.
Socket	The socket number (0-3)

When you are in server mode, it might be desirable to detect the IP address of the connecting client.
You can use this for logging, security, etc.

The IP number MSB, is stored in the LS byte of the variable.

See also
CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , GETDSTPORT 1551, URL2IP 1592

Partial Example
```
Dim L as Long
L = GetdstIP(i)  ' store current IP number of socket i
```

7.143.4 GETDSTPORT

Action
Returns the port number of the peer.

Syntax
Result = **GETDSTPort**(socket)

Remarks

Result	A WORD variable that is assigned with the port number of the peer or destination port number.
Socket	The socket number in the range from 0-3

When you are in server mode, it might be desirable to detect the port number of the connecting client.
You can use this for logging, security, etc.

See also
CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 ,

TCPWRITE 1578, TCPWRITESTR 1579, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , GETDSTIP 1551, URL2IP 1592

Example

```
'-----------------------------------------------------------------------
-----------------
'name                     : servertest_TWI.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : start the easytcp after the chip is
programmed
'                           and create 2 connections
'micro                    : Mega88
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------------
-----------------

$regfile = "m88def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate

$hwstack = 128                                              ' default
use 32 for the hardware stack
$swstack = 128                                              ' default
use 10 for the SW stack
$framesize = 128                                            ' default
use 40 for the frame space
                                                            ' xram
access
Print "Init , set IP to 192.168.1.70"                       ' display a
message
Enable Interrupts                                           ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int1 , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 , Tx
= $55 , Rx = $55 , Chip = W5100 , Spi = 1


Dim Bclient As Byte                                         ' socket
number
Dim Idx As Byte
Dim Result As Word , Result2 As Word                        ' result
Dim S As String * 80
Dim Flags As Byte
Dim Peer As Long
Dim L As Long


Do
  Waitms 1000
  For Idx = 0 To 3
     Result = Socketstat(idx , 0)                           ' get status
     Select Case Result
       Case Sock_established
            If Flags.idx = 0 Then                           ' if we did
not send a welcome message yet
               Flags.idx = 1
               Result = Tcpwrite(idx , "Hello from W3100A{013}{010}")
```

```
   ' send welcome
            End If
            Result = Socketstat(idx , Sel_recv)              ' get number
of bytes waiting
            Print "Received : " ; Result
            If Result > 0 Then
               Do
                 Print "Result : " ; Result
                 Result = Tcpread(idx , S)
                 Print "Data from client: " ; Idx ; " " ; Result ; "  "
; S
                 Peer = Getdstip(idx)
                 Print "Peer IP " ; Ip2str(peer)
                 Print "Peer port : " ; Getdstport(idx)
                 'you could analyse the string here and send an
appropiate command
                 'only exit is recognized
                 If Lcase(s) = "exit" Then
                    Closesocket Idx
                 Elseif Lcase(s) = "time" Then
                    Result2 = Tcpwrite(idx , "12:00:00{013}{010}")' you
should send date$ or time$
                 End If
               Loop Until Result = 0
            End If
       Case Sock_close_wait
            Print "close_wait"
            Closesocket Idx
       Case Sock_closed
            Print "closed"
            Bclient = Getsocket(idx , Sock_stream , 5000 , 64)    ' get
socket for server mode, specify port 5000
            Print "Socket " ; Idx ; " " ; Bclient

            Socketlisten Idx
            Print "Result " ; Result
            Flags.idx = 0                                         '
reset the hello message flag
       Case Sock_listen                                           ' this
is normal
       Case Else
            Print "Socket status : " ; Result
     End Select
  Next
Loop



End
```

7.143.5 GETSOCKET

Action

Creates a socket for TCP/IP communication.

Syntax

Result = **GETSOCKET**(socket, mode, port, param)

Remarks

Result	A byte that is assigned with the socket number you requested. When the operation fails, it will return 255.
socket	A numeric constant or variable with the socket number. The socket number is in range of 0-3. And 0-7 for the W5200 and W5300.
Mode	The socket mode. Use sock_stream(1), sock_dgram(2), sock_ipl_raw (3) or macl_raw(4). The modes are defined with constants. The W5100,W5200,W5300 also have the sock_ppoe(5) mode. For TCP/IP communication you need to specify sock_stream or the equivalent value 1. For UDP communication you need to specify sock_dgram or the equivalent value 2.
Port	This is the local port that will be used for the communication. You may specify any value you like but each socket must have it's own local port number. When you use 0, the value of LOCAL_PORT will be used. LOCAL_PORT is assigned with CONFIG TCPIP. After the assignment, LOCAL_PORT will be increased by 1. So the simplest way is to setup a local port with CONFIG TCPIP, and then use 0 for port.
Param	Optional parameter. Use 0 for default. **W3100** 128 : send/receive broadcast message in UDP 64 : use register value with designated timeout value 32 : when not using no delayed ack 16: when not using silly window syndrome Consult the W3100A documentation for more information. **W5100,W5200,W5300** 128 : enable multicasting in UDP 32 : enable 'No delayed ACK' operation. Only for TCP/IP. In case of UDP multicast : 1 : use IGMP version 1, otherwise V 2. Consult the wiznet documentation for more information.

After the socket has been initialized you can use SocketConnect to connect to a client, or SocketListen to act as a server.

W5100
When GetSocket does not return a valid socket number you can use a SOCKETDISCONNECT 1570 when it is in status &H18. For some reason the socket can remain in status &H18 for over a minutes and a SOCKETDISCONNECT will free the socket quicker.

See also

CONFIG TCPIP 1098, SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, TCPREAD 1576, SOCKETCLOSE 1564 , SOCKETLISTEN 1571 , SOCKETDISCONNECT 1570, URL2IP 1592

Partial Example

```
I = Getsocket(0 , Sock_stream , 5000 , 0)' get a new socket
```

7.143.6 GETTCPREGS

Action
Read a register value from the ethernet chip.

Syntax
var = **GETTCPREGS**(address, bytes)

Remarks

Address	The address of the register. This should not include the base address.
bytes	The number of bytes to read.

Most options are implemented with BASCOM statements or functions. When there is a need to read from the ethernet registers you can use the GETTCPREGS function. It can read multiple bytes.

 It is important that you specify the lowest address. This points to the MSB of the data.

See also
SETTCPREGS 1560

ASM
NONE

Example
See SETTCPREGS 1560

7.143.7 IP2STR

Action
Convert an IP number into it’s string representation.

Syntax
Var = **IP2STR**(num)

Remarks
An IP number is represented with dots like 192.168.0.1.
The IP2STR function converts an IP number into a string.
This function is intended to be used in combination with the BASCOM TCP/IP routines.

Var	The string variable that is assigned with the IP number
Num	A variable that contains the ip number is numeric format.

See also
CONFIG TCPIP 1098, URL2IP 1592

7.143.8 MAKETCP

Action
Creates a TCP/IP formatted long variable.

Syntax
var = **MAKETCP**(b1,b2,b3,b4 [opt])
var = **MAKETCP**(num)

Remarks

var	The target variable of the type LONG that is assigned with the IP number
b1-b4	Four variables of numeric constants that form the IP number. b1 is the MSB of the IP/long b4 is the LSB of the IP/long example var = MakeTCP(192,168,0, varx). We can also use reverse order with the optional parameter : example var = MakeTCP(var3,0,168, 192, **1**). A value of 1 will use reverse order while a value of 0 will result in normal order. When you use a constant, provide only one parameter : example var = MakeTCP(192.168.0.2). Notice the dots !

MakeTCP is a helper routine for the TCP/IP library.

See also
CONFIG TCPIP 1098 , IP2STR 1555, URL2IP 1592

Example
```
NONE
```

7.143.9 SETIPPROTOCOL

Action
Configures socket RAW-mode protocol

Syntax
SETIPPROTOCOL socket, value

Remarks

Socket	The socket number. (0-3)
Value	The IP-protocol value to set.

In order to use W3100A's IPL_RAW Mode, the protocol value of the IP Layer to be used (e.g., 01 in case
of ICMP) needs to be set before socket initialization.
As in UDP, data transmission and reception is possible when the corresponding channel is initialized.

The PING example demonstrates the usage.
As a first step, SETIPPROTOCOL is used :
Setipprotocol Idx , **1**
And second, the socket is initialized :
Idx = Getsocket(idx , **3** , 5000 , 0)

The W3100A data sheet does not provide much more details about the IPR register.

See also
SETTCPREGS 1560, GETSOCKET 1553

ASM
NONE

Example
```
'-----------------------------------------------------------------------------
'name                     : PING_TWI.bas          http://www.faqs.org/rfcs/rfc792.
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : Simple PING program
'micro                    : Mega88
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------
$regfile = "m32def.dat"                                     ' specify the used micr

$crystal = 8000000                                          ' used crystal frequenc
$baud = 19200                                               ' use baud rate
$hwstack = 80                                               ' default use 32 for th
$swstack = 128                                              ' default use 10 for th
$framesize = 80                                             ' default use 40 for th

Const Debug = 1

Const Sock_stream = $01                                     ' Tcp
Const Sock_dgram = $02                                      ' Udp
Const Sock_ipl_raw = $03                                    ' Ip Layer Raw Sock
Const Sock_macl_raw = $04                                   ' Mac Layer Raw Sock
Const Sel_control = 0                                       ' Confirm Socket Status
Const Sel_send = 1                                          ' Confirm Tx Free Buffe
Const Sel_recv = 2                                          ' Confirm Rx Data Size

'socket status
Const Sock_closed = $00                                     ' Status Of Connection
Const Sock_arp = $01                                        ' Status Of Arp
Const Sock_listen = $02                                     ' Status Of Waiting For
Const Sock_synsent = $03                                    ' Status Of Setting Up
Const Sock_synsent_ack = $04                                ' Status Of Setting Up
Const Sock_synrecv = $05                                    ' Status Of Setting Up
```

```
Const Sock_established = $06                                ' Status Of Tcp Connect
Const Sock_close_wait = $07                                 ' Status Of Closing Tcp
Const Sock_last_ack = $08                                   ' Status Of Closing Tcp
Const Sock_fin_wait1 = $09                                  ' Status Of Closing Tcp
Const Sock_fin_wait2 = $0a                                  ' Status Of Closing Tcp
Const Sock_closing = $0b                                    ' Status Of Closing Tcp
Const Sock_time_wait = $0c                                  ' Status Of Closing Tcp
Const Sock_reset = $0d                                      ' Status Of Closing Tcp
Const Sock_init = $0e                                       ' Status Of Socket Init
Const Sock_udp = $0f                                        ' Status Of Udp
Const Sock_raw = $10                                        ' Status of IP RAW


'we do the usual
Print "Init TCP"                                            ' display a message
Enable Interrupts                                           ' before we use config
Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.2
Print "Init done"

Dim Peersize As Integer , Peeraddress As Long , Peerport As Word
Dim Idx As Byte , Result As Word , J As Byte , Res As Byte
Dim Ip As Long
Dim Dta(12) As Byte , Rec(12) As Byte


Dta(1) = 8                                                  'type is echo
Dta(2) = 0                                                  'code

Dta(3) = 0                                                  ' for checksum initiali
Dta(4) = 0                                                  ' checksum
Dta(5) = 0                                                  ' a signature can be ar
Dta(6) = 1                                                  '   signature
Dta(7) = 0                                                  ' sequence number - any
Dta(8) = 1
Dta(9) = 65

Dim W As Word At Dta + 2 Overlay                            'same as dta(3) and dta
W = Tcpchecksum(dta(1) , 9)                                 ' calculate checksum ar

#if Debug
  For J = 1 To 9
    Print Dta(j)
  Next
#endif



Ip = Maketcp(192.168.0.16)                                  'try to check this serv

Print "Socket " ; Idx ; " " ; Idx
Setipprotocol Idx , 1                                       'set protocol to 1
'the protocol value must be set BEFORE the socket is openend

Idx = Getsocket(idx , 3 , 5000 , 0)


Do
   Result = Udpwrite(ip , 7 , Idx , Dta(1) , 9)            'write ping data
   Print Result
   Waitms 100
   Result = Socketstat(idx , Sel_recv)                      'check for data
   Print Result
   If Result >= 11 Then
      Print "Ok"
```

```
        Res = Tcpread(idx , Rec(1) , Result)                    'get data with TCPREAD
        #if Debug
          Print "DATA RETURNED :" ; Res                         '
          For J = 1 To Result
            Print Rec(j) ; " " ;
          Next
          Print
        #endif
      Else                                                      'there might be a probl
         Print "Network not available"
      End If
      Waitms 1000
Loop
```

7.143.1 SETTCP

Action

Configures or reconfigures the TCP/IP chip.

Syntax

SETTCP MAC , IP , SUBMASK , GATEWAY

Remarks

MAC	The MAC address you want to assign to the ethernet chip. The MAC address is a unique number that identifies your chip. You must use a different address for every W3100A chip in your network. Example : 123.00.12.34.56.78 You need to specify 6 bytes that must be separated by dots. The bytes must be specified in decimal notation.
IP	The IP address you want to assign to the ethernet chip. The IP address must be unique for every ethernet chip in your network. When you have a LAN, 192.168.0.10 can be used. 192.168.0.x is used for LAN's since the address is not an assigned internet address.
SUBMASK	The sub mask you want to assign to the W3100A. The sub mask is in most cases 255.255.255.0
GATEWAY	This is the gateway address of the ethernet chip. The gateway address you can determine with the IPCONFIG command at the command prompt : C:\>ipconfig Windows 2000 IP Configuration Ethernet adapter Local Area Connection 2: Connection-specific DNS Suffix . : IP Address. . . . . : 192.168.0.3 Subnet Mask . . . . . : 255.255.255.0 Default Gateway . . . . . : 192.168.0.1 **Use 192.168.0.1 in this case.**

The CONFIG TCPIP statement may be used only once.

When you want to set the TCP/IP settings dynamically for instance when the settings are stored in EEPROM, you can not use constants. For this purpose, SETTCP must be used.

SETTCP can take a variable or a constant for each parameter.

When you set the TCP/IP settings dynamically, you do not need to set them with CONFIG TCPIP. In the CONFIG TCPIP you can use the NOINIT parameter so that the MAC and IP are not initialized which saves code.

See also

GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, TCPREAD 1576, SOCKETCLOSE 1564 , SOCKETLISTEN 1571 , CONFIG TCPIP 1098 , SOCKETDISCONNECT 1570 , GETTCPREGS 1555 , SETTCPREGS 1560

Example

See the DHCP.BAS example from the BASCOM Sample dir.

7.143.13 SETTCPREGS

Action

Writes to an ethernet chip register

Syntax

SETTCPREGS address, var , bytes

Remarks

address	The address of the register. This must be the address of the MSB, or the address with the lowest address. The address should not include the base address.
var	The variable to write.
bytes	The number of bytes to write.

Most options are implemented with BASCOM statements or functions. When there is a need to write to the ethernet chip register you can use the SETTCPREGS command. It can write multiple bytes. It is important that you specify the lowest address. The SETTCPREGS statement will add the base address of the chip to the address so you should not add it yourself. Use the address from the datasheet.
The addresses are different for the W3100,W5100,W5200 and W5300.

Some registers you might want to alter are the Retry Count Register(RCR) and Retry Time Register(RTR).

See also

GETTCPREGS 1555

ASM
NONE

Example
```
'-------------------------------------------------------------------
------------------
'name                       : regs_SPI.bas
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : test custom regs reading writing
'micro                      : Mega88
'suited for demo            : yes
'commercial addon needed    : no
'-------------------------------------------------------------------
------------------
$regfile = "m88def.dat"                                    ' specify
the used micro

$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 80                                              ' default
use 32 for the hardware stack
$swstack = 128                                             ' default
use 10 for the SW stack
$framesize = 80                                            ' default
use 40 for the frame space

Dim L As Long

Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes ,
Polarity = Low , Phase = 0 , Clockrate = 4 , Noss = 0
'Init the spi pins
Spiinit

'we do the usual
Print "Init TCP"                                           ' display a
message
Enable Interrupts                                          ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int1 , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 , Tx
= $55 , Rx = $55 ,  Chip = W5100 , Spi = 1
Print "Init done"

'set the IP address to 192.168.0.135
Settcp 12.128.12.24.56.78 , 192.168.1.135 , 255.255.255.0 , 192.168.1.1

'now read the IP address direct from the registers
L = Gettcpregs(&H0f , 4)
Print Ip2str(l)

Dim B4 As Byte At L Overlay                                ' this byte
is the same as the LSB of L

'now make the IP address 192.168.1.136 by writing to the LSB
B4 = 136
Settcpregs &H0F , L , 4                                    'write

'and check if it worked
L = Gettcpregs(&H0f , 4)
Print Ip2str(l)
```

```
'and with PING you can check again that now it works

End
```

7.143.11 SNTP

Action

This function retrieves the date and time from an SNTP server using the TCP/IP W3100 or W5100.

Syntax

result=**SNTP**(socket,IP)

Remarks

Result	A long or dword that is assigned with the date/time. If there is no data, the result will be 0.
socket	The socket number of the connection.
IP	The IP number of the SNTP server you want to connect to. This may be a number like 192.168.0.2 or a LONG variable that is assigned with an IP number.

SNTP means Network Time Protocol. It is an internet protocol used to synchronize clocks. SNTP uses UTC as reference time.
The SNTP function is intended to be used with a W3100A or W5100 chip. The SNTP function uses UDP routines from the library to fetch the time.

See also

NONE

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : sntp_SPI.bas   RFC 2030
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : test SNTP() function
'micro                    : Mega88
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------


$regfile = "m88def.dat"                                     ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 80                                               ' default
```

```
use 32 for the hardware stack
$swstack = 128                                              ' default
use 10 for the SW stack
$framesize = 80                                             ' default
use 40 for the frame space
$lib "datetime.lbx"                                         ' this
example uses date time routines

Print "Init TCP"                                           ' display a
message
Enable Interrupts                                           ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int1 , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 , Tx
= $55 , Rx = $55 ,  Chip = W5100 , Spi = 1
Print "Init done"

Dim Var As Byte                                             ' for i2c
test


Dim Ip As Long                                              ' IP number
of time server
Dim Idx As Byte                                             ' socket
number
Dim Lsntp As Long                                           ' long SNTP
time

Print "SNTP demo"

'assign the IP number of a SNTP server
Ip = Maketcp(64.90.182.55 )                                 ' assign IP
num NIST time.nist.gov  port 37
Print "Connecting to : " ; Ip2str(ip)


'we will use Dutch format
Config Date = Dmy , Separator = -


'we need to get a socket first
'note that for UDP we specify sock_dgram
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)                ' get socket
for UDP mode, specify port 5000
Print "Socket " ; Idx ; " " ; Idx

'UDP is a connection less protocol which means that you can not listen,
connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the addition
of the IP and PORT
'The SNTP uses port 37 which is fixed in the tcp asm code



Do

   'toggle the variable
   Toggle Var

   Waitms 5000

   Lsntp = Sntp(idx , Ip)                                   ' get time
```

```
from SNTP server
 '  Print Idx ; Lsntp
   'notice that it is not recommended to get the time every sec
   'the time server might ban your IP
   'it is better to sync once or to run your own SNTP server and update
that once a day

   'what happens is that IP number of timer server is send a diagram too
   'it will put the time into a variable lsntp and this is converted to
BASCOM date/time format
   'in case of a problem the variable is 0
   Print Date(lsntp) ; Spc(3) ; Time(lsntp)
Loop


End
```

7.143.13 SOCKETCLOSE

Action

Closes a socket connection.

Syntax

SOCKETCLOSE socket [, prm]

Remarks

Socket	The socket number you want to close in the range of 0-3 (0-7 for W5200/ W5300). When the socket is already closed, no action will be performed.
Prm	An optional parameter to change the behavior of the CloseSocket statement. The following values are possible : • 0 - The code will behave as if no parameter has been set. • 1 - In normal cases, there is a test to see if all data written to the chip has been sent. When you set bit 0 (value of 1) , this test is not performed. • 2 - In normal cases, there is a test to see if the socket is actually closed after the command has been given to the chip. When it is not closed, you can not re-use the socket. The statement will block program execution however and you could test at a later time if the connection has been closed. You may combine the values. So 3 will combine parameter value 1 and 2. It is advised to use option value 1 with care.

You must close a socket when you receive the SOCK_CLOSE_WAIT status.
You may also close a socket if that is needed by your protocol.
You will receive a SOCK_CLOSE_WAIT status when the server closes the connection.

When you use CloseSocket you actively close the connection.
Note that it is not needed to wait for a SOCK_CLOSE_WAIT message in order to close a socket connection.

After you have closed the connection, you need to use GetSocket in order to use the socket number again.

In normal conditions, without using the optional parameter, the statement can block

your code for a short or longer time, depending on the connection speed.

The CLOSESOCKET statement is equivalent with SOCKETCLOSE.

SOCKETCLOSE VS SOCKETDISCONNECT

In the W3x00 chips there was no socket disconnect function. A socket close (SOCKETCLOSE) would create a disconnect.
But in the W5x00 chips, there is an additional function to disconnect a socket. So for these chips you must use SOCKETDISCONNECT to terminate a connection. After that you can still use SOCKETCLOSE to free the resource of the socket.

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, TCPREAD 1576, SOCKETLISTEN 1571 , SOCKETDISCONNECT 1570

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : clienttest.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : start the easytcp.exe program and listen to
port 5000
'micro                    : Mega161
'suited for demo          : no
'commercial addon needed  : yes
'-------------------------------------------------------------------------
-----------------

$regfile = "M161def.dat"
$crystal = 4000000
$baud = 19200
$hwstack = 40                                               ' default
use 40 for the hardware stack
$swstack = 40                                               ' default
use 40 for the SW stack
$framesize = 64                                             ' default
use64 for the frame space

Const Sock_stream = $01                                     ' Tcp
Const Sock_dgram = $02                                      ' Udp
Const Sock_ipl_raw = $03                                    ' Ip Layer
Raw Sock
Const Sock_macl_raw = $04                                   ' Mac Layer
Raw Sock
Const Sel_control = 0                                       ' Confirm
Socket Status
Const Sel_send = 1                                          ' Confirm Tx
Free Buffer Size
Const Sel_recv = 2                                          ' Confirm Rx
Data Size

'socket status
Const Sock_closed = $00                                     ' Status Of
Connection Closed
Const Sock_arp = $01                                        ' Status Of
Arp
Const Sock_listen = $02                                     ' Status Of
```

```
Waiting For Tcp Connection Setup
Const Sock_synsent = $03                                    ' Status Of
Setting Up Tcp Connection
Const Sock_synsent_ack = $04                                ' Status Of
Setting Up Tcp Connection
Const Sock_synrecv = $05                                    ' Status Of
Setting Up Tcp Connection
Const Sock_established = $06                                ' Status Of
Tcp Connection Established
Const Sock_close_wait = $07                                 ' Status Of
Closing Tcp Connection
Const Sock_last_ack = $08                                   ' Status Of
Closing Tcp Connection
Const Sock_fin_wait1 = $09                                  ' Status Of
Closing Tcp Connection
Const Sock_fin_wait2 = $0a                                  ' Status Of
Closing Tcp Connection
Const Sock_closing = $0b                                    ' Status Of
Closing Tcp Connection
Const Sock_time_wait = $0c                                  ' Status Of
Closing Tcp Connection
Const Sock_reset = $0d                                      ' Status Of
Closing Tcp Connection
Const Sock_init = $0e                                       ' Status Of
Socket Initialization
Const Sock_udp = $0f                                        ' Status Of
Udp
Const Sock_raw = $10                                        ' Status of
IP RAW



$lib "tcpip.lbx"                                            ' specify
the tcpip library
Print "Init , set IP to 192.168.0.8"                        ' display a
message
Enable Interrupts                                           ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx =
$55 , Rx = $55

'Use the line below if you have a gate way
'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx
= $55 , Rx = $55

Dim Bclient As Byte                                         ' socket
number
Dim Idx As Byte
Dim Result As Word                                          ' result
Dim S As String * 80

For Idx = 0 To 3                                            ' for all
sockets
  Bclient = Getsocket(idx , Sock_stream , 0 , 0)            ' get socket
for client mode, specify port 0 so loal_port is used
  Print "Local port : " ; Local_port                       ' print
local port that was used
  Print "Socket " ; Idx ; " " ; Bclient
  Result = Socketconnect(idx , 192.168.0.3 , 5000)          ' connect to
easytcpip.exe server
  Print "Result " ; Result
Next
```

```
Do

  If Ischarwaiting() <> 0 Then                              ' is there a
key waiting in the uart?
     Bclient = Waitkey()                                    ' get the
key
     If Bclient = 27 Then
       Input "Enter string to send " , S                    ' send WHO ,
TIME or EXIT
       For Idx = 0 To 3
          Result = Tcpwritestr(idx , S , 255)
       Next
     End If
  End If

  For Idx = 0 To 3
     Result = Socketstat(idx , 0)                           ' get status
     Select Case Result
       Case Sock_established
            Result = Socketstat(idx , Sel_recv)             ' get number
of bytes waiting
            If Result > 0 Then
               Do
                 Result = Tcpread(idx , S)
                 Print "Data from server: " ; Idx ; " " ; S
               Loop Until Result = 0
            End If
       Case Sock_close_wait
            Print "close_wait"
            Closesocket Idx
       Case Sock_closed
            'Print "closed"
     End Select
  Next
Loop
End
```

7.143.1 SOCKETCONNECT

Action

Establishes a connection to a TCP/IP server.

Syntax

Result = **SOCKETCONNECT**(socket, IP, port [,nowait])

Remarks

Result	A byte that is assigned with 0 when the connection succeeded. It will return 1 when an error occurred.
socket	The socket number in the range of 0-3. Or 0-7 for W5200/W5300.
IP	The IP number of the server you want to connect to. This may be a number like 192.168.0.2 or a LONG variable that is assigned with an IP number. Note that the LSB of the LONG, must contain the MSB of the IP number.
Port	The port number of the server you are connecting to.

NoWait	This is an optional parameter. Make it 1 to suppress waiting for a connection. By default, when you create a connection, the code waits for the connect flag. But waiting will block program execution. When you specify, not to wait, the code returns immediately. But you must use SOCKETSTAT 1571 to determine the outcome of the socketconnect. NOWAIT parameter is implemented for : -W5100 -W5200 -W5500

You can only connect to a server. Standardized servers have dedicated port numbers. For example, the HTTP protocol(web server) uses port 80.

After you have established a connection the server might send data. This depends entirely on the used protocol. Most servers will send some welcome text, this is called a banner.

You can send or receive data once the connection is established.

The server might close the connection after this or you can close the connection yourself. This also depends on the protocol.

You need to obtain a valid socket first with the GETSOCKET function.

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579 , TCPREAD 1576, SOCKETCLOSE 1564 , SOCKETLISTEN 1571 , SOCKETDISCONNECT 1570, URL2IP 1592

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : servertest_SPI.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : start the easytcp after the chip is
programmed
'                           and create 2 connections
'micro                    : Mega88
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m88def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate

$hwstack = 128                                             ' default
use 32 for the hardware stack
$swstack = 128                                             ' default
use 10 for the SW stack
$framesize = 128                                           ' default
use 40 for the frame space
```

```
Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes ,
Polarity = Low , Phase = 0 , Clockrate = 4 , Noss = 0
'Init the spi pins
Spiinit                                                    ' xram
access
Print "Init , set IP to 192.168.1.70"                      ' display a
message
Enable Interrupts                                          ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int1 , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 , Tx
= $55 , Rx = $55 , Chip = W5100 , Spi = 1


Dim Bclient As Byte                                        ' socket
number
Dim Idx As Byte
Dim Result As Word , Result2 As Word                       ' result
Dim S As String * 80
Dim Flags As Byte
Dim Peer As Long
Dim L As Long


Do
  Waitms 1000
  For Idx = 0 To 3
     Result = Socketstat(idx , 0)                          ' get status
     Select Case Result
       Case Sock_established
            If Flags.idx = 0 Then                          ' if we did
not send a welcome message yet
               Flags.idx = 1
               Result = Tcpwrite(idx , "Hello from W5100A{013}{010}")
   ' send welcome
            End If
            Result = Socketstat(idx , Sel_recv)            ' get number
of bytes waiting
            Print "Received : " ; Result
            If Result > 0 Then
               Do
                 Print "Result : " ; Result
                 Result = Tcpread(idx , S)
                 Print "Data from client: " ; Idx ; " " ; Result ; "  "
; S
                 Peer = Getdstip(idx)
                 Print "Peer IP " ; Ip2str(peer)
                 Print "Peer port : " ; Getdstport(idx)
                 'you could analyse the string here and send an
appropiate command
                 'only exit is recognized
                 If Lcase(s) = "exit" Then
                    Closesocket Idx
                 Elseif Lcase(s) = "time" Then
                    Result2 = Tcpwrite(idx ,"12:00:00{013}{010}") ' you
should send date$ or time$
                 End If
               Loop Until Result = 0
            End If
       Case Sock_close_wait
            Print "close_wait"
            Closesocket Idx
       Case Sock_closed
```

```
            Print "closed"
            Bclient = Getsocket(idx , Sock_stream , 5000 , 64)    ' get
socket for server mode, specify port 5000
            Print "Socket " ; Idx ; " " ; Bclient

            Socketlisten Idx
            Print "Result " ; Result
            Flags.idx = 0                                          '
reset the hello message flag
       Case Sock_listen                                            ' this
is normal
       Case Else
            Print "Socket status : " ; Result
     End Select
  Next
Loop


End
```

7.143.1 SOCKETDISCONNECT

Action

Disconnects a socket connection.

Syntax

SOCKETDISCONNECT socket

Remarks

Socket	The socket number you want to close in the range of 0-3 (0-7 for W5200/W5300). When the socket is already closed, no action will be performed.

The socketdisconnect statement sends a connection termination request.
You can also use SOCKETCLOSE to close the socket and free it's resources.

After you have closed the connection, you need to use GetSocket in order to use the socket number again.
If you only disconnect the socket, you can use socketconnect witout Getsocket.
The socketdisconnect is only intended for TCP connections. (UDP does not have connections).

 This statement is only available for the W5100/W5200/W5300. The W3100A does not support it.

SOCKETCLOSE VS SOCKETDISCONNECT

In the W3x00 chips there was no socket disconnect function. A socket close (SOCKETCLOSE) would create a disconnect.
But in the W5x00 chips, there is an additional function to disconnect a socket. So for these chips you must use SOCKETDISCONNECT to terminate a connection. After that you can still use SOCKETCLOSE to free the resource of the socket.

See also
CONFIG TCPIP 1098, SOCKETCLOSE 1564, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, TCPREAD 1576, SOCKETLISTEN 1571 , SETTCP 1559, URL2IP 1592

Example
NONE

7.143.16 SOCKETLISTEN

Action
Opens a socket in server(listen) mode.

Syntax
SOCKETLISTEN socket

Remarks

Socket	The socket number you want to use for the server in the range of 0 -3. Or 0-7 for W5200/W5300.

The socket will listen to the port you specified with the GetSocket function.
When a client connects, the socket status changes in sock_established. When a connection is established, you can send or receive data.

After the connection is closed by either the client or the server, a new connection need to be created and the SocketListen statement must be used again.
When the status has changed to sock_closed, there still could be some pending data in the receive buffer. So you could check with the SOCKETSTAT function if there is data waiting. And if data is waiting, you can read it with TCPREAD before opening the socket again.

See also
CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, TCPREAD 1576, SOCKETCLOSE 1564 , SOCKETDISCONNECT 1570

Example
See SOCKETCONNECT 1567 example

7.143.17 SOCKETSTAT

Action
Returns information about a socket.

Syntax
Result = **SOCKETSTAT**(socket , mode)

Remarks

Result	A word variable that is assigned with the result.
Socket	The socket number you want to get information of in the range from 0-3. Or 0-7 for W5200/W5300)
Mode	A parameter which specifies what kind of information you want to retrieve. SEL_CONTROL or 0 : returns the status register value SEL_SEND or 1 : returns the number of bytes that might be placed into the transmission buffer. Or in other words : the free transmission buffer space. SEL_RECV or 2 : returns the number of bytes that are stored in the reception buffer. Or in other words : the number of bytes received.

The SocketStat function contains actual 3 functions. One to get the status of the connection, one to determine how many bytes you might write to the socket, and one to determine how many bytes you can read from the buffer.

When you specify mode 0, one of the following byte values will be returned:

W3100A

Value	State	Description
0	SOCK_CLOSED	Connection closed
1	SOCK_ARP	Standing by for reply after transmitting ARP request
2	SOCK_LISTEN	Standing by for connection setup to the client when acting in passive mode
3	SOCK_SYNSENT	Standing by for SYN,ACK after transmitting SYN for connecting setup when acting in active mode
4	SOCK_SYNSENT_ACK	Connection setup is complete after SYN,ACK is received and ACK is transmitted in active mode
5	SOCK_SYNRECV	SYN,ACK is being transmitted after receiving SYN from the client in listen state, passive mode
6	SOCK_ESTABLISHED	Connection setup is complete in active, passive mode
7	SOCK_CLOSE_WAIT	Connection being terminated
8	SOCK_LAST_ACK	Connection being terminated
9	SOCK_FIN_WAIT1	Connection being terminated
10	SOCK_FIN_WAIT2	Connection being terminated
11	SOCK_CLOSING	Connection being terminated
12	SOCK_TIME_WAIT	Connection being terminated
13	SOCK_RESET	Connection being terminated after receiving reset packet from peer.
14	SOCK_INIT	Socket initializing
15	SOCK_UDP	Applicable channel is initialized in UDP mode.
16	SOCK_RAW	Applicable channel is initialized in IP layer RAW mode
17	SOCK_UDP_ARP	Standing by for reply after transmitting ARP request packet to the destination for UDP

		transmission
18	SOCK_UDP_DATA	Data transmission in progress in UDP RAW mode
19	SOCK_RAW_INIT	W3100A initialized in MAC layer RAW mode

W5100,W5200,W5300

Value	State	Description
0	SOCK_CLOSED	Connection closed
&H11	SOCK_ARP	Standing by for reply after transmitting ARP request
&H14	SOCK_LISTEN	Standing by for connection setup to the client when acting in passive mode
&H15	SOCK_SYNSENT	Standing by for SYN,ACK after transmitting SYN for connecting setup when acting in active mode
&H16	SOCK_SYNRECV	SYN,ACK is being transmitted after receiving SYN from the client in listen state, passive mode
&H17	SOCK_ESTABLISHED	Connection setup is complete in active, passive mode
&H1C	SOCK_CLOSE_WAIT	Connection being terminated
&H1D	SOCK_LAST_ACK	Connection being terminated
&H18	SOCK_FIN_WAIT	Connection being terminated
&H1A	SOCK_CLOSING	Connection being terminated
&H1B	SOCK_TIME_WAIT	Connection being terminated
&H13	SOCK_INIT	Socket initializing
&H22	SOCK_UDP	Applicable channel is initialized in UDP mode.
&H32	SOCK_RAW	Applicable channel is initialized in IP layer RAW mode
&H42	SOCK_MACRAW	Applicable channel is initialized in MAC layer RAW mode
&H5F	SOCK_PPOE	Applicable channel is initialized in PPOE mode

The SocketStat function is also used internal by the library.

For the W5300, if you use ALIGN=2, you need to take in mind that you must read the data buffer if it contains data. Do not call SocketStat again since it will read another 2 bytes to determine the received data size.

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, TCPWRITE 1578, TCPWRITESTR 1579, TCPREAD 1576, SOCKETCLOSE 1564 , SOCKETLISTEN 1571, SOCKETDISCONNECT 1570, URL2IP 1592

Partial Example

```
Tempw = Socketstat(i , 0)' get status
Select Case Tempw
  Case Sock_established
  Case Else
End Select
```

7.143.18TCPCHECKSUM

Action
Return a TCP/IP checksum, also called Internet Checksum, or IP Checksum.

# Syntax
res= **TCPCHECKSUM**(buffer , bytes [,w1] [,w2])

Remarks

Res	A word variable that is assigned with the TCP/IP checksum of the buffer
Buffer	A variable or array to get the checksum of.
Bytes	The number of bytes that must be examined.
w1,w2	Optional words that will be included in the checksum.

Checksum's are used a lot in communication protocols. A checksum is a way to verify that received data is the same as it was sent. In the many Internet Protocols (TCP, UDP, IP, ICMP …) a special Internet checksum is used. Normally the data to calculate the checksum on is stored in an array of bytes, but in some cases like TCP, and UDP, a pseudo header is added. The optional words (w1, w2) can be used for these cases. Most often w1 and w2 will be used for the Protocol number, and the UDP or TCP packet length.

This checksum is calculated by grouping the bytes in the array into 2-byte words. If the number of Bytes is an odd number, then an extra byte of zero is used to make the last 2-byte word. All of the words are added together, keeping the total in a 4-byte Long variable. If the optional words w1, w2, are included, they are also added to the total. Next, the 4-byte Long total is split into two, 2-byte words, and these words are added together to make a new 2-byte Word total. Finally the total is inverted. This is the value returned as Res.

This function using w1, w2, are very useful when working directly with Ethernet chips like the RTL8019AS or with protocols not directly supported by the WIZnet chips.

See the samples directory for more examples of use (IP_Checksum.bas).

You can use it for the PING sample below.

See also
CRC8 809 , CRC16 813, CRC32 817 , CHECKSUM 808

ASM
NONE

Example
```
'-----------------------------------------------------------------------
-----------------
'name                          : PING_TWI.bas                 http://www.faqs.org/
rfcs/rfc792.html
'copyright                     : (c) 1995-2025, MCS Electronics
'purpose                       : Simple PING program
'micro                         : Mega88
'suited for demo               : yes
'commercial addon needed       : no
```

```
'-------------------------------------------------------------------------
------------------
$regfile = "m32def.dat"                                    '    specify
the   used   micro

$crystal = 8000000                                         '   used
crystal       frequency
$baud = 19200                                              '  use   baud
rate
$hwstack = 80                                              '    default
use   32   for   the   hardware   stack
$swstack = 128                                             '    default
use   10   for   the   SW   stack
$framesize = 80                                            '    default
use   40   for   the   frame   space

Const Debug = 1

'we   do   the   usual
Print "Init     TCP"                                       '   display   a
message
Enable Interrupts                                          '   before   we
use   config   tcpip   ,   we   need   to   enable   the   interrupts
Config   Tcpip = Int0,  Mac =   12.128. 12.34. 56.78,   Ip =   192.168. 0.8,  Submask
=   255.255. 255.0,   Gateway =   192.168. 0.1,      Localport =   1000,   Tx =   $55,   Rx =
$55,   Twi =  &H80,    Clock =   400000
Print "Init      done"

Dim    Peersize As Integer ,    Peeraddress As Long ,    Peerport As Word
Dim   Idx As Byte ,    Result As Word ,   J As Byte ,   Res As Byte
Dim   Ip As Long
Dim   Dta( 12)  As Byte ,   Rec( 12)  As Byte


Dta( 1)  = 8                                               '   type   is
echo
Dta( 2)  = 0                                               '   code
Dta( 3)  = 0                                               '    for
checksum          initialization
Dta( 4)  = 0                                               '   checksum
Dta( 5)  = 0                                               '  a
signature   can   be   any   number
Dta( 6)  = 1                                               '     signature
Dta( 7)  = 0                                               '    sequence
number   -   any   number
Dta( 8)  = 1
Dta( 9)  = 65

Dim W As Word At  Dta  +  2 Overlay                        '  same   as
dta(3)    and    dta(4)
W = Tcpchecksum( dta( 1)  ,  9)                            '     calculate
checksum   and   store   in   dta(3)   and   dta(4)

#i f Debug
  For J = 1 To 9
    Print  Dta( j )
  Next
#endif



Ip  = Maketcp( 192.168. 0.16)                              '   try   to
check    this    server

Print "Socket " ; Idx ; " " ; Idx
Setipprotocol Idx , 1                                      ' set
protocol to 1
'the protocol value must be set BEFORE the socket is openend
```

```
Idx = Getsocket(idx , 3 , 5000 , 0)


Do
   Result = Udpwrite(ip , 7 , Idx , Dta(1) , 9)              ' write ping
data
   Print Result
   Waitms 100
   Result = Socketstat(idx , Sel_recv)                       ' check for
data
   Print Result
   If Result >= 11 Then
      Print "Ok"
      Res = Tcpread(idx , Rec(1) , Result)                   ' get data
with TCPREAD !!!
      #if Debug
        Print "DATA RETURNED :" ; Res
        For J = 1 To Result
          Print Rec(j) ; " " ;
        Next
        Print
      #endif
   Else                                                      ' there
might be a problem
      Print "Network not available"
   End If
   Waitms 1000
Loop
```

7.143.1 TCPREAD

Action

Reads data from an open socket connection.

Syntax

Result = **TCPREAD**(socket , var, bytes)

Remarks

Result	A byte variable that will be assigned with **0**, when no errors occurred. When an error occurs, the value will be set to **1**. When there are not enough bytes in the reception buffer, the routine will wait until there is enough data or the socket is closed.
socket	The socket number you want to read data from (0-3). Or 0-7 for W5200/ W5300.
Var	The name of the variable that will be assigned with the data from the socket.
Bytes	The number of bytes to read. Only valid for non-string variables.

When you use TCPread with a string variable, the routine will wait for CR + LF and it will return the data without the CR + LF.
For strings, the function will not overwrite the string.

For example, your string is 10 bytes long and the line you receive is 80 bytes long, you will receive only the first 10 bytes after CR + LF is encountered.

Also, for string variables, you do not need to specify the number of bytes to read since the routine will wait for CR + LF.

For other data types you need to specify the number of bytes.
There will be no check on the length so specifying to receive 2 bytes for a byte will overwrite the memory location after the memory location of the byte.

You should only attempt to read data if you have determined with the SocketStat function, that there is actual data in the receive buffer.

$BIGSTRINGS are not supported by TCPREAD.

See also
CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, SOCKETCLOSE 1564 , SOCKETLISTEN 1571, SOCKETDISCONNECT 1570, URL2IP 1592

Partial Example
```
Result = Socketstat(idx , Sel_recv)              ' get number of bytes
waiting
If Result > 0 Then
   Result = Tcpread(idx , S)
End If
```

7.143.2 TCPREADHEADER

Action
This statement reads the TCP packet header from the specified socket.

Syntax
TCPREADHEADER socket

Remarks
This option is only available for the W5300 which includes a packet header with the packet size when align is set to 0.
TCP packets start with a 2 byte size header.

After you have read the TCP header, you can use TCPDATASIZE to read the number of bytes available in the packet.
TCPDATASIZE is a word variable you need to dimension yourself.

Socket is a constant or variable in the range from 0-7.

See also
UDPREAD 1582, CONFIG TCPIP 1098 , UDPREADHEADER 1585, URL2IP 1592, URL2IP 1592

Example

7.143.2 TCPWRITE

Action
Write data to a socket.

Syntax
Result = **TCPWRITE**(socket , var , bytes)
Result = **TCPWRITE**(socket , EPROM, address , bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
Socket	The socket number you want to send data to in the range from 0-3. Or 0-7 for the W5200/W5300.
Var	A constant string like "test" or a variable. When you send a constant string, the number of bytes to send does not need to be specified.
Bytes	A word variable or numeric constant that specifies how many bytes must be send.
Address	The address of the data stored in the chips internal EEPROM. You need to specify EPROM too in that case.
EPROM	An indication for the compiler so it knows that you will send data from EPROM.

The TCPwrite function can be used to write data to a socket that is stored in EEPROM or in memory.
When you want to send data from an array, you need to specify the element : var (idx) for example.
The amount of data you can send depends on the socket TX size. With CONFIG TCPIP you can define the TX buffer size.
For example, for the W5100, the maximum TX socket size is 2 KB. In this case the maximum data size you can send is 2048 bytes.
Bigger data should be send in multiple chucks.
You should also consider the maximum packet size. If the packet size is 1460, sending more data will send multiple fragmented packets.
If you have enough RAM available, the best option is to use a buffer with the same size as the packet size. But if your memory it limited, you can let the chip handle this.

The following sample function demonstrates how you can send multiple chunks. The sample uses a buffer named eth_buffer() with a size of 2048 bytes.

```
Function Write_databuf(byval Txsize As Word) As Word
   Local Strt As Word
   Strt = 1
   Do
```

```
      If Txsize > 2048 Then
         Write_databuf = Tcpwrite(idx_http , Eth_buffer(strt) , 2048)
         Txsize = Txsize - 2048 : Strt = Strt + 2048
      Else
         Write_databuf = Tcpwrite(idx_http , Eth_buffer(strt) , Txsize)
         Exit Do
      End If
   Loop
   Http_speed = Http_speed + txSize
End function
```

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITESTR 1579, TCPREAD 1576, SOCKETCLOSE 1564 , SOCKETLISTEN 1571, SOCKETDISCONNECT 1570 , SETTCPREGS 1560, URL2IP 1592

Example

```
Result = Tcpwrite(idx , "Hello from W3100A{013}{010}")
```

7.143.27 TCPWRITESTR

Action

Sends a string to an open socket connection.

Syntax

Result = **TCPWRITESTR**(socket , var , param)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
Socket	The socket number you want to send data to (0-3). 0-7 for W5200/ W5300.
Var	The name of a string variable.
Param	A parameter that might be 0 to send only the string or 255, to send the string with an additional CR + LF This option was added because many protocols expect CR + LF at the end of the string.

The TCPwriteStr function is a special variant of the TCPwrite function.
It will use TCPWrite to send the data.

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 ,

TCPWRITE 1578, TCPREAD 1576, SOCKETCLOSE 1564 , SOCKETLISTEN 1571, SOCKETDISCONNECT 1570, URL2IP 1592

Example

```
'-----------------------------------------------------------------------
--------
'                              SMTP.BAS
'                       (c) 1995-2025 MCS Electronics
' sample that show how to send an email with SMTP protocol
'-----------------------------------------------------------------------
--------

$regfile = "m161def.dat"                                    ' used
processor
$crystal = 4000000                                          ' used
crystal
$baud = 19200                                               ' baud rate

Const Debug = -1                                            ' for
sending feeback to the terminal


#if Debug
  Print "Start of SMTP demo"
#endif

Enable Interrupts                                           ' enable
interrupts
'specify  MAC, IP, submask and gateway
'local port value will be used when you do not specify a port value
while creating a connection
'TX and RX are setup to use 4 connections each with a 2KB buffer
Config Tcpip = Int0 , Mac = 00.44.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx
= $55 , Rx = $55

'dim the used variables
Dim S As String * 50 , I As Byte , J As Byte , Tempw As Word
#if Debug
  Print "setup of W3100A complete"
#endif

'First we need a socket
I = Getsocket(0 , Sock_stream , 5000 , 0)
'             ^ socket numer     ^ port
#if Debug
  Print "Socket : " ; I
  'the socket must return the asked socket number. It returns 255 if
there was an error
#endif

If I = 0 Then                                               ' all ok
   'connect to smtp server
   J = Socketconnect(i , 194.09.0. , 25)                    ' smtp
server and SMTP port 25
   '                  ^socket
   '                       ^ ip address of the smtp server
   '                                              ^ port 25 for smtp
    '  DO NOT FORGET to ENTER a valid IP number of your ISP smtp server
   #if Debug
      Print "Connection : " ; J
      Print S_status(1)
```

```
   #endif
   If J = 0 Then                                          ' all ok
      #if Debug
         Print "Connected"
      #endif
      Do
         Tempw = Socketstat(i , 0)                        ' get status
         Select Case Tempw
           Case Sock_established                          ' connection
established
               Tempw = Tcpread(i , S)                     ' read line
               #if Debug
                   Print S                                ' show info
from smtp server
               #endif
               If Left(s , 3) = "220" Then                ' ok
                  Tempw = Tcpwrite(i , "HELO username{013}{010}" )
' send username
                  '                           ^^^ fill in username there
                  #if Debug
                    Print Tempw ; "  bytes written"        ' number of
bytes actual send
                  #endif
                  Tempw = Tcpread(i , S)                   ' get
response
                  #if Debug
                     Print S                               ' show
response
                  #endif
                  If Left(s , 3) = "250" Then              ' ok
                     Tempw = Tcpwrite(i , "MAIL FROM:<tcpip@test.com>
{013}{010}")       ' send from address
                     Tempw = Tcpread(i , S)                ' get
response
                     #if Debug
                        Print S
                     #endif
                     If Left(s , 3) = "250" Then           ' ok
                        Tempw = Tcpwrite(i , "RCPT TO:<tcpip@test.com>
{013}{010}")       ' send TO address
                        Tempw = Tcpread(i , S)             ' get
response
                        #if Debug
                           Print S
                        #endif
                        If Left(s , 3) = "250" Then        ' ok
                           Tempw = Tcpwrite(i , "DATA{013}{010}")
' speicfy that we are going to send data
                           Tempw = Tcpread(i , S)          ' get
response
                           #if Debug
                              Print S
                           #endif
                           If Left(s , 3) = "354" Then     ' ok
                              Tempw = Tcpwrite(i , "From: tcpip@test.com
{013}{010}")
                              Tempw = Tcpwrite(i , "To: tcpip@test.com
{013}{010}")
                              Tempw = Tcpwrite(i , "Subject: BASCOM SMTP
test{013}{010}")
                              Tempw = Tcpwrite(i , "X-Mailer: BASCOM
SMTP{013}{010}")
                              Tempw = Tcpwrite(i , "{013}{010}")
```

```
                                    Tempw = Tcpwrite(i , "This is a test email
from BASCOM SMTP{013}{010}")
                                    Tempw = Tcpwrite(i , "Add more lines as
needed{013}{010}")
                                    Tempw = Tcpwrite(i , ".{013}{010}")
' end with a single dot

                                    Tempw = Tcpread(i , S)          ' get
response
                                    #if Debug
                                      Print S
                                    #endif
                                    If Left(s , 3) = "250" Then     ' ok
                                       Tempw = Tcpwrite(i , "QUIT{013}{010}")
     ' quit connection
                                       Tempw = Tcpread(i , S)
                                       #if Debug
                                          Print S
                                       #endif
                                    End If
                                 End If
                              End If
                           End If
                        End If
                     End If
                  End If
               Case Sock_close_wait
                  Print "CLOSE_WAIT"
                  Closesocket I                                 ' close the
connection
               Case Sock_closed
                  Print "Socket CLOSED"                         ' socket is
closed
                  End
            End Select
         Loop
      End If
End If
End                                                             'end program
```

7.143.2 UDPREAD

Action

Reads data via UDP protocol.

Syntax

Result = **UDPREAD**(socket , var, bytes)

Remarks

Result	A byte variable that will be assigned with **0**, when no errors occurred. When an error occurs, the value will be set to **1**. When there are not enough bytes in the reception buffer, the routine will wait until there is enough data or the socket is closed.
socket	The socket number you want to read data from (0-3). Or 0-7 for W5200/ W5300
Var	The name of the variable that will be assigned with the data from the socket.

Bytes	The number of bytes to read.

Reading strings is not supported for UDP.
When you need to read a string you can use the OVERLAY option of DIM.

There will be no check on the length so specifying to receive 2 bytes for a byte will overwrite the memory location after the memory location of the byte.

W3100

The socketstat function will return a length of the number of bytes + 8 for UDP. This because UDP also includes an 8 byte header. It contains the length of the data, the IP number of the peer and the port number.

The UDPread function will fill the following variables with this header data:

Peersize, PeerAddress, PeerPort

These variables are dimensioned automatically when you use CONFIG TCPIP.

W5100,W5200,W5300

The peersize, peerport and peeraddress have a different order in the W5x00. To avoid mistakes, the compiler will create these variables automatic in the proper order. The NOUDP=1 option can disable this feature if you do not use UDP.
When reading UDP, you need to use the UDPREADHEADER 1585 statement to read the UDP header. After reading the header, the peersize, peerport and peeraddress variables are set.
You then should use the **peersize** variable to determine the number of bytes to retrieve. You must read all these bytes.

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , UDPWRITE 1588, UDPWRITESTR 1589 , UDPREADHEADER 1585 , IP2STR 1555, URL2IP 1592

Example W3100

```
'-------------------------------------------------------------------------
------------------
'name                     : udptest.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : start the easytcp.exe program after the chip
is programmed and
'                          press UDP button
'micro                    : Mega161
'suited for demo          : no
'commercial addon needed  : yes
'-------------------------------------------------------------------------
------------------

$regfile = "m161def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
```

```
$hwstack = 32                                                ' default
use 32 for the hardware stack
$swstack = 10                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space

Print "Init , set IP to 192.168.0.8"                        ' display a
message
Enable Interrupts                                            ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx =
$55 , Rx = $55

'Use the line below if you have a gate way
'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx
= $55 , Rx = $55

Dim Idx As Byte                                              ' socket
number
Dim Result As Word                                           ' result
Dim S(80) As Byte
Dim Sstr As String * 20
Dim Temp As Byte , Temp2 As Byte                             ' temp bytes
'-----------------------------------------------------------------------
---------------------
'When you use UDP, you need to dimension the following variables in
exactly the same order !
Dim Peersize As Integer , Peeraddress As Long , Peerport As Word
'-----------------------------------------------------------------------
---------------------
Declare Function Ipnum(ip As Long) As String                 ' a handy
function

'like with TCP, we need to get a socket first
'note that for UDP we specify sock_dgram
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)                 ' get socket
for UDP mode, specify port 5000
Print "Socket " ; Idx ; " " ; Idx

'UDP is a connection less protocol which means that you can not listen,
connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the addition
of the IP and PORT
Do
   Temp = Inkey()                                            ' wait for
terminal input
   If Temp = 27 Then                                         ' ESC
pressed
      Sstr = "Hello"
      Result = Udpwritestr(192.168.0.3 , 5000 , Idx , Sstr , 255)
   End If
   Result = Socketstat(idx , Sel_recv)                       ' get number
of bytes waiting
   If Result > 0 Then
      Print "Bytes waiting : " ; Result
      Temp2 = Result - 8                                     'the first 8
bytes are always the UDP header which consist of the length, IP number
and port address
```

```
      Temp = Udpread(idx , S(1) , Result)                    ' read the
result
      For Temp = 1 To Temp2
          Print S(temp) ; " " ;                              ' print
result
      Next
      Print
      Print Peersize ; " " ; Peeraddress ; " " ; Peerport    ' these are
assigned when you use UDPREAD
      Print Ipnum(peeraddress)                               ' print IP
in usual format
      Result = Udpwrite(192.168.0.3 , Peerport , Idx , S(1) , Temp2)
  ' write the received data back
   End If
Loop
'the sample above waits for data and send the data back for that reason
temp2 is subtracted with 8, the header size

'this function can be used to display an IP number in normal format
Function Ipnum(ip As Long) As String
    Local T As Byte , J As Byte
    Ipnum = ""
    For J = 1 To 4
      T = Ip And 255
      Ipnum = Ipnum + Str(t)
      If J < 4 Then Ipnum = Ipnum + "."
      Shift Ip , Right , 8
    Next
End Function
End
```

7.143.2 UDPREADHEADER

Action

This statement reads the UDP header from the specified socket.

Syntax

UDPREADHEADER socket

Remarks

UDP packets start with a 8 byte header. This header contains the peer address, port and packet size. The UDPREADHEADER reads the header and places the information into the variables PEERADDRESS, PEERPORT and PEERSIZE.

After you have read the UDP header, you can use PEERSIZE to read the number of bytes available in the packet.

Socket is a constant or variable in the range from 0-3. And 0-7 for the W5200/ W5300.
UDPREADHEADER is only available for the W5x00.

See also

UDPREAD 1582, CONFIG TCPIP 1098 , TCPREADHEADER 1577

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : udptest_SPI.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : start the easytcp.exe program after the chip
is programmed and
'                          press UDP button
'micro                    : Mega88
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------------
------------------

$regfile = "m88def.dat"                                    ' specify
the used micro

$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 64                                              ' default
use 32 for the hardware stack
$swstack = 64                                              ' default
use 10 for the SW stack
$framesize = 50                                            ' default
use 40 for the frame space


Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes ,
Polarity = Low , Phase = 0 , Clockrate = 4 , Noss = 0
'Init the spi pins
Spiinit

Print "Init , set IP to 192.168.1.70"                      ' display a
message
Enable Interrupts                                          ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int1 , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 5000 , Tx
= $55 , Rx = $55 , Chip = W5100 , Spi = 1



Dim Idx As Byte                                            ' socket
number
Dim Result As Word                                         ' result
Dim S(255) As Byte
Dim Sstr As String * 255
Dim Temp As Byte , Temp2 As Byte                           ' temp bytes

Const Showresult = 1

Print "UDP demo"

Dim Ip As Long
Ip = Maketcp(192.168.1.3)                                  'assign IP
num

'like with TCP, we need to get a socket first
'note that for UDP we specify sock_dgram
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)               ' get socket
for UDP mode, specify port 5000
Print "Socket " ; Idx ; " " ; Idx
```

```
'UDP is a connection less protocol which means that you can not listen,
connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the addition
of the IP and PORT
Do
   Temp = Inkey()                                          ' wait for
terminal input
   If Temp = 27 Then                                       ' ESC
pressed
      Sstr = "Hello"
      Result = Udpwritestr(ip, 5000, Idx, Sstr, 255)
   Elseif Temp = 32 Then                                   ' space
      Do
         Waitms 200
         Dim Tel As Long : Incr Tel
         Sstr = "000000000011111111112222222223333333333 " + Str(tel)
         Result = Udpwritestr(ip , 5000 , Idx , Sstr , 255)
      Loop
   End If
   Result = Socketstat(idx, Sel_recv)                      ' get number
of bytes waiting
   If Result > 0 Then
      Print "Bytes waiting : " ; Result

      Udpreadheader Idx                                    ' read the
udp header

      #if Showresult
         Print
         Print Peersize; " "; Peeraddress; " "; Peerport   ' these are
assigned when you use UDPREAD
         Print Ip2str(peeraddress)                         ' print IP
in usual format
      #endif


      If Peersize > 0 Then                                 ' the actual
number of bytes
         Print "read" ; Peersize
         Temp = Udpread(idx, S(1), Peersize)             ' read the
result

        #if Showresult
           For Temp = 1 To Peersize
              Print S(temp); " " ;                        ' print
result
           Next
        Print "done"
        #endif
       Result = Udpwrite(ip, Peerport, Idx, S(1), Peersize) ' write the
received data back
      End If
   End If
Loop
'the sample above waits for data and send the data back for that reason
temp2 is subtracted with 8, the header size


End
```

7.143.2 UDPWRITE

Action
Write UDP data to a socket.

Syntax
Result = **UDPwrite**(IP, port, socket , var , bytes)
Result = **UDPwrite**(IP, port, socket , EPROM, address , bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
IP	The IP number you want to send data to. Use the format 192.168.0.5 or use a LONG variable that contains the IP number.
Port	The port number you want to send data too.
Socket	The socket number you want to send data to(0-3).
Var	A constant string like "test" or a variable. When you send a constant string, the number of bytes to send does not need to be specified.
Bytes	A word variable or numeric constant that specifies how many bytes must be send.
Address	The address of the data stored in the chips internal EEPROM. You need to specify EPROM too in that case.
EPROM	An indication for the compiler so it knows that you will send data from EPROM.

The UDPwrite function can be used to write data to a socket that is stored in EEPROM or in memory.
When you want to send data from an array, you need to specify the element : var (idx) for example.

Note that UDPwrite is almost the same as TCPwrite. Since UDP is a connection-less protocol, you need to specify the IP address and the port number.

 UDP only requires an opened socket. The is no connect or close needed.

See also
CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITESTR 1579, TCPREAD 1576, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , UDPWRITESTR 1589 , UDPREAD 1582 , UDPREADHEADER 1585, URL2IP 1592

Example
See UDPwriteStr 1589

7.143.2 UDPWRITESTR

Action
Sends a string via UDP.

Syntax
Result = **UDPwriteStr**(IP, port, socket , var , param)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
IP	The IP number you want to send data to. Use the format 192.168.0.5 or use a LONG variable that contains the IP number.
Port	The port number you want to send data too.
Socket	The socket number you want to send data to (0-3).
Var	The name of a string variable.
Param	A parameter that might be 0 to send only the string or 255, to send the string with an additional CR + LF This option was added because many protocols expect CR + LF after the string.

The UDPwriteStr function is a special variant of the UDPwrite function.
It will use UDPWrite to send the data.

See also
CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPREAD 1576, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , UDPWRITE 1588, UDPREAD 1582, URL2IP 1592

Example
```
'-----------------------------------------------------------------------
-----------------
'name                    : udptest.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : start the easytcp.exe program after the chip
is programmed and
'                          press UDP button
'micro                   : Mega161
'suited for demo         : no
'commercial addon needed  : yes
```

```
'-------------------------------------------------------------------------
-----------------

$regfile = "m161def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

Const Sock_stream = $01                                     ' Tcp
Const Sock_dgram = $02                                      ' Udp
Const Sock_ipl_raw = $03                                    ' Ip Layer
Raw Sock
Const Sock_macl_raw = $04                                   ' Mac Layer
Raw Sock
Const Sel_control = 0                                       ' Confirm
Socket Status
Const Sel_send = 1                                          ' Confirm Tx
Free Buffer Size
Const Sel_recv = 2                                          ' Confirm Rx
Data Size

'socket status
Const Sock_closed = $00                                     ' Status Of
Connection Closed
Const Sock_arp = $01                                        ' Status Of
Arp
Const Sock_listen = $02                                     ' Status Of
Waiting For Tcp Connection Setup
Const Sock_synsent = $03                                    ' Status Of
Setting Up Tcp Connection
Const Sock_synsent_ack = $04                                ' Status Of
Setting Up Tcp Connection
Const Sock_synrecv = $05                                    ' Status Of
Setting Up Tcp Connection
Const Sock_established = $06                                ' Status Of
Tcp Connection Established
Const Sock_close_wait = $07                                 ' Status Of
Closing Tcp Connection
Const Sock_last_ack = $08                                   ' Status Of
Closing Tcp Connection
Const Sock_fin_wait1 = $09                                  ' Status Of
Closing Tcp Connection
Const Sock_fin_wait2 = $0a                                  ' Status Of
Closing Tcp Connection
Const Sock_closing = $0b                                    ' Status Of
Closing Tcp Connection
Const Sock_time_wait = $0c                                  ' Status Of
Closing Tcp Connection
Const Sock_reset = $0d                                      ' Status Of
Closing Tcp Connection
Const Sock_init = $0e                                       ' Status Of
Socket Initialization
Const Sock_udp = $0f                                        ' Status Of
Udp
Const Sock_raw = $10                                        ' Status of
IP RAW
```

```
$lib "tcpip.lbx"                                            ' specify
the tcpip library
Print "Init , set IP to 192.168.0.8"                       ' display a
message
Enable Interrupts                                           ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx =
$55 , Rx = $55

'Use the line below if you have a gate way
'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx
= $55 , Rx = $55

Dim Idx As Byte                                             ' socket
number
Dim Result As Word                                          ' result
Dim S(80) As Byte
Dim Sstr As String * 20
Dim Temp As Byte , Temp2 As Byte                            ' temp bytes
'-------------------------------------------------------------------------
---------------------
'When you use UDP, you need to dimension the following variables in
exactly the same order !
Dim Peersize As Integer , Peeraddress As Long , Peerport As Word
'-------------------------------------------------------------------------
---------------------
Declare Function Ipnum(ip As Long) As String                ' a handy
function

'like with TCP, we need to get a socket first
'note that for UDP we specify sock_dgram
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)                ' get socket
for UDP mode, specify port 5000
Print "Socket " ; Idx ; " " ; Idx

'UDP is a connection less protocol which means that you can not listen,
connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the addition
of the IP and PORT
Do
   Temp = Inkey()                                           ' wait for
terminal input
   If Temp = 27 Then                                        ' ESC
pressed
      Sstr = "Hello"
      Result = Udpwritestr(192.168.0.3 , 5000 , Idx , Sstr , 255)
   End If
   Result = Socketstat(idx , Sel_recv)                      ' get number
of bytes waiting
   If Result > 0 Then
      Print "Bytes waiting : " ; Result
      Temp2 = Result - 8                                    'the first 8
bytes are always the UDP header which consist of the length, IP number
and port address
      Temp = Udpread(idx , S(1) , Result)                   ' read the
result
```

```
      For Temp = 1 To Temp2
          Print S(temp) ; " " ;                                    ' print
result
      Next
      Print
      Print Peersize ; " " ; Peeraddress ; " " ; Peerport    ' these are
assigned when you use UDPREAD
      Print Ipnum(peeraddress)                                 ' print IP
in usual format
      Result = Udpwrite(192.168.0.3 , Peerport , Idx , S(1) , Temp2)
  ' write the received data back
   End If
Loop
'the sample above waits for data and send the data back for that reason
temp2 is subtracted with 8, the header size

'this function can be used to display an IP number in normal format
Function Ipnum(ip As Long) As String
    Local T As Byte , J As Byte
    Ipnum = ""
    For J = 1 To 4
      T = Ip And 255
      Ipnum = Ipnum + Str(t)
      If J < 4 Then Ipnum = Ipnum + "."
      Shift Ip , Right , 8
    Next
End Function
End
```

7.143.2 URL2IP

Action

This function returns the IP address of an URL.

Syntax

ip=**URL2IP(**URL**)**

Remarks

This function performs a DNS query to the google DNS server with address 8.8.8.8.
It returns either a 0 IP address or the IP address of the URL.

The URL must be a string or string constant.
At the moment, this function is only supported by the W5100 and W5200.

See also

CONFIG TCPIP 1098, GETSOCKET 1553 , SOCKETCONNECT 1567, SOCKETSTAT 1571 , TCPWRITE 1578, TCPWRITESTR 1579, CLOSESOCKET 1564 , SOCKETLISTEN 1571 , BASE64ENC 1549

Example

```
'-----------------------------------------------------------------------
-------------------
```

```
'name                       : PING_SPI.bas
http://www.faqs.org/rfcs/rfc792.html
'copyright                  : (c) 1995-2025, MCS Electronics
'purpose                    : Simple PING program
'micro                      : Mega88
'suited for demo            : yes
'commercial addon needed    : no
'-------------------------------------------------------------------------
------------------
$regfile = "m88def.dat"                                     ' specify
the used micro

$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 80                                               ' default
use 64 for the hardware stack
$swstack = 64                                               ' default
use 64 for the SW stack
$framesize = 180                                            ' default
use 80 for the frame space


Const Cdebug = 1

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0


'Configuration Of The SPI bus
Config Spi = Hard , Interrupt = Off , Data_order = Msb , Master = Yes ,
Polarity = Low , Phase = 0 , Clockrate = 4 , Noss = 0
'Init the spi pins
Spiinit


'we do the usual
Print "Init TCP"                                           ' display a
message
Enable Interrupts                                           ' before we
use config tcpip , we need to enable the interrupts
Config Tcpip = Noint , Mac = 12.128.12.34.56.78 , Ip = 192.168.1.70 ,
Submask = 255.255.255.0 , Gateway = 192.168.1.1 , Localport = 1000 ,
Tx = $55 , Rx = $55 , Chip = W5100 , Spi = 1 , Cs = Portb.2
Print "Init done"

Dim Idx As Byte , Result As Word , J As Byte , Res As Byte
Dim Ip As Long
Dim Dta(12) As Byte , Rec(12) As Byte


Dta(1) = 8                                                  'type is
```

```
echo
Dta(2) = 0                                                  'code

Dta(3) = 0                                                  ' for
checksum initialization
Dta(4) = 0                                                  ' checksum
Dta(5) = 0                                                  ' a
signature can be any number
Dta(6) = 1                                                  '
signature
Dta(7) = 0                                                  ' sequence
number - any number
Dta(8) = 1
Dta(9) = 65

Dim W As Word At Dta(1) + 2 Overlay                         'same as
dta(3) and dta(4)
Dim B As Byte
W = Tcpchecksum(dta(1) , 9)                                 ' calculate
checksum and store in dta(3) and dta(4)

#if Cdebug
  For J = 1 To 9
    Print Dta(j)
  Next
#endif

Ip = Url2ip( "mcselec.com")
Print Ip2str(ip)
If Ip = 0 Then End


Print "Socket " ; Idx ; " " ; Idx
Setipprotocol Idx , 1                                       'set
protocol to 1
'the protocol value must be set BEFORE the socket is openend

Idx = Getsocket(idx , 3 , 5000 , 0)

Do
'   Result = Gettcpregs(&H403 , 2) : Print Hex(result)

'    Print Hex(s_status(1))
   Result = Udpwrite(ip , 7 , Idx , Dta(1) , 9)             'write ping
data                '
   Print "W:" ; Result
   Waitms 300                                               ' depending
on the hops, speed, etc
   Result = Socketstat(idx , Sel_recv)                      'check for
data
   Print "REC:" ; Result
   If Result >= 11 Then
      Print "Ok"
```

```
      Res = Tcpread(idx , Rec(1) , Result)                    'get data
with TCPREAD !!!
      #if Cdebug
        Print "DATA RETURNED :" ; Res                         '
        For J = 1 To Result
          Print Rec(j) ; " " ;
        Next
        Print
      #endif
   Else                                                       'there
might be a problem
      Print "Network not available"
   End If
   Waitms 10000
Loop
```

7.144 TOGGLE

Action

Toggles(inverts) the state of an output pin or bit/Boolean variable. When used on a numeric variable, all bits in the variable are inverted.

Syntax

TOGGLE pin
TOGGLE var

Remarks

pin	Any port pin like PORTB.0 or boolean variable. A port pin must be configured as an output pin before TOGGLE will have effect.
var	A numeric variable like byte, word, integer or long. When you invert a byte, all bits of that byte will be inverted.

With TOGGLE you can simply invert the output state of a port pin.
When the pin is driving a relay for example and the relay is OFF, one TOGGLE statement will turn the relays ON. Another TOGGLE will turn the relays OFF again.

When TOGGLE is used with a variable of the type Byte, Word, Integer or Long, all bits in the variable are toggled. It has the same effect as using the EXOR boolean operand with $FF, $FFFF or $FFFFFFFF

Example:

Toggle Var_byte has the same effect as

Var_byte = Var_byte XOR &HFF

New AVR chips have an enhanced port architecture which allow a toggle of the PORT by setting the PIN register to 1. The DAT files have a setting under the [DEVICE] section named NEWPORT.
When the value is 1, the PIN register will be set to toggle the PORT pin. When the NEWPORT value is set to 0, an XOR will be used to toggle the port pin.

TOGGLE can also be used on numeric variables. It will invert all bits in the variable. It

has the same effect as NOT.
var = NOT var ' invert all bits

See also
CONFIG PORT

ASM
NONE

Example
```
'Bascom Help, Nov 16, 2008
'ToggleNov15_2008.bas
'Program example for use in the Help-files for
'       TOGGLE

'Program has been compiled and tested using Bascom 1.11.9.2.003
'Nard Awater, November 16, 2008


$baud = 19200
$crystal = 16000000
$regfile = "m32def.dat"

$hwstack = 40
$swstack = 20
$framesize = 20

Dim B As Byte , W As Word , I As Integer , L As Long
Led Alias Portb.0                                           'the anode
of the LED connected to PortB.0, cathode with resistor (470 Ohm) to
ground
Config Pinb.0 = Output

B = 0
Reset Led
'Toggle the led
Do
   Print "Led is off "
   Waitms 500
   Toggle Led
   Print "Led is on "
   Waitms 500
   Toggle Led
   Incr B
Loop Until B = 5

'Toggle a bit in a variable
B = &B11110000                                              'assign a
new value: 240 in decimal
Toggle B.0
Print "B in decimal " ; B                                   ' print it:
result = 241 ; bit0 is set
Print Bin(b)                                                ' print it:
result = 11110001
Toggle B.0
Print "B in decimal " ; B                                   ' print it:
result = 240 ; bit0 is reset
Print Bin(b)                                                ' print it:
```

```
result = 11110000


W = &H000F                                                    '15 in
decimal
I = &H00FF                                                    '255 in
decimal
L = &H00CC00DD                                                '13369565 in
decimal
Toggle W
Print "toggled W= " ; W                                       ' print it:
result =  65520
Print Hex(w)                                                  ' print it:
result = &HFFF0

Toggle I
Print "toggled I= " ; I                                       ' print it:
result = -256 ; two's complement !
Print Hex(i)                                                  ' print it:
result = &HFF00

Toggle L
Print "toggled L= " ; L                                       ' print it:
result = -13369566  ;  two's complement !
Print Hex(l)                                                  ' print it:
result = &HFF33FF22

End
```

7.145 TYPE

Action

Defines a memory container using normal data types.

Syntax

```
TYPE SomeName
  mem1 As DataType
  memn As DataType
END TYPE
```

Remarks

Type describes a container of data types. It does not use any memory. Memory allocation is done using DIM.
You start a new Type by using the **TYPE** statement. It must be followed by the type name.
On new lines you enter the member names followed by their data type. This is just like the DIM statement except that you only provide the data type.
You can also use an earlier defined data type.
It is also possible to create an array by specifying an index. This index is one dimensional.
You end the definition on a new line using **END TYPE**.

Some examples :

```
Type Rectest
   Naam As String * 9                                         '0-9,
```

```
10 bytes
   B As Byte                                          '1
byte
   C As Integer                                       '2
bytes
End Type
'total size  13

Type Mem
  Ar(16) As Byte                                      '16
bytes
  X As Rectest                                        '13
bytes
End Type                                              '29
total size

Type Trec3
 J As Byte                                            ' 1
byte
End Type
'total 1 byte

Type Trec2
  N As Integer                                        '2
bytes
  W As Word                                           '2
bytes
  R As Trec3                                          '1
bytes
End Type
'total 5 bytes

Type Trec1
  B As Byte                                           '1
byte
  Q(10) As Byte                                       '10
bytes
  Z(5) As Trec2
'5x2=25 bytes
End Type
'total 36 bytes

Type Tstr
  Naam As String * 16                                 '17
bytes
  B As Byte                                           '1
bytes
  I As Integer                                        '2
bytes
```

```
  W As Word                                '2
bytes
  Dw As Dword                              '4
bytes
  L As Long                                '4
bytes
  S As Single                              '4
bytes
  D As Double                              '8
bytes
  Ar(5) As String * 12
'5x13=65
End Type
'total 107 bytes

Dim Myrec(5) As Trec1
'using DIM you refer to the type name
Dim Rec1 , Rec2 As Rectest
Dim Recar(4) As Mem
Dim Srec As Tstr
```

When you refer to a typed variable you either use the name to refer to the whole record or when you want to access a member, you use a DOT followed by the member name.
When a type contains nested types you use multiple DOTs and member names till you reach the desired member.
The same BASCOM rules apply for typed variables as normal variables.

There are some limitations since the type was not part of the compiler when designed.
Some of the limitations might be changed in the future.
- variable types can be only used in RAM and XRAM. It will not work on ERAM
- you can not perform bit operation on a type member : rec.b.1 = 1 will not work
- boolean/bit types can not be used as members.
- just like arrays, types are global and are passed by reference only

INDIRECT Types
Besides the normal types there is also the indirect type. It works exact the same but when you DIM the variable that uses the type you use an additional specifier named INDIRECT
Dim Somerec As Trec1 **Indirect**

While a normal variable that uses a type uses memory determined at compilation the Indirect variable type has an internal address which need to be set by the user.
An example :

```
Dim Idrec As Tstr Indirect                'claims no memory
Const Cx = Sizeof(idrec)                  'determine the size of the variable

Dim Al(cx) As Byte                        'create an array in memory with the
size of the type
Idrec_adr = Varptr(al(1))                 'set the address to the memory of
al array
```

The address is the name of the variable with a suffix of **_adr**
So what is this good for you wonder?

When the memory would be dynamically allocated and released by a memory manager there would be no fixed memory address. So this variant is intended to be used with a memory manager.
A free memory manager exists. See the help description of Options, Compiler, Output, Extended Constants for more details.

See also

DIM 1228 , SIZEOF 1457

Example

```
'-----------------------------------------------------------------
----------------
'name                     :
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  :
'micro                    : avr64da64
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
----------------

$regfile = "AVRX64da64.dat"
$crystal = 24000000
$hwstack = 40
$swstack = 40
$framesize = 64
'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz
'Config Base = 0
'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'set up the COM por/USART
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1 , Tx_rx_xc_xd_pin =
Alt1_pa4_pa5_pa6_pa7

'a TYPE Defines a data type containing one or more elements
'each element is defined just as you do with DIM
'a TYPE does not occupy any space, it defines how much space is
used when it is DIMensioned
'below are some examples of types.
```

```
Type Rectest
   Naam As String * 9                                    '0-9,
10 bytes
   B As Byte                                             '1
byte
   C As Integer                                          '2
bytes
End Type
'total size  13

Type Mem
  Ar(16) As Byte                                         '16
bytes
  X As Rectest                                           '13
bytes
End Type                                                 '29
total size

Type Trec3
 J As Byte                                               ' 1
byte
End Type
'total 1 byte

Type Trec2
  N As Integer                                           '2
bytes
  W As Word                                              '2
bytes
  R As Trec3                                             '1
bytes
End Type
'total 5 bytes

Type Trec1
  B As Byte                                              '1
byte
  Q(10) As Byte                                          '10
bytes
  Z(5) As Trec2
'5x2=25 bytes
End Type
'total 36 bytes

Type Tstr
  Naam As String * 16                                    '17
bytes
  B As Byte                                              '1
```

```
bytes
  I As Integer                                          '2
bytes
  W As Word                                             '2
bytes
  Dw As Dword                                           '4
bytes
  L As Long                                             '4
bytes
  S As Single                                           '4
bytes
  D As Double                                           '8
bytes
  Ar(5) As String * 12
'5x1365
End Type                                                '107
bytes

'declare som subs for testing
Declare Sub Sbt(rec As Rectest)
Declare Sub Sbtest(byval B1 As Byte , B2 As Byte , Rec As Rectest
)

'dim some vars for test
Dim Ss As String * 10
Dim Ar(2) As Byte
Dim B As Byte
Dim Myrec(5) As Trec1
'using DIM you refer to the type name
Dim Bdum As Byte , Idx As Byte , Qdx As Byte , Zdx As Byte
Dim B1 As Byte
Dim Rec1 , Rec2 As Rectest
Dim Recar(4) As Mem
Dim B2 As Byte
Dim Srec As Tstr

'when you refer to a typed variable you either use the name to
refer to the whole record
'or when you want to access a member, you use a DOT followed by
the member name
'when a type contains nested types you use multiple DOTs and
membernames till you reach the member of interest.
'The same BASCOM rules apply for typed variables as normal
variables.

'There are some limitations. Some of them might be changed in the
near future
'- variable types can be only used in RAM and XRAM. It will not
work on ERAM
```

```
'- you can not do bit operation in a type member : rec.b.1 = 1
will not work
'- boolean/bit types can not be used as members.
'- types are global and are passed by reference only

B = 2
'assign a normal byte

Rec1.b = 2
'assign to the byte member

Srec.naam = "abc"
'assign a string
Srec.ar(1) = "one"                                          'ar
is an index so it must be addressed as index
Srec.ar(b) = "two"
Srec.b = B                                                  'byte
Srec.i = -1234
'integer
Srec.w = 50000                                              'word
Srec.dw = 12345678
'double word
Srec.s = 12.34
'single
Srec.d = 500.1234
'double
Bdum = 1                                                    'some
other byte
Swap Srec.ar(1) , Srec.ar(b)                                'swap
to record members

Print Srec.naam
'print content of member
Print Srec.b
'print byte value

Sbt Rec1                                                    'call
a subroutine
Sbtest Srec.b , Srec.b , Rec1                               'call
another sub

Rec1.naam = "abc"
'assign string member
Myrec(3).z(2).w = &HEEBB
'nested records assignment
Myrec(3).z(b).w = Myrec(3).z(2).w + 100                     'math
operation on a record

Myrec(1).z(2).r.j = 1
```

```
Myrec(2).z(b).r.j = B
Myrec(bdum).z(2).r.j = 3
Myrec(bdum).z(2).r.j = Myrec(2).z(2).r.j
Myrec(1) = Myrec(2)                                          'copy
entire record
Myrec(bdum) = Myrec(2)
Print Myrec(4).z(2).r.j

For Idx = 1 To 5
  Myrec(idx).b = Idx
  For Qdx = 1 To 10
    Myrec(idx).q(qdx) = Qdx
  Next
Next

Rec1 = Rec2                                                  'copy
whole record

End

'when you create a sub/function that pass a typed variable you
need to define the type name
Sub Sbtest(byval B1 As Byte , B2 As Byte , Rec As Rectest)
    B1 = B1 + 1                                              'we
add one which does not matter since we pass a local copy
    Print B1
'print the value
    B2 = B2 + 1                                              'this
is passed by reference , add 1
    Print B2
'print it, the calling value is changed as well
    Rec.b = &HB
'print the record member value
    Rec.c = &HC
End Sub

'another simple test
Sub Sbt(rec As Rectest)
    Rec.b = &HB
    Rec.c = &HC
End Sub
```

7.146 VARPTR

Action

Retrieves the memory-address of a variable.

Syntax
var = **VARPTR**(var2)
var = **VARPTR**("var3")

Remarks

Var	The variable that receives the address of var2.
Var2	A variable to retrieve the address from.
var3	A constant

Sometimes you need to know the address of a variable, for example when you like to peek at it's memory content.
The VARPTR() function assigns this address.
You can also get the address of a register using VARPTR.
The address of registers are constants you can find in the DAT file.

See also
LOADADR 1358 , SIZEOF 1457 , CONFIG VARPTRMODE 1142

Example
```
Dim W As Byte
Print Hex(varptr(w)) ' 0060 depends on processor
```

7.147 VER

Action
Returns the AVR-DOS version

Syntax
result = **VER**()

Remarks

Result	A numeric variable that is assigned with the AVR-DOS version. The version number is a byte and the first release is version 1.

When you have a problem, MCS can ask you for the AVR-DOS version number. The VER() function can be used to return the version number then.

See also
INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848 , FLUSH 790 , PRINT 1501 , LINE INPUT 794 , LOC 795 , LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , GET 1262 , PUT 1402 , FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , WRITE 801 , INPUT 1493

 The VERSION 1606() function is something different. It is intended to include compile time info into the program.

ASM

Calls	_AVRDOSVer
Input	-
Output	R16 loaded with value

Example

```
Print Ver()
```

7.148 VERSION

Action

Returns a string with the date and time of compilation.

Syntax

Var = **VERSION**(frm)

Remarks

Var is a string variable that is assigned with a constant. This version constant is set at compilation time to MM-DD-YY hh:nn:ss Where MM is the month, DD the day of the month, YY the year. hh is the hour is 24-hour format, nn the minutes, and ss the seconds.
When frm is set to 1, the format date will be shown in European DD-MM-YY hh:nn:ss format.
When frm is set to 2, the version info from $VERSION will be used.
When frm is set to 3, the filename will be used.
When frm is set to 4, the version info from $VERSION will be used without the separating dots. So 1.2.3 will become 123.
When frm is a string constant, the string constant will be used.

While it is simple to store the version of your program in the source code, it is harder to determine which version was used for a programmed chip.

The Version() function can print this information to the serial port, or to an LCD display.

See Also

VER 1605 , $VERSION 712

Example

```
Print Version()
```

7.149 WAIT

Action
Suspends program execution for a given time.

Syntax
WAIT seconds

Remarks

seconds	The number of seconds to wait.

No accurate timing is possible with this command.
When you use interrupts, the delay may be extended.

See also
DELAY 1227 , WAITMS 1607

Example
```
WAIT 3  ' wait for three seconds
Print "*"
```

7.150 WAITMS

Action
Suspends program execution for a given time in mS.

Syntax
WAITMS mS

Remarks

Ms	The number of milliseconds to wait. (1-65535)

No accurate timing is possible with this command.
In addition, the use of interrupts can slow this routine.

See also
DELAY 1227 , WAIT 1607 , WAITUS 1608

ASM
WaitMS will call the routine _WAITMS. R24 and R25 are loaded with the number of milliseconds to wait.
Uses and saves R30 and R31.
Depending on the used XTAL the asm code can look like :

```
_WaitMS:
_WaitMS1F:
```

```
 Push R30          ; save Z
 Push R31
_WaitMS_1:
 Ldi R30,$E8       ; delay for 1 mS
 Ldi R31,$03
_WaitMS_2:
 Sbiw R30,1 ; -1
 Brne _WaitMS_2    ; until 1 mS is ticked away
 Sbiw R24,1
 Brne _WaitMS_1    ; for number of mS
 Pop R31
 Pop R30
 Ret
```

Example

```
WAITMS 10  ' wait for 10 mS
Print "*"
```

7.151 WAITUS

Action

Suspends program execution for a given time in uS.

Syntax

WAITUS uS

Remarks

US	The number of microseconds to wait. (1-65535) This must be a constant. Not a variable! **In version 1.12.x.x and higher you can use a variable as well.**

No accurate timing is possible with this command. For accurate timing you can best use a timer.
In addition, the use of interrupts can slow down this routine.

The minimum delay possible is determined by the used frequency.
The number of cycles that are needed to set and save registers is 17.

When the loop is set to 1, the minimum delay is 21 uS. In this case you can better use a NOP that generates 1 clock cycle delay.
At 4 MHz the minimum delay is 5 uS. So a waitus 3 will also generate 5 uS delay.
Above these values the delay will become accurate.

In version 2.0.7.6 the compiler will create different code depending on the $CRYSTAL value and the specified delay.
When you use a constant, the timing is reasonable accurate. When using a variable, the timing accuracy depends on the oscillator speed.
As a general rule : the higher the clock speed, the better the result.

When you really need an accurate delay you should use a timer.
Set the timer to a value and poll until the overflow flag is set. The disadvantage is that you can not use the timer for other tasks during this hardware delay.

The philosophy behind BASCOM is that it should not use hardware resources unless there is no other way to accomplish a task.

See also
DELAY 1227 , WAIT 1607 , WAITMS 1607

Example
```
WAITUS 10  ' wait for 10 uS
Print "*"
```

7.152 WHILE-WEND

Action
Executes a series of statements in a loop, as long as a given condition is true.

Syntax
```
WHILE condition
  statements
WEND
```

Remarks
If the condition is true then any intervening statements are executed until the WEND statement is encountered.
BASCOM then returns to the WHILE statement and checks the condition.
If it is still true, the process is repeated.
If it is not true, execution resumes with the statement following the WEND statement.

So in contrast with the DO-LOOP structure, a WHILE-WEND condition is tested first so that if the condition fails, the statements in the WHILE-WEND structure are never executed.

See also
DO-LOOP 1243

Example
```
'-----------------------------------------------------------------------
-----------------
'name                     : while_w.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demo: WHILE, WEND
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
```

```
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

Dim A As Byte

A = 1                                                      'assign var
While A < 10                                               'test
expression
   Print A                                                 'print var
   Incr A                                                  'increase by
one
Wend                                                       'continue
loop
End
```

7.153 WRITEDAC

Action
This statement will set the DAC output value on the XTINY platform.

Syntax
WRITEDAC value

Remarks
A DAC is a digital to analog converter.

Value is a constant or numeric variable. Xtinies have an 8 bit DAC. DA/DB series have a 10 bit DAC. The 10 bit model need a specific alignment of the bits. That is the reason this statement exists since on Xtiny it will write to the DAC0_DATA register which is 1 byte.
On the AVR128DB28 the DAC0_DATA register is a word register. The two LS bits of the value are located in bit 6 and 7 of the LSB. This means that the value you write must be shifted 6 places to the left and that is what WRITEDAC will do for the platforms that use a 10 bit DAC.

In order to use the DAC you must use the CONFIG DACx statement that will enable the DAC and will set the output pin to output mode.
And you need to set the voltage reference for the DAC using CONFIG VREF 1147.

See also
CONFIG DACx 923 , CONFIG VREF 1147

Example
See the CONFIG DACx example.

7.154 WRITEEEPROM

Action
Write a variables content to the DATA EEPROM.

Syntax
WRITEEEPROM var , address

Remarks

var	The name of the variable that must be stored
address	The address in the EEPROM where the variable must be stored. A new option is that you can provide a label name for the address. See example 2.

This statement is provided for compatibility with BASCOM-8051.

You can better use :
```
Dim V as Eram Byte  'store in EEPROM
Dim B As Byte       'normal variable
B = 10
V = B               'store variable in EEPROM which is the actual writeeeprom
```

When you use the assignment version, the data types must be the same!

According to a data sheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset. It is advised not to use this location.

For security, register R23 is set to a magic value before the data is written to the EEPROM.
All interrupts are disabled while the EEPROM data is written. Interrupts are enabled automatic after the data is written.

It is advised to use the Brownout circuit that is available on most AVR processors. This will prevent that data is written to the EEPROM when the voltage drops under the specified level.

When data is written to the EEPROM, all interrupts are disabled, and after the EEPROM has been written, the interrupts are re-enabled.

In the XMEGA, you need to set the mode to mapped : CONFIG EEPROM 952 = MAPPED.

 When you define a constant named UPDATEEPROM the eprom cells will be only written when the value differs. Instead of just writing the value, the EPROM content is first read and compared to the new value. Only when the new value differs the new value is written to the EEPROM. A memory location can be written to 100.000 times at least.
The constant UPDATEEPROM can have any value. There is only a check if this constant is defined. So even : CONST UPDATEEPROM=0 will use the special update code.

See also
READEEPROM 1415

ASM

NONE

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : eeprom2.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows how to use labels with READEEPROM
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 4000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space

'first dimension a variable
Dim B As Byte
Dim Yes As String * 1

'Usage for readeeprom and writeeprom :
'readeeprom var, address

'A new option is to use a label for the address of the data
'Since this data is in an external file and not in the code the eeprom
data
'should be specified first. This in contrast with the normal DATA lines
which must
'be placed at the end of your program!!

'first tell the compiler that we are using EEPROM to store the DATA
$eeprom

'the generated EEP file is a binary file.
'Use $EEPROMHEX to create an Intel Hex file usable with AVR Studio.
'$eepromhex

'specify a label
Label1:
Data 1 , 2 , 3 , 4 , 5
Label2:
Data 10 , 20 , 30 , 40 , 50

'Switch back to normal data lines in case they are used
$data

'All the code above does not generate real object code
'It only creates a file with the EEP extension

'Use the new label option
```

```
Readeeprom B , Label1
Print B                                                    'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B                                                    'prints 2

Readeeprom B , Label2
Print B                                                    'prints 10
Readeeprom B
Print B                                                    'prints 20

'And it works for writing too :
'but since the programming can interfere we add a stop here
Input "Ready?" , Yes
B = 100
Writeeeprom B , Label1
B = 101
Writeeeprom B

'read it back
Readeeprom B , Label1
Print B                                                    'prints 100
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B                                                    'prints 101
End
```

7.155 X10DETECT

Action

Returns a byte that indicates if a X10 Power line interface is found.

Syntax

Result = **X10DETECT**( )

Remarks

Result	A variable that will be assigned with 0 if there is no Power Line Interface found. 1 will be returned if the interface is found, and the detected mains frequency is 50 Hz. 2 will be returned if the interface is found and the detected mains frequency is 60 Hz.

When no TW-523 or other suitable interface is found, the other X10 routines will not work.

See also

CONFIG X10 1156 , X10SEND 1615

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : x10.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : example needs a TW-523 X10 interface
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m48def.dat"                                    ' specify
the used micro
$crystal = 8000000                                         ' used
crystal frequency
$baud = 19200                                              ' use baud
rate
$hwstack = 32                                              ' default
use 32 for the hardware stack
$swstack = 10                                              ' default
use 10 for the SW stack
$framesize = 40                                            ' default
use 40 for the frame space


'define the house code
Const House = "M"                                          ' use code
A-P

Waitms 500                                                 ' optional
delay not really needed

'dim the used variables
Dim X As Byte

'configure the zero cross pin and TX pin
Config X10 = Pind.4 , Tx = Portb.0
'             ^--zero cross
'                          ^--- transmission pin

'detect the TW-523
X = X10detect()
Print X                                                    ' 0 means
error, 1 means 50 Hz, 2 means 60 Hz

Do
   Input "Send (1-32) " , X
   'enter a key code from 1-31
   '1-16 to address a unit
   '17 all units off
   '18 all lights on
   '19 ON
   '20 OFF
   '21 DIM
   '22 BRIGHT
   '23 All lights off
   '24 extended code
   '25 hail request
   '26 hail acknowledge
   '27 preset dim
   '28 preset dim
   '29 extended data analog
   '30 status on
   '31 status off
```

```
   '32 status request

   X10send House , X                                          ' send the
code
Loop

Dim Ar(4) As Byte
X10send House , X , Ar(1) , 4                                 ' send 4
additional bytes
End
```

7.156 X10SEND

Action
Sends a house and key code with the X10 protocol.

Syntax
X10SEND house , code

Remarks

House	The house code in the form of a letter A-P. You can use a constant, or you can use a variable
Code	The code or function to send. This is a number between 1-32.

The X10SEND command needs a TW-523 interface.
Only ground, TX and Zero Cross, needs to be connected for transmission.
Use CONFIG X10 to specify the pins.

X10 is a popular protocol used to control equipment via the mains. A 110 KHz signal is added to the normal 50/60 Hz , 220/110 V power.

Notice that experimenting with 110V-240V can be very dangerous when you do not know exactly what you are doing !!!

In the US, X10 is very popular and wide spread. In Europe it is hard to get a TW-523 for 220/230/240 V.

I modified an 110V version so it worked for 220V. On the Internet you can find modification information. But as noticed before, MODIFY ONLY WHEN YOU UNDERSTAND WHAT YOU ARE DOING.

A bad modified device could result in a fire, and your insurance will most likely not pay. A modified device will not pass any CE, or other test.

When the TW-523 is connected to the mains and you use the X10SEND command, you will notice that the LED on the TW-523 will blink.

The following table lists all X10 codes.

Code value	Description
1-16	Used to address a unit. X10 can use a maximum of 16 units per house code.
17	All units off

18	All lights on
19	ON
20	OFF
21	DIM
22	BRIGHT
23	All lights off
24	Extended ode
25	Hail request
26	Hail acknowledge
27	Preset dim
28	Preset dim
29	Extended data analog
30	Status on
31	Status off
32	Status request

At www.x10.com you can find all X10 information. The intension of BASCOM is not to learn you everything about X10, but to show you how you can use it with BASCOM.

See also
CONFIG X10 1156 , X10DETECT 1613 , X10SEND 1615

Example
See X10DETECT 1613

Part VIII

8 ASM Libraries and Add-Ons

ASM Libs are libraries that are used by the compiler.
They contain machine language statements for various statements and functions.

A library can also be used to modify an existing function.
For example when you use a conversion routine num<>string with a byte variable only, the routine from the MCS.LIB has some overhead as it can also convert integers, word and longs.

You can specify the MCSBYTE.LIB or MCSBYTE.LBX library then to override the function from MCS.LIB.

When you write a user sub/function that calls a user lib and passed parameters, you must include some code to restore the frame protection.
In the bcd.lib you can find code like :

```
#IF _FPROTECT
  Out sreg,r3                ; restore I flag
#ENDIF
```

The bcd.lib and bin2bcd.bas demonstrate how to write a user lib.
The mylib.lib example contains more details about passing variables.

A lib must be included with the $LIB directive.
When the library contains an overloaded version of a sub/function it is important that you put the $LIB directive early in your code.

You can also change the behavior of a sub by putting the code in your own lib.

For example consider the _FLIPBYTE code from mcs.lib

```
;flip or mirror bits in register R24
;1001_0000 becomes 0000_1001
[_FLIPBYTE]
_FLIPBYTE:
  push r16          ; save regs
  push r17
  ldi r16,8         ; num of bits
_FLIPBYTE2:
  rol r24           ; rotate left through carry
  ror r17           ; get in r17 rotate right
  dec r16           ; next
  brne _FLIPBYTE2
  mov r24, r17
  pop r17
  pop r16
ret
[END]
```

you could put this code into a new lib named fliplib.lib
Then include it with $|LIB "fliplib.lib"
The compiler will use the code from your lib when you include it. The lib code is automatically called when using FLIP() .

You may not share libs where you copied the code from the BASCOM libraries. These are protected by copyright. So when it is required to share libs you need to ask permission.
When you share a forked lib with another user that has a valid license there is no problem. But when you modify some code and put it on the internet, it could be a problem.

You can access variables by using brackets :
lds r24, {bSomeVar} ; load into r24 the content from the byte variable bSomeVar

When you want to compile your own lib using LibManager, you should put a asterisk infrom of the code like :
* lds r24, {bSomeVar} ; load into r24 the content from the byte variable bSomeVar

This will leave the line as it is. This is required because when you compile the lib there is no reference to the variable. This variable exist only in your main program
The * will leave the line alone and it will be compiled during compilation of the main program.

What to save?
There is no need to save registers. You can trash them all.
When it is required to change RAMPZ you should protect the value. Then like with normal asm programming, there are some registers you should not alter like R4,R5, R6,R8-R9 and the Y pointer R28-R29.

8.1 FT800

The FT800 is no toy, this is a proper GPU (Graphics Processor Unit) which has some outstanding abilities.

The Bascom FT800 Library is set up and written to give the user the Framework needed to use this IC.

Please also check some of the video demos to appreciate what you can accomplish!
YouTube demos

Some methods and habits need to change when using the FT800 in respect to a standard graphics LCD.
Here are some points and features (additional points added from the Manufacturers advertisement).

- You now have to update/refresh Screen on every graphics changes (think of it as the old Cartoon drawings flicking through paper).

- You don't need to keep on update/refresh Screen if you want a static image, you can do something else in the meantime.
- For the Experienced, look also at the FT800 Interrupts Int_CmdFlag, looks likes some benefits can be made.
- You can update/refresh the FT800 up to 60Hz, many types of animations can be implemented.
- Don't need many I/O pins, SPI or I2C is all you need (SPI is recommended).
- If you are in the graphics loop careful when you want to access Serial data or other hardware reads, the loop can only cycle at 60Hz so this can slow you down miss data if you are not aware (use other methods).
- Careful using other SPI devices on the same SPI bus, see How to add another SPI device with the FT800 1708
- To create more extensive fancy Graphics, previous experience with other Graphics engines is very helpful, requires some knowledge.
- Unfortunately the FT800 version is only for small LCD screen (largest found is 5" 480x272 - as of 10/2014)
- No custom hardware required, see Getting Started 1707 for some links of ready made boards/bezels to get your project started.
- No real Reset but Hybrid Software type.
- If using any Arduino model boards - use a descent (thick) USB cable if you are trying to program and power using the same cable.

Some USB cables are not good quality, when trying to power the LCD and the Arduino board, you can get a voltage drop getting unexpected results not knowing what is wrong at the time. It is highly recommended you use and external power supply especially if you are a beginner!.

- It's got sound synthesizer and audio playback (mono).

Many of the Documentation and Specifications can be downloaded directly from FTDI's website : FTDI/FT800

Please note:

The Help File for the Bascom FT800 is very much taken from FTDI's FT800 Series Programmer Guide.PDF with some changes.
Currently the Help file is a Work in Progress which means it may contain some error (s) and may not be complete, so if in doubt try to consult with the FT800 Series Programmer Guide.PDF from FTDI if/when having any difficulties. Some of the FTDI explanations are not clear and require better more work, though some sperate Document have been released giving more detail.

And last if you find any errors or have suggestions/improvements or even any feedback, please send an email to support@mcselec.com.

FT801

In version 2079, support for FT801 is included. The INC files have been renamed to reflect this. 800 is renamed into 80x. This means that in order to use the updates and/or new features, you need to change the names of the used include files in your project.

FT810

In version 2080, support for FT810 is included. The INC files have been renamed to reflect this. 800 is renamed into 81x. This means that in order to use the updates and/or new features, you need to change the names of the used include files in your project.

Note from MCS

The text above and all FT800 help topics and sample files are written by Peter Maroudas. Peter made the BASCOM implementation possible.
MCS has written the low level ASM FT800 library and the required compiler changes and modifications of the include files based on Peters work.

For version 2079, Peter included FT801 support.

See CONFIG FT800 959 for configuration of the library.

8.1.1 Commands

Summary of Command Groups

Setting Graphics State

AlphaFunc 1625	Set the alpha test function
BitmapHandle 1627	Set the bitmap handle
BitmapLayout 1627	Set the source bitmap memory format and layout for the current handle
BitmapSize 1628	Set the screen drawing of bitmaps for the current handle
BitmapSource 1630	Set the source address for bitmap graphics
BitmapTransformA-F 1631	Set the components of the bitmap transform matrix

BlendFunc 1633 Set pixel arithmetic
Cell 1635 Set the bitmap cell number for the VERTEX2F command
Clear_B 1635 Clear buffers to preset values
ClearColorA 1636 Set clear value for the alpha channel
ClearColorRGB 1637 Set clear values for red, green and blue channels
ClearStencil 1639 Set clear value for the stencil buffer
ClearTag 1639 Set clear value for the tag buffer
Color_A 1688 Set the current color alpha
ColorMask 1688 Enable or disable writing of color components
ColorRGB 1689 , Set the current color red, green and blue
ColorRBGdw 1690
LineWidth 1692 Set the line width
PointSize 1693 Set point size
RestoreContext 1696 Restore the current graphics context from the context stack
SaveContext 1697 Push the current graphics context on the context stack

ScissorSize 1697 Set the size of the scissor clip rectangle
ScissorXY 1698 Set the top left corner of the scissor clip rectangle
StencilFunc 1699 Set function and reference value for stencil testing
StencilMask 1700 Control the writing of individual bits in the stencil planes
StencilOp 1700 Set stencil test actions
Tag 1701 Set the current tag value
TagMask 1702 Control the writing of the tag buffer

Commands that begin and finish the display list

Begin_G 1625 Start drawing a graphics primitive
End_G 1691 Finish drawing a graphics primitive
CmdDlStart 1650 Start a New Display List

Commands to draw graphic objects

CmdText 1680 Draw Text
CmdButton 1643 Draw a Button
CmdClock 1645 Draw an analog clock
CmdBgColor 1642 Set the background color
CmdFgColor 1650 Set the foreground color
CmdGradColor 1656 Set the 3D effects for CmdButton and CmdKeys highlight color
CmdGauge 1652 Draw a gauge
CmdGradient 1657 Draw a smooth color gradient
CmdKeys 1659 Draw a row of keys
CmdProgress 1667 Draw a progress bar
CmdScrollbar 1672 Draw a scroll bar
CmdSlider 1675 Draw a slider
CmdDial 1649 Draw a rotary dial control
CmdToggle 1682 Draw a toggle switch
CmdNumber 1665 Draw a decimal number

Drawing Actions

Vertex2f 1702 Supply a vertex with fractional coordinates
Vertex2ii 1703 Supply a vertex with positive integer coordinates
CmdSetFont 1674 Set up a custom font
CmdTrack 1684 Track touches for a graphic object

Commands to operate on memory

CmdMemCRC 1663 Compute a CRC-32 for memory
CmdMemZero 1665 Write zero to a block of memory
CmdMemSet 1664 Fill memory with a byte value
CmdMemWrite 1664 Write bytes into memory
CmdMemCpy 1663 Copy a block of memory
CmdAppend 1642 Append memory to display list
CmdGetPtr 1655 Get the End memory address of inflated data

Commands for loafing image data into FT80x memory

CmdInflate 1658 Decompress data into memory
CmdLoadImage 1662 Load a JPEG image

Commands for setting the bitmap transform matrix

CmdLoadIdentity 1661 Set the current matrix to identity
CmdTranslate 1686, CmdTranslateP 1687 Apply a translation to the current matrix
CmdScale 1671 Apply a scale to the current matrix
CmdRotate 1669, CmdRotateA 1670 Apply a rotation to the current matrix
CmdSetMatrix 1674 Write the current matrix as a bitmap transform
CmdGetMatrix 1655 Retrieves the current matrix coefficients

Execution control

Jump 1691 Execute commands at another location in the display list
Macro_R 1693 Execute a single command from a macro register
Call_C 1634 Execute a sequence of commands at another location in the display list
Return_C 1696 Return from a previous CALL command
Display_E 1690 End the display list
CmdSwap 1680 Swap de current display list

Other Commands

CmdColdStart 1648 Set co-processor engine state to default values
CmdInterrupt 1659 Trigger interrupt INT_CMDFLAG
CmdRegRead 1668 Read a register value
CmdCalibrate 1644 Execute the touch screen calibration routine
CmdSpinner 1677 Start an animated spinner
CmdStop 1680 Stop any spinner, screensaver or sketch
CmdScreenSaver 1672 Start an animated screen saver
CmdSketch 1674 Start a continuous sketch update
CmdSnapshot 1677 Take a snapshot of the current screen
CmdLogo 1662 Play device logo animation

Co-Processor Engine commands

BASCOM high level commands

ClearScreen 1640	Clears the LCD with a black background
UpdateScreen 1704	Executes the commands in FIFO and refreshes LCD
WaitCmdFifoEmpty 1705	Waits for execution of commands in FIFO buffer
CMDFTSTACK 1651	Send data from the soft stack
CMD8 1640	Send a byte to the FT800 graphic processor.
CMD16 1641	Send a word to the FT800 graphic processor.
CMD32 1641	Send a dword to the FT800 graphic processor.
WR8 1705	Write an address and a byte parameter to the FT800.
WR16 1706	Write an address and a word parameter to the FT800.
WR32 1706	Write an address and a dword parameter to the FT800.

Errors
The FTERROR byte variable contains 4 flags you can examine.
FtError.0 = WaitCmdFifoEmpty Sub when Overflowed
FtError.1 = WaitCmdFifoEmpty Sub when TimeOut
FtError.2 = FreeSpaceFt Sub when OverFlowed
FtError.3 = FreeSpaceFt Sub when TimeOut

8.1.1.1 AlphaFunc

Action
Specify the Alpha Test Function

Syntax
AlphaFunc ref, func

Remarks

ref	Specifies the reference value for the alpha test. The initial value is 0
func	Specifies the test function, one of **NEVER**, **LESS**, **LEQUAL**, **GREATER**, **GEQUAL**, **EQUAL**, **NOTEQUAL**, or **ALWAYS**. The default value is **ALWAYS**.

Graphics Context
The values of func and ref are part of the graphics context, as described in section 4.1 in FT800 Series Programmer Guide.PDF from FTDI.

8.1.1.2 Begin_G

Action
Begin drawing a Graphics Primitive.

Syntax
Begin_G prim

Remarks

prim	BITMAPS	Bitmap Drawing Primitive
	FTPOINTS	Point Drawing Primitive
	LINES	Line Drawing Primitive
	LINE_STRIP	Line Strip Drawing Primitive
	EDGE_STRIP_R	Edge Strip Right side Drawing Primitive
	EDGE_STRIP_L	Edge Strip Left side Drawing Primitive
	EDGE_STRIP_A	Edge Strip Above Drawing Primitive
		Edge Strip Below Drawing Primitive
		Rectangle Drawing Primitive

	EDGE_STRIP_BRECTS	

All primitives supported by the FT800 are defined in the table above. The primitive
to be drawn is selected by the Begin_G 1625 command. Once the primitive is selected, it will be
valid till the new primitive is selected by the Begin_G 1625 command.

Please Note: The primitive drawing operation will not be performed until Vertex2ii 1703 or Vertex2f 1702 is executed.

See also

END_G 1691 , VERTEX2F 1702, VERTEX2II 1703

Example

```
' Pseudocode
Begin_G   Lines
Vertex2F (FT_DispWidth / 4) * 16, (FT_DispHeight - 25) / 2 * 16
Vertex2F (FT_DispWidth / 4) * 16, (FT_DispHeight + 25) / 2 * 16
ColorRGB 0, 128, 0
LineWidth 10 * 16

Begin_G   FTPoints
Vertex2F   50,5,00
Vertex2F    110,15,0,0
```

```
' Drawing points, lines and bitmap
Begin_G FTPOINTS
Vertex2II 50, 5, 0, 0
Vertex2II 110, 15, 0, 0
Begin_G LINES
Vertex2II 50, 45, 0, 0
Vertex2II 110, 55, 0, 0
Begin_G BITMAPS
Vertex2II 50, 65, 31,&H45
Vertex2II 110, 75, 31,&H46
```

8.1.1.3 BitmapHandle

Action
Specify the Bitmap Handle

Syntax
BitmapHandle handle

Remarks

handle	Bitmap Handle. The initial value is 0. Valid range values **0** to **31**.

Handles 16 to 31 are defined by the FT800 for built-in font.
Handle 15 is defined in the co-processor engine commands CmdGradient 1657, CmdButton 1643, and CmdKeys 1659.
Users can define new bitmaps using handles from 0 to 14.
If there is no co-processor engine command CmdGradient 1657, CmdButton 1643 and CmdKeys 1659 in the current display list, users
can even define a bitmap using handle 15.

Graphics Context
The value of handle is part of the graphics context, as described in section 4.1 in FT800 Series Programmer Guide.PDF from FTDI.

See also
BitmapLayout 1627, BitmapSize 1628

8.1.1.4 BitmapLayout

Action
Specify the source bitmap memory format and layout for the current handle.

Syntax
BitmapLayout format, linestride, height

Remarks

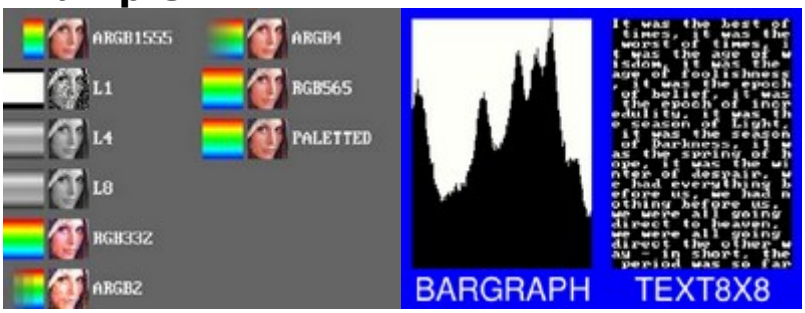
format	Bitmap Pixel Formats. **ARGB1555** **FT_L1** **FT_L4** **FT_L8** **RGB332** **ARGB2** **ARGB4** **RGB565** **PALETTED** **TEXT8x8**

	TEXTVGA BARGRAPGH
linestride	Bitmap linestride, in bytes. Please note the alignment requirement which is described below.
height	Bitmap height, in lines

The bitmap formats supported are FT_L1, FT_L4, FT_L8, RGB332, ARGB2, ARGB4, ARGB1555, RGB565 and PALETTED.
For FT_L1 format the linestride must be a multiple of 8 bits.
For FT_L4 format the linestride must be multiple of 2 nibbles (Aligned to byte).

For more details about alignment, please refer to the FT800 Series Programmer Guide.PDF from FTDI.

See also
BitmapHandle 1627, BitmapSize 1628, BitmapSource 1630

Example


8.1.1.5 BitmapSize

Action
Specify the Screen Drawing Bitmap Size (for the current Bitmap Handle)

Syntax
BitmapSize Filter, Wrapx , Wrapx ,Width, Height

Remarks

Filter	Bitmap Filtering Mode, **NEAREST** or **BILINEAR**
Wrapx	Bitmap x wrap mode, **REPEAT** or **BORDER**
Wrapx	Bitmap y wrap mode, **REPEAT** or **BORDER**
Width	Drawn bitmap Width, in Pixels
Height	Drawn bitmap Height, in Pixels

This command controls the drawing of bitmaps: the on-screen size of the bitmap, the behavior for wrapping, and
the filtering function. Please note that if Wrapx or Wrapy is using REPEAT then the corresponding memory layout dimension
(BitmapLayout 1627 linestride or height) must be power of two, otherwise the result is

undefined.

See also

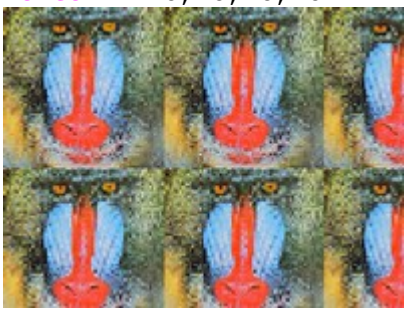
BitmapHandle 1627, BitmapLayout 1627, BitmapSource 1630

Example

```
' Pseudocode

' Drawing a 64 x 64 bitmap
BitmapSource 0
BitmapLayout RGB565, 128, 64
BitmapSIZE NEAREST, BORDER, BORDER, 64, 64
Begin_G BITMAPS
Veterx2II 48, 28, 0, 0
```

```
'Reducing the size to 32 x 50
BitmapSource 0
BitmapLayout RGB565, 128, 64
BitmapSize NEAREST, BORDER, BORDER, 32, 50
Begin_G BITMAPS
Vertex2II 48, 28, 0, 0
```

```
' Using the REPEAT wrap mode to tile the bitmap
BitmapSource 0
BitmapLayout RGB565, 128, 64
BitmapSize NEAREST,  REPEAT, REPEAT, 160, 120
Begin_G BITMAPS
Vertex2II 0, 0, 0, 0
```



' 4X zoom - 128 X 128 - using a bitmap transform

```
BitmapSource 0
BitmapLayout RGB565, 128, 64
BitmapTransformA 128
BitmapTransformE 128
BitmapSize NEAREST, BORDER,BORDER, 128, 128
Begin_G BITMAPS
Vertex2II 16, 0, 0, 0
```

8.1.1.6 BitmapSource

Action

Specify the source address of bitmap data in FT800 graphics memory RAM_G

Syntax

BitmapSource Addr

Remarks

Addr	Bitmap address in graphics FT800 SRAM, aligned with respect to the bitmap format. For example, if the bitmap format is RGB565/ARGB4/ ARGB1555, the bitmap source shall be aligned to 2 bytes.

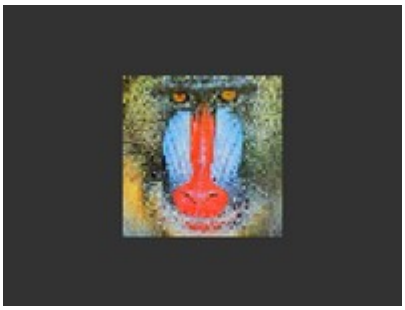
The bitmap source address is normally the address in main memory where the bitmap graphic data is loaded.

See also

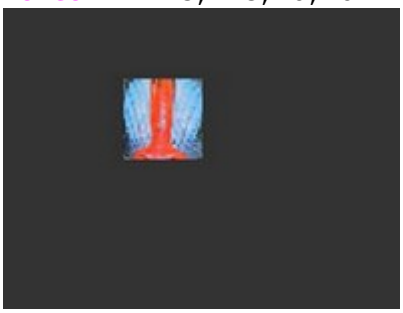
BitmapSize 1628, BitmapLayout 1627

Example

```
' Drawing a 64 x 64 bitmap, loaded at address 0
BitmapSource 0
BitmapLayout RGB565, 128, 64
BitmapSize NEAREST, BORDER, BORDER, 64, 64
Begin_G BITMAPS
Veterx2II 48, 28, 0, 0
```



```
Using the same graphics data, but with source and size changed to show
only a 32 x 32 detail
BitmapSource 128 * 16 + 32
BitmapLayout RGB565, 128, 64
BitmapSize NEAREST, BORDER, BORDER, 32, 32
Begin_G BITMAPS
Vertex2II 48, 28, 0, 0
```



8.1.1.7 BitmapTransform

Action

Specify the A-F coefficient of the Bitmap Transform Matrix.

Syntax

BitmapTransform CoefValue , CoefName

Remarks

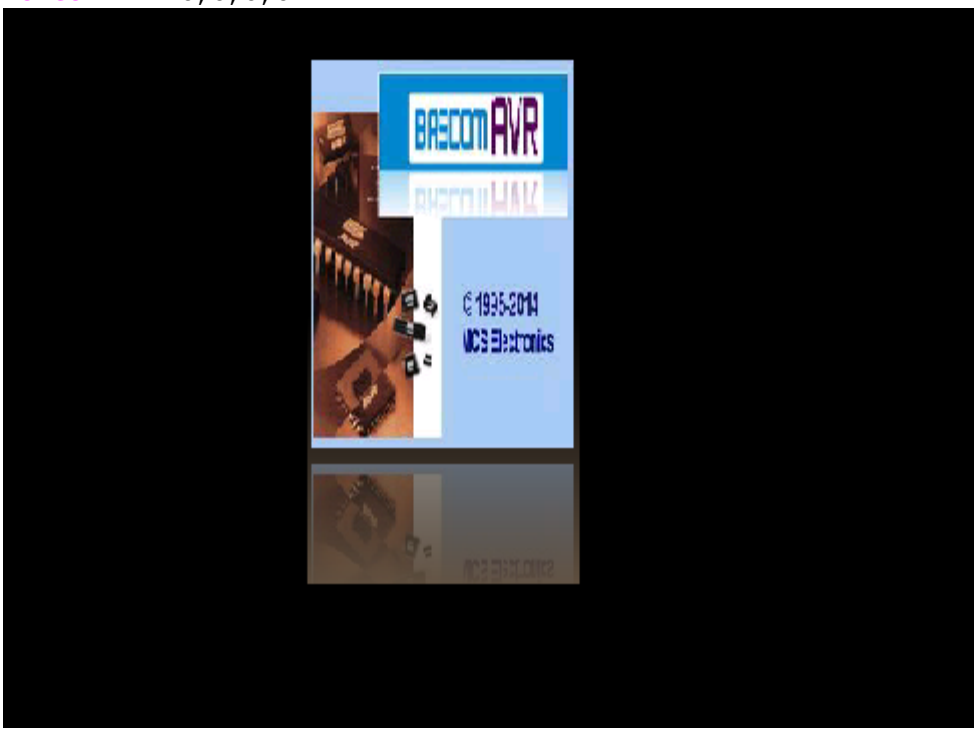
CoefValue	Coefficient value of the Bitmap Transform Matrix in signed 8.8 bit fixed-point form. The initial value is 256.
CoefName	Coeeficient name. There are coefficient A-F. You need to specify a capital letter A,B,C,D,E or F.

BitmapTransform A-F coefficients are used to perform bitmap transform functionalities such as scaling, rotation and translation.

Example

```
' Pseudocode
' A value of 0.5 (128) causes the bitmap appear double width:
BitmapSource 0
BitmapLayout RGB565, 128,64
BitmapTransform 128, A
BitmapSize Nearest, Border, Border
Begin_G Bitmaps
Vertex2II 16,0,0,0
```


```
' Pseudocode
' A value of 2.0 (512) gives a half-width bitmap:
BitmapSource 0
BitmapLayout RGB565, 128,64
BitmapTransform 512, A
BitmapSize Nearest, Border, Border
Begin_G Bitmaps
Vertex2II 16,0,0,0
```



8.1.1.8 BlendFunc

Action
Specify pixel arithmetic.

Syntax
BlendFunc src, dst

Remarks

src	Specifies how the source blending factor is computed. One of **ZERO**, **ONE**, **SRC_ALPHA**, **DST_ALPHA**,**ONE_MINUS_SRC_ALPHA** or **ONE_MINUS_DST_ALPHA**.
dst	Specifies how the destination blending factor is computed, One of **ZERO**, **ONE**, **SRC_ALPHA**, **DST_ALPHA**,**ONE_MINUS_SRC_ALPHA** or **ONE_MINUS_DST_ALPHA**.

The blend function controls how new color values are combined with the values already in the color buffer.
Given a pixel value source and a previous value in the color buffer destination, the computed color is:

source × src + destination × dst

for each color channel: red, green, blue and alpha.

For more details please refer to the FT800 Series Programmer Guide.PDF from FTDI.

See also
Color_A 1688

Example
```
' Pseudocode

' The default blend function of (SRC_ALPHA, ONE_MINUS_SRC_ALPHA) causes drawing
' to overlay the destination using the alpha value
Begin_G BITMAPS
Vertex2II 50, 30, 31, &H47
Color_A 128
Vertex2II 60, 40, 31, &H47
```

```
' A destination factor of zero means that destination pixels are not used
Begin_G BITMAPS
BlendFunc SRC_ALPHA, ZERO
Vertex2II 50, 30, 31, &H47
Color_A 128
```

```
Vertex2II 60, 40, 31, &H47
```

```
' Using the source alpha to control how much of the destination to keep
Begin_G BITMAPS
BlendFunc ZERO, SRC_ALPHA
Vertex2II 60, 40, 31, &H47
```

8.1.1.9 Call_C

Action

Execute a sequence of commands at another location in the Display List (RAM_DL).

Syntax

Call_C dest

Remarks

dest	The destination address in RAM_DL which the display command is to be switched. FT800 has the stack to store the return address. To come back to the next command of source address, the RETURN command can help.

Call_C and Return_C have a 4 level stack in addition to the current pointer.
Any additional Call_C/Return_C done will lead to unexpected behavior.

See also

JUMP 1691, RETURN_C 1696 , MACRO_R 1693 , DISPLAY_E 1690

8.1.1.10 Cell

Action
Specify the bitmap Cell number for the Vertex2f command.

Syntax
Cell Cell

Remarks

Cell	Bitmap Cell number.

See Also
VERTEX2F 1702

8.1.1.11 Clear_B

Action
Clear buffers to preset values.
(This is similar to CLS)

Syntax
Clear_B C,S,T

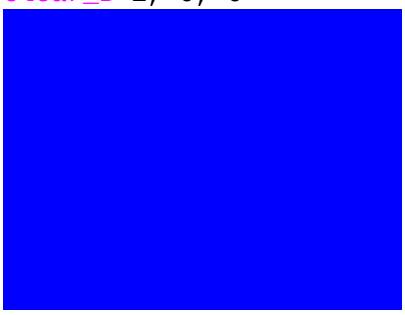
Remarks

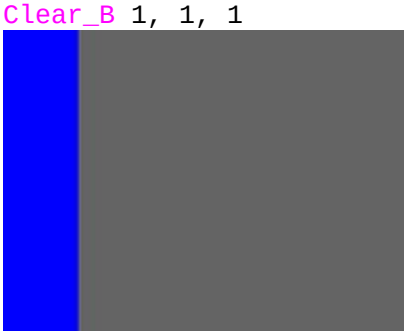
C	Clear **C**olor buffer. Setting this bit to **1** will clear the color buffer of the FT800 to the preset value. Setting this bit to **0** will maintain the color buffer of the FT800 with an unchanged value. The preset value is defined in command ClearColorRGB 1637 for the RGB channel and ClearColorA 1636 for the alpha channel.
S	Clear **S**tencil buffer. Setting this bit to **1** will clear the stencil buffer of the FT800 to the preset value. Setting this bit to **0** will maintain the stencil buffer of the FT800 with an unchanged value. The preset value is defined in command ClearStencil 1639.
T	Clear **T**ag buffer. Setting this bit to **1** will clear the tag buffer of the FT800 to the preset value. Setting this bit to **0** will maintain the tag buffer of the FT800 with an unchanged value. The preset value is defined in command ClearTag 1639.

The scissor test and the buffer write masks affect the operation of the clear. Scissor limits the cleared rectangle, and the buffer write masks limit the affected buffers.

The state of the alpha function, blend function, and stenciling do not affect the clear.

See also
ClearColorA 1636, ClearStencil 1639, ClearTag 1639, ClearColorRGB 1637

Example
```
' Pseudocode
' To Clear the LCD to bright blue:
ClearColorRGB 0, 0, 255
Clear_B 1, 0, 0
```


```
' Pseudocode
' To clear part of the screen to gray, part to blue using scissor rectangles:
ClearColorRGB 100, 100, 100
Clear_B 1, 1, 1
ClearColorRGB 0, 0, 255
ScissorScize 30, 120
Clear_B 1, 1, 1
```


8.1.1.12 ClearColorA

Action
Specify the clear value for the alpha channel.

Syntax
ClearColorA Alpha

Remarks
Alpha	Alpha value used when the color buffer is cleared. The initial value is **0**

Sets the alpha value applied to drawn elements - points, lines, and bitmaps. How the alpha value affects image pixels depends on BlendFunc 1633, the default behavior is a transparent blend.

See also

ColorRGB 1689, BlendFunc 1633

Example

```
' Pseudocode
' Drawing three characters with transparency 255, 128, and 64
Begin_G BITMAPS
Vertex2II 50, 30, 31, &H47
Color_A 128
Vertex2II 58, 38, 31, &H47
Color_A 64
Vertex2II 66, 46, 31, &H47
```

8.1.1.13 ClearColorRGB

Action

Specify the clear values for Red, Green and Blue channels.

Syntax

ClearColorRGB Red, Green, Blue

Remarks

Red	Red value used when the color buffer is cleared. The initial value is **0**
Green	Green value used when the color buffer is cleared. The initial value is **0**
Blue	Blue value used when the color buffer is cleared. The initial value is **0**

Sets the color values used by a following Clear_B 1635

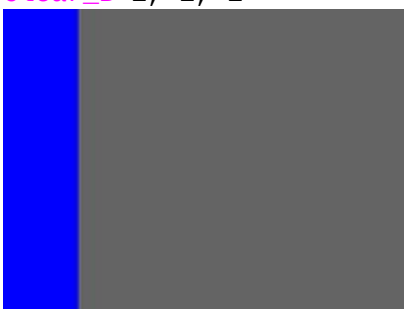
See also

ClearColorA 1636, Clear_B 1635 , ClearColorRGBdw 1638

Example

```
' Pseudocode
' To clear the screen to bright blue:
ClearColorRGB 0, 0, 255
Clear_B 1, 1, 1
```

```
' To clear part of the screen to gray, part to blue using scissor rectangles:
ClearColorRGB 100, 100, 100
Clear_B 1, 1, 1
ClearColorRGB 0, 0, 255
ScissorScize 30, 120
Clear_B 1, 1, 1
```



8.1.1.14 ClearColorRGBdw

Action

Specify the clear values for Red, Green and Blue channels.

Syntax

ClearColorRGBdw RGB

Remarks

RGB	Value in the range of **0** to **&H00FFFFFF**, Red is the most significant 8 bits and Blue is the least. So **&H**ff0000 is bright Red.

Sets the color values used by a following Clear_B 1635

The following colors are defined by constants.

Color	Value
Black	&H000000
White	&HFFFFFF
Red	&HFF0000
Lime	&H00FF00
Blue	&H0000FF
Yellow	&HFFFF00
Cyan	&H00FFFF
Magenta	&HFF00FF
Silver	&HC0C0C0
Grey	&H808080
Maroon	&H800000

Olive	&H808000
Green	&H008000
Purple	&H800080
Teal	&H008080
Navy	&H000080
Brown	&H703800
Orange	&H00A5FF

See also
ClearColorA 1636, Clear_B 1635 , ClearColorRGB 1637

Example
```
' Pseudocode
' To clear the screen to bright blue:
ClearColorRGBdw &H0000FF
Clear_B 1, 1, 1
```

8.1.1.15 ClearStencil

Action
Specify clear value for the stencil buffer.

Syntax
ClearStencil s

Remarks
s	Value used when the stencil buffer is cleared. The initial value is **0**

See also
Clear_B 1635

8.1.1.16 ClearTag

Action
Specify clear value for the tag buffer.

Syntax
ClearTag t

Remarks
t	Value used when the tag buffer is cleared. The initial value is **0**.

See also

Tag 1701, TagMask 1702, Clear_B 1635

8.1.1.17 ClearScreen

Action
Clears the LCD with a Black Background.

Syntax
ClearScreen

Remarks
NONE

8.1.1.18 CMD8

Action
This statement will send a byte to the FT800 graphic processor.

Syntax
CMD8 prm

Remarks
CMD8 expects a numeric parameter. It will call the _cmd8 assembler code in FT800.LIB

See also
CMD16 1641 , CMD32 1641 , WR8 1705 , WR16 1706 , WR32 1706

Example
```
Sub Cmdinflatex(byval Ptr As Dword , Byref Varaddress As Word , Byval Count As Dword)

  Local Length As Dword

  Cmd32 Cmd_inflate
  Cmd32 Ptr

  For Length = 1 To Count
    Tb = Cpeek(varaddress)
    Cmd8 Tb
    Incr Varaddress
  Next

  Alignfifo Count

End Sub
```

8.1.1.19 CMD16

Action
This statement will send a word to the FT800 graphic processor.

Syntax
CMD16 prm

Remarks
CMD16 expects a numeric parameter. It will call the _cmd16 assembled code in FT800.LIB

See also
CMD8 1640 , CMD32 1641 , WR8 1705 , WR16 1706 , WR32 1706

Example
```
Sub Cmdprogress(bystack X As Integer , Bystack Y As Integer , Bystack W As Integer , Bystack H As Integer , Bystack
Options As Word , Bystack Value As Word , Bystack Range As Word)

  ' Draws a Progress Bar

  ' Options  Are
  ' OPT_3D = 0
  ' OPT_FLAT

  Cmd32 Cmd_progress
  cmdftstack 14
  Cmd16 &H0000

  ' was a total of 18 bytes, to align with 4byte boundary it had to be offset of 20
End Sub
```

8.1.1.20 CMD32

Action
This statement will send a dword to the FT800 graphic processor.

Syntax
CMD32 prm

Remarks
CMD32 expects a numeric parameter. It will call the _cmd32 assembled code in FT800.LIB

See also
CMD8 1640 , CMD16 1641 , WR8 1705 , WR16 1706 , WR32 1706

Example
```
Sub Cmdnumber(bystack X As Integer , Bystack Y As Integer , Bystack Fontx As Integer , Bystack Options As Word ,
Bystack Num As Long)
'-----------------------------------------------------------------------------------------------------------
  ' Draws a Decimal Number
  ' No Justification = 0
```

```
  'OPT_CENTERX
  'OPT_CENTERY
  'OPT_CENTER
  'OPT_RIGHTX
  'OPT_SIGNED

  Cmd32 Cmd_number
  cmdftstack 12
End Sub
```

8.1.1.21 CmdAppend

Action

Appends a block of memory to the current display list memory address where the offset is specified in REG_CMD_DL.

Syntax

CmdAppend Ptr, Num

Remarks

Ptr	Start of source commands in main memory
Num	Number of bytes to copy. This must be a multiple of 4

After appending is done, the co-processor engine will increase the REG_CMD_DL by num to make sure the display list is in order.

Example

```
' Pseudocode
CmdAppend 0, 40    ' Copy 10 commands from main memory address 0
Display_E          ' finish the display list
```

8.1.1.22 CmdBgColor

Action

Set the Background Color.

Syntax

CmdBgColor rgb

Remarks

rgb	New Background color, as a 24-bit RGB number. Red is the most significant 8 bits and Blue is the least. So &Hff0000 is bright Red. Background color is applicable for things that the user can move. example: behind gauges and sliders etc..

See also

CmdFgColor 1827

Example

```
' Pseudocode
xOffset = 40
```

```
yOffset = 80
' Draw horizontal Toggle bars
CmdBgColor &H800000
CmdFgColor &H410105
CmdToggle xOffset, yOffset, 30, 27, 0, 65535, "-ve" + gap + "+ve"
CmdFgColor &H0b0721
CmdBgColor &H000080

' The top scrollbar uses the default foreground color, the others with a changed color
CmdScrollBar 20, 30, 120, 8, 0, 10, 40, 100
CmdFgColor &H402000
CmdScrollBar 20, 60, 120, 8, 0, 30, 40,100
CmdFgColor &H202020
CmdScrollBar 20, 90, 120, 8, 0, 50, 40, 100
```

8.1.1.23 CmdButton

Action

Draw a button.

Syntax

CmdButton x, y, w, h ,font, options, text

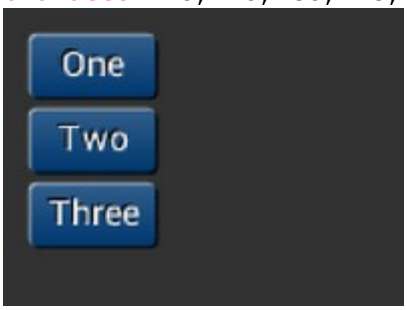
Remarks

x	x-coordinate of button top-left, in pixels
y	y-coordinate of button top-left, in pixels
w	width of button, in pixels
h	height of button, in pixels
font	Internal Fonts 16-31, User Defined Fonts 0-14
options	By default the button is drawn with a 3D effect, **OPT_FLAT** removes the 3D effect.
text	Text to display, valid printable ASCII code 32 to 127. For Custom/User Defined font, the ASCII code is from 1 to 127.

Example

```
' Pseudocode
CmdButton 10, 10, 50, 25, 26, 0, "One"
CmdButton 10, 40, 50, 25, 26, 0, "Two"
CmdButton 10, 70, 50, 25, 26, 0, "Three"

' A 140x00 pixel button with large text
CmdButton 10, 10, 140, 100, 31, 0,"Press!"
```


```
' Several smaller buttons
CmdButton 10, 10, 50, 25, 26, 0, "One"
CmdButton 10, 40, 50, 25, 26, 0, "Two"
CmdButton 10, 70, 50, 25, 26, 0, "Three"
```




```
' Changing button color
CmdFgColor &Hb9b900
CmdButton 10,  10,  50,  25,  26,  0, "Banana"
CmdFgColor &Hb97300
CmdButton 10,  40,  50,  25,  26,  0, "Orange"
CmdFgColor &Hb90007
CmdButton 10,  70,  50,  25,  26,  0, "Cherry"
```

8.1.1.24 CmdCalibrate

Action

Execute the touch screen calibration routine.

Syntax

CmdCalibrate

Remarks

The calibration procedure collects three touches from the touch screen, then

computes and loads an appropriate matrix into REG_TOUCH_TRANSFORM_A-F. To use it, create a display list and then use CmdCalibrate 1644. The co-processor engine overlays the touch
targets on the current Display List, gathers the calibration input and updates REG_TOUCH_TRANSFORM_A-F.

The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

8.1.1.25 CmdCalibratex

Action
Execute the touch screen calibration routine.
This is all in one routine with displaying prompts on the screen and updating of the REG_TOUCH_TRANSFORM_A-F registers.

Syntax
CmdCalibratex

Remarks
The calibration procedure collects three touches from the touch screen, then computes and loads an appropriate matrix into REG_TOUCH_TRANSFORM_A-F. To use it, create a display list and then use CmdCalibrate 1645. The co-processor engine overlays the touch
targets on the current Display List, gathers the calibration input and updates REG_TOUCH_TRANSFORM_A-F.

Note: You can Automatically let Bascom do the Screen calibration for you.
Or if you want to force an Screen calibration at anytime:

1. Press on the LCD panel
2. Reset your project
3. When you release from the LCD the Screen calibration message will appear.

To enable this mode, set LcdCal = 1 from the FT800.inc file.

```
Const LcdCal          = 1 ' Prompts for LCD Calibration (if not previously done)
```

See also
CmdCalibrate 1644

8.1.1.26 CmdClock

Action
Draw a Analog Clock.

Syntax

CmdClock x, y, r, options, h, m, s, ms

Remarks

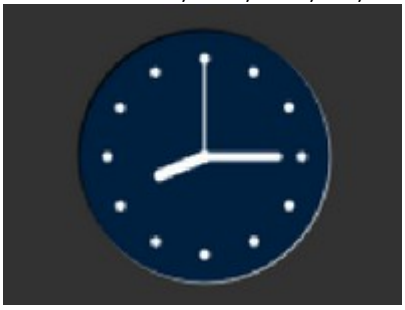
x	x-coordinate of clock center, in pixels
y	y-coordinate of clock center, in pixels
r	Radius of the gauge, in pixels
options	By default the clock dial is drawn with a 3D effect and the name of this option is **OPT_3D**. Option **OPT_FLAT** removes the 3D effect. With option **OPT_NOBACK**, the background is not drawn. With option **OPT_NOTICKS**, the twelve hour ticks are not drawn. With option **OPT_NOSECS**, the seconds hand is not drawn. With option **OPT_NOHANDS**, no hands are drawn. With option **OPT_NOHM**, no hour and minutes hands are drawn.
h	hours
m	minutes
s	seconds
ms	milliseconds

The details of physical dimension are:
- The 12 tick marks are placed on a circle of radius r*(200/256).
- Each tick is a point of radius r*(10/256)
- The seconds hand has length r*(200/256) and width r*(3/256)
- The minutes hand has length r*(150/256) and width r*(9/256)
- The hours hand has length r*(100/256) and width r*(12/256)

Refer to sections 5.7 Widgets physical dimensions and 5.7 Widget color settings in the FT800 Series Programmer Guide.PDF from FTDI
for more information.

Example

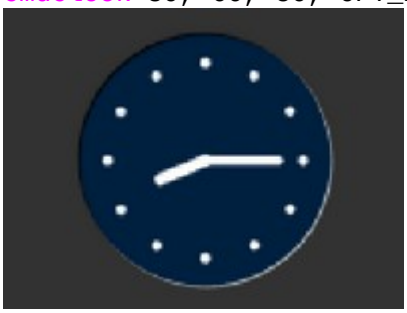
```
' A clock with radius 50 pixels, showing a time of 8.15
CmdClock 80, 60, 50, 0, 8, 15, 0, 0
```



```
' Setting the background color
CmdBgColor &H401010
CmdClock 80, 60, 50, 0, 8, 15, 0, 0
```

```
' Without the 3D look
CmdClock 80, 60, 50, OPT_FLAT, 8, 15, 0, 0

' The time fields can have large values. Here the hours are (7 x 3600s) and minutes
' are (38 x 60s), and seconds is 59. Creating a clock face showing the time as 7.38

CmdClock 80, 60, 50, 0, 0, 0, (7 * 3600) + (38 * 60) + 59, 0
```

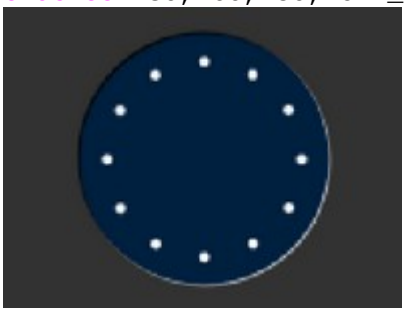
```
' No seconds hand
CmdClock 80, 60, 50, OPT_NOBACK, 8, 15, 0, 0
```



```
' No background
CmdClock 80,  60,  50,  OPT_NOBACK,  8, 15, 0, 0
```

```
' No Ticks
CmdClock 80, 60, 50, OPT_NOTICKS, 8, 15, 0, 0
```

```
' No Hands
CmdClock 80, 60, 50, OPT_NOHANDS, 8, 15, 0, 0
```



8.1.1.27 CmdColdStart

Action

This command sets co-processor engine to reset default states.

Syntax

CmdColdStart

Remarks

Example

```
' Pseudocode
CmdFgColor &H00C040
CmdGradColor &H000000
CmdButton 2, 32, 76, 56, 26, 0, "custom"
CmdColdStart
CmdButton 82, 32, 76, 56, 26, 0, "default"
```

8.1.1.28 CmdDial

Action
Draw a rotary dial control.

Syntax
CmdDial x, y, r ,options, val

Remarks

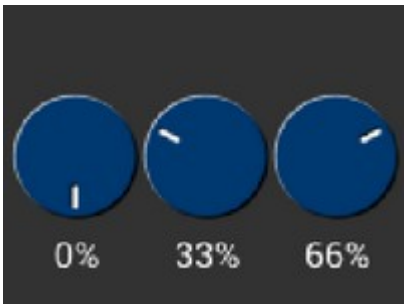
x	x-coordinate of dial center, in pixels
y	y-coordinate of dial center, in pixels
r	radius of dial, in pixels
options	By default the dial is drawn with a 3D effect. Options **OPT_FLAT** removes the 3D effect.
val	Specify the position of dial points by setting a value between **0** and **65535** inclusive. **0** means that the dial points straight down, &H4000 left, &H8000 up, and &Hc000 right.

Example
```
' Pseudocode

' A dial set to 50%
CmdDial 80, 60, 55, 0, &H8000
```

```
' Without the 3D look
CmdDial 80, 60, 55, OPT_FLAT, &H8000
```

```
' Dials set to 0%, 33% and 66%
CmdDial 28, 60 , 24, 0, 0
CmdText 28, 100, 26, OPT_CENTER, "0%"
CmdDial 80, 60, 24, 0, &H5555
CmdText 80, 100, 26, OPT_CENTER, "33%"
CmdDial 132, 60, 24, 0, &HAAAA
```

```
CmdText 132, 100, 26, OPT_CENTER, "66%"
```



8.1.1.29 CmdDlStart

Action

Start a New Display List.
When the co-processor engine executes this command, it waits until the display list is ready for writing, then sets Reg_Cmd_DL to zero.

Syntax

CmdDlStart

Remarks

In most of FTDI's FT800 C/C++ examples you will notice this command is used at the beginning of each loop or graphic routine.

Note: The Bascom FT800 Lib calls CmdDlStart from within **UpdateScreen** 1357 so it's not required in most circumstances.

8.1.1.30 CmdFgColor

Action

Set the Foreground Color.

Syntax

CmdFgColor rgb

Remarks

rgb	New Foreground color, as a 24-bit RGB number. Red is the most significant 8 bits and Blue is the least. So &Hff0000 is bright Red. Foreground color is applicable for things that the user can move such as handles and buttons.

See also

CmdBgColor 1827

Example

```
' Pseudocode
xOffset = 40
```

```
yOffset = 80
' Draw horizontal Toggle bars
CmdBgColor &H800000
CmdFgColor &H410105
CmdToggle xOffset, yOffset, 30, 27, 0, 65535, "-ve" + gap + "+ve"
CmdFgColor &H0b0721
CmdBgColor &H000080

' The top scrollbar uses the default foreground color, the others with a changed color
CmdScrollBar 20, 30, 120, 8, 0, 10, 40, 100
CmdFgColor &H703800
CmdScrollBar 20, 60, 120, 8, 0, 30, 40,100
CmdFgColor &H387000
CmdScrollBar 20, 90, 120, 8, 0, 50, 40, 100
```

8.1.1.31 CMDFTSTACK

Action

This FT800 command will send data from the soft stack to the FT800 processor.

Syntax

CMDFTSTACK bts [,opt]

Remarks

bts	The number of bytes to pop from the stack.
opt	An optional parameter to change stack clean up. When no parameter or 0 is specified, the soft stack will be cleaned up. But when a string is passed you can not clean up the stack since the pointers would point to the wrong address. In such a case specify a numeric value like 2 so the compiler will not clean up the stack. You must clean up the stack before the code returns. You can do this with the ADIW asm command. Please make sure you adjust with the same amount of bytes as you passed.

See also

FT800 1619 , CMD32 1641

Example

```
'-----------------------------------------------------------------------------------------------
Sub Cmdbutton(bystack X As Integer , Bystack Y As Integer , Bystack W As Integer , Bystack H As Integer , Bystack
Fontx As Integer , Bystack Options As Word , Byval S As String)
'-----------------------------------------------------------------------------------------------
  ' Draws Keyboard like buttons

  ' Options  Are
  ' OPT_3D = 0
  ' OPT_FLAT
```

```
  If Asc(S) = 0 Or Asc(S) > 127 then
    !adiw yl,12       ; manual clean up stack
    Exit Sub
  End if

  Cmd32 Cmd_button
  cmdftstack 12,2  'pop and transmit 12 bytes, option 2 means, no stack clean up
  Cmdstr S         'because we access this string we could not clean up
!  adiw yl,12       ; manual clean up stack
End Sub
```

8.1.1.32 CmdGauge

Action

Draw a Gauge.

Syntax

CmdGauge x, y, r, options, major, minor, val, range

Remarks

x	X-coordinate of gauge center, in pixels
y	Y-coordinate of gauge center, in pixels
r	Radius of the gauge, in pixels
options	By default the gauge dial is drawn with a 3D effect and the value of options is zero. OPT_FLAT removes the 3D effect. With option OPT_NOBACK, the background is not drawn. With option OPT_NOTICKS, the tick marks are not drawn. With option OPT_NOPOINTER, the pointer is not drawn.
major	Number of major subdivisions on the dial, 1-10 minor
minor	Number of minor subdivisions on the dial, 1-10
val	Gauge indicated value, between 0 and range, inclusive range
range	Maximum value

The details of physical dimension are:

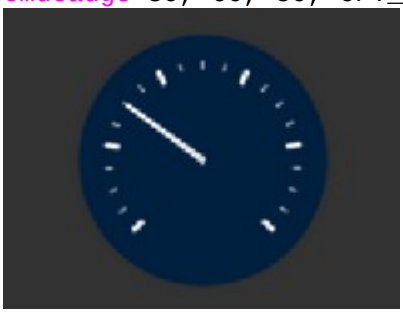
- The tick marks are placed on a 270 degree arc, clockwise starting at southwest position
- Minor ticks are lines of width r*(2/256), major r*(6/256)
- Ticks are drawn at a distance of r*(190/256) to r*(200/256)
- The pointer is drawn with lines of width r*(4/256), to a point r*(190/256) from the center
- The other ends of the lines are each positioned 90 degrees perpendicular to the pointer direction, at a distance r*(3/256) from the center

Refer to sections 5.7 Widgets physical dimensions and 5.7 Widget color settings in the FT800 Series Programmer Guide.PDF from FTDI
for more information.

Example

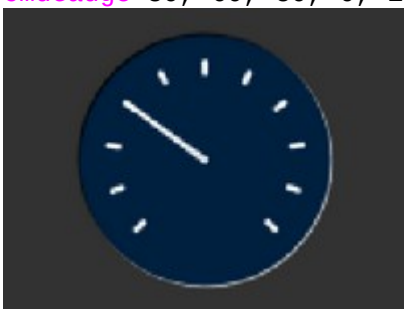
```
' Pseudocode

' A gauge with radius 50 pixels, five divisions of four ticks each, indicating 30%
CmdGauge 80, 60, 50, 0, 5, 4, 30, 100
```

```
' Without the 3D look
CmdGauge 80, 60, 50, OPT_FLAT, 5, 4, 30, 100
```

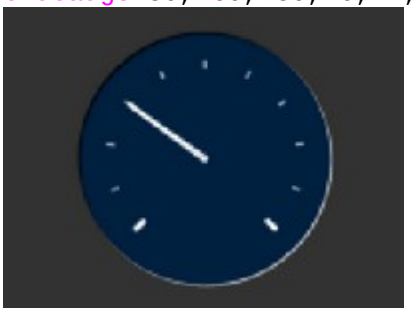


```
' Ten major divisions with two minor divisions each
CmdGauge 80, 60, 50, 0, 10, 2, 30, 100
```

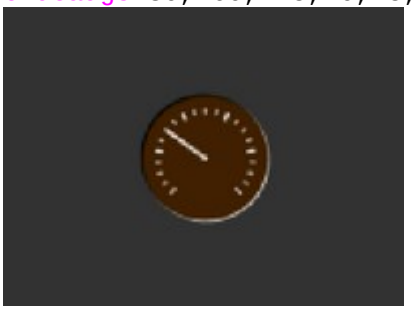
```
' Setting the minor divisions to 1 makes them disappear
CmdGauge 80, 60, 50, 0, 10, 1, 30, 100
```



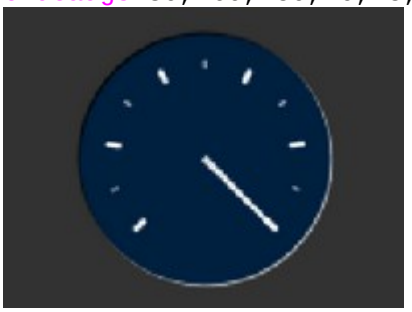
```
' Setting the major divisions to 1 gives minor divisions only
CmdGauge 80, 60, 50, 0, 1, 10, 30, 100
```



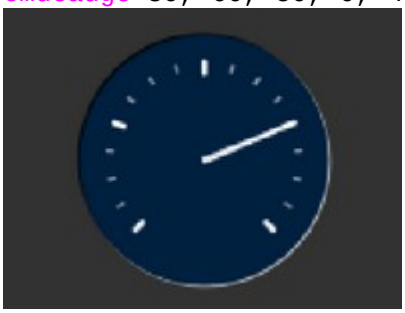
```
' A smaller gauge with a brown background
CmdBgColor &H402000
CmdGauge 80, 60, 25, 0, 5, 4, 30, 100
```



```
' Scale 0-1000, indicating 1000
CmdGauge 80, 60, 50, 0, 5, 2, 1000, 1000
```



```
' Scaled 0-65535, indicating 49152
CmdGauge 80, 60, 50, 0, 4, 4, 49152, 65535
```



```
' No background
CmdGauge 80, 60, 50, OPT_NOBACK, 4, 4, 49152, 65535
```

8.1.1.33 CmdGetMatrix

Action
Retrieves the current matrix coefficients.

Syntax
CmdGetMatrix a, b ,c, d, e, f

Remarks

a	Output parameter; written with matrix coefficient a. See BitmapTransform 1631 for formatting.
b	Output parameter; written with matrix coefficient b. See BitmapTransform 1631 for formatting.
c	Output parameter; written with matrix coefficient c. See BitmapTransform 1631 for formatting.
d	Output parameter; written with matrix coefficient d. See BitmapTransform 1631 for formatting.
e	Output parameter; written with matrix coefficient e. See BitmapTransform 1631 for formatting.
f	Output parameter; written with matrix coefficient f. See BitmapTransform 1631 for formatting.

To retrieve the current matrix within the context of co-processor engine. Please note the matrix within the context of co-processor engine will not apply to the bitmap transformation until it is passed to graphics engine through CmdGetMatrix 1655.

Example

8.1.1.34 CmdGetPtr

Action
Get the end memory address of inflated data.

Syntax
CmdGetPtr result

Remarks

result	The end address of decompressed data done by CmdInflate 1658. The starting address of decompressed data as was specified by CmdInflate 1658, while the end address of decompressed data can be retrieved by this command. It is one out parameter and can be passed in as any value with CmdGetPtr 1655 to RAM_CMD.

8.1.1.35 CmdGradColor

Action
Set the 3D button highlight color.

Syntax
CmdGradColor c

Remarks

c	New highlight gradient color, as a 24-bit RGB number. Red is the most significant 8 bits, blue is the least. So &Hff0000 is bright red.

Gradient is supported only for Button and Keys widgets.

Example
```
' Pseudocode

' Changing the gradient color: white (the default), red, green and blue
CmdFgColor &H101010
CmdButton 2, 2, 76, 56, 31, 0, "W"
CmdGradColor &Hff0000
CmdButton 82, 2, 76, 56, 31, 0, "R"
CmdGradColor &H00ff00
CmdButton 2, 62, 76, 56, 31, 0, "G"
CmdGradColor &H0000ff
CmdButton 82, 62, 76, 56, 31, 0,"B"
```


```
' The gradient color is also used for keys
CmdFgColor &H101010
CmdKeys 10, 10, 140, 30, 26, 0, "abcde"
CmdGradColor &Hff0000
CmdKeys 10, 50, 140, 30, 26, 0, "fghij"
```

8.1.1.36 CmdGradient

Action
Draw a smooth color gradient.

Syntax
CmdGradient x0, y0, rgb0, x1, y1, rgb1

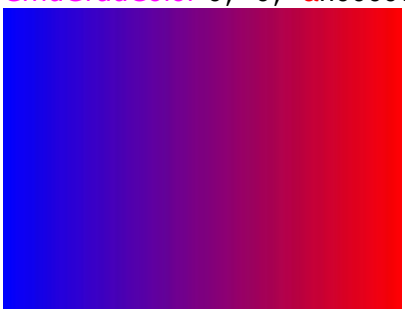
Remarks

x0	x-coordinate of point 0, in pixels
y0	y-coordinate of point 0, in pixels
rgb0	Color of point 0, as a 24-bit RGB number. r is the most significant 8 bits, b is the least. So &hff0000 is bright red.
x1	x-coordinate of point 1, in pixels
y1	y-coordinate of point 1, in pixels
rgb1	Color of point 1.

All the colour step values are calculated based on smooth curve interpolated from the rgb0 to rgb1 parameter.
The smooth curve equation is independently calculated for all three colors and the equation used is R0 + t * (R1 - R0), where t is interpolated between 0 and 1.
Gradient must be used with Scissor function to get the intended gradient display

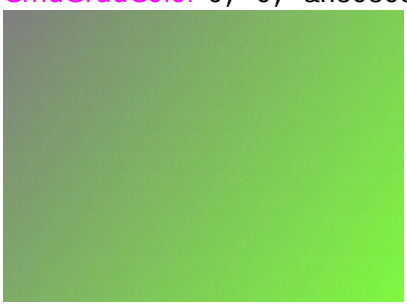
Example
```
' Pseudocode
ClearScreen
ColorRGB 255, 255, 255
ScissorSize wScissor, hScissor
' Horizontal gradient effect
ScissorXY xOffset, yOffset ' Clip the Display
CmdGradient xOffset, yOffset, &H808080, xOffset + wScissor, yOffset, &HFFFF00
```

Example
```
' Pseudocode

' A horizontal gradient from blue to red
CmdGradColor 0, 0, &H0000ff, 160, 0, &Hff0000
```

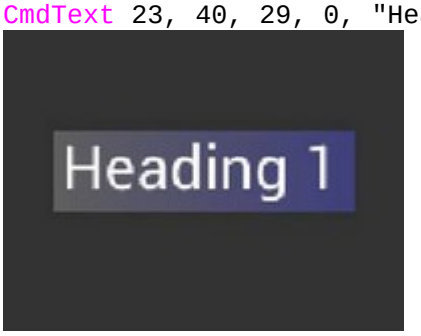


```
' A vertical gradient
CmdGradColor 0, 0, &H808080, 0, 120, &H80ff40
```

```
' The same colors in a diagonal gradient
CmdGradColor 0, 0, &H808080, 160, 120, &H80ff40
```



```
'Using a scissor rectangle to draw a gradient stripe as a background for
a title
ScissorXY 20, 40
ScissorSize 120, 32
CmdGradient 20,  0,  &H606060,  140,  0, &H404080
CmdText 23, 40, 29, 0, "Heading 1"
```



8.1.1.37 CmdInflate

Action

Decompress data into memory.

Syntax

CmdInflate ptr

Remarks

ptr	Destination address. The data byte should immediate follow in the command buffer

If the number of bytes is not a multiple of 4, then 1, 2 or 3 bytes should be appended to ensure 4-byte alignment of the next command. These padding bytes can have any value Command layout.

Example

```
' See demos - FT800 Gauges.bas (Sub IntroFTDI),  DigitTest.bas
```

8.1.1.38 CmdInterrupt

Action
Trigger an Interrupt Int_CmdFlag

Syntax
CmdInterrupt ms

Remarks

ms	Delay before interrupt triggers, in milliseconds. The interrupt is guaranteed not to fire before this delay. If ms is zero, the Interrupt fires immediately.

When the co-processor engine executes this command, it triggers Interrupt Int_CmdFlag

8.1.1.39 CmdKeys

Action
draw a row of keys.

Syntax
CmdKeys x, y, w ,h, font, options, char

Remarks

x	x-coordinate of keys top-left, in pixels
y	y-coordinate of keys top-left, in pixels
w	The width of the keys
h	The height of the keys
font	Bitmap handle to specify the font used in key label. The valid range is from **0** to **31**
options	By default the keys are drawn with a 3D effect and the value of option is zero. **OPT_FLAT** removes the 3D effect. If **OPT_CENTER** is given the keys are drawn at minimum size centered within the w x h rectangle. Otherwise the keys are expanded so that they completely fill the available space. If an ASCII code is specified, that key is drawn 'pressed' - i.e. in background color with any 3D effect removed.
char	Key labels, one character per key. The TAG value is set to the ASCII value of each key, so that key presses can be detected using the REG_TOUCH_TAG register.

The gap between keys is 3 pixels.
For OPT_CENTERX case, the keys are (font width + 1.5) pixels wide ,otherwise

keys are sized to fill available width.

Example

```
' Pseudocode

' A row of keys
CmdKeys 10, 10, 140, 30, 26, 0, "12345"
```

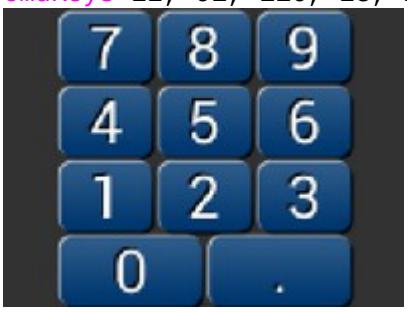

```
' Without the 3D look
CmdKeys 10,  10,  140,  30,  26, OPT_FLAT, "12345"
```


```
' Default vs. Centered
CmdKeys 10, 10, 140, 30, 26, 0, "12345"
CmdKeys 10, 60, 140, 30, 26, OPT_CENTER, "12345"
```


```
' Setting the options to show '2' key pressed ('2' is ASCII code &H32)
CmdKeys 10,  10,  140,  30,  26,  &H32, "12345"
```


```
' A calculator-style keyboard using font 29
CmdKeys 22,  1, 116, 28, 29, 0, "789"
CmdKeys 22, 31, 116, 28, 29, 0, "456"
CmdKeys 22, 61, 116, 28, 29, 0, "123"
```

```
CmdKeys 22, 91, 116, 28, 29, 0, "0."
```




```
' A compact keyboard drawn in font 20
CmdKeys 2, 2, 156, 21, 20, OPT_CENTER, "qwertyuiop"
CmdKeys 2, 26, 156, 21, 20,OPT_CENTER, "asdfghijkl"
CmdKeys 2, 50, 156, 21, 20,OPT_CENTER, "zxcvbnm"
CmdButton 2,74, 156, 21, 20, 0, ""
```


```
' Showing  the  f  (ASCII  &H66)  key pressed
CmdKeys 2, 2, 156, 21, 20, &H66 OR OPT_CENTER, "qwertyuiop"
CmdKeys 2, 26, 156, 21, 20, &H66 OR OPT_CENTER, "asdfghijkl"
CmdKeys 2, 50, 156, 21, 20, &H66 OR OPT_CENTER, "zxcvbnm"
CmdButton 2, 74, 156, 21, 20, 0, ""
```

8.1.1.40 CmdLoadIdentity

Action

Set the current matrix to the identity matrix.

Syntax

CmdLoadIdentity

Remarks

This command instructs the co-processor engine of the FT800 to set the current

matrix to the identity matrix, so that co-processor engine is able to form the new matrix as requested by CmdScale 1671, CmdRotate 1669, CmdTranslate 1686 command.

For more information on the identity matrix, please see section 2.5.5 Bitmap transformation matrix from the FT800 Series Programmer Guide.PDF (from FTDI).

8.1.1.41 CmdLoadImage

Action
Load a JPEG image.

Syntax
CmdLoadImage ptr, options

Remarks

ptr	Destination address
options	By default, option **OPT_RGB565** means the loaded bitmap is in RGB565 format. Option **OPT_MONO** means the loaded bitmap to be monochrome, in L8 format. The command appends Display List commands to set the source, layout and size of the resulting image. Option **OPT_NODL** prevents this - nothing is written to the display list. OPT_NODL can be OR'ed with OPT_MONO or OPT_RGB565.

The data byte should immediately follow in the command buffer. If the number of bytes is not a multiple of 4, then 1, 2 or 3 bytes
should be appended to ensure 4-byte alignment of the next command. These padding bytes can have any value.
The application on the host processor has to parse the JPEG header to get the properties of the JPEG image and decide to decode. Behavior is unpredictable in cases of non baseline jpeg images or the output data generated is more than the RAM_G size.

Example
```
' See demos - FT800 Demo2.bas, FT800 LoadImage.bas
```

8.1.1.42 CmdLogo

Action
Play device logo animation.

Syntax
CmdLogo

Remarks
The logo command causes the co-processor engine to play back a short animation of the FTDI logo.
During logo playback the MCU should not access any FT800 resources. After 2.5 seconds have elapsed, the co-processor
engine writes zero to REG_CMD_READ and REG_CMD_WRITE, and starts waiting for commands. After this command is complete,
the MCU shall write the next command to the starting address of RAM_CMD.

Example
```
' see it working - FT800 Gauges.bas, FT800 Keyboard.bas, FT800 Signals.bas and FT80
```

8.1.1.43 CmdMemCpy

Action
Copy a block of memory.

Syntax
CmdMemCpy dst, src, num

Remarks

dst	address of the destination memory block
src	address of the source memory block
num	number of bytes to copy

The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Example
```
' Pseudocode

' To copy 1K byte of memory from 0 to &H8000
CmdMemCpy &H8000, 0, 1024
```

8.1.1.44 CmdMemCrc

Action
Compute a CRC-32 for memory.

Syntax
CmdMemCrc ptr, num, result

Remarks

ptr	Starting address of the memory block
num	Number of bytes in the source memory block
result	Output parameter; written with the CRC-32 after command execution. The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Example

```
' Pseudocode

' To compute the CRC-32 of the first 1K byte of FT800 memory, first record the valu
' of REG_CMD_WRITE, execute the command, wait for completion, then read  the 32-bit

x = Rd16(REG_CMD_WRITE)
CmdMemCrc 0, 1024, 0
Print Rd32(RAM_CMD + x + 12)
```

8.1.1.45 CmdMemSet

Action

Fill memory with a byte value.

Syntax

CmdMemSet ptr, value, num

Remarks

ptr	Starting address of the memory block
value	Value to be written to memory
num	Number of bytes in the memory block

The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Example

```
' Pseudocode

' To write 0xff the first 1K of main memory
CmdMemSet 0, 255, 1024
```

8.1.1.46 CmdMemWrite

Action

Write bytes into memory.

Syntax

CmdMemWrite ptr, num,

Remarks

ptr	The memory address to be written
result	Number of bytes to be written

The data byte should immediately follow in the command buffer. If the number of bytes is not a multiple of 4, then
1, 2 or 3 bytes should be appended to ensure 4-byte alignment of the next command, these padding bytes can have any value.
The completion of this function can be detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.
Caution: if using this command, it may corrupt the memory of the FT800 if used improperly.

Example

```
' Pseudocode
' To change the backlight brightness to 64 (half intensity) for a particular screen
...
CmdSwap ' finish the display list
CmdDlStart ' wait until after the swap
CmdMemWrite REG_PWM_DUTY, 4 ' write to the PWM_DUTY register
```

8.1.1.47 CmdMemZero

Action

Write zero to a block of memory.

Syntax

CmdMemZero ptr, num

Remarks

ptr	Starting address of the memory block
num	Number of bytes in the memory block

The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Example

```
' Pseudocode

' To erase the first 1K of main memory
CmdMemZero 0, 1024
```

8.1.1.48 CmdNumber

Action

Draw a decimal number.

Syntax

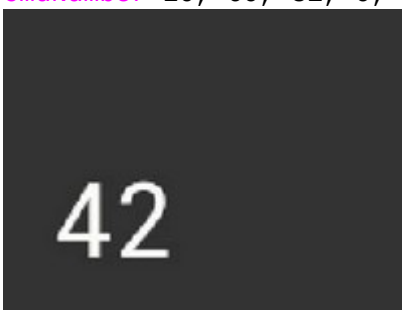
CmdNumber x, y, font, options, n

Remarks

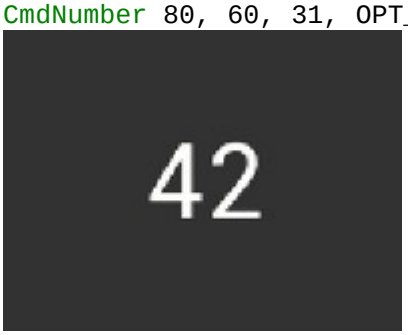
x	x-coordinate of text base, in pixels
y	y-coordinate of text base, in pixels
font	font to use for text, **0**-**31**. See ROM and RAM Fonts
options	By default (x,y) is the top-left pixel of the text. **OPT_CENTERX** centers the text horizontally **OPT_CENTERY** centers it vertically. **OPT_CENTER** centers the text in both directions. **OPT_RIGHTX** right-justifies the text, so that the x is the rightmost pixel. By default the number is displayed with no leading zeroes, but if a width 1-9 is specified in the options, then the number is padded if necessary with leading zeroes so that it has the given width. If **OPT_SIGNED** is given, the number is treated as signed, and prefixed by a minus sign if negative.
n	The number to display, either unsigned or signed 32-bit. NOTE : while -2147483648 is valid for a long, the FT800 will show -18446744071562067968 (128 bit signed number) which seems a bug in the FT800.

Example

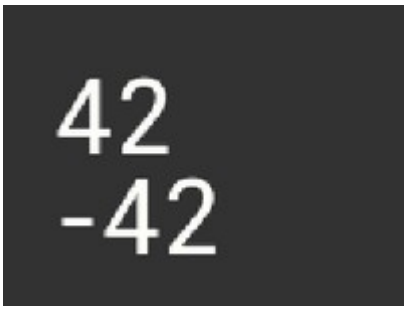
```
' Pseudocode

' A number
CmdNumber 20, 60, 31, 0, 42
```



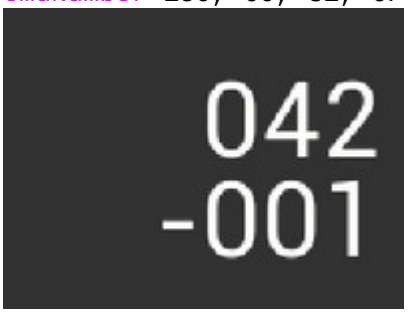

```
' Centered
CmdNumber 80, 60, 31, OPT_CENTER, 42
```




```
' Signed output of positive and negative numbers
CmdNumber 20, 20, 31, OPT_SIGNED, 42
CmdNumber 20, 60, 31, OPT_SIGNED, -42
```




```
' Forcing width to 3 digits, right-justified
CmdNumber 150, 20, 31, OPT_RIGHTX OR 3, 42
CmdNumber 150, 60, 31, OPT_SIGNED OR OPT_RIGHTX OR 3, -1
```



8.1.1.49 CmdProgress

Action

Draw a progress bar.

Syntax

CmdProgress x, y, w, h, options, val, range

Remarks

x	x-coordinate of progress bar top-left, in pixels
y	y-coordinate of progress bar top-left, in pixels
w	width of progress bar, in pixels
h	height of progress bar, in pixels
options	By default the progress bar is drawn with a 3D effect and the value of options is zero. Options **OPT_FLAT** removes the 3D effect and its value is 256.
val	Displayed value of progress bar, between **0** and **range** inclusive
range	Maximum value

The details of physical dimensions are:

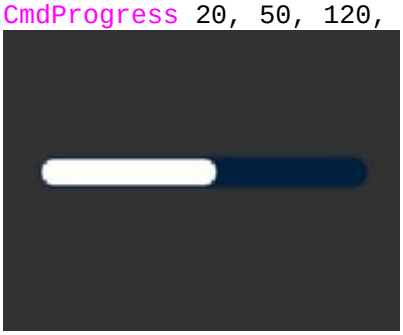
- x,y,w,h give outer dimensions of progress bar. Radius of bar (r) is min (w,h)/2
- Radius of inner progress line is r * (7/8)

Example

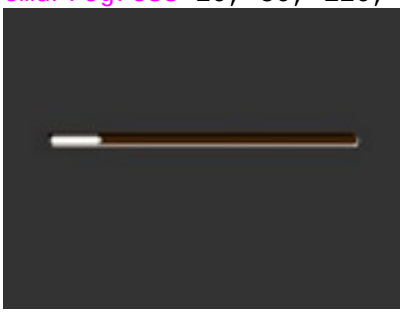
```
' Pseudocode

' A progress bar showing 50% completion
```

```
CmdProgress 20, 50, 120, 12, 0, 50,100
```

```
' Without the 3D look
CmdProgress 20, 50, 120, 12, OPT_FLAT, 50, 100
```



```
' A 4 pixel high bar, range 0-65535, with a brown background
CmdBgColor &H402000
CmdProgress 20, 50, 120, 4, 0, 9000, 65535
```



8.1.1.50 CmdRegRead

Action

Read a register value.

Syntax

CmdRegRead ptr, result

Remarks

ptr	Address of register to read
result	The register value to be read at ptr address

Example

```
' Pseudocode
' To capture the exact time when a command completes:
x = Rd16(REG_CMD_WRITE
CmdRegread REG_CLOCK, 0
```

```
...
```

8.1.1.51 CmdRotate

Action
Apply a rotation to the current matrix.

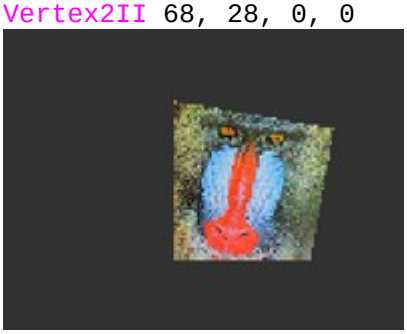
Syntax
CmdRotate anle

Remarks

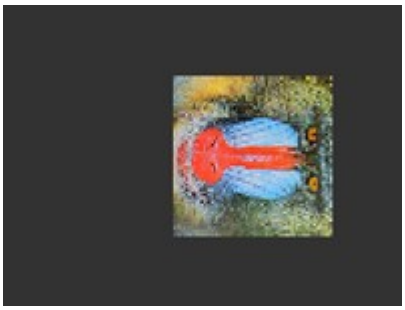
angle	Clockwise rotation angle, in units of 1/65536 of a circle

Remarks
CMDROTATEA 1670

Example
```
' Pseudocode

' To rotate the bitmap clockwise by 10 degrees with respect to the top left of the
Begin_G BITMAPS
CmdLoadIdentity
CmdRotate 10 * 65536 / 360
CmdSetMatrix
Vertex2II 68, 28, 0, 0
```



```
' To rotate the bitmap counter clockwise by 33 degrees top left of the bitmap
Begin_G BITMAPS
CmdLoadIdentity
CmdRotate -33 * 65536 / 360
CmdSetMatrix
Vertex2II 68, 28, 0, 0

' Rotating a 64 x 64 bitmap around its center
Begin_G BITMAPS
CmdLoadIdentity
CmdTranslate 65536 * 32, 65536 * 32
CmdRotate 90 * 65536 / 360
CmdTranslate 65536  *  -32,  65536  *  -32
CmdSetMatrix
Vertex2II 68, 28, 0, 0
```



8.1.1.52 CmdRotateA

Action
Apply a rotation in degrees to the current matrix

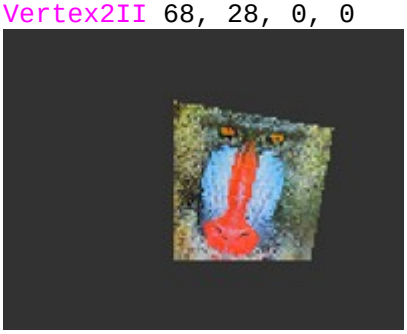
Syntax
CmdRotateA angle

Remarks

angle	Clockwise rotation angle, in degrees (0-360)

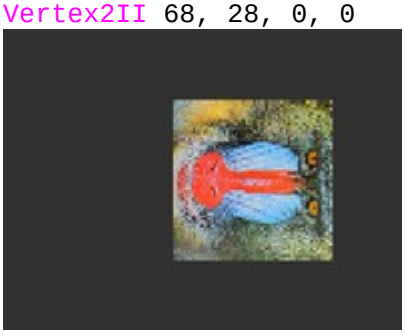
See Also
CMDROTATE 1669

Example
```
' Pseudocode

' To rotate the bitmap clockwise by 10 degrees with respect to the top left of the
Begin_G BITMAPS
CmdLoadIdentity
CmdRotate 10
CmdSetMatrix
Vertex2II 68, 28, 0, 0
```



```
' To rotate the bitmap counter clockwise by 33 degrees top left of the bitmap
Begin_G BITMAPS
CmdLoadIdentity
CmdRotate -33
CmdSetMatrix
Vertex2II 68, 28, 0, 0

' Rotating a 64 x 64 bitmap around its center
Begin_G BITMAPS
CmdLoadIdentity
```

```
CmdTranslate 65536 * 32, 65536 * 32
CmdRotate 90
CmdTranslate 65536  *  -32,  65536  *  -32
CmdSetMatrix
Vertex2II 68, 28, 0, 0
```



8.1.1.53 CmdScale

Action

Apply a scale to the current matrix.

Syntax

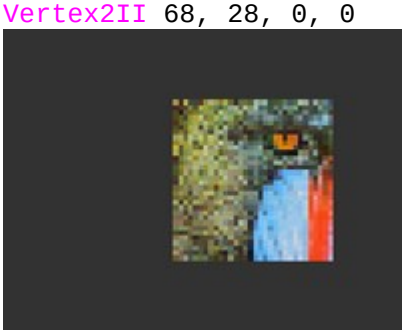
CmdScale sx, sy

Remarks

sx	x scale factor, in signed 16. 16 bit fixed-point form
sy	y scale factor, in signed 16. 16 bit fixed-point form

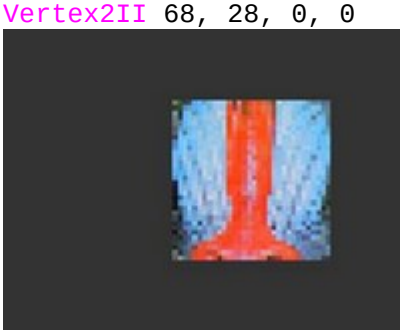
Example

```
' Pseudocode

' To zoom a bitmap 2X
Begin_G BITMAPS
CmdLoadIdentity
CmdScale 2 * 65536, 2 * 65536
CmdSetMatrix
Vertex2II 68, 28, 0, 0
```



```
' To zoom a bitmap 2X around its center
Begin_G BITMAPS
CmdLoadIdentity
CmdTranslate 65536 * 32, 65536 * 32
CmdScale 2 * 65536, 2 * 65536
CmdTranslate 65536  *  -32,  65536  *  -32
CmdSetMatrix
```

```
Vertex2II 68, 28, 0, 0
```



8.1.1.54 CmdScreenSaver

Action
Start an animated screensaver.

Syntax
CmdScreenSaver

Remarks
After the screensaver command, the co-processor engine continuously updates REG_MACRO_0 with VERTEX2F with varying (x,y) coordinates. With an appropriate display list, this causes a bitmap to move around the screen without any MCU work.
Command CMD_STOP stops the update process.
Note that only one of CmdSketch 1674, CmdScreenSaver 1672 or CmdSpinner 1677 can be active at one time.

REG_MACRO_0 is updated with respect to frequency of frames displayed (depending on the display registers configuration).
Typically for 480x272 display the frame rate is around 60 frame per second.

Example
```
' see it working in FT800 Demo4.bas (Sub Screensaver)
```

8.1.1.55 CmdScrollBar

Action
Draw a scroll bar.

Syntax
CmdScrollBar x, y, w, h, options, val, range, size, range

Remarks

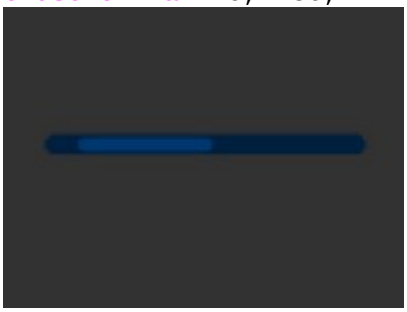
x	x-coordinate of scroll bar top-left, in pixels
y	y-coordinate of scroll bar top-left, in pixels
w	Width of scroll bar, in pixels. If width is greater than height, the scroll bar is

	drawn horizontally
h	Height of scroll bar, in pixels. If height is greater than width, the scroll bar is drawn vertically
options	By default the scroll bar is drawn with a 3D effect and the value of options is zero. Options **OPT_FLAT** removes the 3D effect and its value is 256
val	Displayed value of scroll bar, between **0** and range inclusive **range**
range	Maximum value

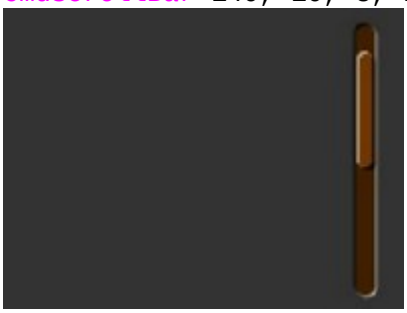
Example

```
' Pseudocode

' A scroll bar indicating 10-50%
CmdScrollBar 20, 50, 120, 8, 0, 10, 40, 100
```

```
' Without the 3D look
CmdScrollBar 20,  50,  120,  8,  OPT_FLAT, 10, 40, 100
```



```
' A brown-themed vertical scroll bar
CmdBgColor &H402000
CmdFgColor &H703800
CmdScrollBar 140, 10, 8, 100, 0, 10, 40, 100
```



8.1.1.56 CmdSetFont

Action
Set up a custom font.

Syntax
CmdSetFont font, ptr

Remarks

font	The bitmap handle from **0** to **14**. Bitmap handle **15** can be used conditionally
ptr	The metric block address in RAM. 4 bytes aligned is required.

CmdSetFont is used to register one custom defined bitmap font into the FT800 coprocessor engine. After registration, the FT800
co-processor engine is able to use the bitmap font with its co-processor command.

Details on how to set up custom font, please refer to ROM and RAM Fonts from FTDI's FT800 Series Programmer Guide.PDF

Example

```
' See demos - DigitTest.bas and FT800 Demo3.bas
```

8.1.1.57 CmdSetMatrix

Action
Write the current matrix to the Display List.

Syntax
CmdSetMatrix

Remarks
The co-processor engine assigns the value of the current matrix to the bitmap transform matrix of the graphics engine by generating Display List commands, i.e. BitmapTransformA-F. After this command, the following bitmap rendering operation will be affected by the new transform matrix.

8.1.1.58 CmdSketch

Action
Start a continuous sketch update.

Syntax FT800
CmdSketch x, y, w, h, ptr, format

Syntax FT801
CmdSketch x, y, w, h, ptr, format , freq

Remarks

x	x-coordinate of sketch area top-left, in pixels
y	y-coordinate of sketch area top-left, in pixels
w	Width of sketch area, in pixels
h	Height of sketch area, in pixels
ptr	Base address of sketch bitmap
format	Format of sketch bitmap, either **L1** or **L8**
freq	The oversampling frequency. The typical value is 1500 to make sure the lines are connected smoothly. The value zero means no oversampling operation.

FT800

Please note that update frequency of bitmap data in graphics memory depends on sampling frequency of ADC built-in circuit of FT800, which is up to 1000 Hz.

FT801

CmdSketch - Capacitive touch specific sketch This command has the same functionality as CmdSketch except it has done the optimization for a Capacitive Touch Panel.
Because Capacitive Touch Panels have lower sampling frequencies (around 100 Hz) to report the coordinates, the sketch functionality updates less frequently compared to resistive touch.
CmdSketch introduces a linear interpolation algorithm to provide a smoother effect when drawing the output line.

After the sketch command, the co-processor engine continuously samples the touch inputs and paints pixels into a bitmap, according to the touch (x, y). This means that the user touch inputs are drawn into the bitmap without any need for MCU work. Command CmdStop 1680 stops the sketch process.

Note that only one of CmdSketch 1674, CmdScreenSaver 1672 or CmdSpinner 1677 can be active at one time.

Example

```
' see demo - FT800 Sketch.bas also FT800 Demo4.bas (SUB Sketch)
```

8.1.1.59 CmdSlider

Action

Draw a slider.

Syntax

CmdSlider x, y, w, h, options, val, range, size, range

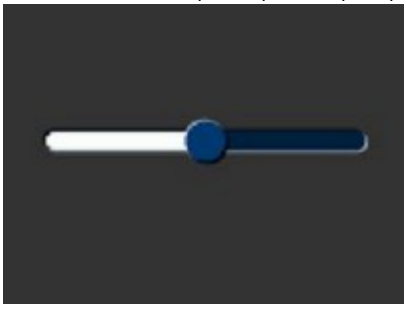
Remarks

x	x-coordinate of scroll bar top-left, in pixels
y	y-coordinate of scroll bar top-left, in pixels
w	Width of slider, in pixels. If width is greater than height, the scroll bar is drawn horizontally

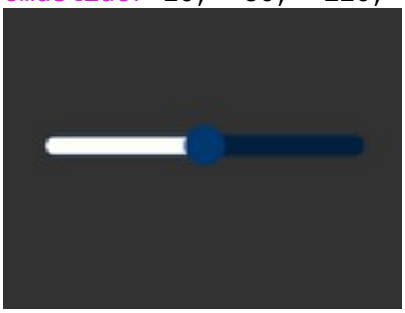
h	Height of slider, in pixels. If height is greater than width, the scroll bar is drawn vertically
options	By default the slider is drawn with a 3D effect. **OPT_FLAT** removes the 3D effect
val	Displayed value of slider, between **0** and **range** inclusive
range	Maximum value

Example

```
' Pseudocode

' A slider set to 50%
CmdSlider 20, 50, 120, 8, 0, 50, 100
```



```
' Without the 3D look
CmdSlider 20,  50,  120,  8,  OPT_FLAT, 50, 100
```



```
' A  brown-themed  vertical  slider  with  range  0-65535
CmdBgColor &H402000
CmdFgColor &H703800
CmdSlider 76, 10, 8, 100, 0, 20000, 65535
```

8.1.1.60 CmdSnapShot

Action
Take a snapshot of the current screen.

Syntax
CmdSnapShot ptr

Remarks

ptr	Snapshot destination address, in RAM_G

This command causes the co-processor engine to take a snapshot of the current screen, and write the result into RAM_G as a ARGB4 bitmap.The size of the bitmap is the size of the screen, given by the REG_HSIZE and REG_VSIZE registers.

During the snapshot process, the display should be disabled by setting REG_PCLK to 0 to avoid display glitch.
Because co-processor engine needs to write the result into the destination address, the destination address must be never used or referenced by graphics engine.

Note: If you want to actual take Screen Captures - see FT800 Capture.Bas

Example

```
' See demo - FT800 Demo4.bas (Sub Snapshot)
```

8.1.1.61 CmdSpinner

Action
Start an animated spinner.

Syntax
CmdSpinner x, y, style, range

Remarks

x	The X coordinate of top left of spinner
y	The Y coordinate of top left of spinner
style	The style of spinner. Valid range is from **0** to **3**
range	The scaling coefficient of spinner. 0 means no scaling

The spinner is an animated overlay that shows the user that some task is continuing. To trigger the spinner, create a display list and then use CMD_SPINNER. The co-processor engine overlays the spinner on the current display list, swaps the display list to make it
visible, then continuously animates until it receives CMD_STOP. REG_MACRO_0 and REG_MACRO_1 registers are utilized to perform the animation kind of effect. The frequency of points movement is with respect to the display frame rate configured.

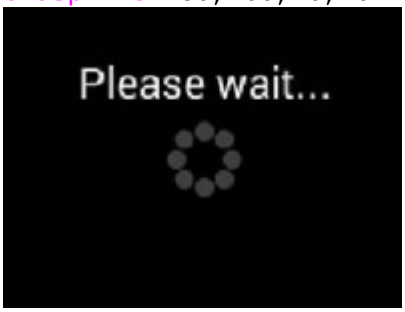
Typically for 480x272 display panels the display rate is ~60fps.

For style **0** and 60fps the point repeats the sequence within 2 seconds.
For style **1** and 60fps the point repeats the sequence within 1.25 seconds.
For style **2** and 60fps the clock hand repeats the sequence within 2 seconds.
For style **3** and 60fps the moving dots repeat the sequence within 1 second.

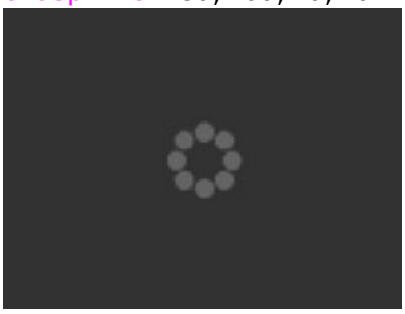
Note that only one of CmdSketch 1674, CmdScreenSaver 1672 or CmdSpinner 1677 can be active at one time.

Example

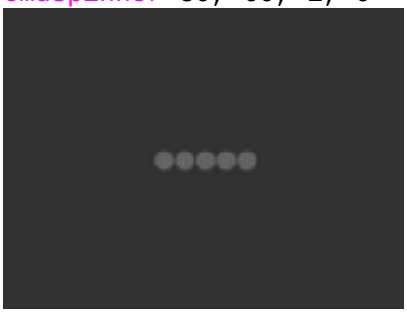
```
' Pseudocode

' Create a display list, then start the spinner
Clear_B 1,1,1
CmdText 80, 30, 27, OPT_CENTER, "Please wait..."
CmdSpinner 80, 60, 0, 0
```



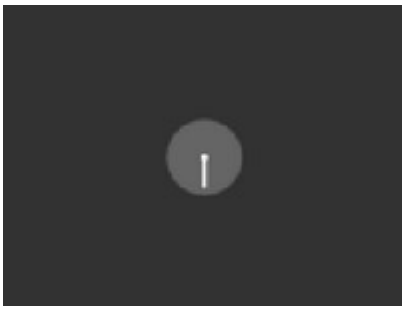

```
' Spinner style 0, a circle of dots
CmdSpinner 80, 60, 0, 0
```



```
' Style 1, a line of dots
CmdSpinner 80, 60, 1, 0
```

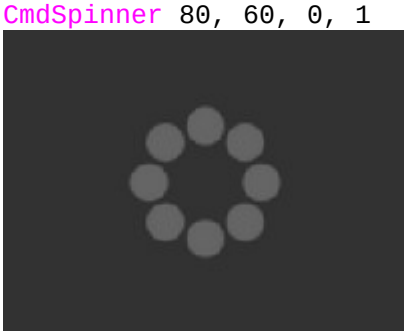


```
' Style 2, a rotating clock hand
CmdSpinner 80, 60, 2, 0
```

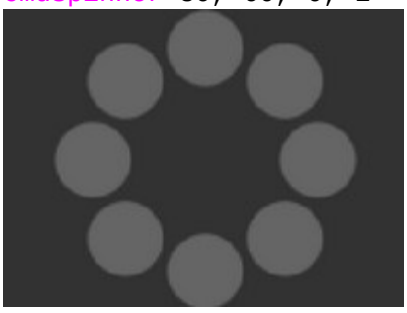


```
' Style 3, two orbiting dots
CmdSpinner 80, 60, 3, 0
```

```
' Half screen, scale 1
CmdSpinner 80, 60, 0, 1
```



```
' Full screen, scale 2
CmdSpinner 80, 60, 0, 2
```



8.1.1.62 CmdStop

Action
Stop any of spinner, screensaver or sketch.

Syntax
CmdStop

Remarks
For CmdSpinner 1677 and CmdScreenSaver 1672, REG_MACRO_0 and REG_MACRO_1 will be stopped updating.
For CmdSketch 1674 the bitmap data in RAM_G will be stopped updating.

Example
```
' See FT800 Demo1.bas - Sub Widget_Spinner
' FT800 Demo4.bas - SUB Sketch, Sub Screensaver
```

8.1.1.63 CmdSwap

Action
Swap the current Display List

Syntax
CmdSwap

Remarks
When the co-processor engine executes this command, it requests a display list swap
immediately after current display list is scanned out. Internally, the co-processor engine
implements this command by writing to Reg_DlSwap

Note: The Bascom FT800 Lib calls CmdSwap from within **UpdateScreen** 1357 so it's not required in most circumstances.

8.1.1.64 CmdText

Action
Draw Text.

Syntax
CmdText x, y, font, options, string

Remarks

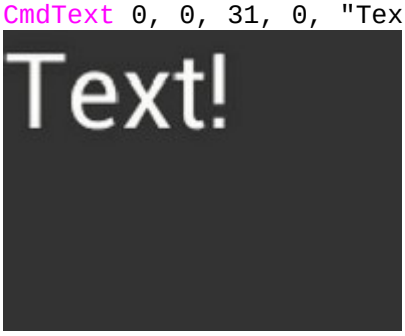
x	x-coordinate of text base, in pixels
y	y-coordinate of text base, in pixels

font	Internal Fonts **16**-**31**, User Defined Fonts **0**-**14**
options	By default (x,y) is the top-left pixel of the text (options = **0**). **OPT_CENTERX** centers the text horizontally **OPT_CENTERY** centers it vertically **OPT_CENTER** centers the text in both directions **OPT_RIGHTX** right-justifies the text, so that the x is the rightmost pixel.
string	text to display

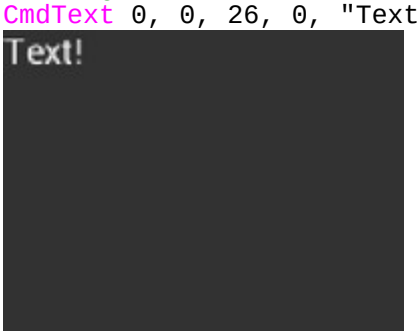
Example

```
ClearScreen
ColorRGB &H80, &H80, &H00
CmdText FT_DispWidth/ 2, FT_DispHeight/ 2, 31, OPT_CENTER, "Bascom is here"

UpdateScreen
```

```
' Plain text at (0,0) in the largest font
CmdText 0, 0, 31, 0, "Text!"
```

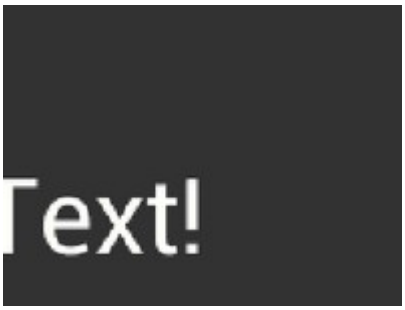



```
' Using a smaller font
CmdText 0, 0, 26, 0, "Text!"
```

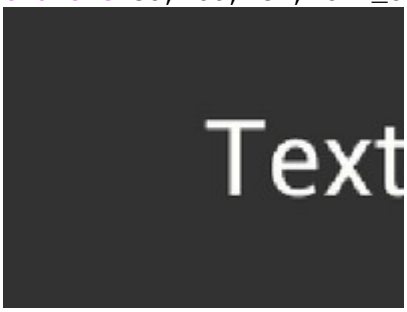



```
' Centered horizontally
CmdText 80, 60, 31, OPT_CENTERX, "Text!"
```


```
' Right-justified
CmdText 80, 60, 31, OPT_RIGHTX, "Text!"
```




```
' Centered vertically
CmdText 80, 60, 31, OPT_CENTERY, "Text!"
```




```
' Centered both horizontally and vertically
CmdText 80, 60, 31, OPT_CENTER, "Text!"
```

8.1.1.65 CmdToggle

Action

Draw a toggle switch.

Syntax

CmdToggle x, y, w, font, options, state, char

Remarks

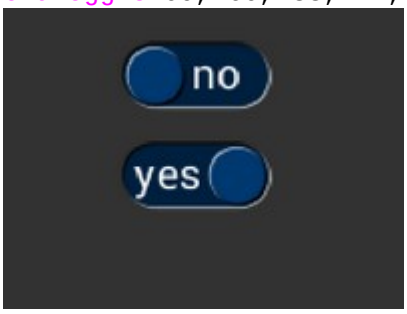
x	x-coordinate of top-left of toggle, in pixels

y	y-coordinate of top-left of toggle, in pixels
w	width of toggle, in pixels
font	font to use for text, **0**-**31**
options	By default the toggle is drawn with a 3D effect and the value of options is zero. Options **OPT_FLAT** removes the 3D effect.
state	state of the toggle: **0** is off, **65535** is on
char	String label for toggle. To seperate the labels use 'gap' ie: "off" + gap + "on"

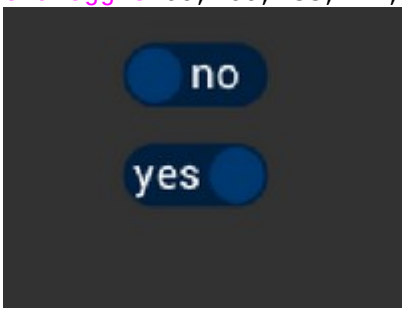
The details of physical dimension are
- Outer bar radius I is font height*(20/16)
- Knob radius is r-1.5

Example

```
' Pseudocode

' Using a medium font, in the two states
CmdToggle 60, 20, 33, 27, 0, 0,"no" + gap + "yes"
CmdToggle 60, 60, 33, 27, 0, 65535, "no" + gap + "yes"
```




```
' Without the 3D look
CmdToggle 60, 20, 33, 27, OPT_FLAT, 0, "no" + gap + "yes"
CmdToggle 60, 60, 33, 27, OPT_FLAT, 65535, "no" + gap + "yes"
```




```
' With different background and foreground colors
CmdBgColor &H402000
CmdFgColor &H703800
CmdToggle 60, 20, 33, 27, 0, 0, "no" + gap + "yes"
CmdToggle 60, 60, 33, 27, 0, 65535, "no" + gap + "yes"
```

8.1.1.66 CmdTrack

Action

Track touches for a graphics object.

Syntax

CmdTrack x, y, w, h, tag

Remarks

x	For linear tracker functionality, x-coordinate of track area top-left, in pixels. For rotary tracker functionality, x-coordinate of track area center, in pixels.
y	For linear tracker functionality, y-coordinate of track area top-left, in pixels. For rotary tracker functionality, y-coordinate of track area center, in pixels.
w	Width of track area, in pixels.
h	Height of track area, in pixels. A w and h of (1,1) means that the tracker is rotary, and reports an, angle value in REG_TRACKER. A w and h of (0,0) disables the track functionality of co-processor engine.
tag	tag of the graphics object to be tracked, **1**-**255**

This command will enable co-processor engine to track the touch on the particular graphics object with one valid tag value assigned. Then, co-processor engine will update the REG_TRACKER periodically with the frame rate of LCD display panel. Co-processor engine tracks the graphics object in rotary tracker mode and linear tracker mode:

- Rotary tracker mode – Track the angle between the touching point and the center of graphics object specified by tag value.

The value is in units of 1/65536 of a circle. 0 means that the angle is straight down, &H4000 left, &H8000 up, and &HC000 right
from the center.

- Linear tracker mode – If parameter w is greater than h, track the relative distance of touching point to the width of graphics

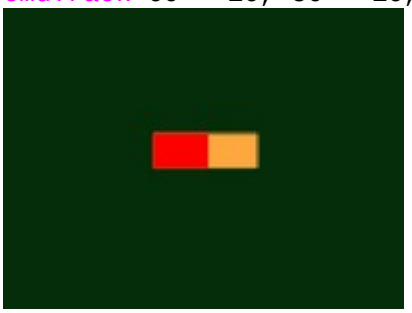
object specified by tag value. If parameter w is not greater than h, Track the relative distance of touching point to the height of graphics object specified by tag value. The value is in units of 1/65536 of the width or height of graphics object.
The distance of touching point refers to the distance from the top left pixel of graphics object to the coordinate of touching point.

Example

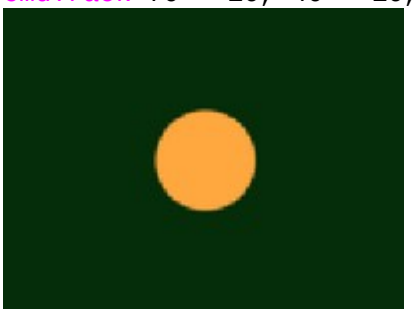
Note: see demo files for more examples

```
' Pseudocode

' Horizontal track of rectangle dimension 40x12 pixels and the present touch is at
ClearColorRGB 5, 45, 110
ColorRGB 255, 168, 64
```

```
Clear_B 1 ,1 ,1
Begin_G RECTS
Vertex2F 60 * 16, 50 * 16
Vertex2F 100 * 16, 62 * 16
ColorRGB 255, 0, 0
Vertex2F 60 * 16,50 * 16
Vertex2F 80 * 16,62 * 16
ColorMask 0 ,0 ,0 ,0
Tag 1
Vertex2F 60 * 16,50 * 16
Vertex2F 100 * 16,62 * 16
CmdTrack 60 * 16, 50 * 16, 40, 12, 1
```

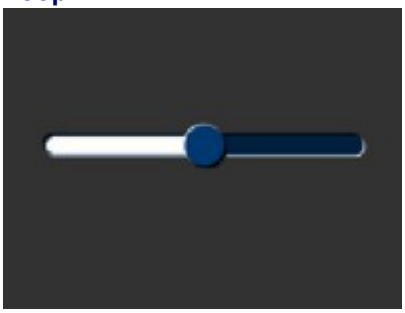


```
' Circular track centered at (80,60) display location
ClearColorRGB 5, 45, 110
ColorRGB 255, 168, 64
Clear_B 1 ,1 ,1
Begin_G RECTS
Vertex2F 70 * 16,40 * 16
Vertex2F 82 * 16,80 * 16
ColorRGB 255, 0, 0
Vertex2F 70 * 16,40 * 16
Vertex2F 82 * 16,60 * 16
ColorMask 0 ,0 ,0 ,0
Tag 1
Vertex2F 70 * 16,40 * 16
Vertex2F 82 * 16,80 * 16
CmdTrack 70 * 16, 40 * 16, 12, 40, 1
```



```
' To draw a dial with tag 33 centered at (80, 60), adjustable by touch
angle = &H8000
CmdTrack 80, 60, 1, 1, 33

Do
      Tag 33
      CmdDial 80, 60, 55, 0, angle
      .....
      tracker = Rd32(REG_TRACKER)
      If tracker AND 255 = 33 Then
         angle = tracker * 1000
      .....
      End If
Loop
```

```
' To make an adjustable slider with tag 34

val = &H8000
CmdTrack 20, 50, 120, 8, 34
Do
      ...
      Tag 34
      CmdSlider 20, 50, 120, 8, val, 65535
      ...
      tracker = Rd32(REG_TRACKER)
      If tracker AND 255 = 33 Then
        val = tracker * 1000
      End If
      ...
Loop
```



8.1.1.67 CmdTranslate

Action

Apply a translation to the current matrix.

Syntax

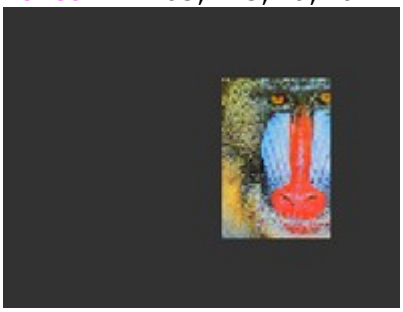
CmdTranslate tx, ty

Remarks

tx	x translate factor, in signed 16.16 bit fixed-point form
ty	y translate factor, in signed 16.16 bit fixed-point form

Example

```
' Pseudocode

' To translate the bitmap 20 pixels to the right
Begin_G BITMAPS
CmdLoadIdentity
CmdTranslate 20 * 65536, 0
CmdSetMatrix
Vertex2II 68, 28, 0, 0
```



```
' To translate the bitmap 20 pixels to the left
Begin_G BITMAPS
CmdLoadIdentity
CmdTranslate -20 * 65536, 0
CmdSetMatrix
Vertex2II 68, 28, 0, 0
```

8.1.1.68 CmdTranslateP

Action

Apply a translation to the current matrix.

Syntax

CmdTranslateP tx, ty

Remarks

tx	x translate factor
ty	y translate factor

Note: This is the same command as CmdTranslate 1686 except you can enter direct Pixel values instead of having to multiply by it 65536 to convert to a Pixel.

8.1.1.69 Color_A

Action
Set the current color alpha.

Syntax
Color_A alpha

Remarks

alpha	Alpha for the current color. **0** to **255**, the initial value is **255**

Sets the alpha value applied to drawn elements - points, lines, and bitmaps. How the alpha value affects image pixels depends on BlendFunc 1633 the default behavior is a transparent blend.

See also
ColorRGB 1689, BlendFunc 1633

Example
```
' Pseudocode

' Drawing three characters with transparency 255, 128, and 64
Begin_G BITMAPS
Vertex2II 50, 30, 31, &H47
Color_A 128
Vertex2II 58, 38, 31, &H47
Color_A 64
Vertex2II 66, 46, 31, &H47
```

8.1.1.70 ColorMask

Action
Enable or disable writing of color components.

Syntax
ColorMask r, g ,b ,a

Remarks

r	Enable or disable the red channel update of the FT800 color buffer. The initial value is **1** and means enable
g	Enable or disable the green channel update of the FT800 color buffer. The

	initial value is **1** and means enable
b	Enable or disable the blue channel update of the FT800 color buffer. The initial value is **1** and means enable
a	Enable or disable the alpha channel update of the FT800 color buffer. The initial value is **1** and means enable

The color mask controls whether the color values of a pixel are updated. Sometimes it is used to selectively update only the
red, green, blue or alpha channels of the image. More often, it is used to completely disable color updates while updating the
tag and stencil buffers.

See also
TagMask 1702

Example
```
' Pseudocode

'Draw a '8' digit in the middle of the screen. Then paint an invisible 40-pixel  ci
'touch area into the tag buffer

Begin_G BITMAPS
Vertex2II 68, 40, 31, &H38
PointSize 40 * 16
ColorMask 0, 0, 0, 0
Begin_G FTPOINTS
Tag &H38
Vertex2II 80, 60, 0, 0
```

8.1.1.71 ColorRGB

Action
Set the current color red, green and blue.

Syntax
ColorRGB red, green ,blue

Remarks
red	Red value for the current color. **0** to **255** , initial value is 255
green	green value for the current color. **0** to **255** , initial value is 255
blue	blue value for the current color. **0** to **255** , initial value is 255

Sets red, green and blue values of the FT800 color buffer which will be applied to the following draw operation.

See also
Color_A 1688 , ColorRGBdw 1690

Example
```
' Pseudocode

' Drawing three characters with different colors
Begin_G BITMAPS
Vertex2II 50, 38, 31, &H47
ColorRGB 255, 100, 50
Vertex2II 80, 38, 31, &H47
ColorRGB 50, 100, 255
Vertex2II 110, 38, 31,&H47
```


8.1.1.72 ColorRGBdw

Action
Set the current color red, green and blue.

Syntax
ColorRGBdw rgb

Remarks

rgb	Value in the range of **0** to **&H00FFFFFF**, Red is the most significant 8 bits and Blue is the least. So **&H**ff0000 is bright Red.

Sets red, green and blue values of the FT800 color buffer which will be applied to the following draw operation.

Note: this is the same as ColorRGB 1689 except you can now parse the whole rgb values in a dword

See also
Color_A 1688 , ColorRGB 1689

8.1.1.73 Display_E

Action
End the display list. FT800 will ignore all the commands following this command.

Syntax
Display_E

See Also
CALL_C 1634 , JUMP 1691, RETURN_C 1696 , MACRO_R 1693

Remarks

Note: The Bascom FT800 Lib calls Display_E from within **UpdateScreen** 1357 so it's not required in most circumstances.

8.1.1.74 End_G

Action
End drawing a graphics primitive.

Syntax
End_G

Remarks
It is recommended to have an End_G for each Begin_G 1625.

**For advanced users you can avoid the usage of End_G in order to save extra graphics instructions in the
Display List RAM.**

See also
BEGIN_G 1625 , VERTEX2F 1702, VERTEX2II 1703

8.1.1.75 Jump

Action
Execute commands at another location in the Display List.

Syntax
Jump dest

Remarks

dest	Display list address to be jumped

See also
CALL_C 1634 , RETURN_C 1696 , MACRO_R 1693 , DISPLAY_E 1690

8.1.1.76 LineWidth

Action
Specify the width of lines to be drawn with primitive LINES in 1/16th pixel precision.

Syntax
LineWdth width

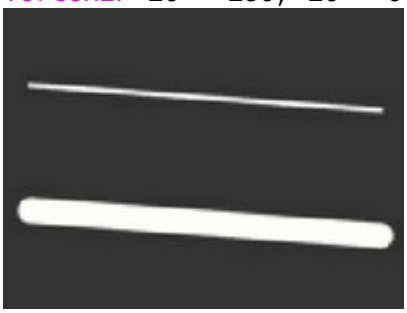
Remarks

width	Line width in 1/16 pixel. The initial value is 16, range is **16** to **4095**

Sets the width of drawn lines. The width is the distance from the center of the line to the outermost drawn pixel, in units
of 1/16 pixel. The valid range is from 16 to 4095 in terms of 1/16th pixel units.

Please note the LineWidth command will affect the LINES, LINE_STRIP, RECTS, EDGE_STRIP_A/B/R/L primitives.

Example
```
' Pseudocode

' The second line is drawn with a width of 80, for a 5 pixel radius
Begin_G LINES
Vertex2F 16 * 10, 16 * 30
Vertex2F 16 * 150, 16 * 40
LineWidth 80
Vertex2F 16 * 10, 16 * 80
Vertex2F 16 * 150, 16 * 90
```



8.1.1.77 Macro_R

Action
Execute a single command from a macro register.

Syntax
Macro_R m

Remarks

m	Macro register to read. Value **0** means the FT800 will fetch the command from REG_MACRO_0 to execute. Value **1** means the FT800 will fetch the command from REG_MACRO_1 to execute. The content of REG_MACRO_0 or REG_MACRO_1 shall be a valid display list command, otherwise the behavior is undefined.

See Also
CALL_C 1634 , JUMP 1691, RETURN_C 1696 , DISPLAY_E 1690

8.1.1.78 PointSize

Action
Specify the radius of points.

Syntax
PointSize size

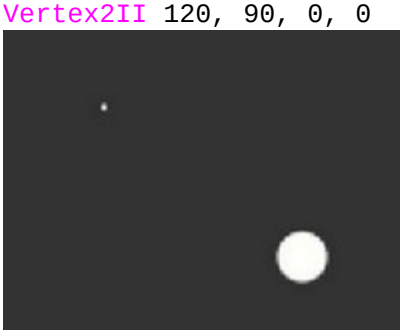
Remarks

size	Point radius in 1/16 pixel. range **16** to **8191**, the initial value is 16

Sets the size of drawn points. The width is the distance from the center of the point to the outermost drawn pixel, in
units of 1/16 pixels. The valid range is from 16 to 8191 with respect to 1/16th pixel unit.

Example
```
' Pseudocode

' The second point is drawn with a width of 160, for a 10 pixel radius
Begin_G FTPOINTS
```

```
Vertex2II 40, 30, 0, 0
PointSize 160
Vertex2II 120, 90, 0, 0
```



8.1.1.79 RD8

Action
This function returns a BYTE from the FT800 processor.

Syntax
var **= RD8(**address**)**

Remarks
NONE

See also
RD16 1694 , RD32 1695 , CMD32 1641, WR32 1706

Example
```
Sub Dlswap()
'-----------------------------------------------------------------------------------------------------------
  ' API to check the status of previous DLSWAP and perform DLSWAP of new DL
  ' Check for the status of previous DLSWAP and if still not done wait for few ms and check again

  Local Swap_done As Byte

  ' Perform a new DL swap
  Wr8 Reg_dlswap , Dlswap_frame

  ' Wait till the swap is done
  While Swap_done > 0
    Swap_done = Rd8(reg_dlswap)
    If Dlswap_done <> Swap_done Then
      Waitms 10
    End If
  Wend

End Sub                                 ' Dlswap
```

8.1.1.80 RD16

Action
This function returns a WORD from the FT800 processor.

Syntax
var **= RD16(**address**)**

Remarks
NONE

See also
RD8 1694, RD32 1695, CMD32 1641, WR32 1706

Example
```
Sub Dlswap()
'-------------------------------------------------------------------------------------------------
  ' API to check the status of previous DLSWAP and perform DLSWAP of new DL
  ' Check for the status of previous DLSWAP and if still not done wait for few ms and check again

  Local Swap_done As Byte

  ' Perform a new DL swap
  Wr8 Reg_dlswap , Dlswap_frame

  ' Wait till the swap is done
  While Swap_done > 0
    Swap_done = Rd8(reg_dlswap)
    If Dlswap_done <> Swap_done Then
      Waitms 10
    End If
  Wend

End Sub                                 ' Dlswap
```

8.1.1.81 RD32

Action
This function returns a DWORD from the FT800 processor.

Syntax
var **= RD32(**address**)**

Remarks
NONE

See also
RD8 1694, RD16 1694, CMD32 1641, WR32 1706

Example
```
Sub Dlswap()
'-------------------------------------------------------------------------------------------------
  ' API to check the status of previous DLSWAP and perform DLSWAP of new DL
  ' Check for the status of previous DLSWAP and if still not done wait for few ms and check again

  Local Swap_done As Byte

  ' Perform a new DL swap
  Wr8 Reg_dlswap , Dlswap_frame

  ' Wait till the swap is done
  While Swap_done > 0
    Swap_done = Rd8(reg_dlswap)
    If Dlswap_done <> Swap_done Then
      Waitms 10
```

```
    End If
  Wend

End Sub                                 ' Dlswap
```

8.1.1.82 RestoreContext

Action
Restore the current graphics context from the context stack.

Syntax
RestoreContext

Remarks
Restores the current graphics context. Four (4) levels of SAVE and RESTORE are available in the FT800.
Any extra RestoreContext will load the default values into the present context.

See also
SaveContext 1697

Example
```
' Pseudocode

' Saving and restoring context means that the second 'G' is drawn in red, instead c
Begin_G BITMAPS
ColorRGB 255, 0, 0
SaveContext
ColorRGB 50, 100, 255
Vertex2II 80, 38, 31, &H47
RestoreContext
Vertex2II 110, 38, 31,&H47
```

8.1.1.83 Return_C

Action
Return from a previous Call_C 1634 command.

Syntax
Return_C

Remarks

Call_C[1634] and Return_C have 4 levels of stack in addition to the current pointer. Any additional Call_C/[1634]Return_C done wil
lead to unexpected behavior.

See also

CALL_C[1634] , JUMP[1691], MACRO_R[1693] , DISPLAY_E[1690]

8.1.1.84 SaveContext

Action

Push the current graphics context on the context stack.

Syntax

SaveContext

Remarks

Saves the current graphics context Any extra SaveContext will throw away the earliest saved context.

See also

RestoreContext[1696]

Example

```
' Pseudocode

' Saving and restoring context means that the second 'G' is drawn in red, instead c
Begin_G BITMAPS
ColorRGB 255, 0, 0
SaveContext
ColorRGB 50, 100, 255
Vertex2II 80, 38, 31, &H47
RestoreContext
Vertex2II 110, 38, 31, &H47
```

8.1.1.85 ScissorSize

Action

Specify the size of the scissor clip rectangle.

Syntax
ScissorSize width, height

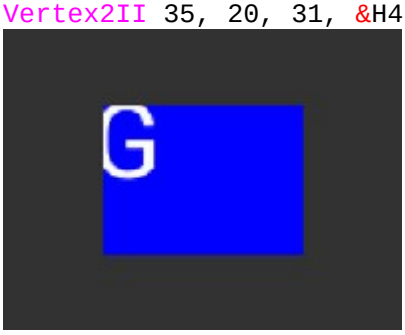
Remarks

width	The width of the scissor clip rectangle, in pixels. The initial value is 512. The valid value range is from **0** to **512**.
height	The height of the scissor clip rectangle, in pixels. The initial value is 512. The valid value range is from **0** to **512**

Sets the width and height of the scissor clip rectangle, which limits the drawing area.

See Also
SCISSORXY 1698

Example
```
' Pseudocode

' Setting a 40 x 30 scissor rectangle clips the clear and bitmap drawing
ScissorXY 40, 30
ScissorSize 80, 60
ClearColorRGB 0, 0, 255
Clear_B 1, 1, 1
Begin_G BITMAPS
Vertex2II 35, 20, 31, &H47
```



8.1.1.86 ScissorXY

Action
Specify the size of the scissor clip rectangle.

Syntax
ScissorXY x, y

Remarks

x	The x coordinate of the scissor clip rectangle, in pixels. The initial value is 0
y	The y coordinate of the scissor clip rectangle, in pixels. The initial value is 0

Sets the top-left position of the scissor clip rectangle, which limits the drawing area.

See Also
SCISSORSIZE 1697

Example
```
' Pseudocode

' Setting a 40 x 30 scissor rectangle clips the clear and bitmap drawing

ScissorXY 40, 30
ScissorSize 80, 60
ClearColorRGB 0, 0, 255
Clear_B 1, 1, 1
Begin_G BITMAPS
Vertex2II 35, 20, 31, &H47
```


8.1.1.87 StencilFunc

Action
Set function and reference value for stencil testing.

Syntax
StencilFunc func, ref, mask

Remarks

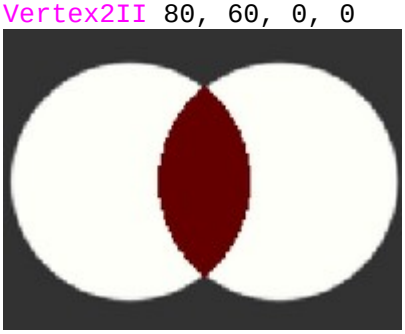
func	Specifies the test function, one of **NEVER**, **LESS**, **LEQUAL**, **GREATER**, **GEQUAL**, **EQUAL**, **NOTEQUAL**, or **ALWAYS**. The initial value is ALWAYS.
ref	Specifies the reference value for the stencil test, range **0** to **255**, the initial value is 0
mask	Specifies a mask that is ANDed with the reference value and the stored stencil value, range **0** to **255** The initial value is 255

Stencil test rejects or accepts pixels depending on the result of the test function defined in func parameter, which operates
on the current value in the stencil buffer against the reference value.

See also
StencilOp 1700, StencilMask 1700

Example

```
' Pseudocode

' Draw two points, incrementing stencil at each pixel, then draw the pixels with va
StencilOp INCR, INCR
PointSize 760
Begin_G FTPOINTS
Vertex2II 50, 60, 0, 0
Vertex2II 110, 60, 0, 0
StencilFunc EQUAL, 2, 255
ColorRGB 100, 0, 0
Vertex2II 80, 60, 0, 0
```



8.1.1.88 StencilMask

Action

Control the writing of individual bits in the stencil planes.

Syntax

StencilMask mask

Remarks

mask	The mask used to enable writing stencil bits, range **0** - **255,** the initial value is 255

See also

StencilFunc 1699, StencilOp 1700, TagMask 1702

8.1.1.89 StencilOp

Action

Set stencil test actions.

Syntax

StencilOp sfail, spass

Remarks

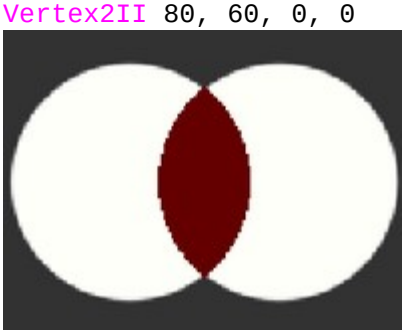
sfail	Specifies the action to take when the stencil test fails, one of **KEEP**, **ZERO**, **REPLACE**, **INCR**, **DECR** and **INVERT**. The initial value is KEEP

spass	Specifies the action to take when the stencil test passes, one of the same constants as sfail. The initial value is KEEP

The stencil operation specifies how the stencil buffer is updated. The operation selected depends on whether the stencil test
passes or not.

See also
StencilFunc 1699, StencilMask 1700

Example
```
' Pseudocode

' Draw two points, incrementing stencil at each pixel, then draw the pixels with va
StencilOp INCR, INCR
PointSize 760
Begin_G FTPOINTS
Vertex2II 50, 60, 0, 0
Vertex2II 110, 60, 0, 0
StencilFunc EQUAL, 2, 255
ColorRGB 100, 0, 0
Vertex2II 80, 60, 0, 0
```



8.1.1.90 Tag

Action
Attach the tag value for the following graphics objects drawn on the screen.

Syntax
Tag s

Remarks
s	Tag value. Valid value range is from **1** to **255**

The initial value of the tag buffer of the FT800 is specified by command ClearTag 1639 and taken effect by command Clear_B 1635.
Tag command can specify the value of the tag buffer of the FT800 that applies to the graphics objects when they are drawn
on the screen. This Tag value will be assigned to all the following objects, unless the TagMask 1702 command is used to disable it.

Once the following graphics objects are drawn, they are attached with the tag value successfully.
When the graphics objects attached with the tag value are touched, the register REG_TOUCH_TAG will be updated
with the tag value of the graphics object being touched.

If there is no Tag commands in one display list, all the graphics objects rendered by the display list will report tag value as 255
in REG_TOUCH_TAG when they were touched.

See also
ClearTag 1639, TagMask 1702

8.1.1.91 TagMask

Action
Control the writing of the tag buffer.

Syntax
TagMask mask

Remarks

mask	Allow updates to the tag buffer. The initial value is **one** (**1**) and it means the tag buffer of the FT800 is updated with the value given by the Tag 1701 command. Therefore, the following graphics objects will be attached to the tag value given by the TAG command. The value **zero** (**0**) means the tag buffer of the FT800 is set as the default value, rather than the value given by Tag 1701 command in the display list.

Every graphics object drawn on screen is attached with the tag value which is defined in the FT800 tag buffer.
The FT800 tag buffer can be updated by Tag 1701 command.
The default value of the FT800 tag buffer is determined by ClearTag 1639 and Clear_B 1635 commands.
If there is no ClearTag 1639 command present in the Display List, the default value in tag buffer shall be 0.
TagMask 1702 command decides whether the FT800 tag buffer takes the value from the default value of the FT800 tag buffer or
the Tag 1701 command of the Display List.

See also
Tag 1701, ClearTag 1639, StencilMask 1700, ColorMask 1688

8.1.1.92 Vertex2f

Action
Start the operation of graphics primitives at the specified screen coordinate, in 1/16th pixel precision.

Syntax
Vertex2f x, y

Remarks

x	x-coordinate in 1/16 pixel precision (Integer)
y	y-coordinate in 1/16 pixel precision (Integer)

The range of coordinates can be from -16384 to +16383 in terms of 1/16 th pixel units.
The Vertex2F command allows negative coordinates. It also allows fractional coordinates, because it specifies screen (x,y) in
units of 1/16 of a pixel.

Please note the negative x coordinate value means the coordinate in the left virtual screen from (0, 0), while the negative y coordinate value means the coordinate in the upper virtual screen from (0, 0). If drawing on the negative coordinate position, the drawing operation will not be visible

See also
BEGIN_G 1625 , END_G 1691 , VERTEX2II 1703

Example

```
ClearColorRGB 5, 45, 10
ColorRGB 255, 168, 64
Clear_B 1 ,1 ,1
Begin_G EDGE_STRIP_R
Vertex2F 16 * 16, 16 * 16
Vertex2F (FT_DispWidth * 2 / 3)  * 16, (FT_DispHeight * 2 / 3) * 16
Vertex2F (FT_DispWidth - 80) * 16, (FT_DispHeight - 20) * 16

UpdateScreen
```

8.1.1.93 Vertex2ii

Action
Start the operation of graphics primitive at the specified coordinates in pixel precision.

Syntax
Vertex2ii x, y, handle, cell

Remarks

x	x-coordinate in pixels, from **0** to **511**
y	y-coordinate in pixels, from **0** to **511**
handle	Bitmap handle. The valid range is from **0** to **31**. From 16 to 31, the bitmap handle is dedicated to the FT800 built-in font.

cell	Cell number. Cell number is the index of bitmap with same bitmap layout and format. For example, for handle 31, the cell 65 means the character "A" in the largest built in font.

The Vertex2II command only allows positive screen coordinates.
If the bitmap is partially off screen, for example during a screen scroll, then it is necessary to
specify negative screen coordinates (with Vertex2F).

The handle and cell parameters will be ignored unless the graphics primitive is specified as bitmap by command Begin_G 1625, prior to this command.

See Also
BEGIN_G 1625 , END_G 1691 , VERTEX2F 1702

Example

```
Clear_B 1, 1, 1 ' Clear Screen
BitmapSource RAM_G
BitmapLayout BARGRAPH, 256, 1
BitmapSize NEAREST, Border, Border, 256, 256
Begin_G BITMAPS
ColorRGB 255, 0, 0
' Display bargraph At hoffset, voffset location
Vertex2II 0, 0, 0, 0
Vertex2II 256, 0, 0, 1
ColorRGB 0, 0, 0
Vertex2II 0, 4, 0, 0
Vertex2II 256, 4, 0, 1

UpdateScreen
```

8.1.1.94 UpdateScreen

Action
Executes the Commands in the FIFO and Display the Graphics.

Syntax
UpdateScreen

Remarks
UpdateScreen High level command which executes the following commands

Display_E 1690
CmdSwap 1680
CmdDlStart 1650
WaitCmdFifoEmpty 1705

Generally you insert this command towards the end of the loop or when you need to update the LCD.

Example

```
' Pseudocode

Do
  ClearScreen
  BitmapLayout PALETTED, Ft_DispWidth , Ft_DispHeight
  BitmapSize NEAREST, BORDER, BORDER, Ft_DispWidth, Ft_DispHeight
  ...
  ...
  UpdateScreen
Loop
```

8.1.1.95 WaitCmdFifoEmpty

Action

Executes Commands in the FIFO buffer.

Syntax

WaitCmdFifoEmpty

Remarks

WaitCmdFifoEmpty polls a loop checking the state of the Reg_Cmd_Read and Reg_Cmd_Write registers
to see whether the FT800 has executed the commands in the FIFO buffer.

If the your code is long you have to be careful it's not more than 4K otherwise you can get
overflows/corruption.

Inserting WaitCmdFifoEmpty in area of your code allows you to execute parts of your code
instantly, but be aware it won't display any Graphics and don't use it for Graphics Display
(use UpdateScreen) 1704

8.1.1.96 WR8

Action

This statement will write an address and a byte parameter to the FT800.

Syntax

WR8 address , prm

Remarks

The address need to be an address in the FT800 address range. See FT800 manual for more info.
The parameter (prm) is a word numeric value. It depends on the address which parameter value you may send.

When you want to write to the FIFO buffer you can best use CMD8.

See also
CMD8 1640 , CMD16 1641, CMD32 1641 , WR16 1706, WR32 1706

Example

```
Wr8 Reg_GPIO_Dir , &H83
Wr8 Reg_GPIO , &H83

Wr16 Reg_Touch_rzThresh , 1200

Wr32 Ram_DL + 0,  &H02FFFFFF
```

8.1.1.97 WR16

Action
This statement will write an address and a word parameter to the FT800.

Syntax
WR16 address , prm

Remarks
The address need to be an address in the FT800 address range. See FT800 manual for more info.
The parameter (prm) is a word numeric value. It depends on the address which parameter value you may send.

When you want to write to the FIFO buffer you can best use CMD16.

See also
CMD8 1640 , CMD16 1641, CMD32 1641 , WR8 1705, WR32 1706

Example

```
Wr8 Reg_GPIO_Dir , &H83
Wr8 Reg_GPIO , &H83

Wr16 Reg_Touch_rzThresh , 1200

Wr32 Ram_DL + 0,  &H02FFFFFF
```

8.1.1.98 WR32

Action
This statement will write an address and a dword parameter to the FT800.

Syntax
WR32 address , prm

Remarks

The address need to be an address in the FT800 address range. See FT800 manual for more info.
The parameter (prm) is a dword numeric value. It depends on the address which parameter value you may send.

When you want to write to the FIFO buffer you can best use CMD32.

See also

CMD8 1640 , CMD16 1641, CMD32 1641 , WR8 1705, WR16 1706

Example

```
Wr8 Reg_GPIO_Dir , &H83
Wr8 Reg_GPIO , &H83

Wr16 Reg_Touch_rzThresh , 1200

Wr32 Ram_DL + 0,  &H02FFFFFF
```

8.1.2 Getting Started

During the Development of the Bascom FT800 library, the following FT800/LCD hardware was used:

Model	Company	LCD Type/Size	URL
Gameduino 2	excamera	4.3" resistive touch display	excamera.com/sphinx/gameduino2
4DLCD-FT843	4D Systems	4.3" resistive touch display	www.4dsystems.com.au
VM800C35A	FTDI	3.5" resistive touch display	www.ftdichip.com
VM800C43A	FTDI	4.3" resistive touch display	www.ftdichip.com
VM800C50A	FTDI	5.0" resistive touch display	www.ftdichip.com

Minimum **Microcontroller** requirements would be ATMEGA328P or an Arduino. These boards are very common and are good value for money for any beginner who wants to start learning.

For the Advanced or Professional it would be advisable to chose a micro with plenty of Flash (>32k).

If using a microSD/SDHC (with AVR-DOS) a good start would be 4KB of SRAM (>2KB)

Have a look at the various DEMO's 1710. The Help shows the output you should get.

8.1.3 How to add another SPI device with the FT800

Bascom continuously Streams Data to the SPI bus trying to minimize additional commands sent over the SPI bus by taking advantage of some the the FT800 capabilities. Because of method used you have to be aware you can just add another SPI device and just let your micro talk to it.

What happens here, is that the CHIP Select line is held LOW for most of the time (depending on what code the Bascom FT800 is running at the time), if another SPI device wants to communicate with that micro then the data from that device will also be sent to the FT800 which means that you will get unexpected results!.

Wait, don't fear, here is some example code to show you how it can be done easily.

Our friend is **Endtransfer**

In the example below AVR-DOS needs to enable the Chip Select to do it's job (reading/writing), before doing so you have to call **Endtransfer** which tells the Micro to Set the Chip Select line to the FT800.

Note: The Chip Select line for the FT800 should/will automatically Reset next time it has to execute commands.

```
'-----------------------------------------------------------------------------------------
- -
Sub LoadJpeg( Byval file As Byte)
'-----------------------------------------------------------------------------------------
- -
    ' API's used to upload the image to GRAM from SD card

    Local fsize As Dword
    Local BlockLen As Word, Ptr1 As Word, Ptr2 As Word, Ptr3 As Word

    Endtransfer '<--------
    Open imagename(file) for Binary as #1

      fsize = Lof(1)

     Ptr1 = 1 ' Start at the first byte
     BlockLen = Chunk
    While fsize > 0
       If fsize > Chunk Then BlockLen = Chunk Else BlockLen = fsize
         fsize = fsize - BlockLen
       Endtransfer '<--------
       Get #1, Dat, Ptr1, BlockLen

         ALign4 BlockLen

        Ptr2 = BlockLen
        Ptr3 = _base

       While Ptr2 > 0
          Cmd8 aDat(Ptr3)
          Incr Ptr3
          Decr Ptr2
       Wend

       EndTransfer '<--------
         WaitCmdFifoEmpty

        Ptr1 = Ptr1 + BlockLen
    Wend
    Close #1

End Sub ' LoadJpeg
```

8.1.4 How to Screen Capture

How to Screen Capture

There is nothing better than been able to produce nice Screen captures from your Graphics Display (instead of using a camera) when wanting to write a manual or a help file explaining the different screen operations/features at what they do.

The process is quite simple to implement into your program generating a Screen capture output. You can use the supplied code or you can modify the code and produce your own version.
If you look at **FT800 Capture.Bas** it demonstrates the Screen capture using two routines.

Sub ScreenShot: is a demo originally from James Bowman (Gameduino2) which takes a snapshot and just outputs the data via Serial (which you have to write your own PC serial capture program).

Sub ScreenShot2: is the same as above except it uses additional control codes for handshaking and stopping the program. A sample PC (Windows) program called Capture FT800.exe demonstrate the capture process which when successful produces a BMP file.

Capture FT800.exe waits for a ACK to acknowledge a ready to receive message so transmission can start, once transfer begins and then finishes it receives a EOT acknowledge end of transmission., Additional to this if the user wants to stop/quit transmission the program will send an ESC character to notify the hardware to stop sending data .
The easiest way to begin is to add Screenshot.inc to your code:

```
$Include "FT800.inc"
$Include "FT800_Functions.inc"

$Include "ScreenShot.inc" ' ç ==== add this line
```

Then decide where in your program you want to call ScreenShot2 so it can start the capturing process (working with Capture FT800.exe).
This example it's called at the end of the program:

```
Do
   Demo
Loop

 ScreenShot2

End
```

This sample is called within a certain code area, straight after the screen is displayed.

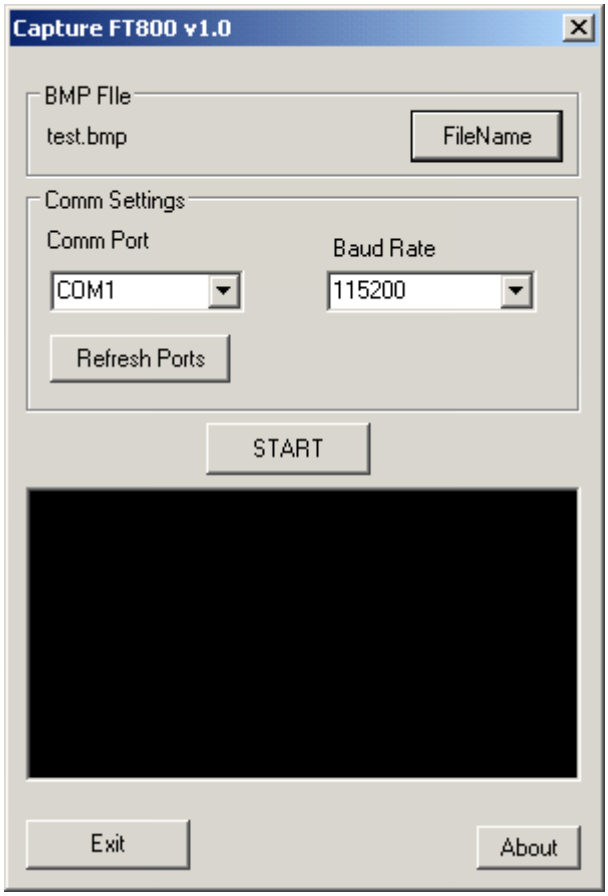
```
    ClearScreen
   ColorRGB  255,   255,   255
   BitmapSource  RAM_G
      BitmapLayout   Header_Format( 1+_base) ,     Header_Stride( 1+_base) ,
Header_Height( 1+_base)
   BitmapSize  NEAREST,    Border,    Border,    Header_Width( 1+_base) ,
Header_Height( 1+_base)
   Begin_G  BITMAPS '    start    drawing    bitmaps
   Const DA =    FT_DispWidth  /  4
   Ln1  =    Header_Width( 1+_base)  /  2
   Const DB =    FT_DispHeight  /  2
   Ln2  =    Header_Height( 1+_base)  /  2
     BMoffsetx  = DA  -  Ln1
```

```
BMoffsety = DB - Ln2
  Vertex2II  BMoffsetx,    BMoffsety,  0,  0
UpdateScreen
```

ScreenShot2

Using the *Capture FT800.exe:*

Note: when possible use the highest baud rate possible to decrease the wait time of receiving transmission. Don't forget to make sure the Hardware baud rate matches the Capture FT800 baud rate! (it won't time time out if wrong).



1) Chose your Comm port
2) Select the Baud rate of your Hardware
3) You can either enter a filename or it can prompt you at the end of the capture.
4) Press Start when ready, if successful you will see a message.

8.1.5 Demos

There are around 67 various demos which have been ported from various sources (please respect some of the credits notes in some of
the sample demos). These have been tested and work, but some may not be optimized for the 3.5" display.

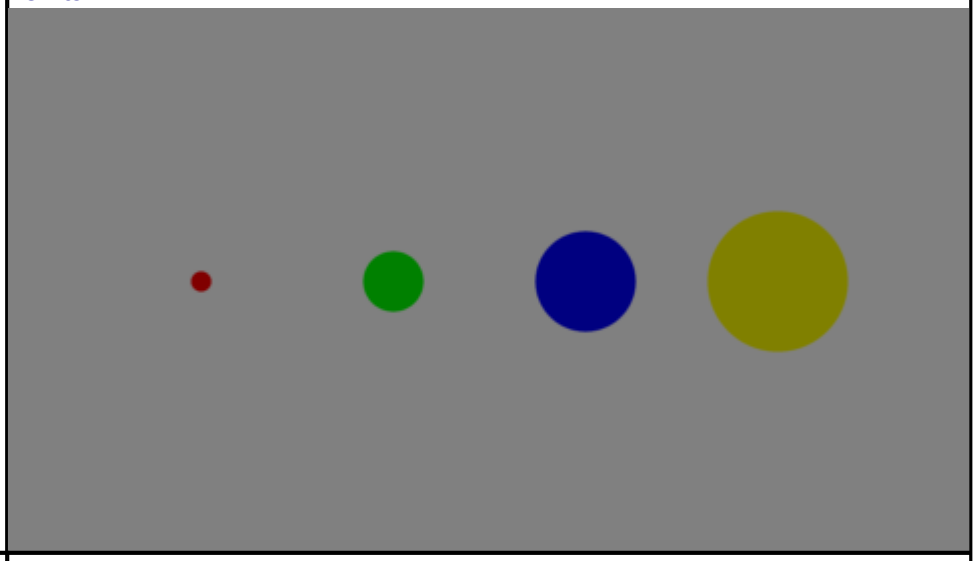
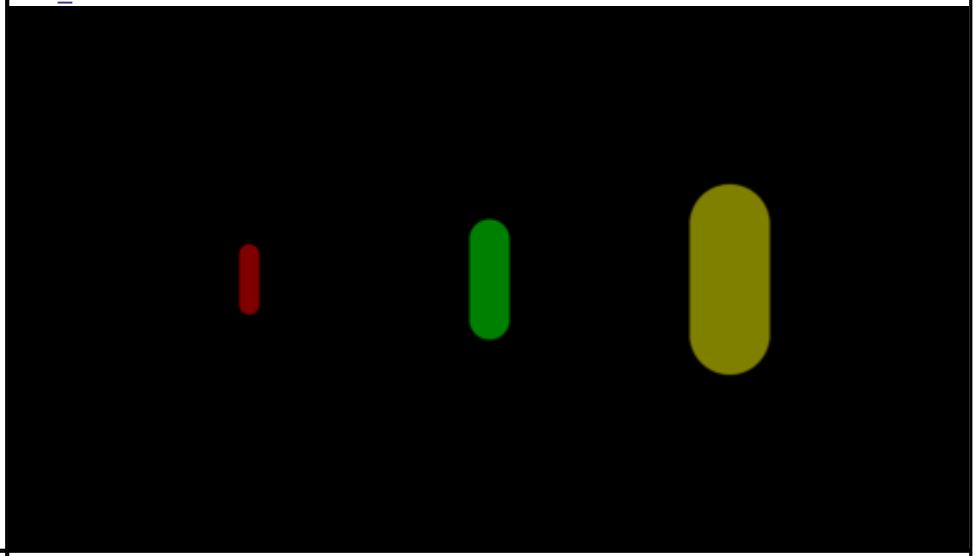
If you want to see more fantastic demo's click here YouTube demos

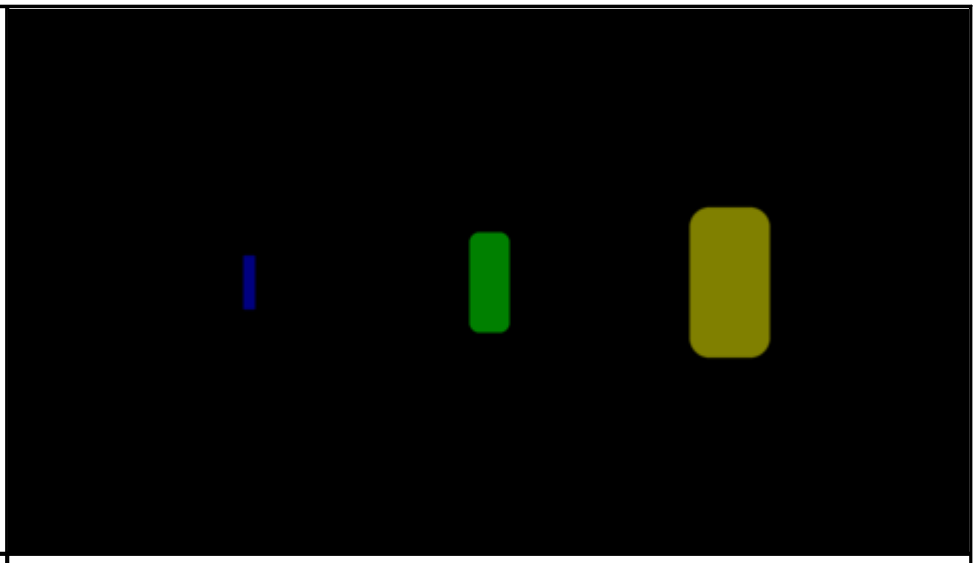
Since the all original samples where from C/C++ sources, you may notice some demos could be further be improved or optimized or think why was is done that way. Some samples have been modified and have been enhanced and improved using Bascom's rich commands, others have been left as is.

Also look in the Demo folder for the **Snapshots** folder which has all the Screen captures of the demo's.

Note: Some demos require an microSD/SDHC card to store the various pictures and fonts etc.

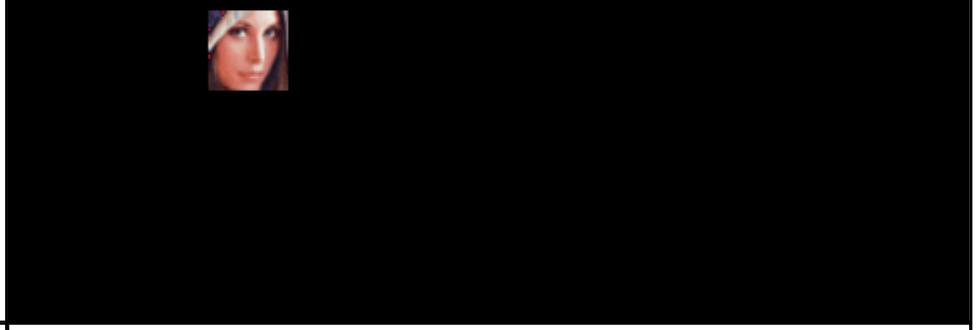
Here are the list of demo names:

Filename	Routine
Demo0	Points 
	Line_s 
	Rectangles



Bitmap

BitmapPalette

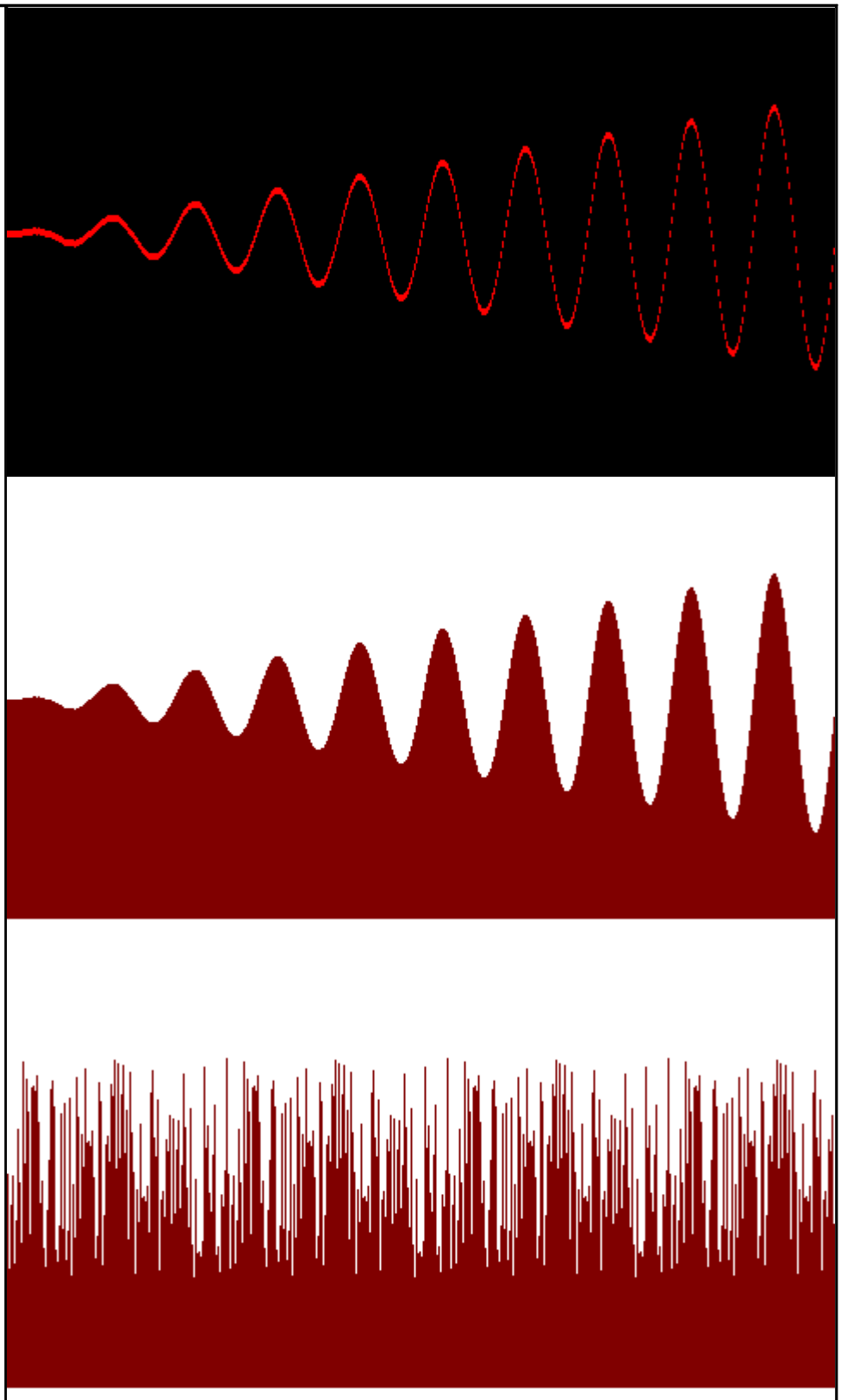


Fonts

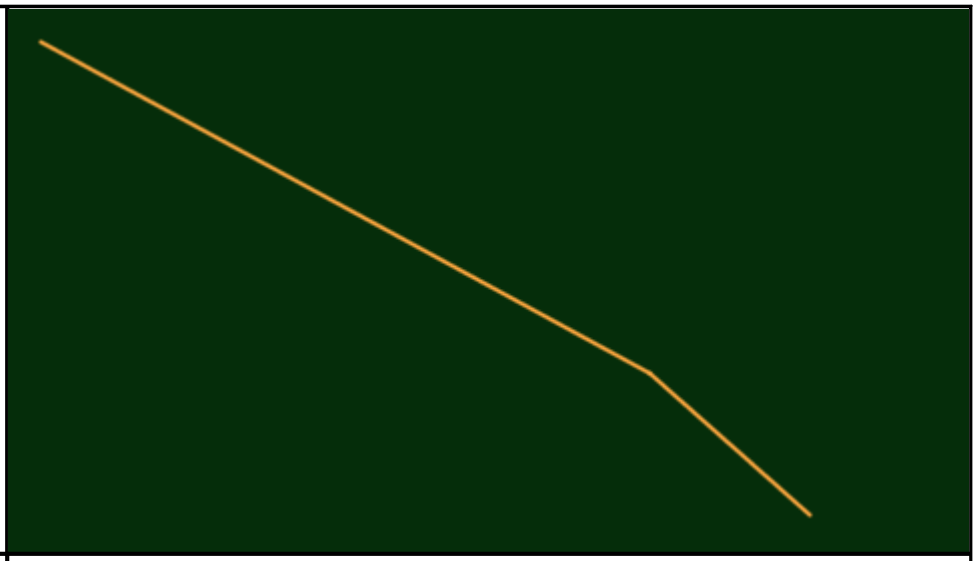
Text_8x8

Text_VGA

Bar_graph



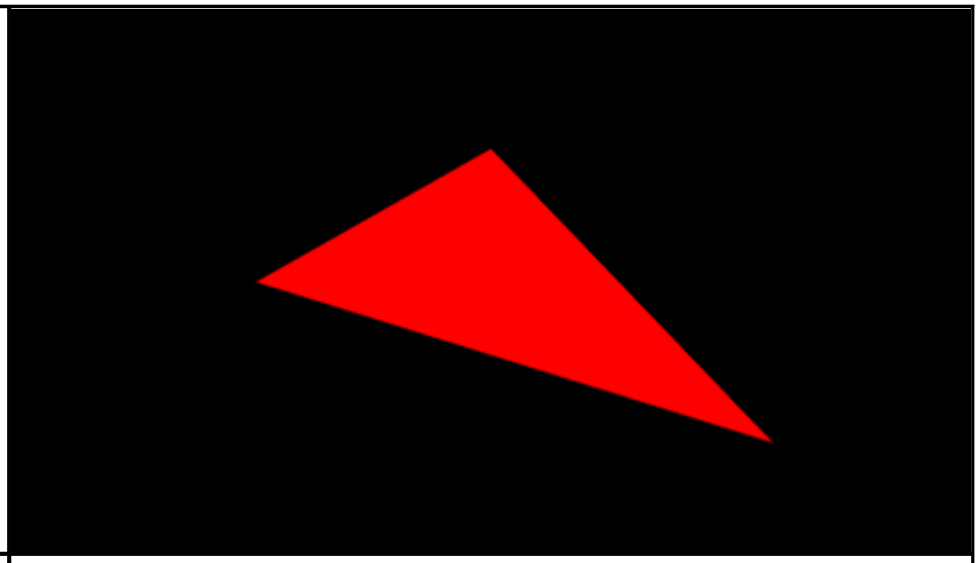
LineStrips



EdgeStrips

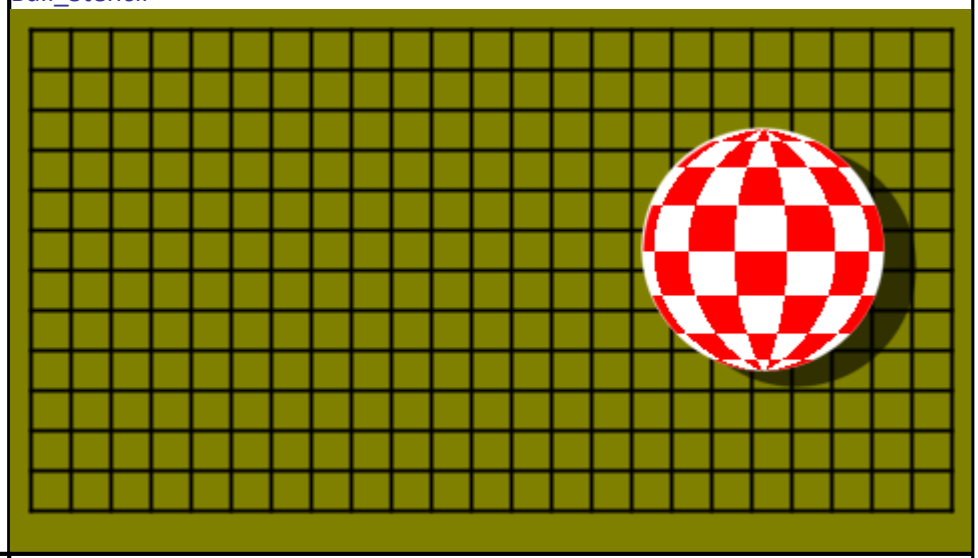
Scissor

Polygon

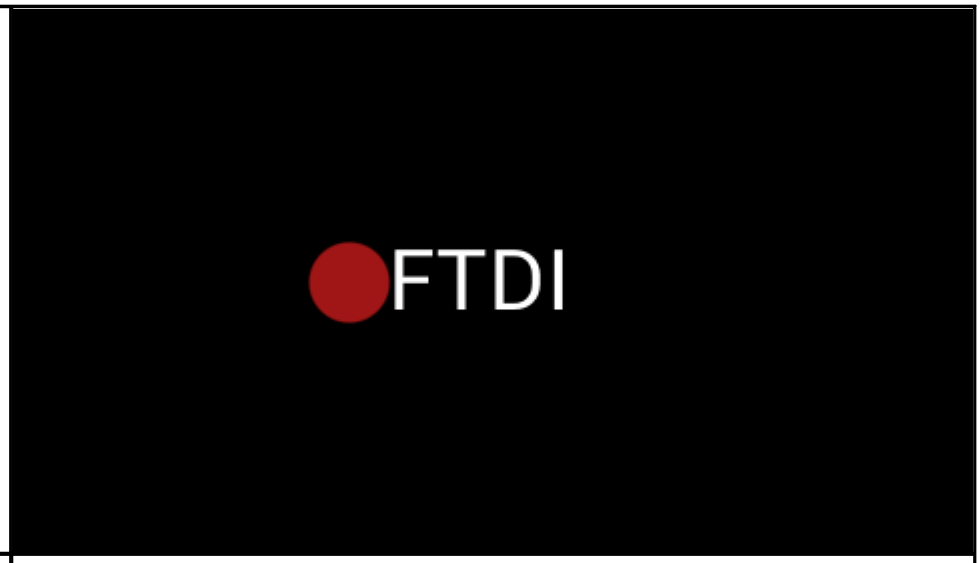


Cube

Ball_Stencil



FtdiString

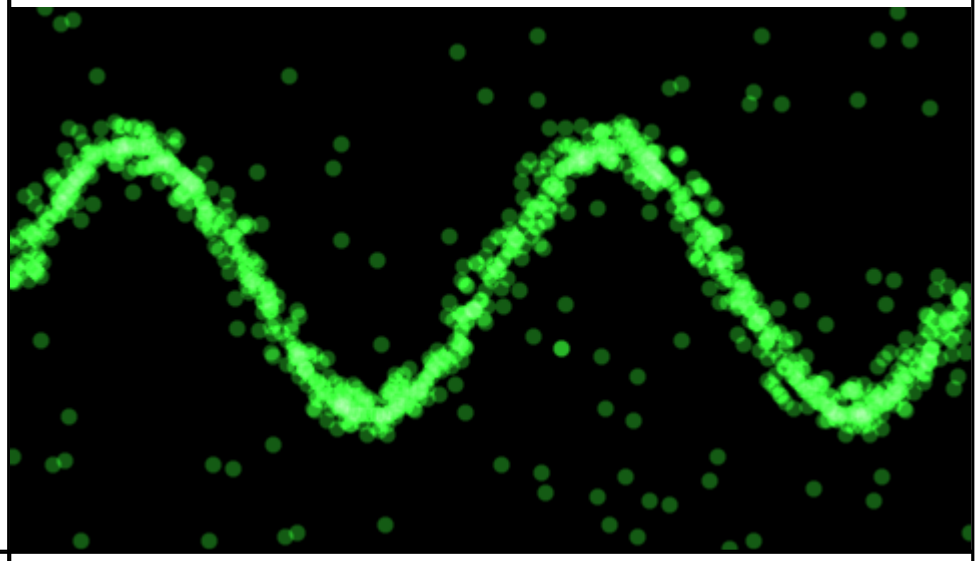


StreetMap

AdditiveBlendText

MacroUsage

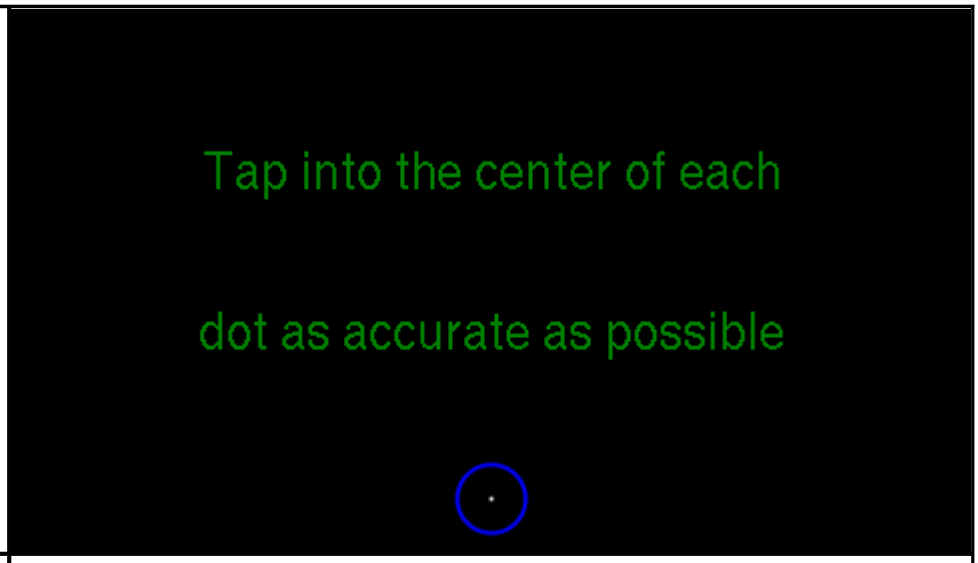
AdditiveBlendPoints



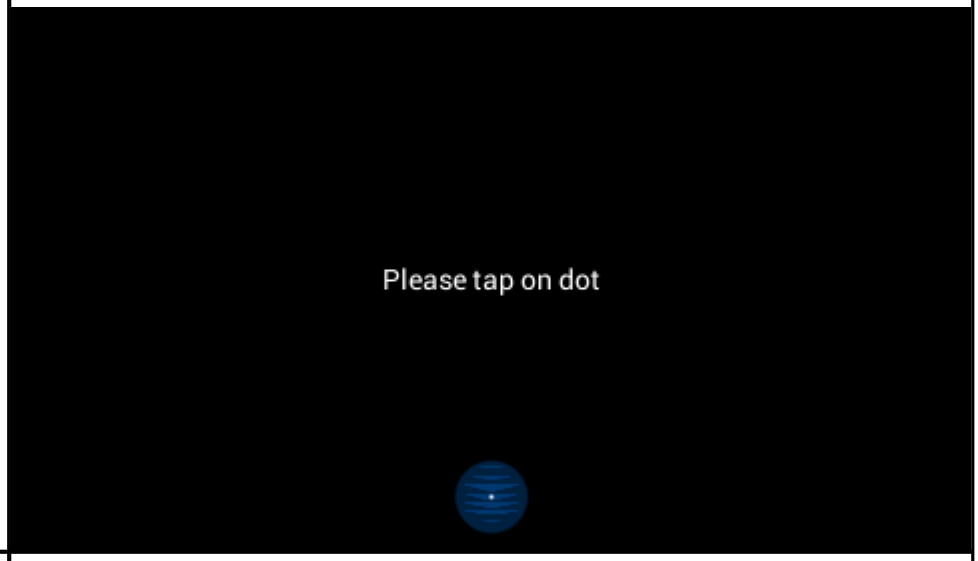
Demo1

Logo

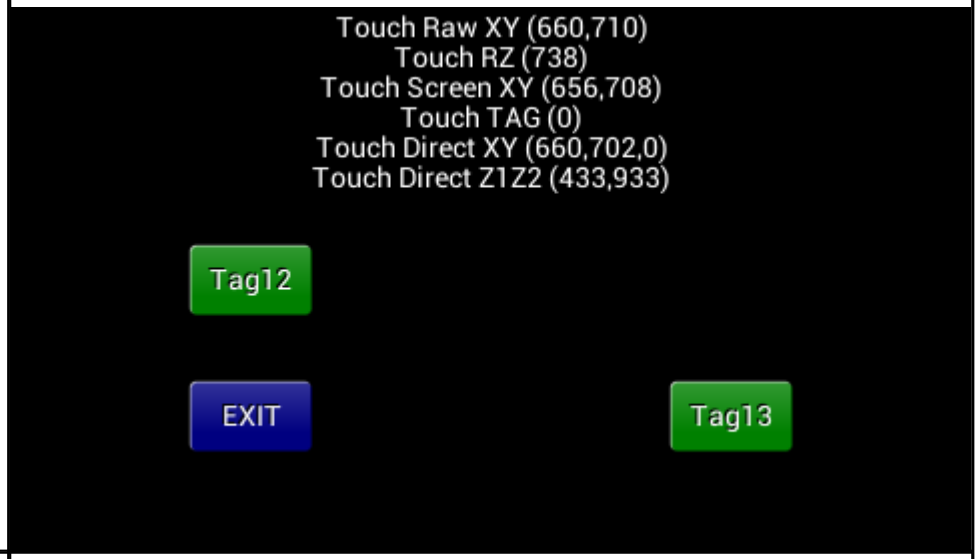
Calibrate1



Calibrate2

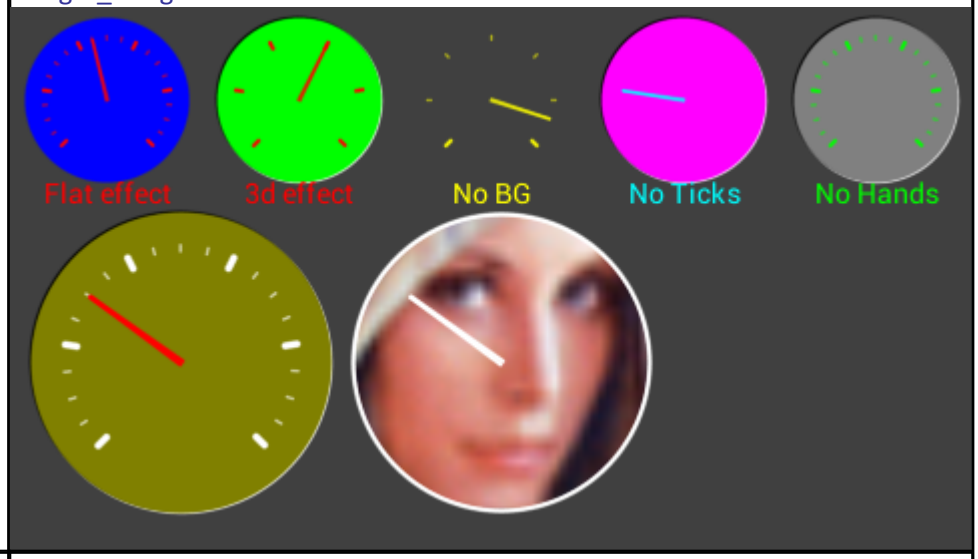


Touch

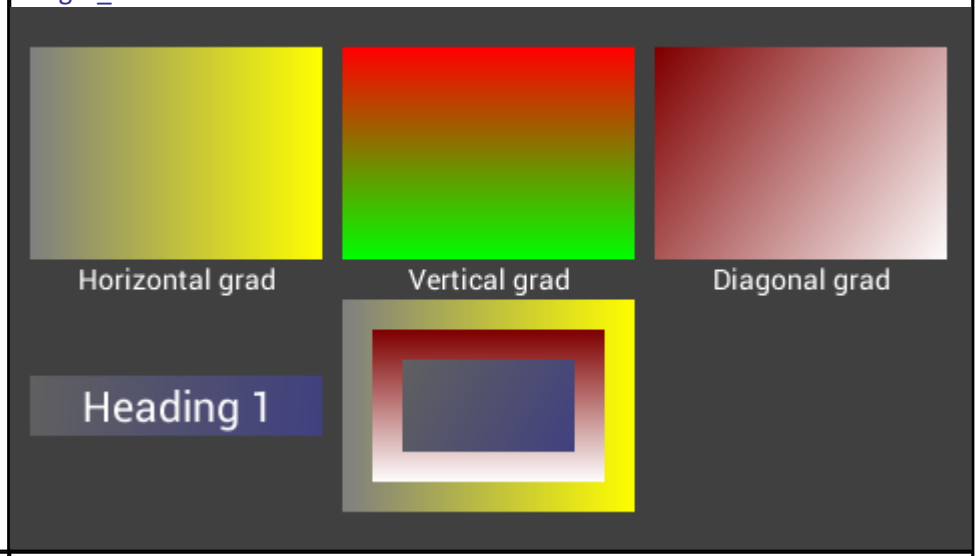


Widget_Clock

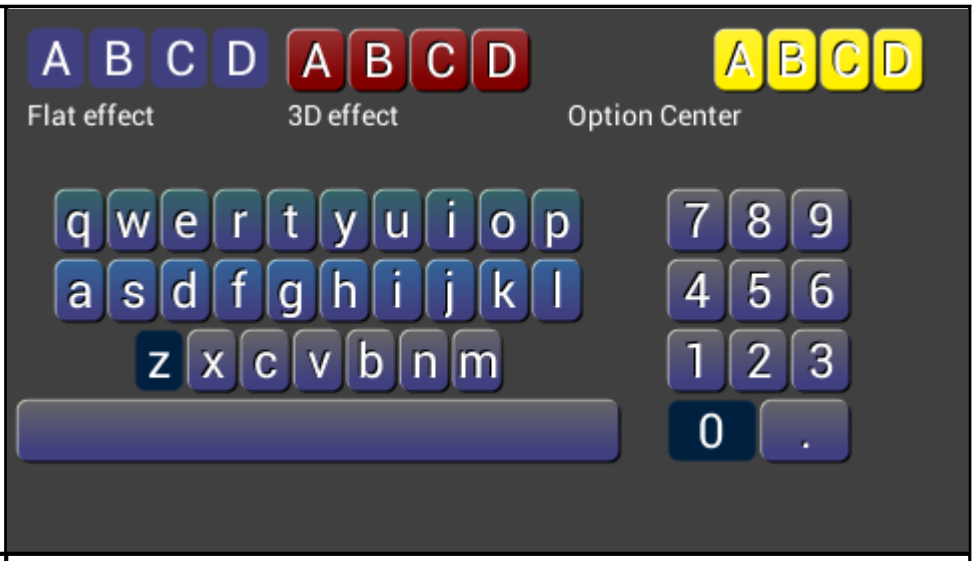
Widget_Gauge



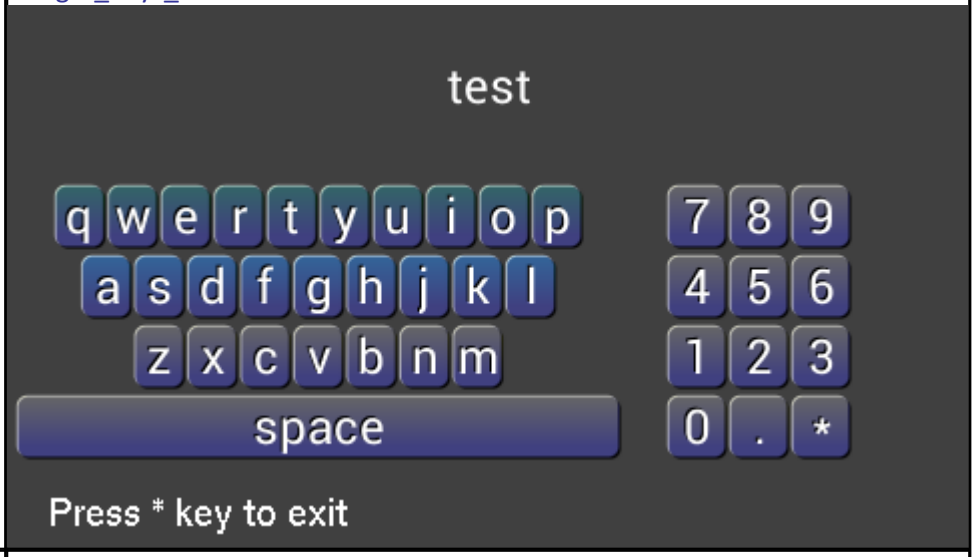
Widget_Gradient



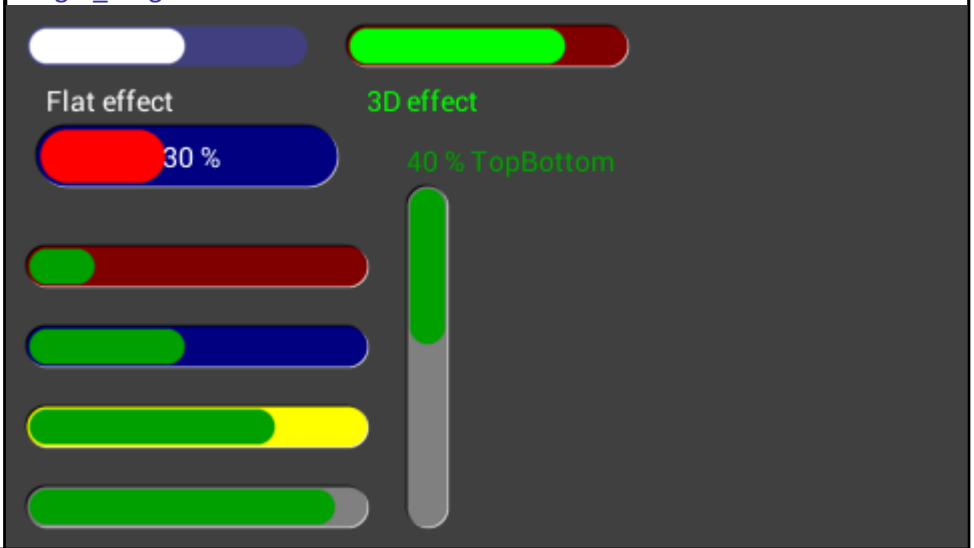
Widget_Keys



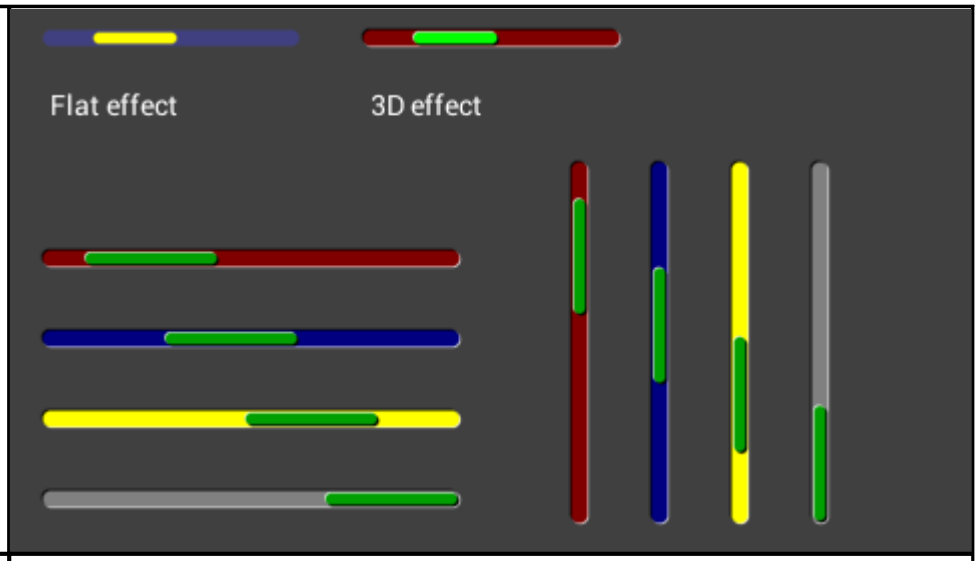
Widget_Keys_Interactive



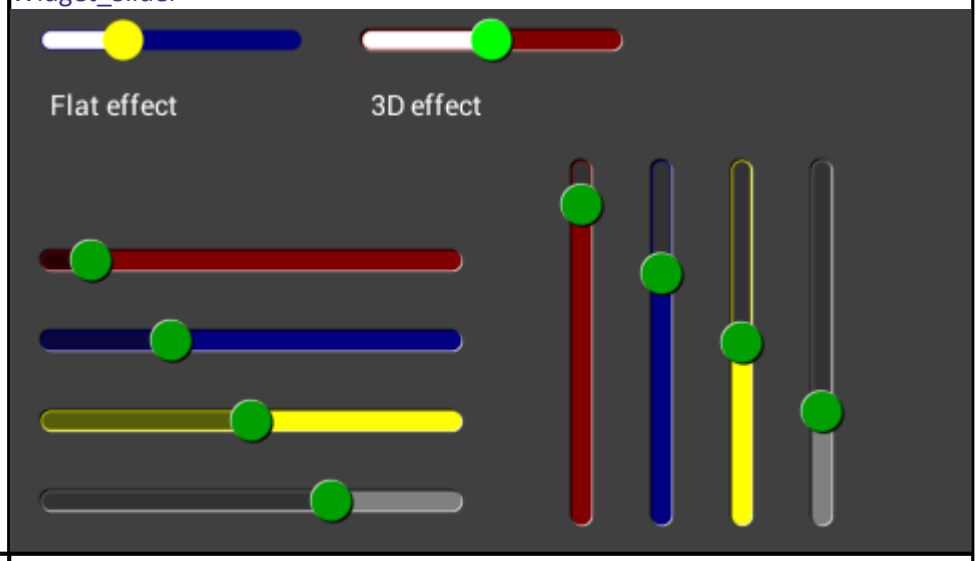
Widget_Progressbar



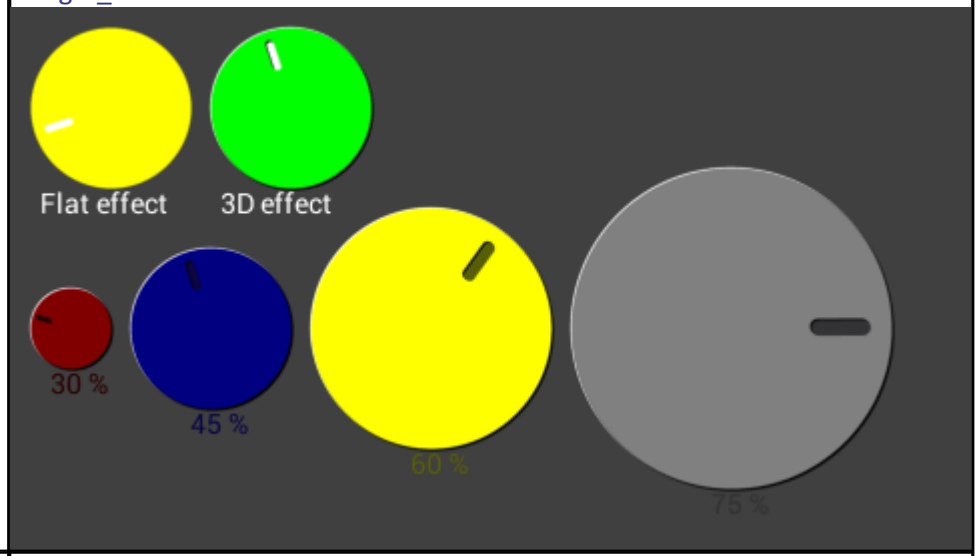
Widget_Scroll



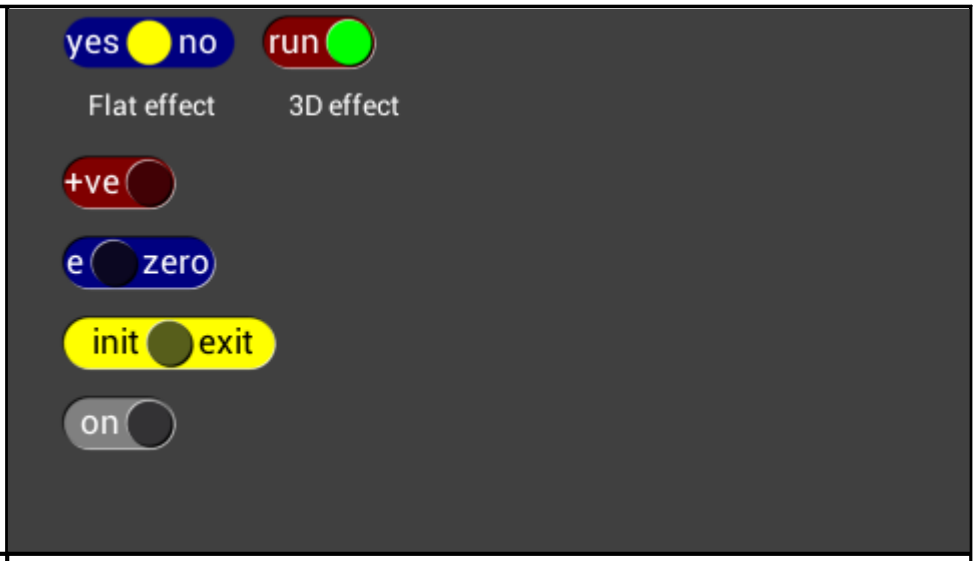
Widget_Slider



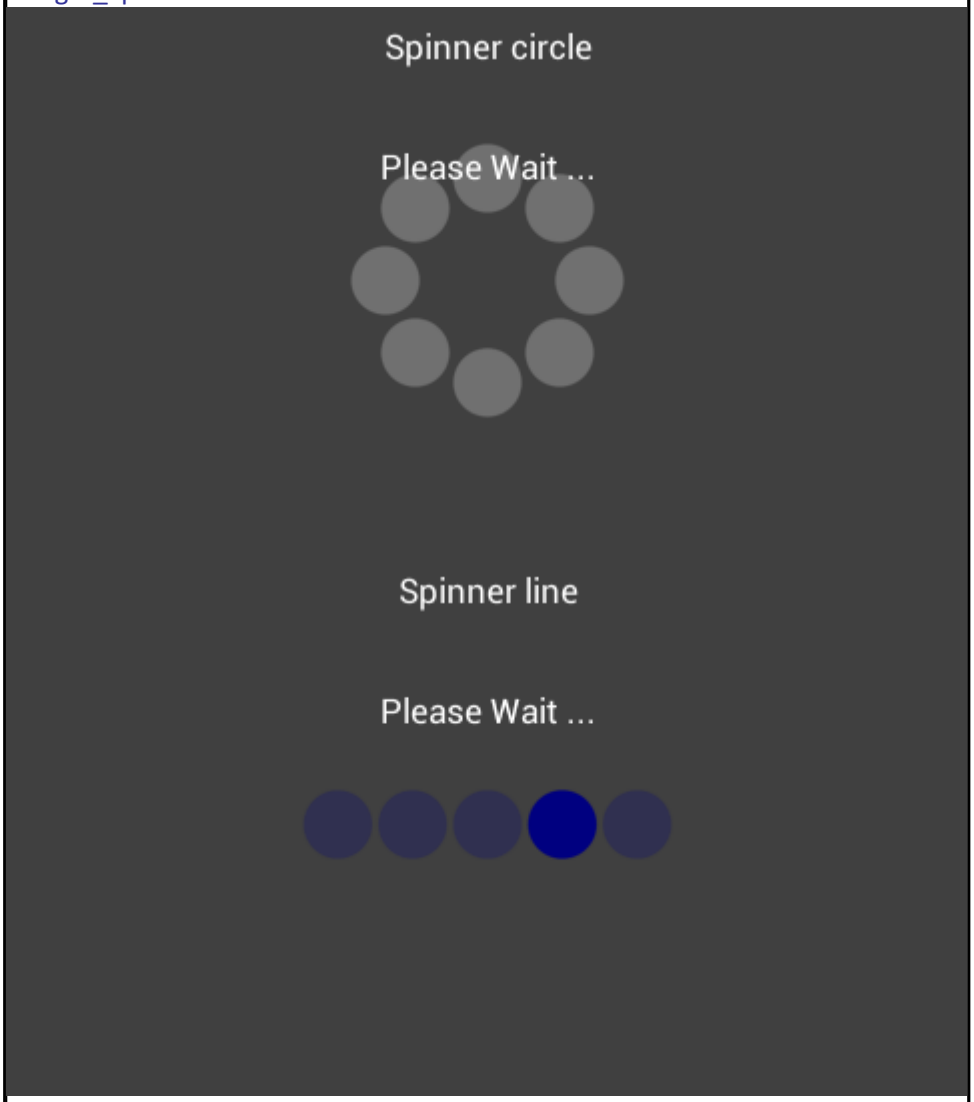
Widget_Dial

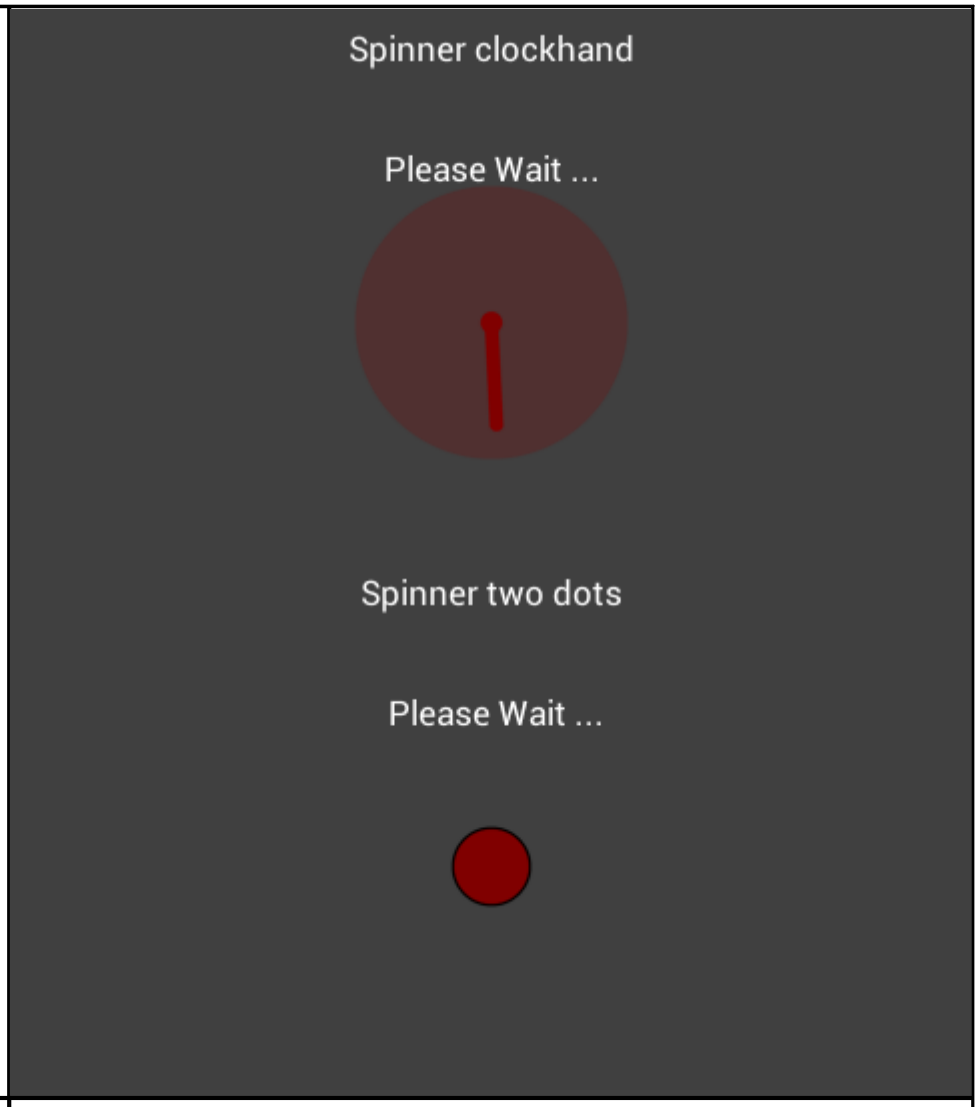


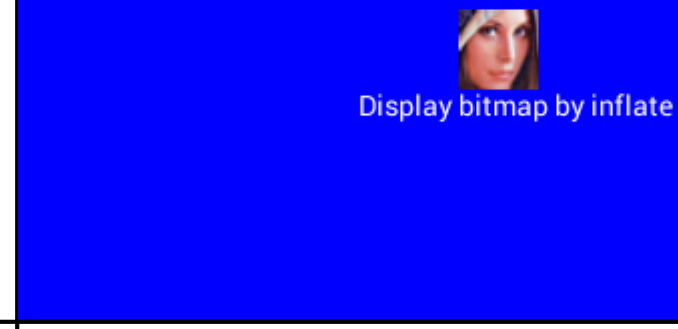
Widget_Toggle



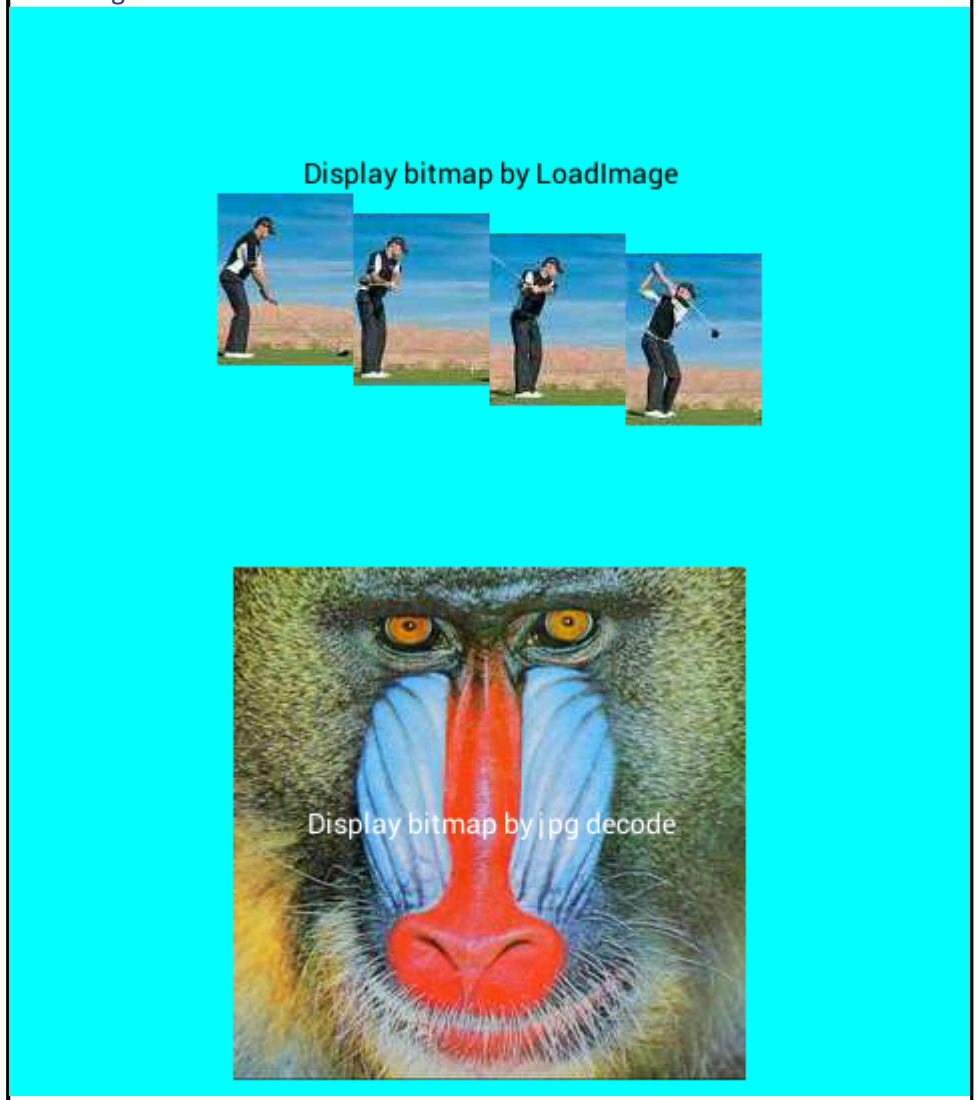
Widget_Spinner

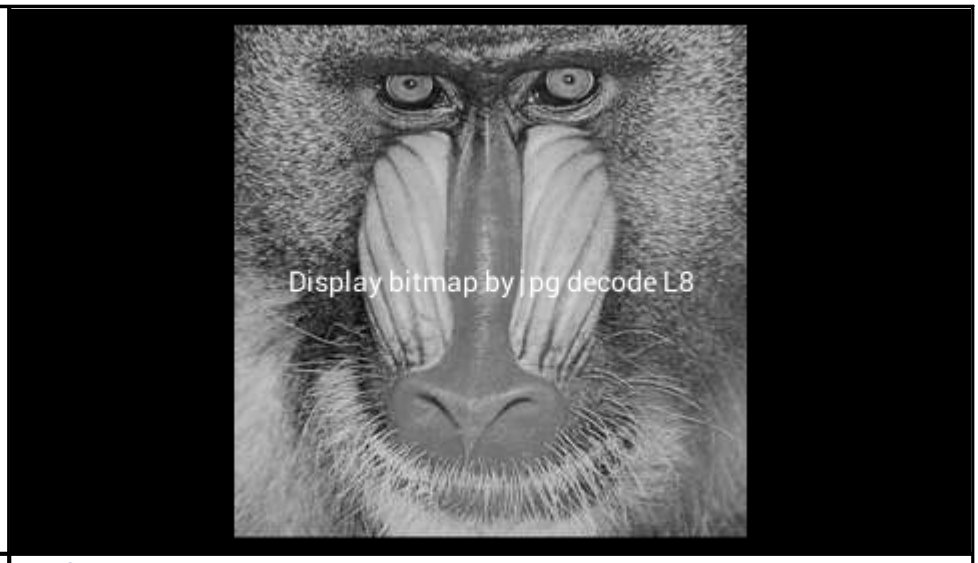


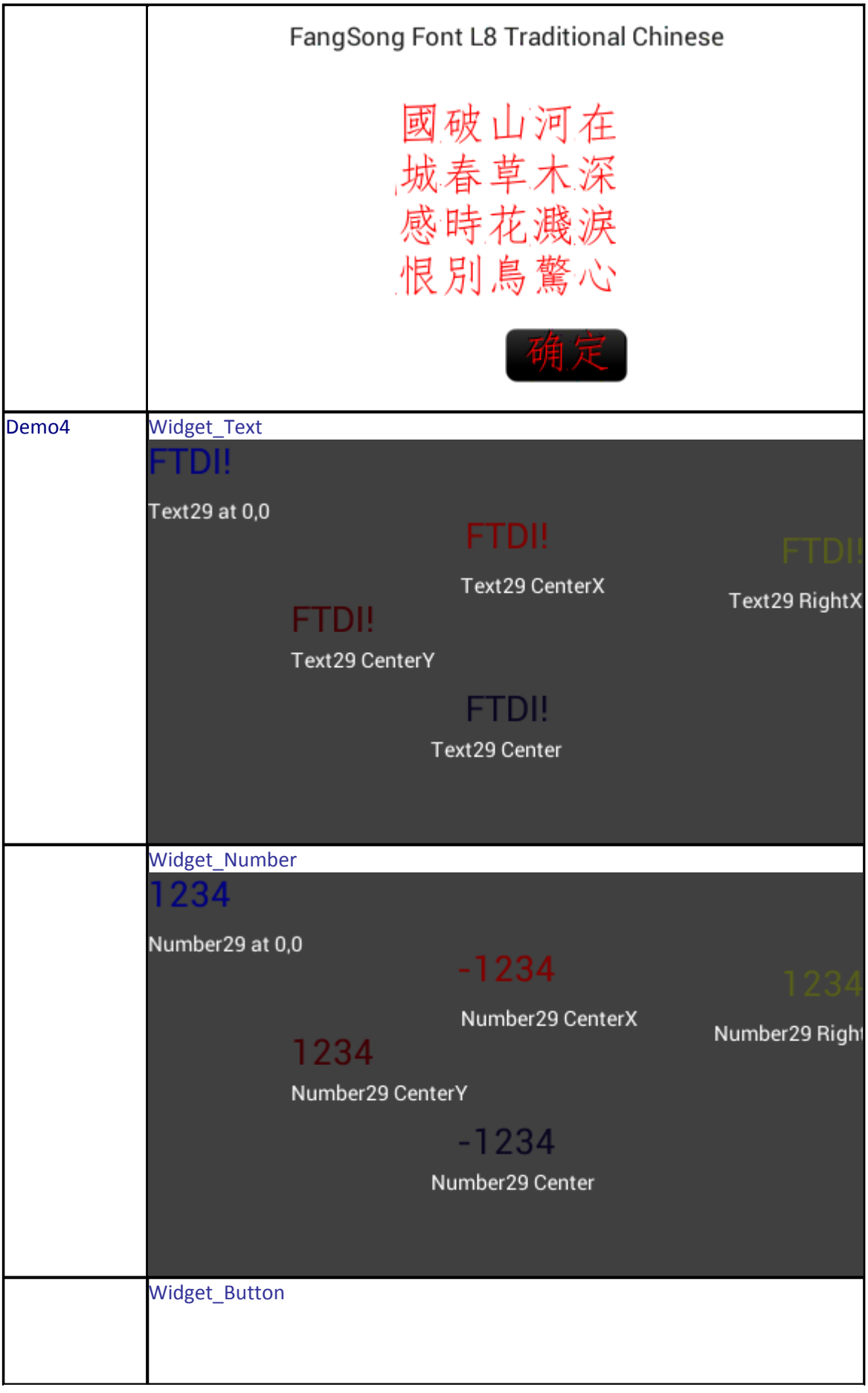
	
	PowerMode Scenario 6
Demo2	Inflate

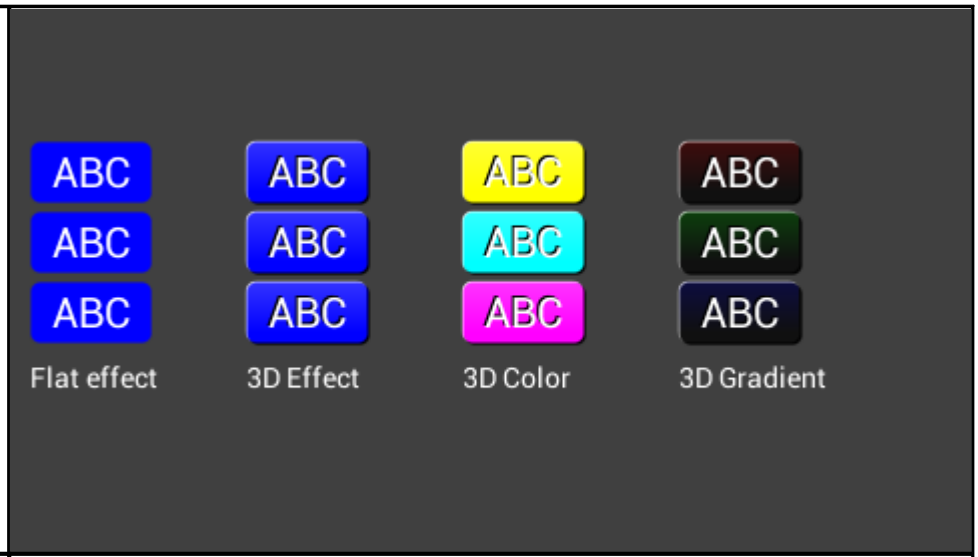


Loadimage

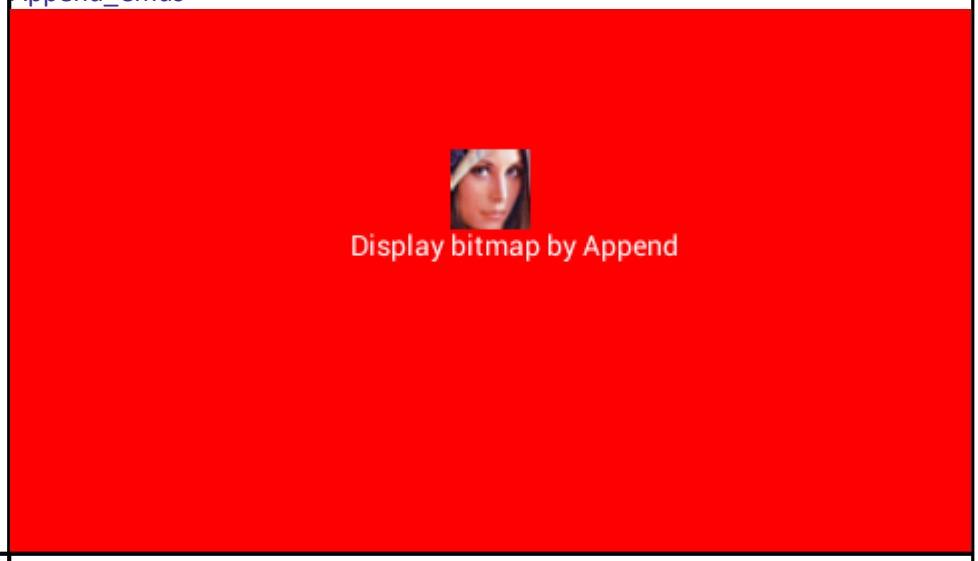


	
Demo3	Set_font SetFont - format L4 **The quick brown fox jumps** **over the lazy dog.** **1234567890**
	Set_font2 SetFont - format L4 (using $Inc) **The quick brown fox jumps** **over the lazy dog.** **1234567890**
	ChineseFont

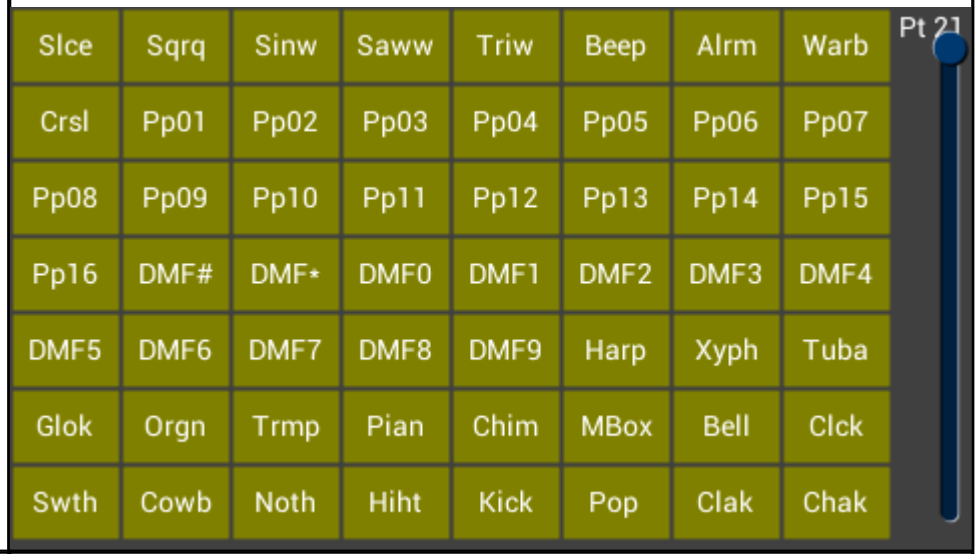




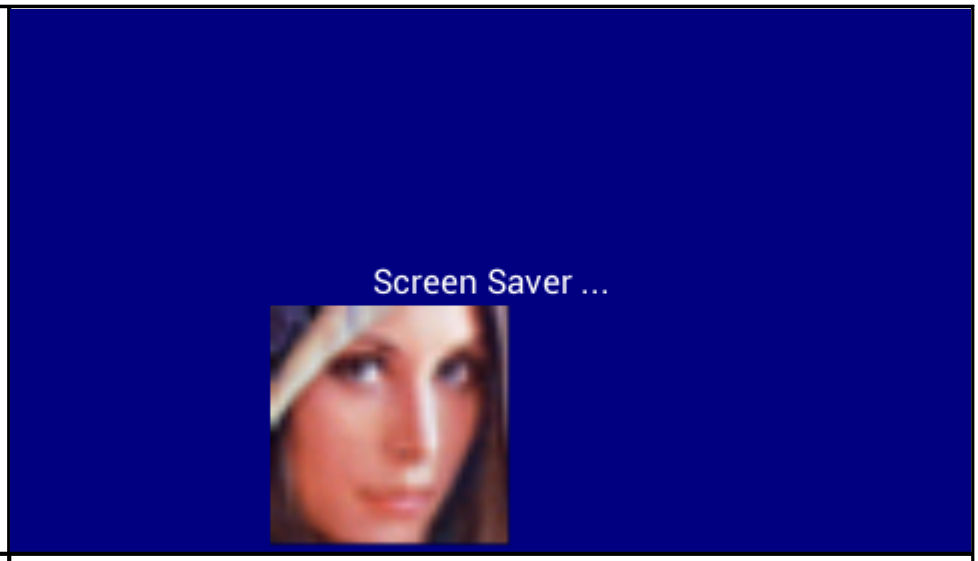
Append_Cmds



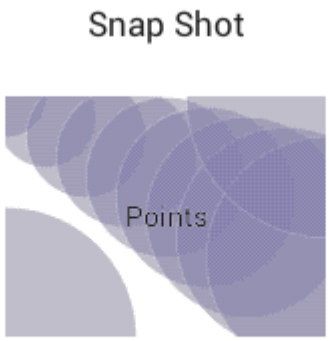
Sounds



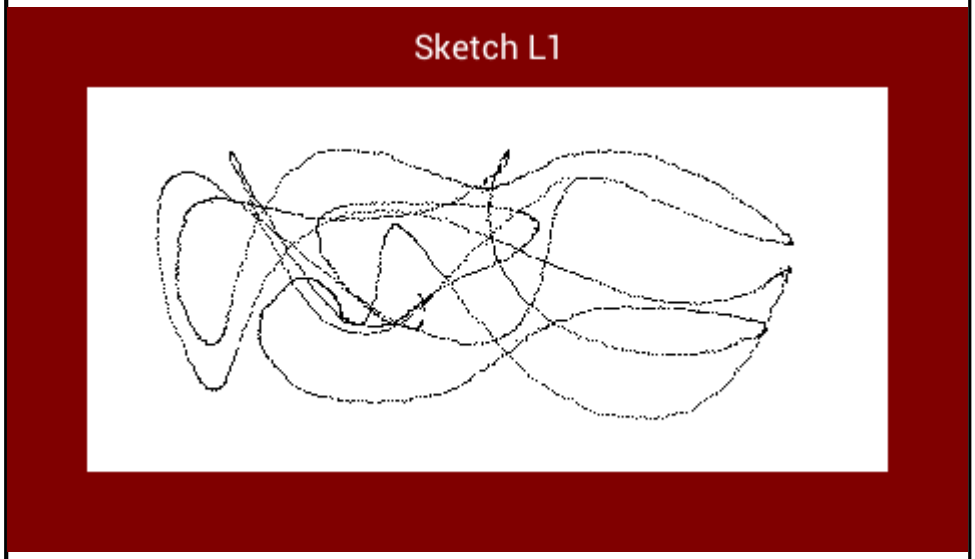
Screensaver

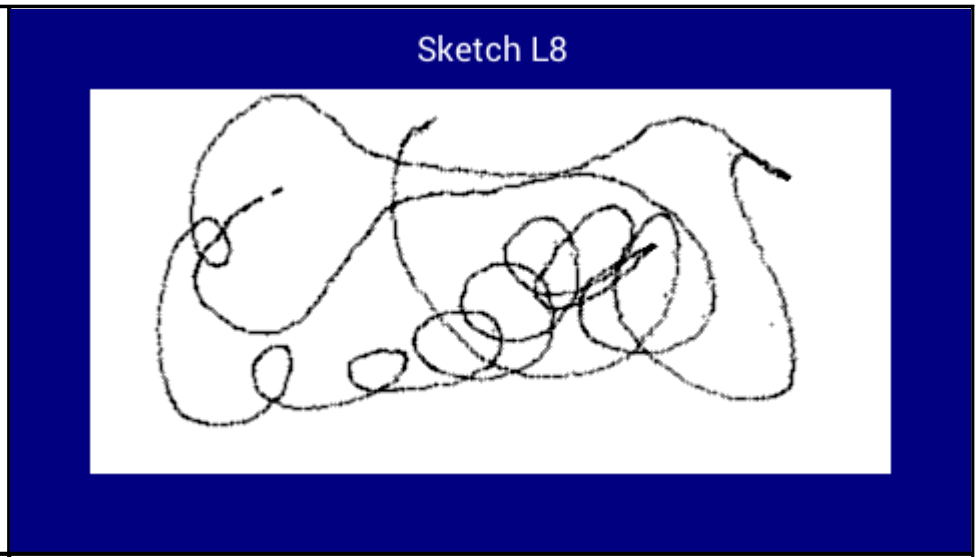


Snapshot

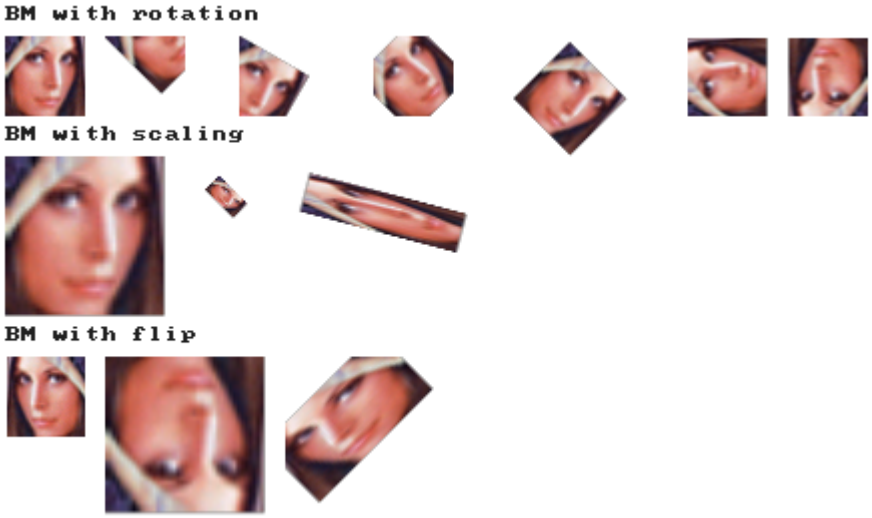


Sketch

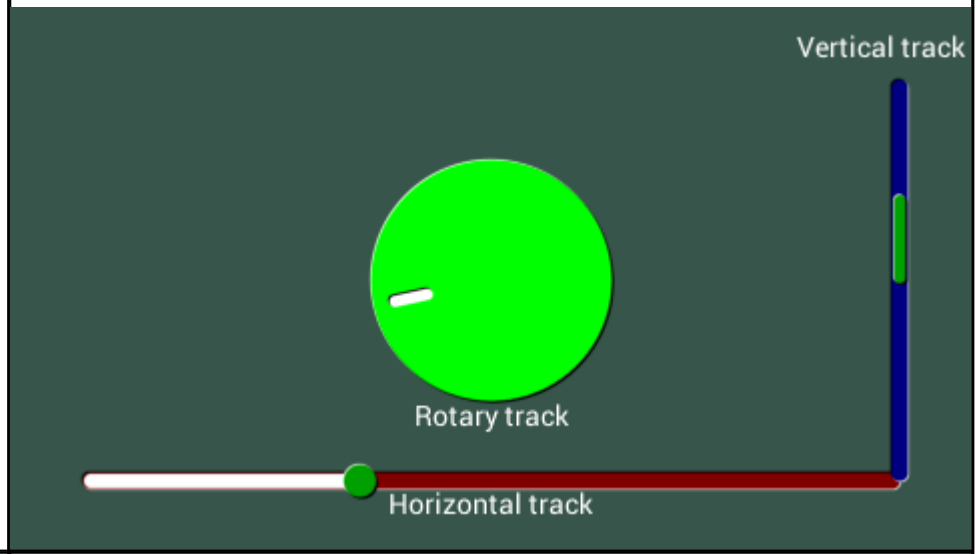




Matrix

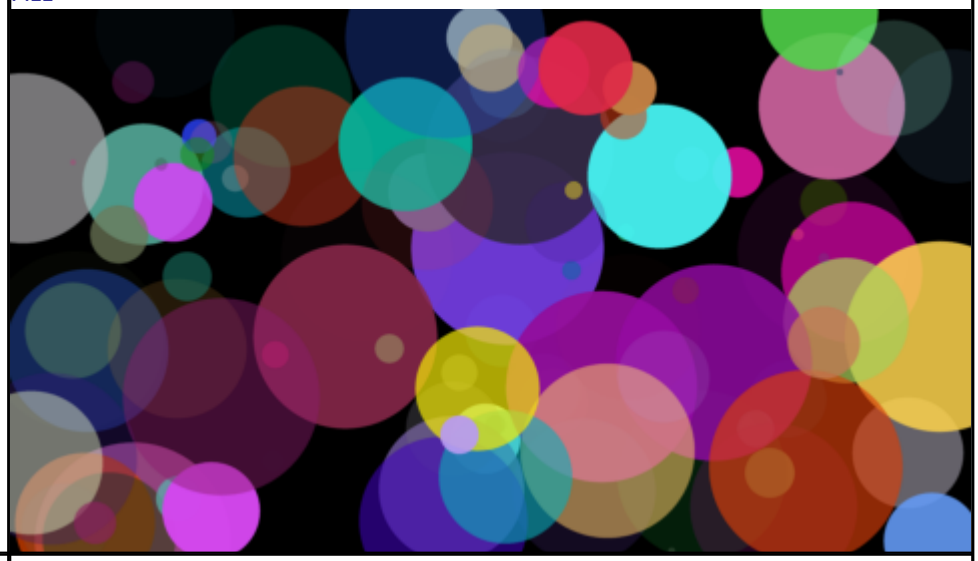
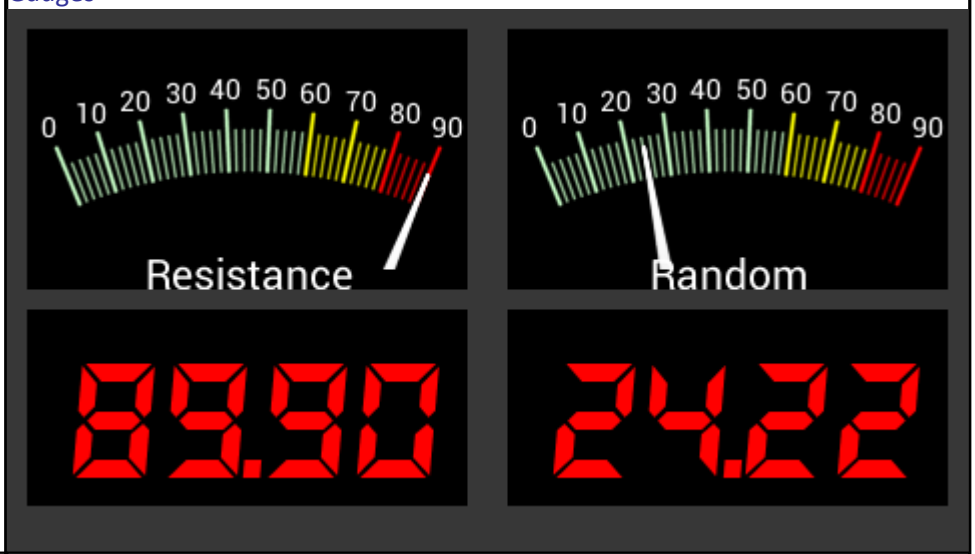


Track

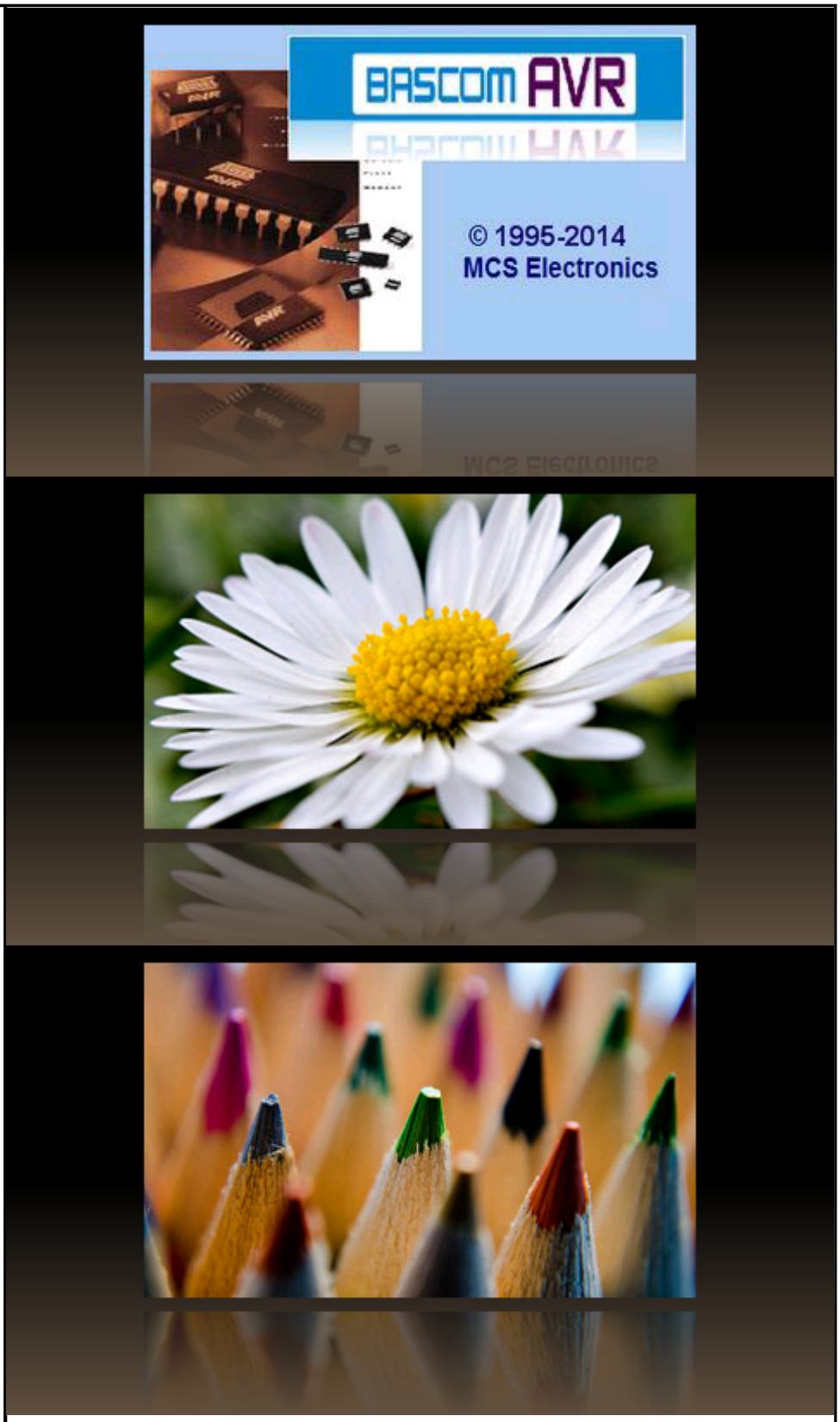


DigitTest

Digit

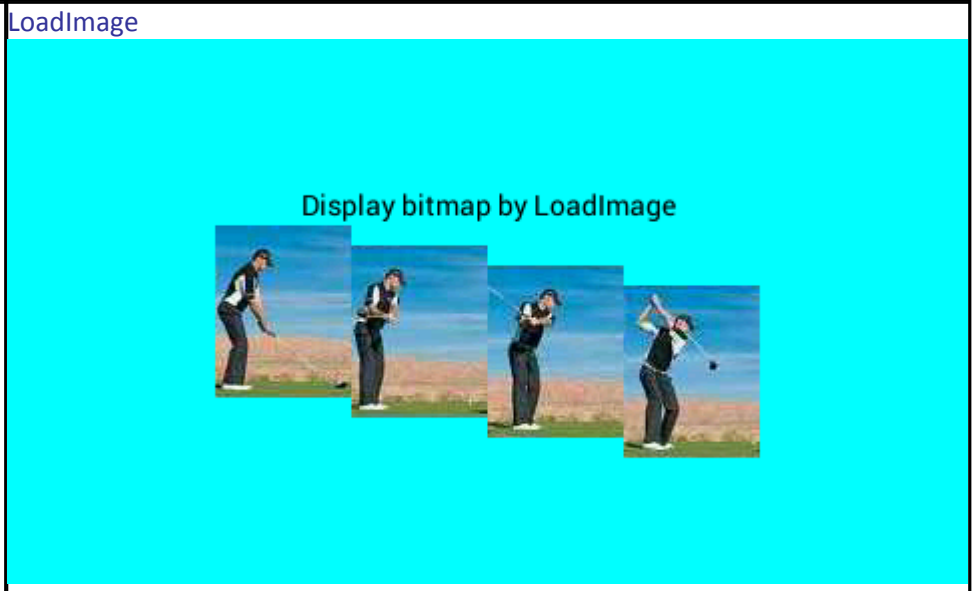
	
FT800 Capture	ScreenShot2
FT800 Demo_Fizz	Fizz 
FT800 Gauges	Gauges 
FT800	Imageviewer

ImageViewer

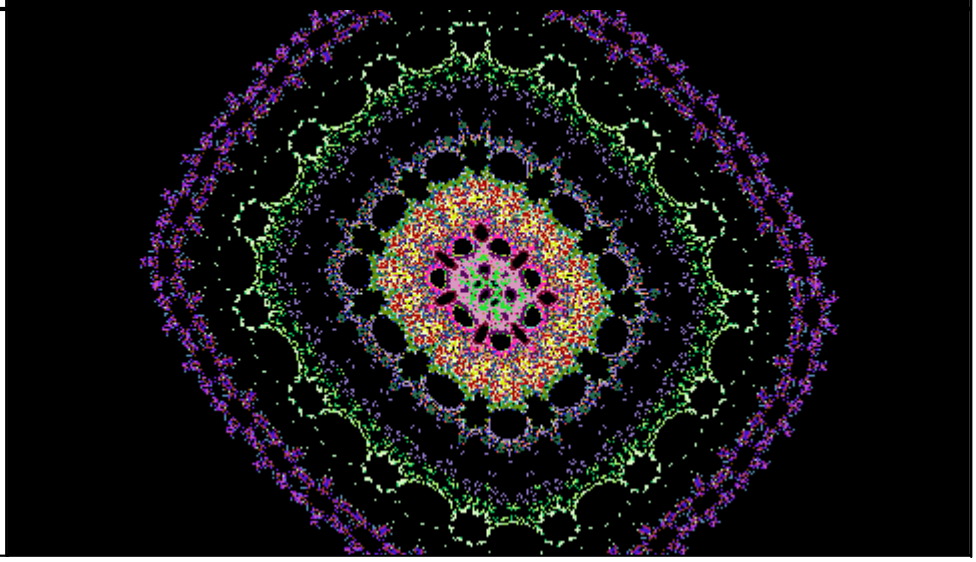


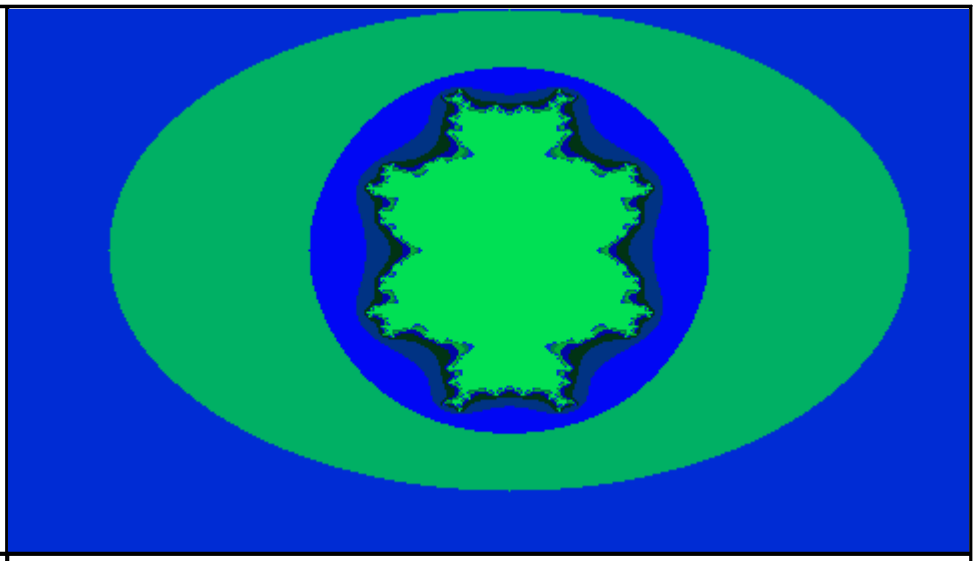
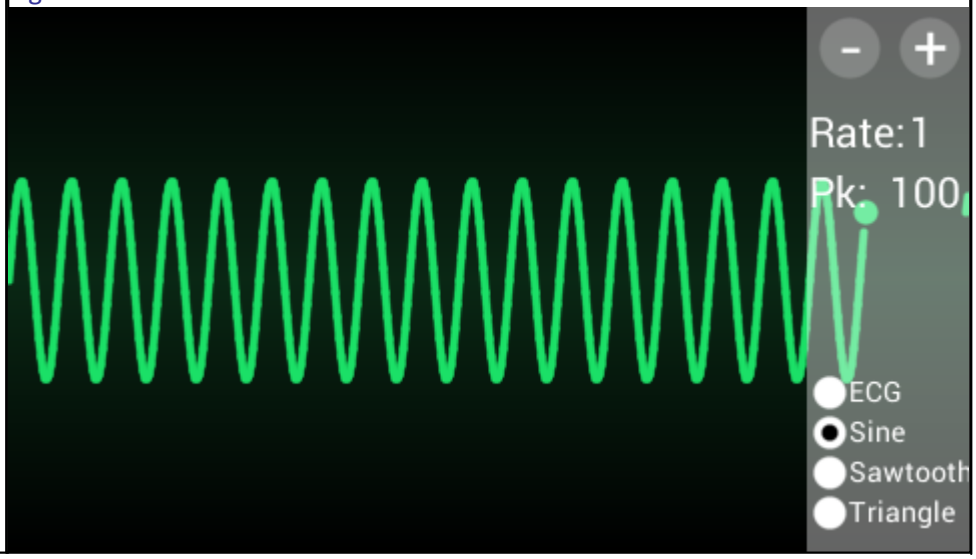
FT800 Keyboard

FT800 Loadimage



FT800
Mandelbrot1

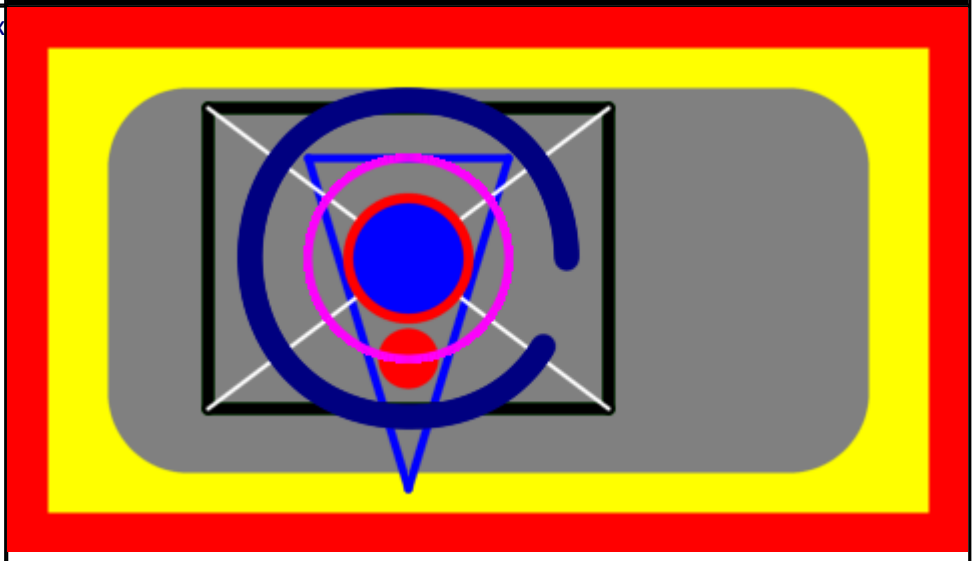


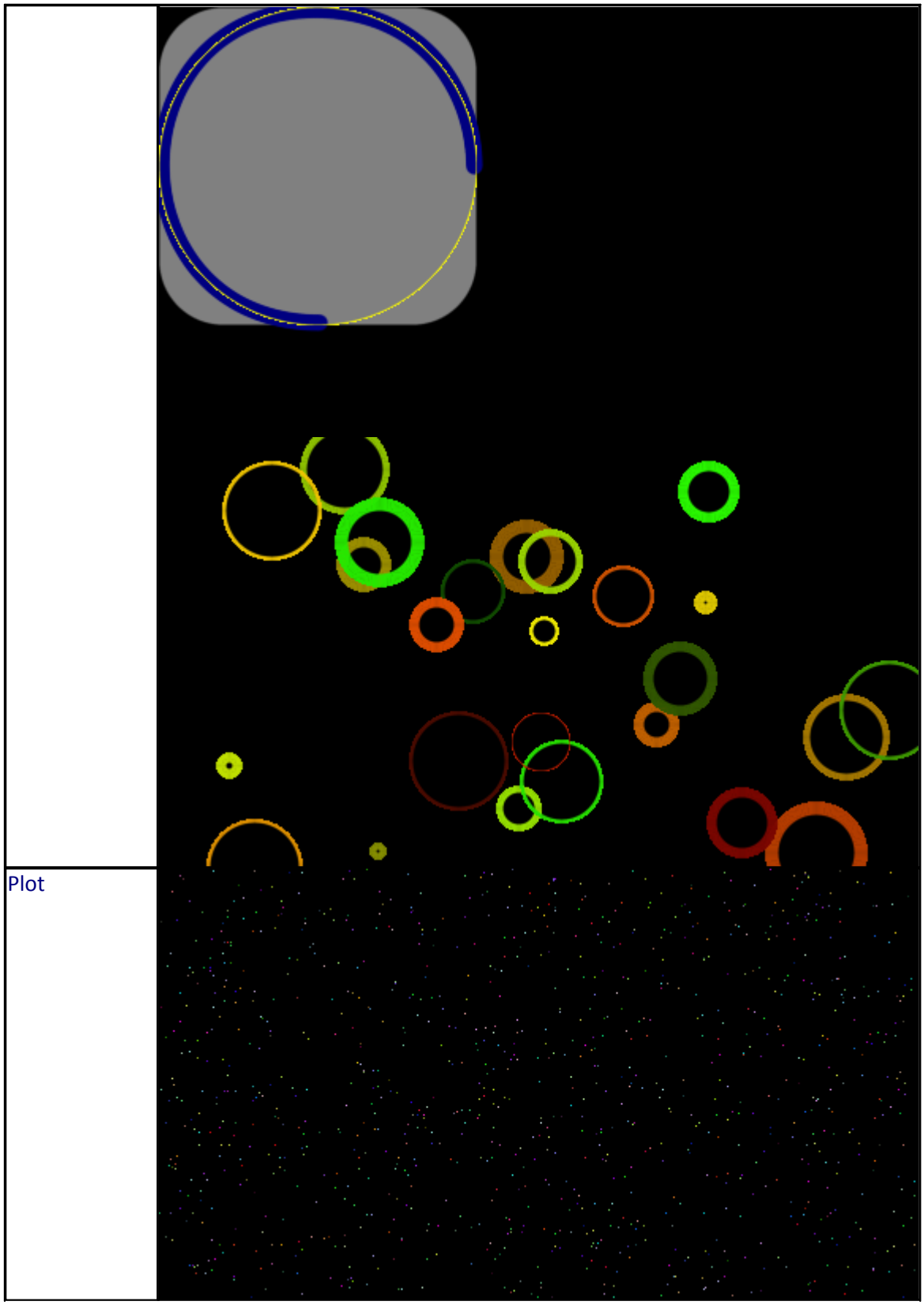
FT800 Mandelbrot2	
Ft800 Player	Player 
FT800 Signals	Signals 

FT800 Sketch

FT800 Sprites

Line_Circle_Box





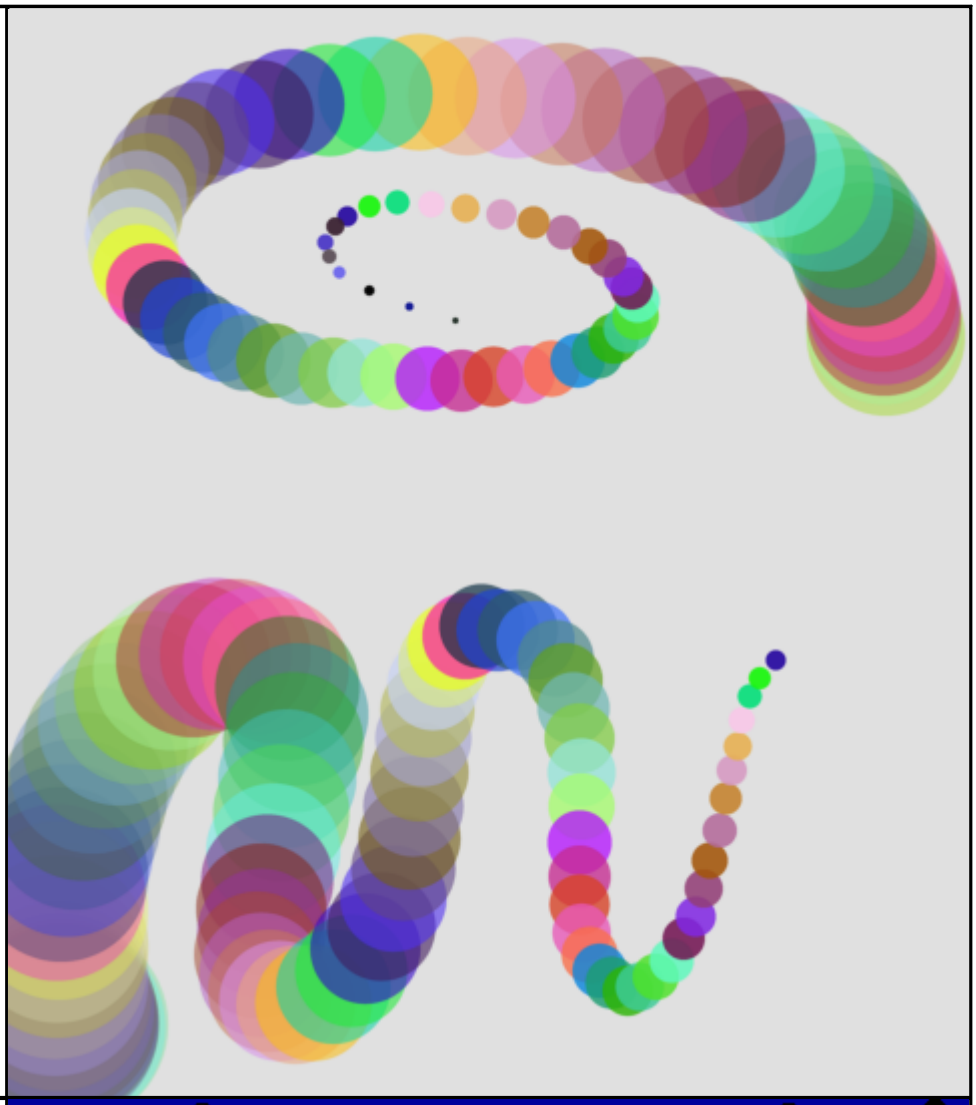
Plot

Test Clock

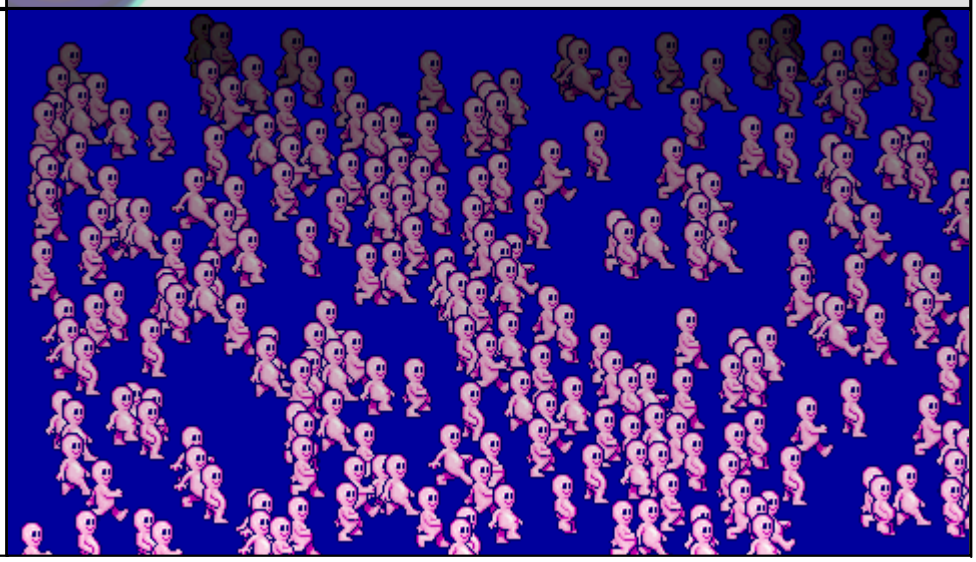
FT800 RadioButton

Test Clock 2

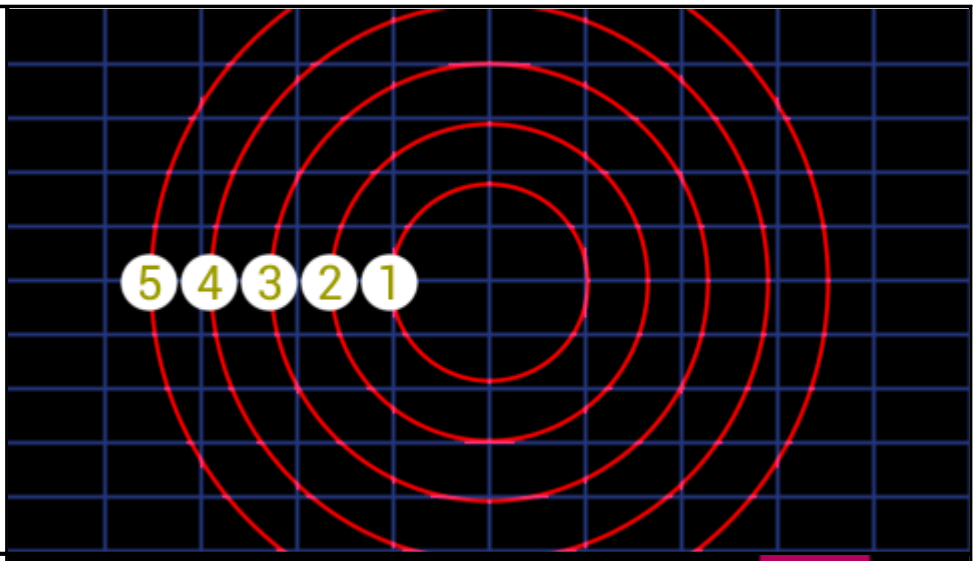
FT800 Blobs



FT800 Walk

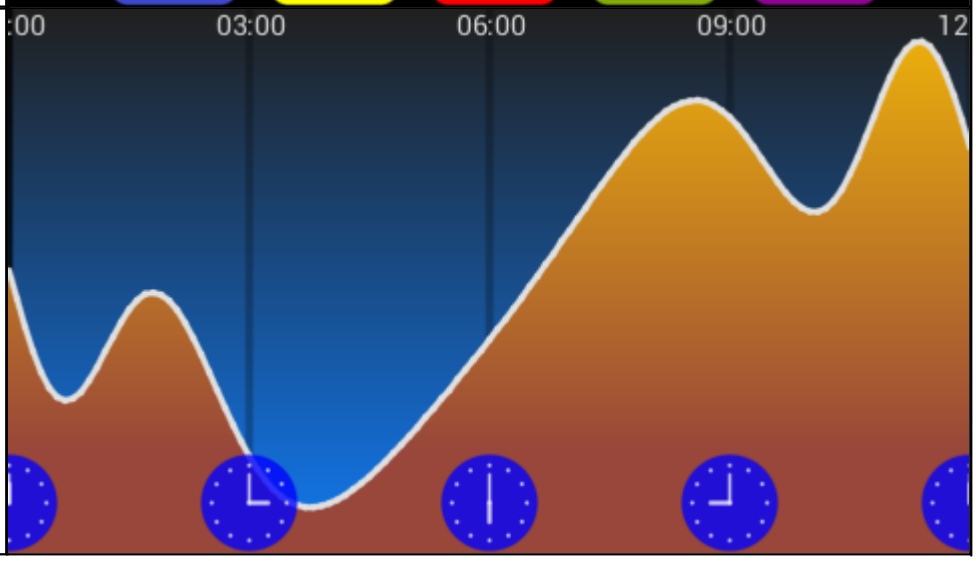


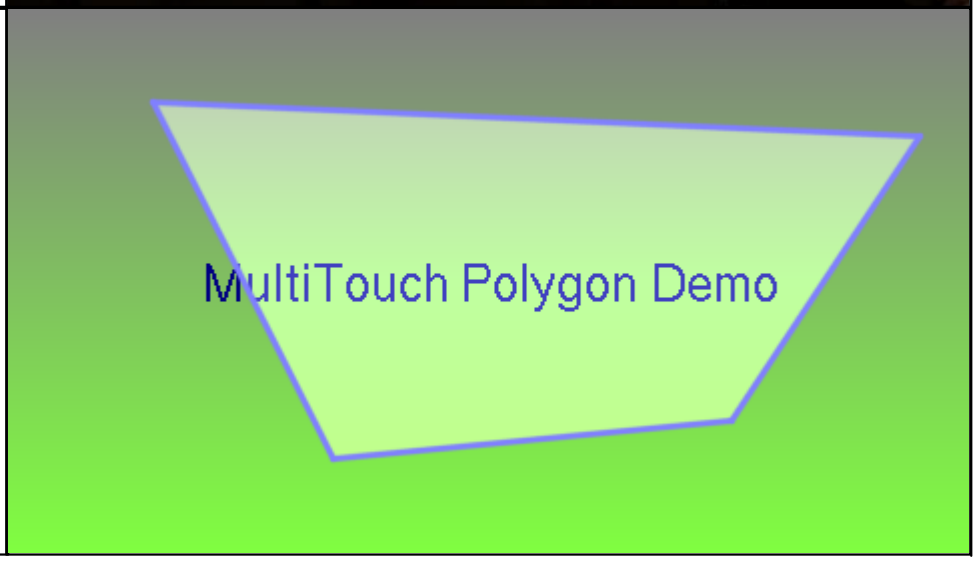
FT801 Circles.bas



FT801 Bars.bas

FT801 Graph - Capacitive Touch.bas



FT801 Polygon - Capacitive Touch.bas	
FT801 Polygon2 - Capacitive Touch.bas	

8.2 EXTENDED I2C

Action
Instruct the compiler to use parts of the extended i2c library

Syntax
$LIB = "i2c_extended.lib"

Remarks
The I2C library was written when the AVR architecture did not have extended registers. The designers of the AVR chips did not preserve enough space for registers. So when they made bigger chips with more ports they ran out of registers.
They solved it to use space from the RAM memory and move the RAM memory from &H60 to &H100.
In the free space from &60 to &H100 the new extended register were located.

While this is a practical solution, some ASM instructions could not be used anymore. This made it a problem to use the I2C statements on PORTF and PORTG of the

Mega128.
The extended i2c library is intended to use I2C on portF and portG on the M64 and M128.
It uses a bit more space then the normal I2C lib.

Best would be that you use the TWI interface and the i2c_twi library as this uses less code. The disadvantage is that you need fixed pins as TWI used a fix pin for SCL and SDA.

See also
I2C 1306

ASM
NONE

Example
```
'-----------------------------------------------------------------------
--------
'                            (c) 1995-2025 MCS Electronics
'                  This demo shows an example of I2C on the M128 portF
' PORTF is an extened port and requires a special I2C driver
'-----------------------------------------------------------------------
--------


$regfile = "m128def.dat"                                    ' the used
chip
$crystal = 8000000
$baud = 19200                                               ' baud rate

$lib "i2c_extended.lib"

Config Scl = Portf.0                                        ' we need to
provide the SCL pin name
Config Sda = Portf.1                                        ' we need to
provide the SDA pin name


Dim B1 As Byte , B2 As Byte
Dim W As Word At B1 Overlay



I2cinit                                                     ' we need to
set the pins in the proper state


Dim B As Byte , X As Byte
Print "Mega128 master demo"


Print "Scan start"
For B = 1 To 254 Step 2
  I2cstart
  I2cwbyte B
  If Err = 0 Then
     Print "Slave at : " ; B
  End If
```

```
  I2cstop
Next
Print "End Scan"


Do
  I2cstart
  I2cwbyte &H70                                          ' slave
address write
  I2cwbyte &B10101010                                    ' write
command
  I2cwbyte 2
  I2cstop
  Print Err


  I2cstart
  I2cwbyte &H71
  I2crbyte B1 , Ack
  I2crbyte B2 , Nack
  I2cstop
  Print "Error : " ; Err                                 ' show error
  Waitms 500                                             'wait a bit
Loop
End
```

8.3 FM24C16

The FM24C16 library is a library that uses a RAMTRON I2C serial EEPROM.
Ramtron memory chips are as quick as RAM and can be overwritten almost unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM.

By using : $lib "fm24c16.lib"
The EEPROM read and write routines from the library will be used instead of the internal EEPROM.
Thus you can still use : Dim BE as ERAM Byte
And you can use READEEPROM and WRITEEEPROM, but instead of using the internal EEPROM, the external I2C EEPROM is used.
The lib is for the FM24C16. It uses I2C/TWI.


This library is only included in the full version. It is not included with the DEMO.

Example

```
'-------------------------------------------------------------------------
-----------------
'name                   : 24C256 simple RW test.bas
'copyright              : (c) 1995-2025, MCS Electronics
'purpose                : Testing Read/Write operation with external
EEPROM
'micro                  : Mega8535
'suited for demo        : no
'commercial addon needed : no
'-------------------------------------------------------------------------
-----------------

$regfile = "m8535.dat"                                   ' specify
the used micro
```

```
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 64                                               ' default
use 32 for the hardware stack
$swstack = 20                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

$lib "i2c_twi.lbx"                                          ' we do not
use software emulated I2C but the TWI

Config Scl = Portc.0                                        ' we need to
provide the SCL pin name
Config Sda = Portc.1                                        ' we need to
provide the SDA pin name

I2cinit                                                     ' we need to
set the pins in the proper state

Config Twi = 100000                                         ' wanted
clock frequency

' External EEPROM Config
$eepromsize = &H8000
$lib "fm24c64_256.lib"

Dim A(101) As Eram Byte
Dim B As Byte
Dim C As Byte
Dim D As Byte

Do
   Input "Data to write ? (0-255)" , D

   Print "Reading content of EEPROM (via ERAM Byte)"
   For C = 0 To 100
      B = A(c)
      Print "Read ";C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 1

   Print "Writing data to EEPROM (via ERAM Byte)"
   For C = 0 To 100
      A(c) = D
      Print "Write " ; C ; ":" ; D ; "/" ; Hex(d)
      Waitms 4
   Next

   Wait 1

   Print "Reading back data from EEPROM (via ERAM Byte)"
   For C = 0 To 100
      B = A(c)
      Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 2
```

```
   Input "Data to write ? (0-255)" , D

   Print "Reading content of EEPROM (via READEEPROM)"
   For C = 0 To 100
      Readeeprom B , C
      Print "Read ";C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 1

   Print "Writing data to EEPROM (via WRITEEEPROM)"
   For C = 0 To 100
      Writeeeprom D , C
      Print "Writing " ; C ; ":" ; D ; "/" ; Hex(d)
      Waitms 4
   Next

   Wait 1

   Print "Reading content of EEPROM (via READEEPROM)"
   For C = 0 To 100
      Readeeprom B , C
      Print "Read ";C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 2

Loop

End
'--------------------------------------------------------------------------
--------
```

8.4 FM24C64_256

The FM24C64_256 library is a library that uses a RAMTRON I2C serial EEPROM. Ramtron memory chips are as quick as RAM and can be overwritten almost unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM.

By using : $lib "fm24c64_256.lib"
The EEPROM read and write routines from the library will be used instead of the internal EEPROM.
Thus you can still use : Dim BE as ERAM Byte
And you can use READEEPROM and WRITEEEPROM, but instead of using the internal EEPROM, the external I2C EEPROM is used.
The lib is for the FM24C64 to FM24C256. It uses I2C/TWI.
It was also tested with the M24512 an 1 MBit EEPROM.

While intended for RAMTRON memory it will work with most normal I2C EEPROM as well.
You should test it since some normal EEPROM might require a certain delay after the write.

 This library is only included in the full version. It is not included with the DEMO.

For an example see FM24C16 1743

8.5 FM24C64_256-XMEGA

FM24C64_256-XMEGA is the XMEGA version of the FM24C64_256 1745 library.

This library is a library that uses a RAMTRON I2C serial EEPROM.
Ramtron memory chips are as quick as RAM and can be overwritten almost unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM.

By using : $lib "FM24C64_256-XMEGA.lib"
The EEPROM read and write routines from the library will be used instead of the internal EEPROM.
Thus you can still use : Dim BE as ERAM Byte
And you can use READEEPROM and WRITEEEPROM, but instead of using the internal EEPROM, the external I2C EEPROM is used.

Since Xmega has up to 4 different TWI channels, you need to define which channel is used.
You need to do so by defining a constant in your code named **cFRAM_CHANNEL** and give it a value of 1 for TWIC, 2 for TWID, 4 for TWIE or 8 for TWIF.

 This library is only included in the full version. It is not included with the DEMO.

In version 2086 you can also read write strings.

Example

```
'(

The fm24c64_256-XMEGA library is a library that uses a RAMTRON I2C
serial EEPROM.
Ramtron memory chips are as quick as RAM and can be overwritten almost
unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM.

By using : $lib "fm24c64_256-xmega.lib"
The EEPROM read and write routines from the library will be used
instead of the internal EEPROM.
Thus you can still use : Dim BE as ERAM Byte
And you can use READEEPROM and WRITEEEPROM, but instead of using the
internal EEPROM, the external I2C EEPROM is used.
The lib is for the FM24C515. It uses I2C / TWI.

You must define a constant in your code with a constant that defines
the twi interface :
CONST cfram_channel = 1 'twic
CONST cfram_channel = 2 'twid
CONST cfram_channel = 4                                    'twie
```

```
CONST cfram_channel = 8 'twif

This library is only included in the full version. It is not included
with the DEMO.
This library is especial for XMEGA and serves as a sample.
reading/writing strings is NOT supported but can be added by the user


')
'-----------------------------------------------------------------------
------------------
'name                     : 24C512-xmega-simple-RW test-TWIE.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : Testing Read/Write operation with external
EEPROM on TWIE
'micro                    : xmega128A1
'suited for demo          : no
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "xM128a1def.dat"
$crystal = 32000000                                         ' 32MHz
$hwstack = 128
$swstack = 128
$framesize = 128

Config BASE = 0                                             ' arrays
start at 0

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
'for debug we send some data to the UART
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8

Config Twie = 100000                                        ' CONFIG
TWI will ENABLE the TWI master interface
const cfram_channel = 4                                     ' this
constant is required by the fm24c64_256-xmega lib
                                                            ' set it to
1 for TWIC , 2 for TWID , 4 for TWIE and 8 for TWIF
Open "twie" For Binary As #4                                ' when not
using default twic, you must use a channel

const _twi_stop_1 = 1                                       ' just test
i2cstop option, see help

Dim Twi_start As Byte                                       ' always
required for xmega i2c
I2Cinit #4
```

```
$eepromsize = &H400                                          ' set it to
the size of your EEPROM
$lib "fm24c64_256-xmega.lib"                                 ' include
lib


dim ee(100) as eram byte                                     ' dim an
EEPROM array
Dim B , adres As byte

print "Writing EEEPROM"
for adres = 0 to 10
   print adres ; ",";
   ee(adres) = adres
   waitms 20                                                 ' ONLY FOR
NORMAL EEPROM , REMOVE FOR RAMTRON
next
print

print "read EEEPROM"

for adres = 0 to 10
   b = ee(adres)
   print adres;"-";b
next

end
```

8.6 FM25C256

The FM24C256 library is a library that uses a RAMTRON SPI serial EEPROM. Ramtron memory chips are as quick as RAM and can be overwritten almost unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM. You can also increase the size of the EEPROM this way.

By using : $lib "fm25c256.lib"
The EEPROM read and write routines from the library will be used instead of the internal EEPROM.
Thus you can still use : Dim BE as ERAM Byte
And you can use READEEPROM and WRITEEEPROM, but instead of using the internal EEPROM, the external I2C EEPROM is used.
The lib is for the FM25C256. It uses SPI

For the SPI you have to define the pins. The pin named fram_so is connected to SO of the FRAM. SI is connected to SI.
A sample is shown below. The clock, cs and SI pins need to be configured as output

pins.

Fram_cs Alias Portl.7 : Const Fram_csp = 7 : Const Fram_csport = Portl
Fram_so Alias Pind.1 : Const Fram_sop = 1 : Const Fram_soport = Pind
Fram_si Alias Portd.0 : Const Fram_sip = 0 : Const Fram_siport = Portd
Fram_sck Alias Portl.6 : Const Fram_sckp = 6 : Const Fram_sckport = Portl

 This library is only included in the full version. It is not included with the DEMO.

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : 25C256 simple RW test.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : Testing Read/Write operation with external
EEPROM
'micro                    : Mega8535
'suited for demo          : no
'commercial addon needed  : no
'-----------------------------------------------------------------------
------------------

$regfile = "m8535.dat"                                      ' specify
the used micro
$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 64                                               ' default
use 32 for the hardware stack
$swstack = 20                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space



' External EEPROM Config
Config Portb.4 = Output
Config Portb.7 = Output
Config Portb.5 = Output
Fram_cs Alias Portb.4 : Const Fram_csp = 4 : Const Fram_csport = Portb
Fram_so Alias Pinb.6 : Const Fram_sop = 6 : Const Fram_soport = Pinb
Fram_si Alias Portb.5 : Const Fram_sip = 5 : Const Fram_siport = Portb
Fram_sck Alias Portb.7 : Const Fram_sckp = 7 : Const Fram_sckport =
Portb

$eepromsize = &H8000
$lib "fm25c256.lib"

Dim A(101) As Eram Byte
Dim B As Byte
Dim C As Byte
Dim D As Byte

Do

   Input "Data to write ? (0-255)" , D
```

```
   Print "Reading content of EEPROM (via ERAM Byte)"
   For C = 0 To 100
      B = A(c)
      Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 1

   Print "Writing data to EEPROM (via ERAM Byte)"
   For C = 0 To 100
      A(c) = D
      Print "Write " ; C ; ":" ; D ; "/" ; Hex(d)
      Waitms 4
   Next

   Wait 1

   Print "Reading back data from EEPROM (via ERAM Byte)"
   For C = 0 To 100
      B = A(c)
      Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 2

   Input "Data to write ? (0-255)" , D

   Print "Reading content of EEPROM (via READEEPROM)"
   For C = 0 To 100
      Readeeprom B , C
      Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 1

   Print "Writing data to EEPROM (via WRITEEEPROM)"
   For C = 0 To 100
      Writeeeprom D , C
      Print "Writing " ; C ; ":" ; D ; "/" ; Hex(d)
      Waitms 4
   Next

   Wait 1

   Print "Reading content of EEPROM (via READEEPROM)"
   For C = 0 To 100
      Readeeprom B , C
      Print "Read " ; C ; ":" ; B ; "/" ; Hex(b)
      Waitms 4
   Next

   Wait 2


Loop

End
'-------------------------------------------------------------------------
--------
```

Example 2, shared bus

```
'                              Using the FM25C256 library

'  The FM25C256 library uses the CYPRESS FM25W256  chip (before named FM25C256 by Ramtron)
'  This chip is based in FRAM technology, which makes it much faster than an EEPROM  and has a much
'  longer life (100.000.000.000.000 read/writes)
'  To give an idea of speed, writting a byte to an XMEGA192A3 internal EEPROM takes more than 10580us
'  while writing a byte  to the FM25W256 chip using the FM25C256 library  takes 32,5us in this example;
'  this is more than 325 times  faster.

' NOTES:
'     -  This library allows you to use an external EEPROM INSTEAD of the internal EEPROM (you cannot use both)
'     -  Do not use the "Config Eeprom = " command when using this library
'     -  The FM25C256 library uses software SPI; therefore,  if you need to share the SPI bus with another chip
'        that uses  HW SPI, you must:
'           - Configure the HW SPI normally  (with the "Config SpiX =" command in XMEGA chips)  as needed for
'             the other chip
'           - Disable HW SPI before reading or writing to EEPROM, and enable it after.

' In this example, there are two chips connected to the SPIC bus of an XMEGA192A3, an accelerometer BMA180
' and the FM25W256 FRAM chip.

' The HW SPIC of the XMEGA192A3  is configured at the beginning to allow for the BMA180 to be read while the
'  FM25W256 is not used.
'-------------------------------------------------------------------------------

$regfile = "xm192a3def.dat"
$hwstack = 256
$swstack = 256
$framesize = 256
'-------------------------------------------------------------------------------

'   For  16MHz crystal
Config Osc = Disabled , Extosc = Enabled , Range = 12mhz_16mhz , Startup = Xtal_1kclk , 32khzosc = Enabled
' Set PLL OSC conditions:
Osc_pllctrl = &B1100_0010                       ' Reference external oscillator, set the PLL' multiplication factor to 2 (bits
0 - 4)
Set Osc_ctrl.4                                  ' Enable PLL Oscillator
Bitwait Osc_status.4 , Set                        ' wait until the pll clock reference source is stable
Clk_ctrl = &B0000_0100                           ' switch system clock to pll
' Prescale
Config Sysclock = Pll , Prescalea = 1 , Prescalebc = 1_1
$crystal = 32000000
'-------------------------
Const Fclock = 32000000
'-------------------------------------------------------------------------------

'Config Eeprom = Mapped                         ' Do not put this command when using an external EEPROM
 '-------------------------------------------------------------------------------

 Config Priority = Static , Vector = Application , Lo = Enabled , Med = Enabled , Hi = Enabled
 Enable Interrupts

'================================ COM1 (C2 C3)  C0  ================================

' COM1       RS232_1
Config Com1 = 230400 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8
Config Serialin = Buffered , Size = 254
Config Serialout = Buffered , Size = 254
Open "COM1:" For Binary As #1

'==================================    SPIC for FRAM    ================================

' External EEPROM Config
Fram_cs Alias Porta.7 : Const Fram_csp = 7 : Const Fram_csport = Porta : Config Porta.7 = Output
Fram_si Alias Portc.5 : Const Fram_sip = 5 : Const Fram_siport = Portc : Config Portc.5 = Output
Fram_sck Alias Portc.7 : Const Fram_sckp = 7 : Const Fram_sckport = Portc : Config Portc.7 = Output
Fram_so Alias Port.6 : Const Fram_sop = 6 : Const Fram_soport = Pinc

$eepromsize = &H8000                            ' Size, in bytes, of the FM25W256 memory
'-------------------------------------------------------------------------------

$lib "fm25c256.lib"                             '
'NOTE:
'While using the lib, the hardware SPI should be disabled. you can do this by writing to the SPIx_CTRL register
```

```
' SPIC_CTRL.6=0   'disable SPI
'Then use the eeprom commands, and re-enable the SPI after that : SPIC_CTRL.6=1
'Also notice that clock level must be low at entrance for FM25W256
'    Fram_sck = 0                                ' Need to put this before accesing the chip
     'eprom commands here
 ' Before re-enable hw spi, set clock pin to high, and enabe with spic_ctrl.6=1
'-------------------------------------------------------------------------------

 ' Configure HW SPIC to use a  BMA180
   Config Spic = Hard , Master = Yes , Mode = 3 , Clockdiv = Clk8 , Data_order = Msb , Ss = None
'  Open device
   Open "SPIC" For Binary As #10

    Bma_ss Alias Portc.4 : Config Portc.4 = Output : Bma_ss = 1       ' /SS del bma180

'-------------------------------------------------------------------------------

Dim Dwtemp_ee As Eram Dword
Dim Dwtemp As Dword
Dim N As Byte
N = 0
Dim I As Byte

 Dim Acel_x As Integer
'-------------------------------------------------------------------------------

Do
' ------------------------
   Incr N
' ------------------------
' Disable HW SPi before writing to EEPROM  FM25W256
   Spic_ctrl.6 = 0
   Fram_sck = 0                                ' Clock level must be low at entrance for fm25256
' Write to EEPROM FM25W256
   Dwtemp = N                                  ' Convert Byte to Dword. When writing to EEPROM variables must be of the
same type
   Dwtemp_ee = Dwtemp                              ' This takes 51,1us
' Read from EEPROM FM25W256
   Dwtemp = Dwtemp_ee                              ' This takes 42,2us
' Enable  HW SPI. It must be done with SCK high
   Fram_sck = 1
   Spic_ctrl.6 = 1                            ' Enable HW SPI
' ------------------------
'  Show  value stored and then retrieved from EEPROM
   Print #1 , N ; ":" ; Dwtemp ; "  ";
' ------------------------
   Gosub Read_bma_x
   Print #1 , Acel_x ; "mG"
' ------------------------
   Waitms 500
' ------------------------
Loop

'-------------------------------------------------------------------------------

'                              READ THE BMA180 X AXIS ACCELERATION
'-------------------------------------------------------------------------------

 Dim Bma_adr_byte As Byte
 Dim Spi_byte As Byte
 Dim Msb_itemp As Integer
 Dim Lsb_itemp As Integer
' Dim Aceleracion_tmp As Integer
 Const Acc_x_msb = &H3
 Const Acc_x_lsb = &H2
 '-------------------------------------------------------------------------------

Read_bma_x:
   '_____________________________ Read  Acel_X_LSB
     Bma_ss = 0
     Bma_adr_byte = Acc_x_lsb                      ' X_LSB
     Bma_adr_byte.7 = 1                          ' Read command
      Print #10 , Bma_adr_byte                     ' Send address
      Input #10 , Spi_byte                      ' Read  spibyte= | d5 d4 d3 d2  d1  d0 | 0 | 1  |
     Bma_ss = 1                                 ' De-select BMA 180
     Shift Spi_byte , Right , 2
     Lsb_itemp = Spi_byte
```

```
'_____________________________ Read Acel_X_MSB
   Bma_ss = 0
   Bma_adr_byte = Acc_x_msb                    ' X_MSB
   Bma_adr_byte.7 = 1                        ' Read command
   Print #10 , Bma_adr_byte                   ' Send address
   Input #10 , Spi_byte                     ' Read spibyte= |d13 d12  d11 d10 d9  d8 d7 d6 |
   Bma_ss = 1                              ' De-select BMA180
   Msb_itemp = Spi_byte
   Shift Msb_itemp , Left , 6
   Lsb_itemp = Lsb_itemp Or Msb_itemp
   Lsb_itemp.14 = Spi_byte.7
   Lsb_itemp.15 = Spi_byte.7

   Acel_x = Lsb_itemp

Return
'_____________________________________________________________________________

End
```

8.7 HEXVAL

The HEXVAL library contains an enhanced version of the HEXVAL code. The library was made by MWS.
The default HEXVAL function does not ignore spaces. The routine from the hexval.lib does ignore spaces.
It will also set the ERR flag to 1 if invalid characters are found. Valid characters are 0-9, A-F,a-f

Usage : $lib "hexval.lbx"

8.8 I2C_MULTIBUS

While XMEGA supports multiple TWI busses, the normal AVR only supports on TWI or no I2C bus. The I2C_MULTIBUS library supports up to 16 I2C busses.
See CONFIG I2CBUS 974

8.9 I2C_TWI

I2C Software vs. Hardware Routines

By default BASCOM will use software routines when you use I2C statements. This because when the first AVR chips were introduced, there was no TWI yet. Atmel named it TWI because Philips is the inventor of I2C. But TWI is the same as I2C. This I2C/TWI peripheral performs all the tasks in hardware so less code is required. But it is limited to the designated pins for SCL and SDA.

So BASCOM allows you to use I2C on every AVR chip. Most newer AVR chips have build in hardware support for I2C. With the I2C_TWI lib you can use the TWI which has advantages as it require less code.

To force BASCOM to use the TWI, you need to insert the following statement into your code:

$LIB "I2C_TWI.LBX"

You also need to choose the correct SCL and SDA pins with the CONFIG SCL and CONFIG SDA statements.

The TWI will save code but the disadvantage is that you can only use the fixed SCL and SDA pins.

For XMEGA the default is using the hardware TWI. You can force bascom to use the software solution using $FORCESOFTI2C 635

 You should not reference the I2C_TWI in Xmega or Xtiny ! Xmega and Xtiny use a different TWI interface.

See also: Using the I2C protocol 297, CONFIG TWI 1116 , I2CV2 1763

8.10 I2C_TWI-MULTI

The I2C_TWI-MULTI library is intended to be used with normal AVR processors which have 2 or more TWI interfaces.
An example of such a processor is the ATMEGA328**PB**

In order to support multiple busses, this library need to be included using the $LIB directive.
Further you need to create a byte variable named _i2cchannel in your code.

This variable will hold the bus or TWI number.
By default it will be 0 and thus the usual TWI hardware will be used : Portc.5 and Portc.4

By setting the variable to 1, the second TWI hardware will be used : Porte.0 and Porte.1

Further you need to use CONFIG TWI1 instead of CONFIG TWI in order to specify the clock rate for the second TWI : Config Twi1 = 100000

All other code will remain compatible.

Example

```
'-----------------------------------------------------------------------
----------
'name                     : m328pb.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates M328pb
'micro                    : Mega328pb
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
----------
$regfile = "m328pbdef.dat"
$crystal = 8000000
$baud = 19200
$hwstack = 40
$swstack = 40
$framesize = 40

' USART   TX    RX
```

```
'   0     D.1   D.0
'   1     B.3   B.4

' ISP programming
' MOSI-PB3     MISO-PB4   SCK-PB5

' TWI    SDA   SCL
'  0     C.4   C.5
'  1     E.0   E.1

'Configuration

Config Clockdiv = 1                                          'make sure
we get 8 Mhz from internal osc

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

'we have 2 TWI interfaces
Config Scl = Portc.5                                         ' we need
to provide the SCL pin name
Config Sda = Portc.4                                         ' we need
to provide the SDA pin name

Config Sda1 = Porte.0                                        'use this
for the second TWI
Config Scl1 = Porte.1

Config Twi = 100000                                          'speed 100
KHz
Config Twi1 = 100000                                         'speed 100
KHz

'some constants for the signature row
Const Device_signature_byte1 = 0
Const Device_signature_byte2 = 2
Const Device_signature_byte3 = 4

Const Rc_oscillator_calibration = 1

Const Serial_number_byte0 = &H0E
Const Serial_number_byte1 = &H0F
Const Serial_number_byte2 = &H10
Const Serial_number_byte3 = &H11
Const Serial_number_byte4 = &H12
Const Serial_number_byte5 = &H13
Const Serial_number_byte6 = &H14
Const Serial_number_byte7 = &H15
Const Serial_number_byte8 = &H16
Const Serial_number_byte9 = &H17
```

```
$lib "I2C_TWI-MULTI.lib"                                    'important
for using 2 TWI interfaces

Dim _i2cchannel As Byte                                     ' you MUST
dim this variable yourself when using the above lib
Dim B As Byte                                               'just a
used byte

I2cinit                                                     'default
TWI init
I2cinit Twi1                                                'optional
specify TWI1 to init that interface

Open "com2:" For Binary As #2                               'create a
channel to reference the UART

'print the chip ID
Print "ID : " ; Hex(readsig(device_signature_byte1)) ; Hex(readsig(
device_signature_byte2)) ; Hex(readsig(device_signature_byte3))

'all I2C statements will work the same. All you need to do is to set
the _i2cchannel variable to 0 or 1
_i2cchannel = 1                                             'try the
second bus

Print "Scan start"
For B = 0 To 254 Step 2                                     'for all
odd addresses
  I2cstart
  I2cwbyte B                                                'send
address
  If Err = 0 Then                                           'we got an
ack
     Print "Slave at : " ; B ; " hex : " ; Hex(b) ; " bin : " ; Bin(b)
  End If
  I2cstop                                                   'free bus
Next


Do
   Print "COM1"
   Print #2 , "COM2"
   Waitms 1000
Loop
```

8.11 I2C_USI

The I2C_USI library is an alternative I2C master library. It is intended to be used with processors that have an USI interface.
Using the hardware is better since it will use less processor resources.
If a processor has TWI, use the TWI
If a processors has USI, use the USI
If a processor has no hardware I2C, use the default built in software routines.

To use the USI in master mode, use CONFIG USI 1138.

```
'-----------------------------------------------------------------------
'               (c) 1995-2025 MCS Electronics
'                 USI-MASTER.bas
' USI used as TWI master demo
'-----------------------------------------------------------------------

$regfile = "attiny2313.dat"
$crystal = 8000000
$hwstack = 40
$swstack = 16
$framesize = 24
$baud = 19200

config usi = twimaster , mode = fast

dim b as byte

i2cinit

do
  i2cstart
  i2cwbyte &H40                         'send slave WRITE address for PCF8574
  i2cwbyte &B10101010                       'send a pattern
  i2crepstart                          'repeated start

  i2cwbyte &H41                           'send slave READ address
  i2crbyte b , ack                       'read a byte
  i2crbyte b , nack                       'and again
  i2cstop                             'end transaction and free bus

  waitms 100                            'some delay not required only when you print
loop
```

8.12 I2C_USI_SLAVE

The I2C_USI_SLAVE library is a library that ships with the I2C slave add on package. The purpose of the lib is to offer I2C/TWI slave support for processors that have an USI interface.

USI master support is bundled with the commercial version of Bascom. This library is named i2_USI.

The USI has two interrupts. One to detect the START condition and one to detect a counter overflow.
Unfortunately Atmel did not define an interrupt for STOP condition. This means that it is not possible to detect a STOP condition with an interrupt.
You can read the STOP condition bit from the USISR register but no interrupt will fire as for the START condition.
So in practice, the **Twi_stop_received** label will be called just before the I2CSTART is called or when data is clocked by the master.

The following example demonstrates how to receive multiple bytes. It emulates an I2c, EEPROM memory chip.

Example

```
'-----------------------------------------------------------------------
---------------
'                              (c) 1995-2025 MCS Electronics
'       This demo demonstrates the USI I2C slave and emulates an
```

```
EEPROM chip
'          This is part of the I2C Slave library which is a
commercial addon library
'          Not all AVR chips have an USI !!!!
'-------------------------------------------------------------
---------------
' This is a simple sample. the master sends the address of the
slave, the WORD address
'  of the memory location, and a byte to store or read
'-------------------------------------------------------------
--------------
' The matching master code to write
'   i2cstart : i2cwbyte &H40 : i2cwbyte low(address) : i2cwbyte
high(address) : i2cwbyte value : i2cstop
' The mathing master code to read
'   i2cstart : i2cwbyte &H40 : i2cwbyte low(address) : i2cwbyte
high(address) : i2crepstart : i2cwbyte &H41 : i2cRbyte value,
nack : i2cstop
'See also the eeprom_master.bas

$regfile = "attiny2313.dat"
'$regfile = "attiny85.dat"
$crystal = 8000000
$hwstack = 44
$swstack = 16
$framesize = 28
config CLOCKDIV=1
'I2C pins on tiny2313 connected like :
'PB5 SDA
'PB7 SCL

'I2C pins on tiny85 connected like :
'PB0 SDA
'PB2 SCL

config BASE=0
'arrays start at address 0

Const Cprint = 0                                            'make
0 for chips that have NO UART, make 1 when the micro has a UART
and you want to show data on the terminal

#if cPrint
   Config Com1 = 19200 , Synchrone = 0 , Parity = None ,
Stopbits = 1 , Databits = 8 , Clockpol = 0
   print "USI DEMO"
#endif
```

```
config usi = twislave , address = &H40
'bascom uses 8 bit i2c address (7 bit shifted to the left with
one bit)

dim epr(128) as Eram byte                  'for easy access to the
memory
dim wAdres as Word, bValue as Byte
dim bAdresL as Byte at Wadres overlay      'overlay with wAdres
LSB
dim bAdresH as Byte at Wadres+1 overlay    'overlay with wAdres
MSB

'do not forget to enable global interrupts since USI is used in
interrupt mode
enable interrupts                                              'it
is important you enable interrupts

do
   !  nop                                                      ;
nothing to do here
loop



'The following labels are called from the library. You need to
insert code in these subroutines
'Notice that the PRINT commands are remarked.
'You can unmark them and see what happens, but it will increase
code size
'The idea is that you write your code in the called labels. And
this code must execute in as little time
'as possible. So when you slave must read the A/D converter, you
can best do it in the main program
'then the data is available when the master requires it, and you
do not need to do the conversion which cost time.


'A master can send or receive bytes.
'A master protocol can also send some bytes, then receive some
bytes
'The master and slave address must match.

'the following labels are called from the library  when master
send stop or start
Twi_start_received:
   #if cprint
      Print "Master sent start or repeated start"
   #endif
Return
```

```
Twi_stop_received:
   #if cprint
      Print "Master sent stop"
   #endif
Return

'master sent our slave address and will now send data
Twi_addressed_goread:
   #if cprint
      Print "We were addressed and master will send data"
   #endif
Return


Twi_addressed_gowrite:
   #if cprint
      Print "We were addressed and master will read data"
   #endif
Return

'this label is called when the master sends data and the slave
has received the byte
'the variable TWI holds the received value
Twi_gotdata:
   #if cprint
      Print "received : " ; Twi ; " byte no : " ; Twi_btw ; "-";
usidr
   #endif
   Select Case Twi_btw
      Case 1 : bAdresL=TWI 'first byte is LSB
      Case 2 : bAdresH=TWI 'second byte is MSB
      case 3 :
         #if cprint
            print "address:" ; wAdres
         #endif
         epr(wAdres)=twi 'write to eeprom in case we receive a
third byte which should only happen when we write to the slave
   End Select

'if you want to auto inc wAdres, use this code instead:
'   Select Case Twi_btw
'      Case 1 : bAdresL=TWI 'first byte is LSB
'      Case 2 : bAdresH=TWI 'second byte is MSB
'      case else : epr(wAdres)=twi 'write to eeprom in case we
receive a third byte which should only happen when we write to
the slave
'                  incr wAdres
'   End Select
```

```
Return

'this label is called when the master receives data and needs a
byte
'the variable twi_btr is a byte variable that holds the index of
the needed byte
'so when sending multiple bytes from an array, twi_btr can be
used for the index
Twi_master_needs_byte:
   #if cprint
      Print "Master needs byte : " ; Twi_btr
      print "address:" ; wAdres
   #endif
   twi=epr(wAdres) 'return the data from EEPROM
   'when you want to support auto adres increase add this :
   'incr wAdres
Return
```

The following master sample can be used with the slave sample.

Master sample

```
'-------------------------------------------------------------------
--------------
'                        eeprom_master.bas
'                    demo for USI eeprom slave
'
'
'-------------------------------------------------------------------
--------------
$Regfile= "m88pdef.dat"
$crystal=8000000
$HWstack=40
$SWstack=50
$FrameSize=40
$baud=19200
$lib "i2c_twi.lbx"              ' we do not use software emulated
I2C but the TWI

config CLOCKDIV=1               ' no need to change fuse byte, we
set the divider to 1
Config Sda = Portc.4                                        ' I2C
Bus konfigurieren
Config Scl = Portc.5

Dim Address As Word
Dim Value As Byte
'!!!!!!!!!!!!!!!!!!!
'osccal=46              'REMARK THIS LINE, THIS WAS REQUIRED for
```

```
the test chip
'!!!!!!!!!!!!!!!!!!!

Print "Start"
I2cinit                          ' init i2c
For Address = 0 To 10                                       ' just
test a bit
   value=address+10
   print "write "; address ; ":";value

   I2cstart : I2cwbyte &H40                                 'slave
address
   I2cwbyte Low(address)                                    'LSB
first
   I2cwbyte High(address)                                   'MSB
   I2cwbyte Value                                           'write
value
   I2cstop
   Waitms 500
next

print "Read"
For Address = 0 To 10
   ' The mathing master code to read
   I2cstart : I2cwbyte &H40                                 'send
slave WRITE address
   I2cwbyte Low(address) :  I2cwbyte High(address) :        'send
eeprom address
   I2crepstart                                              'repeated
start
   I2cwbyte &H41                                            'write
slave READ address
   I2crbyte Value , Nack                                    'read
eeprom value
   I2cstop
   print address;":";value
Next Address                                                  '
increment address byte

end

'EXPECTED OUTPUT
'(
Start
write 0:10
write 1:11
write 2:12
write 3:13
write 4:14
```

```
write 5:15
write 6:16
write 7:17
write 8:18
write 9:19
write 10:20
Read
0:10
1:11
2:12
3:13
4:14
5:15
6:16
7:17
8:18
9:19
10:20
')
```

8.13 I2CV2

I2C Software

By default BASCOM will use software routines when you use I2C statements. This because when the first AVR chips were introduced, there was no TWI interface. Atmel named it TWI because Philips is the inventor of I2C. But TWI is the same as I2C.

When your processor has a TWI interface you can best use this TWI interface.
By default the software master i2c routines use the library named i2c.lib. This library does not maintain a clock/data state so when i2cstart or i2cstop is generated, the clock and data lines need to be set to the proper state before the start/stop condition can be generated. This can result in small glitches. Most slave chips will not notice them but some do.
For this purpose the i2c master library has been rewritten so that clock and data have a known state/level at all times. This allows to create glitch free clock/data.
To use this library use the $LIB directive : $LIB "I2CV2.LIB"
This will make the compiler use this library. One thing to be aware of : a repeated start can only be created by using the I2CREPSTART 1306 statement.
This is a difference with the default i2c.lib

When you want to use soft I2C on an XTINY or XMEGA you need to use the $FORCESOFTI2C 635 directive.
Example

```
$forcesofti2c                              ' force soft i2c
$lib "i2cv2.lib"                           ' use this one
```

See also: Using the I2C protocol 297, CONFIG TWI 1116 , I2CV2 1763

8.14 MCSBYTE

The numeric<>string conversion routines are optimized when used for byte, integer, word and longs.

When do you use a conversion routine ?
- When you use STR() , VAL() or HEX().
- When you print a numeric variable
- When you use INPUT on numeric variables.

To support all data types the built in routines are efficient in terms of code size.
But when you use only conversion routines on bytes there is a overhead.

The mcsbyte.lib library is an optimized version that only support bytes.
Use it by including : $LIB "mcsbyte.lbx" in your code.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager 132.

See also
mcsbyteint.lib 1764

8.15 MCSBYTEINT

The numeric<>string conversion routines are optimized when used for byte, integer, word and longs.

When do you use a conversion routine ?

-When you use STR() , VAL() or HEX().
-When you print a numeric variable
-When you use INPUT on numeric variables.

To support all data types the built in routines are efficient in terms of code size.
But when you use only conversion routines on bytes there is a overhead.

The mcsbyteint.lib library is an optimized version that only support bytes, integers and words.
Use it by including : $LIB "mcsbyteint.lbx" in your code.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

See also
mcsbyte.lib 1764

8.16 PULSEIN

The full version includes a lib named pulsein.lib. It overloads the PULSEIN 1400 statement. This special lib allows to set a custom timeout and delay.
You need to add the following to your code :

```
const cPulseIn_Timeout = 0     'This is the default timeout value. When you increase
the value you will get a shorter time out period.
dim bPulseIn_Delay as byte :  bPulseIn_Delay = 10   'For 10 uS units , the default is
1
$lib "pulsein.lib"  'include the lib to overload the function
```

The library is compatible with the default lib.

8.17 SERIN

This is an alternative library that adds timeout support to the SERIN 1506 statement.
Development was sponsored by a customer.
To use this library instead of the default SERIN code, you need to add it to the configuration using the $LIB 665 directive

```
$lib "serin.lib"
```

Then you need to dimension a DWORD or LONG variable named **SERIN_TIMEOUT** which is used for the timeout.
You assign a time out value to this variable. A higher value will cause a longer time out.
The time out is not in uSec or mSec but relative to the processor speed.
Thus using a clock of 1 Mhz will have a longer time out than a processor clock of 16 MHz.

How it works : The value of SERIN_TIMEOUT is copied to 4 registers. When the software is waiting for a certain bit level, instead of looping, it will decrease the registers and when they reach 0, the code ends. This will prevent that the processor locks up when you have bad signals.

This lib is only available in the full version.

8.18 TCPIP

The TCPIP library allows you to use the W3100A internet chip from www.iinchip.com

There are also libraries for W5100, W5200 and W5300.
MCS has developed a special development board that can get you started quickly with TCP/IP communication. Look at http://www.mcselec.com for more info.

All tcpip lbx files areshipped with BASCOM-AVR

The following functions are provided:

CONFIG TCPIP 1098	Configures the W3100 chip.
GETSOCKET 1553	Creates a socket for TCP/IP communication.
SOCKETCONNECT 1567	Establishes a connection to a TCP/IP server.
SOCKETSTAT 1571	Returns information of a socket.

TCPWRITE 1578	Write data to a socket.
TCPWRITESTR 1579	Sends a string to an open socket connection.
TCPREAD 1576	Reads data from an open socket connection.
CLOSESOCKET 1564	Closes a socket connection.
SOCKETLISTEN 1571	Opens a socket in server(listen) mode.
GETDSTIP 1551	Returns the IP address of the peer.
GETDSTPORT 1551	Returns the port number of the peer.
BASE64DEC 1547	Converts Base-64 data into the original data.
BASE64ENC 1549	Convert a string into a BASE64 encoded string.
MAKETCP 1556	Encodes a constant or 4 byte constant/variables into an IP number
UDPWRITE 1588	Write UDP data to a socket.
UDPWRITESTR 1589	Sends a string via UDP.
UDPREAD 1582	Reads data via UDP protocol.

The MCS webshop offers the WIZ810MJ ethernet module, and a special converter board so it has few connections.
WIZ810MJ module
TCPADB5100 adapter board.

8.19 M128-1wire-PortF

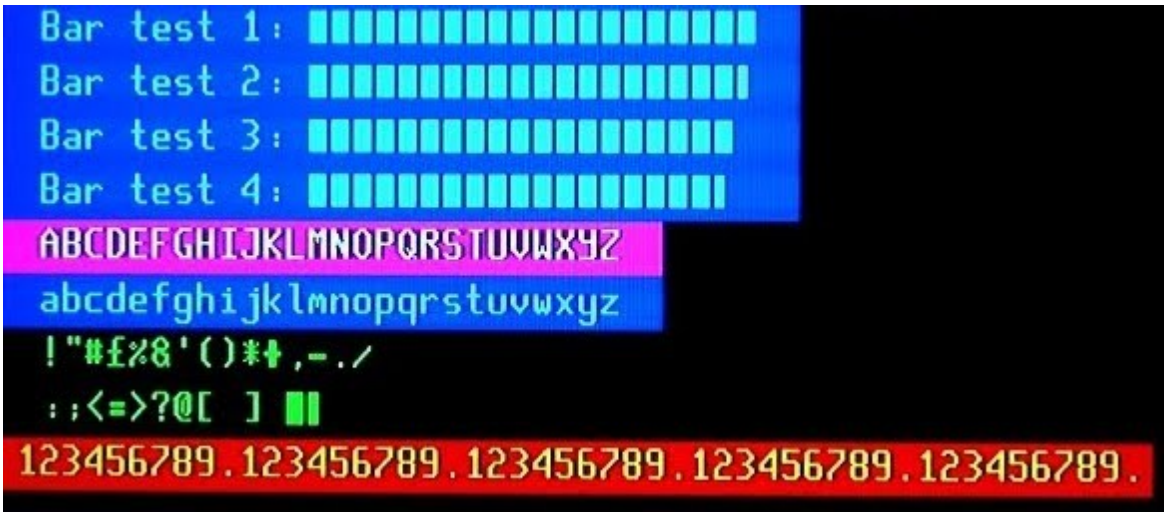
This user contributed library is only for the atmega128 when 1wire is used on PORTF. Normally the port registers DDR, PORT and PIN are grouped and this is used to work with pointers.
PORTF is however incompatible since it is grouped different. This library uses fixed addresses.

- When using this library you can not use 1wire devices on other ports. This because this lib overloads the default library.
- The EXTENDED=1 option from CONFIG 1WIRE 857 may not be used in combination with this library.

8.20 TVOUT

The TVOUT add on is an add on that allows you to show text in color on a TV using the SCART connector.

The add on is free for personal use but for commercial use you need to buy a license from the author (Graham Carnell).



This is a photo of the TV display function working on a flat panel LCD TV set

The actual display is perfectly straight, some distortion is seen here caused by the camera optics.

The TV code is free for personal use but that support is not included.
For commercial application you do need a license.

Company licence (unlimited copies for company use) now available for commercial use. Includes built & tested board, pre-programmed sample IC, TV generation software module including all pixel data which can be edited, plus full support from the developer by phone and e-mail.

TV Code Features

* Generates a 55 column color TV character display from an AVR MPU without any extra ICs
* Connects via a standard SCART socket giving a sharp RGB output signal (not composite video)
* Flexible RAM use - display RAM can be as large or small as needed
* Completely interrupt driven software - transparent to user
* Character pixel data can be edited or replaced by the user to allow custom characters

The TV software can be customized to special user requirements (e.g. differing character sizes) but is time critical machine code.

ICs supported

This code is for the ATMega 48/88/168/328 ICs. A PCB is available for testing and / or production. The code can be ported to other ATMega AVR ICs which have 512 bytes or more RAM, and a clock of 16-20 MHz.

BASCOM versions

When using the company licence (which allows as many copies as you need) you will also require a registered copy of BASCOM to allow sufficient Flash program memory for most projects, as the binary include file for the TV output code is 2.5K, leaving only 1.5K of available space for your program out of the 4K maximum space allowed

in the demo version of BASCOM.

Orders
The software and hardware is made by Eximia Projects.
Your order will be shipped from the UK, directly from the manufacturer.
You will receive a binary include file.
You will also receive a free development board PCB. This board is used in production and does not has an ISP connector. But it has a tested processor and all other components. All you need is to connect 5V and a TV and it will show a demo.

Support is included in the cost of the development package. You can be assured that this support will smooth your way to producing a product with a TV output - you will not be left to struggle on your own.
Before purchasing the package you might want to E-mail to check if the TV software will be compatible with your planned product, for example if there are many interrupts running or a heavy CPU load.

If you want to use a different Atmel AT Mega IC to the one the software is designed for (AT Mega 48 / 88 / 168 / 328) the first step to take is to contact Eximia Projects and let me know what your design requires. I can then let you know what extra steps you will need to take (if any) to get the TV software to work with any specific hardware. You can contact Graham Carnell at gmcarnell@gmail.com
A PCB you can buy from Agricom :
http://agricom.gr/eshop/product_info.php?cPath=26_38&products_id=986&language=en

How to use BASCOM with the TV software module

The TV display is very simple to use from a programming point of view.
As it runs entirely within the ATMega chip, it uses internal RAM to hold the display data. This means that all you have to do to write to the display is write bytes to the RAM which is allocated to the TV screen area.

To start with you need to include the following 7 lines of code in your BASCOM program:

```
On Oc2a Tvinterrupt Nosave
Goto Main
!.org $100
$inc Tvinterrupt , Nosize , "tvinc.bin"
Return
Main:
$include "tv.inc.bas"
```

You don't need to worry about these statements, you can just cut & paste this into your program and it will work.

After including these lines of code you can make the TV display work simply by moving bytes to the screen area in the internal RAM.

The amount of RAM used by the display is very flexible. It can be any number of bytes up

to the maximum possible which is 12 lines of around 56 bytes per line, maximum 672 bytes.

The minimum number of bytes which can be used is just one! This byte would be the "End of Screen" code which has to be the very last byte of any screen. In fact, if the interrupt is disabled, then no RAM is needed at all, and you will also have full CPU usage until you enable the interrupt again. To disable the TV software all you need to do is:

```
DISABLE OC2A
```

this will halt the TV code and allow all RAM to be used by your application, then

```
ENABLE OC2A
```

to start the TV code again. You will need to make sure the screen RAM area contains sensible display data before enabling the interrupts.

You can use all of the on-chip RAM to do calculations and for temporary storage, you only need to free enough RAM as you need for the screen while it is actually displaying.

More Info

http://sites.google.com/site/bascomtvhelp/

PDF with detailed information can be downloaded from here

Bascom TV FAQs

Q: How can I add TV output to my BASCOM application?

A: If you are using an ATMega48/88/168/328 everything is already set up for you to use. If you want to use another ATMega IC, you need to refer to the technical information and make sure that the IC you are using has enough hardware resources (CPU speed, SPI port, RAM etc). You can't just add TV output to any AVR chip - ATTiny ICs are not supported as the TV code uses the hardware multiply instruction, and only ATMega ICs have enough RAM.

Q: What about low power applications?

A: Whilst TV output is enabled, power consumption will be around the maximum given in the data sheet for the IC at the speed and voltage used. TV output should be disabled when not needed, then the IC can benefit from all the low power and sleep modes available. This would be relevant to any device which is normally in low power mode, but can have a TV attached to display data only when required.

Q: What about CPU intensive applications?

A: As explained in the previous Q/A about low power, the TV output can be switched off

(by disabling the relevant interrupt) so the full CPU power is available, however most applications can easily run in the spare time (approx 20%) of the CPU when running at 16 or 20 MHz.

Sample hardware:

TECHNICAL INFORMATION

IMPORTANT

You do not have to read this information - BASCOM and the TV code will automatically set up the hardware as required. If you use the supplied PCB this ensures the TV output will work without any knowledge of the module.
These technical details are for reference.

The TV code has various fixed hardware requirements as follows.

Clock:

The clock MUST use an external crystal to produce a stable screen.
Normally 20 MHz but 16 MHz can be used (20% less columns).

SPI port:

The SPI hardware is used by the TV code and cannot be used for other purposes while the TV code is running.

RAM use:

* Amount of RAM used can be very small - EndScreen code marks end of RAM used
* Lines are variable length so only visible characters [excepting space] use RAM

RAM Addresses:

* The address of RAM used by the TV code is fixed at $100 (start of RAM in ATMega 48/88/168/328 ICs)
* The first six bytes of RAM are used to store variables for the interrupt code
* The first byte of RAM used for the screen area is at address $106

Maximum RAM use:

* For a full screen of 55 characters by 12 lines, RAM used = 660 bytes
* In an ATMega48, used screen RAM will need to be kept below around 450 bytes for use with BASCOM

GPIO register:

In ATMega 48/88/168/328 ICs there is a "GPIO" register at $1E. Bit 0 of this is used by the code.
The other 7 bits are unused and can be changed by the user software.

Pixel data:

* The pixel data used for the characters shown on screen starts at a fixed Flash ROM word address
* All pixel data can be edited or replaced by the user to allow custom characters

Timers:

* Timer 2, an 8 bit timer, is reserved for use by this code
* Timer 2 causes a 64µS interrupt and can be used for a system "tick"
* Timers 0 and 1 are unused

Reset and interrupt vectors:

* The interrupt vectors for Timer 2 compare match A and B are both used
* Timer/Counter2 Compare Match B vector points to the TV interrupt code

Control characters:

* End of line code = EndLineCode = $0D
* End of screen code = EndScreenCode = $0C
* Set colour to yellow chars on a red background = SetYellowCode = $10
* Set colour to green chars on a black background = SetGreenCode = $11
* Set colour to cyan chars on a blue background = SetBlueCode = $12
* Set colour to white chars on a magenta background = SetWhiteCode = $16

All of Port B is reserved:

* PB0 Sync
* PB1 Blue
* PB2 Red
* PB3 Green
* PB4 & PB5 [2] allocated SPI pins
* PB6 & PB7 [2] Used for XTAL

No pins on Ports C or D are used

Interrupt Code:

* Triggered by Timer 2 interrupt
* Runs every 64 uS
* Consumes up to 80% of CPU time [worst case]

Fuses:

Only the low fuse needs to be changed:

CKDIV8 must be high (unprogrammed) so clock will be at full freq

1 CKDIV8 Divide clock by 8 (default 0 = programmed)
1 CKOUT Clock output (default 1 = unprogrammed) i.e. clock output off
1 SUT1 Select start-up time (default 1 = unprogrammed)
1 SUT2 Select start-up time (default 0 = programmed)

SUT1,2 = 11 selects Crystal Oscillator, slowly rising power (in case of PSU problems)

0 CKSEL3 0
1 CKSEL2 0
1 CKSEL1 1
1 CKSEL0 0

0111 in CKSEL 3210 selects full swing oscillator, slowly rising power

So lfuse = $F7

Example

```
' Serial input demo.
' NOTES:
' Uses an array of bytes for Screen RAM
' Uses Tilde char "~" = $7E for new screen

$crystal = 20000000
$BAUD = 19200
```

```
' Reserve screen variable area at start of RAM
' Assembler interrupt code has two byte variables IN BETWEEN two word
variables
Dim ScreenAddr1 as word at $100
Dim RAMVar1 as byte at $102
Dim RAMVar2 as byte at $103
Dim ScreenAddr2 as word at $104

' Reserve Screen RAM - can reserve as much or as little as required
Dim ScreenRAM(600) as byte at $106
Dim Addr As Word
Dim CharIn as Byte

Const NewScreen = $7E

'Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'Config is not needed unless settings differ from default

On Oc2a Tvinterrupt Nosave
Goto Main
!.org $100
$inc Tvinterrupt , Nosize , "tvinc.bin"
Return


Main:
   ' Set up clock division - only need to do this if DIV8 fuse not set,
as default fuse setting is div. by 8
   Config Clockdiv = 1
   '        CLKPR=$80
   '        CLKPR=0
   ' Setup timer 2
   TCCR2B=$02
   OCR2A=158
   OCR2B=160
   TIMSK2=&b00000110
   ' Now set up sleep mode [SMCR = Sleep Mode Control Register] - must
be enabled or TV code cannot work accurately
   SMCR=1
   ' Set PORTB to all outputs for video signal
   DDRB=$FF
   ' Enable & config SPI
   SPCR=$54
   SPSR=1
   ' Init RAM variables for interrupt code
   RAMVar1=0
   RAMVar2=0
   ScreenAddr1=$106
   ScreenAddr2=$106

   ENABLE OC2A
   '        ENABLE OC2B
   ENABLE INTERRUPTS
   ' Now continue with user code



   Addr=1        ' Set Addr to address of first screen location

   Do
      CharIn=INKEY()
      If CharIn>0 then
```

```
            ScreenRAM(Addr)=CharIn
            Incr Addr
            ScreenRAM(Addr)=13      ' Make sure there is always an end of
screen character
         End If
         If CharIn=NewScreen then
            Addr=1
            ScreenRAM(Addr)=13
         End If
         If Addr>599 then Addr=599       ' Make sure cannot write past end
of screen Ram
      Loop

      ' Decimal {012} = $0C = END OF LINE MARKER
      ' Decimal {013} = $0D = END OF SCREEN MARKER

      ' Decimal {017} = $11 = Green on black
      ' Decimal {019} = $13 = Cyan on blue
      ' Decimal {020} = $14 = Yellow on red
      ' Decimal {022} = $16 = White on magenta

   End
```

8.21 RAINBOWBSC

This lib is based on the rainbow 1.2 lib from Galahat. See also : http://bascom-forum.de/mediawiki/index.php/Rainbow_Lib
The rainbowbsc.lib is essentially the same lib providing the same functionality. Some code is moved to CONFIG RAINBOW, and the routines are renamed in order to to give conflicts with existing/future statements/functions.

See CONFIG RAINBOW 1033

8.22 LCD

8.22.1 LCD4BUSY

BASCOM supports LCD displays in a way that you can choose all pins random. This is great for making a simple PCB but has the disadvantage of more code usage. BASCOM also does not use the WR-pin so that you can use this pin for other purposes.

The LCD4BUSY.LIB can be used when timing is critical.

The default LCD library uses delays to wait until the LCD is ready. The lcd4busy.lib is using an additional pin (WR) to read the status flag of the LCD.

The db4-db7 pins of the LCD must be connected to the higher nibble of the port.

The other pins can be defined.

```
'-----------------------------------------------------------------------
' (c) 1995-2025 MCS Electronics
' lcd4busy.bas shows how to use LCD with busy check
'-----------------------------------------------------------------------
'code tested on a 8515
$regfile="8515def.dat"
```

```
'stk200 has 4 MHz
$crystal= 4000000

'define the custom library
'uses 184 hex bytes total

$lib"lcd4busy.lib"

'define the used constants
'I used portA for testing
Const _lcdport =Porta
Const _lcdddr =Ddra
Const _lcdin =Pina
Const _lcd_e = 1
Const _lcd_rw = 2
Const _lcd_rs = 3


'this is like always, define the kind of LCD
ConfigLcd= 16 * 2

'and here some simple lcd code
Cls
Lcd"test"
Lowerline
Lcd"this"
End
```

8.22.2 LCD4_anypin_oled_RS0010

This LCD driver is intended to be used with the OLED LCD RS0010.

This LCD text driver can be used with any pin. It supports the WR pin in which case the LCD will be used in busy mode.

A typical sample is shown below.

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
$hwstack=32
$swstack = 16
$framesize=24



$lib "lcd4_anypin_oled_RS0010.lib"                          'override
default lib with OLED lib

'Config Lcd Sets The Portpins Of The Lcd
Config Lcdpin = Pin , Db4 = Portb.2 , Db5 = Portb.3 , Db6 = Portb.4 ,
Db7 = Portb.5 , E = Portb.1 , Rs = Portb.0
Config Lcd = 16x2                                           '16*2 type
LCD screen
```

```
Dim V As Byte

Cls
Lcd "ABC" ; Chr(253)
Lowerline
Lcd "test"
Const Test = " this is a test"                              ' Just  A
Test

Lcdfont 0                                                   'select
first font

Cls
Dim X As Byte , Y As Byte
X = &B1000_0000 + 0
Lcdcmd &B0001_1111                                          'gmode
Lcdcmd X                                                    'X  (0-99)
Lcdcmd &B0100_0000                                          'Y  (0-1)

'send data
For V = 1 To 80
   Lcddata &B10101010
   Waitms 100
Next
End
```

8.22.3 LCD_RX1602A5

This LCD driver is based on O-Family AQM0802A Library.
It is suited for I2C displays RX1602A5. It was developed for, and sponsored by Lab microelectronic GmbH

All you need to do is connect the LCD to the I2C pins and configure LCD like : config lcd = 16x2 , chipset = st7032
A sample you find under CONFIG LCD 992
Of course you need a functional I2C or TWI bus. Both soft and HW TWI are supported.

8.22.4 LCD4.LIB

The built in LCD driver for the PIN mode is written to support a worst case scenario where you use random pins of the microprocessor to drive the LCD pins.

This makes it easy to design your PCB but it needs more code.
When you want to have less code you need fixed pins for the LCD display.

With the statement $LIB "LCD4.LBX" you specify that the LCD4.LIB will be used.

The following connections are used in the asm code:

Rs = PortB.0
RW = PortB.1 we dont use the R/W option of the LCD in this version so connect to ground
E = PortB.2

E2 = PortB.3 optional for lcd with 2 chips
Db4 = PortB.4 the data bits must be in a nibble to save code
Db5 = PortB.5
Db6 = PortB.6
Db7 = PortB.7

You can change the lines from the lcd4.lib file to use another port.
Just change the address used :
.EQU LCDDDR=$17 ; change to another address for DDRD ($11)
.EQU LCDPORT=$18 ; change to another address for PORTD ($12)

See the demo lcdcustom4bit.bas in the SAMPLES dir.

Note that you still must select the display that you use with the CONFIG LCD 992 statement.

See also the lcd42.lib 1777 for driving displays with 2 E lines.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

8.22.5 LCD4E2

The built in LCD driver for the PIN mode is written to support a worst case scenario where you use random pins of the microprocessor to drive the LCD pins.

This makes it easy to design your PCB but it needs more code.

When you want to have less code you need fixed pins for the LCD display.
With the statement $LIB "LCD4E2.LBX" you specify that the LCD4.LIB will be used.

The following connections are used in the asm code:
Rs = PortB.0
RW = PortB.1 we don't use the R/W option of the LCD in this version so connect to ground
E = PortB.2
E2 = PortB.3 the second E pin of the LCD
Db4 = PortB.4 the data bits must be in a nibble to save code
Db5 = PortB.5
Db6 = PortB.6
Db7 = PortB.7

You can change the lines from the lcd4e2.lib file to use another port.
Just change the address used :
.EQU LCDDDR=$17 ; change to another address for DDRD ($11)
.EQU LCDPORT=$18 ; change to another address for PORTD ($12)

See the demo lcdcustom4bit2e.bas in the SAMPLES dir.

Note that you still must select the display that you use with the CONFIG LCD 992 statement.

See also the lcd4.lib 1776 for driving a display with 1 E line.

A display with 2 E lines actually is a display with 2 control chips. They must both be controlled. This library allows you to select the active E line from your code.

In your basic code you must first select the E line before you use a LCD statement.

The initialization of the display will handle both chips.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

8.22.6 GLCD

GLCD.LIB (LBX) is a library for Graphic LCD's based on the T6963C chip.

The library contains code for LOCATE 1347, CLS 1322, PSET 1348, LINE 1344, CIRCLE 1319, SHOWPIC 1355 and SHOWPICE 1355.

8.22.7 GLCDSED

GLCDSED.LIB (LBX) is a library for Graphic LCD's based on the SEDXXXX chip.

The library contains modified code for this type of display.
New special statements for this display are :

LCDAT 1339
SETFONT 1351
GLCDCMD 1333
GLCDDATA 1333

See the SED.BAS sample from the sample directory

8.22.8 PCF8533

COLOR LCD

Color displays were always relatively expensive. The mobile phone market changed that. And Display3000.com , sorted out how to connect these small nice colorfully displays.
You can buy brand new Color displays from Display3000. MCS Electronics offers the same displays.
There are two different chip sets used. One chip set is from EPSON and the other from Philips. For this reason there are two different libraries. When you select the wrong one it will not work, but you will not damage anything.
LCD-EPSON.LBX need to be used with the EPSON chip set.
LCD-PCF8833.LBX need to be used with the Philips chip set.

Config Graphlcd = Color , Controlport = Portc , Cs = 1 , Rs = 0 , Scl = 3 , Sda = 2

Controlport	The port that is used to control the pins. PORTA, PORTB, etc.
CS	The chip select pin of the display screen. Specify the pin number. 1 will mean PORTC.1
RS	The RESET pin of the display
SCL	The clock pin of the display
SDA	The data pin of the display

As the color display does not have a built in font, you need to generate the fonts yourself.
You can use the Fonteditor 238 for this task.

A number of statements accept a color parameter. See the samples below in **bold**.

LINE	Line(0 , 0) -(130 , 130) , **Blue**
LCDAT	Lcdat 100 , 0 , "12345678" , **Blue , Yellow**
CIRCLE	Circle(30 , 30) , 10 , **Blue**
PSET	32 , 110 , **Black**
BOX	Box(10 , 30) -(60 , 100) , **Red**

See Also

LCD Graphic converter 1862

Example

```
'
------------------------------------------------------------------------------
' The support for this display has been made possible by Peter Küsters from (c) Display3000
' You can buy the displays from Display3000 or MCS Electronics
'
------------------------------------------------------------------------------'
'
$lib "lcd-pcf8833.lbx"                                      'special color display support

$regfile = "m88def.dat"                                     'ATMega 8, change if using different processors
$crystal = 8000000                                          '8 MHz

'First we define that we use a graphic LCD
Config Graphlcd = Color , Controlport = Portc , Cs = 1 , Rs = 0 , Scl = 3 , Sda = 2

'here we define the colors

Const Blue = &B00000011     'predefined contants are making programming easier
Const Yellow = &B11111100
Const Red = &B11100000
Const Green = &B00011100
Const Black = &B00000000
Const White = &B11111111
Const Brightgreen = &B00111110
Const Darkgreen = &B00010100
Const Darkred = &B10100000
Const Darkblue = &B00000010
Const Brightblue = &B00011111
Const Orange = &B11111000


'clear the display
Cls

'create a cross
Line(0 , 0) - (130 , 130) , Blue
Line(130 , 0) - (0 , 130) , Red

Waitms 1000
```

```
'show an RLE encoded picture
Showpic 0 , 0 , Plaatje
Showpic 40 , 40 , Plaatje

Waitms 1000

'select a font
Setfont Color16x16
'and show some text
Lcdat 100 , 0 , "12345678" , Blue , Yellow


Waitms 1000
Circle( 30 , 30) , 10 , Blue

Waitms 1000
'make a box
Box( 10 , 30) - (60 , 100) , Red

'set some pixels
Pset 32 , 110 , Black
Pset 38 , 110 , Black
Pset 35 , 112 , Black
End


Plaatje:
$bgf "a.bgc"


$include "color.font"
$include "color16x16.font"
```

8.22.9 LCD-EPSON

This chip is compatible with PCF8533.

8.22.10 LCD_DOGS104a_I2C

This is a user contributed lbx for the EADOGS104 with the SSD1803A.

The SAMPLES\LCDGRAPH folder contains the sample :

```
'-----------------------------------------------------------------
'                         DOGS-104.bas
'       Demonstration for DOGS 104-A display
'                   (c) R. Müller-Westermann
'                        HB9EFQ@yahoo.com
'-----------------------------------------------------------------

$regfile = "m168def.dat"
$crystal = 1000000
'$sim

$hwstack = 32
$swstack = 32
$framesize = 64

$lib "Lcd_dogs104a_i2c.lbx"
```

```
'LCD  ------------------------------------------------------------
'chipset:DOGS104V3
'DOGS104 Display can use either &H78 if pin SA0 of module is set to GND
'or &H7A if SA0 of module is set to VDD for I²C communication.
'Pullup resistors on SDA and SCL lines of less or equal to 3.9kOhm
@3.3V
'are recommended.

 Const Dogs104_adr_w = &H78                                 'I²C write
address
 Const Dogs104_adr_r = &H79                                 'I²C read
address


 'LCD has 2 view options. If LCD_view is set to 0 characters are being
 'displayed in bottom view (6 o'clock). If set to 1 characters are
being
 'displayed in top view (12 0'clock)
 Const Lcd_view = 0                                         'bottom
view
 'Const Lcd_view = 1                                         'top View


 'configuration is needed for defining start address of LCD RAM
 Config Lcd = 20x2


 'LCD comes with 3 different character sets. These can be accessed by
setting
 'LCD_ROM
 Const Lcd_rom = 1                                          'ROM A
 'Const Lcd_rom = 2                                          'ROM B
 'Const Lcd_rom = 3                                          'ROM C


 'there are 2 custom procedures witch provide number of lines switching
at
 'runtime. You can either choose 2 line mode with double hight fonts or
regular
 '4 line mode. This is the standard mode used by Initlcd.
 $external 2line_mode
 $external 4line_mode

'LCD  ------------------------------------------------------------

Declare Sub 2line_mode
Declare Sub 4line_mode

'TWI---------------------------------------------------------------
Config Scl = Portc.5
Config Sda = Portc.4
```

```
I2cinit
'TWI-----------------------------------------------------------------


Initlcd
Waitms 100

'As with any other LCD module, you can define up to 8 additional
characters
'by using the regular Bascom command

 '-------------------------------------------------------------------
    Deflcdchar 1 , 32 , 32 , 4 , 10 , 17 , 10 , 4 , 32      ' circle
 '-------------------------------------------------------------------

 Cls

 Waitms 100

 Cursor Off

 Locate 1 , 1 : Lcd Chr(1)

 Wait 2

 'standard initialization of LCD is set to 4 line mode

 Cls

 Locate 1 , 1 : Lcd "line 1"
 Locate 2 , 1 : Lcd "line 2"
 Locate 3 , 1 : Lcd "line 3"
 Locate 4 , 1 : Lcd "line 4"

 Wait 2

'-----------------------------------

 Cls

 'if needed LCD can be switched to 2 line mode
 2line_mode

 Locate 1 , 3 : Lcd "line 1"
 Locate 2 , 3 : Lcd "line 2"

 Wait 2

 ' ... and back to 4 line mode

 4line_mode

 Cls
```

```
  Locate 1 , 3 : Lcd "line 1 "
  Locate 2 , 3 : Lcd "line 2"
  Locate 3 , 3 : Lcd "line 3"
  Locate 4 , 3 : Lcd "line 4"

  Wait 2

'if desired you can put the LCD module in power down mode. This saves
some
'400µA.
'Any other command applicable for DOGS104A using SSD1803A controller
can be
'issued by using regular Rcall _Lcd_control command with preloaded
'R24 register.

 Display Off
 Waitms 100

 'power down ----------------

 R24 = &B00111010                                          '8 bit data
RE1, REV0
 Lcdcmd R24



 R24 = &B00000011                                          'power down
 Lcdcmd R24

 R24 = &B00111000                                          '8 bit data
RE0, IS0
 Lcdcmd R24

 'power down ----------------

 Wait 2

'... and power up again. LCD RAM remains unchanged.

 'power up -----------------

 R24 = &B00111010                                          '8 bit data
RE1, REV0
 Lcdcmd R24

 R24 = &B00000010                                          'power up
 Lcdcmd R24

 R24 = &B00111000                                          '8 bit data
RE0, IS0
 Lcdcmd R24
```

```
 'power up -----------------

 Waitms 100
 Display On

 Locate 4 , 1 : Lcd "powered up"
End
```

8.22.11 glcdR7565R

The glcdR7565R.lib is intended to be used with 128x64 displays using the ST7565R chip.

```
'-----------------------------------------------------------------
'                   (c) 1995-2025, MCS
'                    xm128A1-ST7565R.bas
'  This sample demonstrates the ST7565R chip with an Xmega128A1
'  Display used : 64128N SERIES from DisplayTech
'  this is a parallel display with read/write options
'-----------------------------------------------------------------
-

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40

'include the following lib and code, the routines will be
replaced since they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 38400 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8
$lib "glcdST7565r.lbx"                                  ' specify
the used lib
$lib "glcd.lbx"                                        ' and this
one of you use circle/line etc

'the display was connected with these pins
Config Graphlcd = 128 * 64eadogm ,dataport=portj,  Cs1 = Porth.0
, A0 = Porth.2 , rst= Porth.1 , wr = Porth.3 , Rd = Porth.4,c86=
porth.6

cls

Setfont Font8x8tt ' set font
```

```
dim y as byte

'You can use locate but the columns have a range from 1-128
'When you want to show somthing on the LCD, use the LDAT command
'LCDAT Y , COL, value
Lcdat 1 , 1 , "11111111"
Lcdat 2 , 1 , "ABCDEFGHIJKL1234"
Lcdat 3 , 1 , "MCS Electronics" , 1    ' inverse
Lcdat 4 , 1 , "MCS Electronics"

Waitms 3000
Setfont My12_16 ' use a bigger font

Cls
Lcdat 1 , 1 , "112345678"                                    'a
bigger font
Waitms 3000                                                  '
wait

Line(0 , 0) -(127 , 64) , 1                                  'make
line
Waitms 2000  'wait 2 secs
Line(0 , 0) -(127 , 64) , 0
'remove line by inverting the color

For Y = 1 To 20
   Circle(30 , 30) , Y , 1                                   '
growing circle
   Waitms 100
Next

End


$include "font8x8TT.font"
$include "my12_16.font"
```

8.22.12 glcdSSD1325_96x64

This lib is for SSD1325 based displays. This lib supports screen 96x64.
The lib is based on bascom code from Robert Wolgajew.

SSD1325 is used for OLED displays. Each pixel can have 16 tints.
The usual graphic statements are supported.
Images such as bitmaps can be converted into 16 grey tone images.
The ssd1325 conversion tool you can download from the MCS web server.

The sample below is using porta pins to control BS1 and BS2. Of course you would connect them to VDD directly.

The pins used, and bascom pins names are :

SSD pin	BASCOM pin
WR	WR
RD	RD
D0-D7	PORTx

D/C	A0
RES	RST
CS	CS1
VCC	VCC

The VCC pin controls a 12V generator.

Since the display is using a pallet of 16 grey tones, you must specify the foreground and background colors with LCDAT.

```
'-----------------------------------------------------------------------------
'                             (c) 1995-2025 MCS Electronics
'                              oled_ssd1325.bas
'  demonstrates OLED display 96x64 with SSD1325 chip
'  Based on bascom SSD1325 code from Robert Wolgajew
'-----------------------------------------------------------------------------
$regfile = "m8535.dat"
$crystal = 3686000
$hwstack = 48
$swstack = 48
$framesize = 48


'normally the BS1 and BS2 pins would be connected to VCC on the PCB
'but the test PCB used 2 port pins
Config Porta.0 = Output
Config Porta.1 = Output
Porta.0 = 1
Porta.1 = 1


$lib "glcdSSD1325_96x64.lbx"                                ' include the lib

'vcc is 12V and must be enabled later. This means vcc needs a control pin.
Config Graphlcd = 96x64ssd1325 , Dataport = Portc , Wr = Portd.6 , Rd = Portd.7 , Cs1 =
Portd.3 , A0 = Portd.5 , Rst = Portd.4 , Vcc = Portd.2


Cls                                                         'as usual clear display

Dim J As Byte , K As Byte , W As Word

Line(0 , 0) - (95 , 63) , 15                                ' diagonal line
Line(0 , 63) - (95 , 0) , 6                                 ' diagonal line with other
color


Pset 1 , 0 , 15                                             'set a pixel

Setfont Color8x8                                            ' font to use
Lcdat 20 , 0 , "123" , 15 , 0                               'and show some text

Waitms 3000

Showpic 0 , 0 , Plaatje


End
'include font
$include "color8x8.font"
'$include "color16x16.font"


Plaatje:
$bgf "ssd1325.bgc"
```

8.22.13 GLCDEADOGMXL240-7-I2C

This library was sponsored by a customer.
The library supports the EADOGMXL240-7 in I2C mode.
The library supports all the usual graphical LCD commands.

Example

```
'-----------------------------------------------------------------
---------------
'                           eadogxl240-7.bas
'                     (c) 1995-2025  MCS Electronics
'  Sample to demo the EADOGXL240-7 LCD in I2C mode
'
'-----------------------------------------------------------------
---------------
$regfile = "M328pdef.dat"                                   ' the
used chip
$crystal = 8000000                                          '
frequency used
$baud = 19200                                               '
baud rate
$hwstack = 40
$swstack = 40
$framesize = 40

Config Scl = Portc.5                                        ' we
need to provide the SCL pin name
Config Sda = Portc.4                                        ' we
need to provide the SDA pin name

$lib "i2c_twi.lbx"                                          ' we
do not use software emulated I2C but the TWI
Config Twi = 400000
'speed 400 KHz
I2cinit

$lib "glcdEADOGMXL240-7-I2C.lib"
'override the default lib with this special one
#if _build < 2078
  Dim ___lcdrow As Byte , ___lcdcol As Byte
#endif

Config Graphlcd = Custom , Cols = 240 , Rows = 128 , Lcdname =
"EADOGXL240-7"

Cls

Setfont Font8x8tt

'You can use locate but the columns have a range from 1-240
'When you want to show somthing on the LCD, use the LDAT command

Lcdat 1 , 1 , "11111111"
Lcdat 2 , 1 , "88888888"
Lcdat 12 , 64 , "MCS Electronics" , 1
```

```
Showpic 60 , 0 , Plaatje

Circle(30 , 30) , 20 , 255
Line(0 , 0) -(239 , 127) , 255                                    '
diagonal line
Line(0 , 127) -(239 , 0) , 255                                    '
diagonal line

End

$include "font8x8TT.font"


Plaatje:
   $bgf "ks108.bgf"
'include the picture data
```

8.22.14 GLCDdSSD1306-I2C

This library is based on work of Ben Zijstra and Heiko/Hkipnik
The library supports the SSD1306 graphical LCD in I2C mode.
Since the display can not read data back, the library supports only the graphical write statements. Commands like LINE, PSET and CIRCLE which need to alter a single pixel are not supported.

XMEGA

For use with XMEGA you need to define 2 constants in your code.
const TWI_ADR = interface
const TWI_CH = num

The interface must point to the TWI control register, this could be : TWIC_CTRL but aldo TWID_CTRL, TWIE_CTRL and TWIF_CTRL
The TWI_CH constant with the value num, must be 1 for TWIC, 2 for TWID, 4 for TWIE and 8 for TWIF
The reference to : `$lib "i2c_twi.lbx"` must be removed.

Example

```
'-----------------------------------------------------------------
---------------
'                        SSD1306-I2C.BAS
'                     (c) 1995-2025 MCS Electronics
'          Sample to demo the 128x64 I2C OLED display
'
'-----------------------------------------------------------------
---------------
$regfile = "m88pdef.dat"
$hwstack = 32
$swstack = 32
```

```
$framesize = 32
$crystal = 8000000
Config Clockdiv = 1                                         '
make sure the chip runs at 8 MHz

Config Scl = Portc.5                                        '
used i2c pins
Config Sda = Portc.4
Config Twi = 400000                                         ' i2c
speed

I2cinit
$lib "i2c_twi.lbx"                                          ' we
do not use software emulated I2C but the TWI
$lib "glcdSSD1306-I2C.lib"                                  '
override the default lib with this special one

#if _build < 20784
  Dim ___lcdrow As Byte , ___lcdcol As Byte                 ' dim
these for older compiler versions
#endif

Config Graphlcd = Custom , Cols = 128 , Rows = 64 , Lcdname =
"SSD1306"
Cls
Setfont Font8x8tt                                           '
select font

Lcdat 1 , 1 , "BASCOM-AVR"
Lcdat 2 , 10 , "1995-2020"
Lcdat 8 , 5 , "MCS Electronics" , 1
Waitms 3000

Showpic 0 , 0 , Plaatje

End


$include "font8x8TT.font"                                   '
this is a true type font with variable spacing


Plaatje:
   $bgf "ks108.bgf"                                         '
include the picture data
```

8.22.15 LCD_I2C_PCF8574

The LCD_I2C_PCF8574 library is made by O-Family. This library supports multiple LCD.
The library is based on an old library from Kent Andersson. The old lib used bascom code but for best performance should use ASM.
O-Family made this possible. He also extended the lib so multiple LCD can be used.

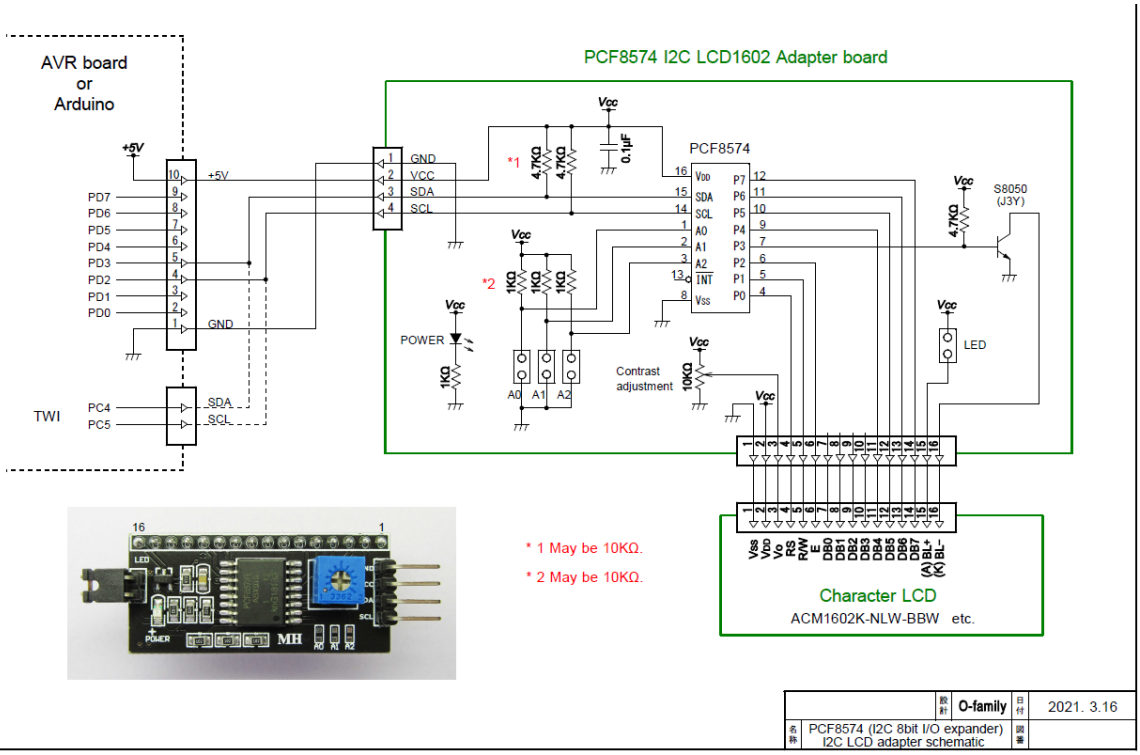
The LCD are normal text LCD. Unlike graphical LCD, TEXT LCD have a kind of standard.
So most text LCD would be usable. The important part is that an I2C port extended chip is used : the PCF8574.
The PCF8574 also has a brother/sister : an identical chip with the A suffix. The only difference is that it has a different base address.

Normally an LCD is operated in 4 bit or 8 bit parallel pin mode. The PCF chip is connected to the LCD data and control lines. And the driver sends the proper I2C commands to control the LCD. This way you can use I2C which is great since it can be used of a greater distance.

XMEGA

For use with XMEGA you need to define 2 constants in your code.
const TWI_ADR = interface
const TWI_CH = num

The interface must point to the TWI control register, this could be : TWIC_CTRL but aldo TWID_CTRL, TWIE_CTRL and TWIF_CTRL
The TWI_CH constant with the value num, must be 1 for TWIC, 2 for TWID, 4 for TWIE and 8 for TWIF



The circuit above show how to connect things. Converter boards exist that can be

soldered right to the LCD.
But you can also wire this yourself.
The A0-A1-A2 select the address of the PCF chip.
More info in the forum 1332.

Two samples you can find in the SAMPLES\LCDTEXT folder

SAMPLE

```
$programmer = 22
'ARDUINO (using stk500v1 protocol)
   '
   '  ************************************
   '  *  PCF8574 I2C LCD Adapter test     *
   '  *    For multiple LCDs   2021/ 3/24 *
   '  ************************************
   '
$regfile = "m328pdef.dat"                                   'Set
the AVR to use.
$crystal = 16000000                                         'Set
the AVR clock.
   '
$hwstack = 64                                               'Set
the capacity of the hardware stack.
$swstack = 10                                               'Set
the capacity of the software stack.
$framesize = 24                                             'Set
the capacity of the frame area.
   '
   '  * PCF8574 I2C LCD Adapter settings *
   '
Const I2c_select = 1
'0:Software I2C , 1:TWI
#if I2c_select = 0
   '------[For software I2C]------
   Config I2cdelay = 10                                     'SCL
clock frequency = approx. 42KHz. (At AVR clock 16MHz) (* Maximum
100KHz)
   Config Scl = Portd.2                                     'Set
the port pin to connect the SCL line of the I2C bus.
   Config Sda = Portd.3                                     'Set
the port pin to connect the SDA line of the I2C bus.
   I2cinit
'Initialize the SCL and SDA lines of the I2C bus.
   '-------------------------------
#else
   '------[For TWI]-----------------
```

```
   $lib "i2c_twi.lib"
'Incorporate the hardware I2C/TWI library.
   Config Twi = 100000                                      'I2C
bus clock = 100KHz
   Config Scl = Portc.5                                     'You
must specify the SCL pin name.
   Config Sda = Portc.4                                     'You
must specify the SDA pin name.
   I2cinit
'Initialize the SCL and SDA lines of the I2C bus.
   '------------------------------
#endif
Dim Pcf8574_lcd As Byte : Pcf8574_lcd = &H4E             'PCF8574
slave address. (&H40,&H42,&H44,&H46,&H48,&H4A,&H4C,&H4E)
Dim Backlight As Byte : Backlight = 1                    'LCD
backlight control. (0: off, 1: on)
$lib "lcd_i2c_PCF8574.LIB"
'Incorporate the library of I2C LCD PCF8574 Adapter.
Config Lcd = 20x4                                        'Set the
LCD to 20 characters and 4 lines.
Initlcd
'Initialize the LCD.
   '
   '  * When installing the second and subsequent LCDs  *
   '
Pcf8574_lcd = &H4C                                       'The
slave address of the second PCF8574.
(&H40,&H42,&H44,&H46,&H48,&H4A,&H4C,&H4E)
Initlcd
'Initialize the second LCD.
   '
Pcf8574_lcd = &H4A                                       'The
slave address of the third PCF8574.
(&H40,&H42,&H44,&H46,&H48,&H4A,&H4C,&H4E)
Initlcd
'Initialize the third LCD.

   '
   '  ****************
   '  * Display test *
   '  ****************
   '
Pcf8574_lcd = &H4E                                       'Specify
the first LCD.
   '
Locate 1 , 1                                             'Display
of title.
Lcd "PCF8574"
   '
```

```
Locate 2 , 2
Lcd "I2C LCD Adapter"
   '
Deflcdchar 2 , &H02 , &H04 , &H0C , &H1E , &H0F , &H06 , &H04 , &
H08       'Write the custom character [Lightning] to the LCD.
Locate 1 , 15                                                 'Display
custom characters.
Lcd Chr(2) ; "1"
   '
Locate 1 , 9                                                  'Display
the slave address of PCF8574.
Lcd "[" ; Hex(pcf8574_lcd) ; "]"
   '
   '  * Second LCD *
   '
Pcf8574_lcd = &H4C                                            'Specify
the second LCD.
   '
Locate 1 , 1                                                  'Display
of title.
Lcd "PCF8574"
   '
Locate 2 , 2
Lcd "I2C LCD Adapter"
   '
Deflcdchar 3 , &H02 , &H04 , &H0C , &H1E , &H0F , &H06 , &H04 , &
H08       'Write the custom character [Lightning] to the LCD.
Locate 1 , 15                                                 'Display
custom characters.
Lcd Chr(3) ; "2"
   '
Locate 1 , 9                                                  'Display
the slave address of PCF8574.
Lcd "[" ; Hex(pcf8574_lcd) ; "]"
   '
   '  * Third LCD *
   '
Pcf8574_lcd = &H4A                                            'Specify
the third LCD.
   '
Locate 1 , 1                                                  'Display
of title.
Lcd "PCF8574"
   '
Locate 2 , 4
Lcd "I2C LCD Adapter"
   '
Deflcdchar 4 , &H02 , &H04 , &H0C , &H1E , &H0F , &H06 , &H04 , &
H08       'Write the custom character [Lightning] to the LCD.
```

```
Locate 1 , 19                                               'Display
custom characters.
Lcd Chr(4) ; "3"
   '
Locate 1 , 9                                                'Display
the slave address of PCF8574.
Lcd "[" ; Hex(pcf8574_lcd) ; "]"
   '
Locate 3 , 3
Lcd "-- 3rd Line --"
   '
Locate 4 , 4
Lcd "20x4 Display "
   '
Locate 4 , 20                                               'Display
the cursor.
Cursor On , Blink
End
```

8.23 AVR-DOS

8.23.1 AVR-DOS File System

AVR-DOS is a Disk Operating System (DOS) for Atmel AVR microcontroller.
The AVR-DOS file system is written by Josef Franz Vögel. He can be contacted via the BASCOM forum.
Josef has put a lot of effort in writing and especially testing the routines.

Topics of AVR-DOS File System:
1. Introduction
2. Important Steps to configure AVR-DOS
3. Requirements
4. Steps to get started with an ATMEGA (and with MMC.lib)
5. Getting started with an ATMEGA and ATXMEGA with MMCSD_HC.LIB
6. Memory Usage of DOS – File System
7. Error Codes
8. Buffer Status: Bit definitions of Buffer Status Byte (Directory, FAT and File)
9. Validity of the file I/O operations regarding the opening modes
10. SD and SDHC specs and pin-out
11. Example **1** for getting started with an ATMEGA and ATXMEGA with MMCSD_HC.LIB
12. Example **1**: Following the Config_MMCSD_HC.INC which is included in the main example program
13. Example **1**: Following the Config_AVR-DOS.inc which is included in the main example program
14. Example **2**: SD and SDHC Card Analysis Example Demo program
(Show the Card Capacity and the Card-Register CSD, CID, OCR and SD_Status)

Introduction

AVR-DOS provide the needed libraries to handle:
- The file system like open and/or create a file, send to or read from a file (Binary files and ASCII files)
- Interface functions (drivers) for Compact Flash 1823, hard disk, SD-Cards, SDHC (also

microSD or microSDHC). See SD and SDHC pinout below.

See also: New CF-Card Drivers 1827, Elektor CF-Interface 1825

The Filesystem works with:

- FAT16 formatted partitions
- FAT32 formatted partitions
- Short file name (8.3)
- Files with a long file name can be accessed by their short file name alias
- Files in Root Directory. The root dir can store 512 files. Take in mind that when you use long file names, less filenames can be stored.
- Files in Root directory and sub directories
- LBA mode (Logical block addressing) which is a linear addressing scheme where blocks are located by an integer index.

SD-card is a further development of the former MMC (Multi Media Card).
FAT = File Allocation Table and is the name of the file system architecture (FAT16 means 16-Bit version of FAT).

An SD or SDHC card is working at 2.7V ... 3.6V so for ATMEGA running at 5V you need a voltage converter or voltage divider. ATXMEGA are running at 2.7V ... 3.6V anyway so you can connect the sd-card direct to the ATXMEGA pin's.

 It is very important to use a proper level converter when using high clock rates (above 8 MHz). When using a FET/resistor as a level converter make sure there is enough pull up for a proper clock pulse.

Everything is written in Assembler to ensure a fast and compact code.
The intention in developing the DOS – file system was to keep close to the equivalent VB functions.

 Note that it is not permitted to use the AVR-DOS file system for commercial applications without the purchase of a license. A license comes with the ASM source. You can buy a user license that is suited for most private users.
When you develop a commercial product with AVR-DOS you need the company license. The ASM source is shipped with both licenses.

 Josef nor MCS Electronics can be held responsible for any damage or data loss of your memory cards or disk drives.

FAT16-FAT32

In the root-directory of a FAT16, you have a maximum of 512 directory entries. This limit is defined in the Partition Boot sector. In a FAT16 subdirectory there is a limit of 65535 (2^16 - 1) entries. This Limit depends of the type of the directory entry pointer used in AVR-DOS and can not be increased.

On a FAT32 Partition you have in all kind of directories (Root and Sub) the limit of 65535 entries like the FAT16 Subdirectory.

Please take into account, that long-File-Name Entries will use more than one Directory-Entry depending on the length of the file-name.
So if you use a card with files created on a PC, there a normally more Directory Entries used, than the count of file names.

Important Steps to configure AVR-DOS

1. Driver interface Library (select one of the following):
   For compactFlash:
```
$include "Config_CompactFlash_ElektorIF_M128.bas"
$include "Config_CompactFlash_M128.bas"
```

   For Hard Drives:
```
$include "Config_HardDisk.bas"
```

   For SD-Cards:
```
$include "Config_MMC.bas"
```

   For SD-cards and SDHC cards (works also with ATXMEGA !):
```
$include "config_MMCSD_HC.inc"
```

2. After calling the Driver interface library you need check the Error Byte which is **Gbdriveerror** and which is output of the function DRIVEINIT() 781. If the output is 0 (no error) you can include the AVR-DOS configuration file. Otherwise you should output the error number.

```
If    Gbdriveerror = 0 Then
  $include "Config_AVR-DOS.inc"
End If
```

3. In case of **Gbdriveerror = 0** (No Error) you can Initialize the file system with INITFILESYSTEM 792(1) where 1 is the partition number. For the Error Output var you need to dim a byte variable like `Dim Btemp1 As Byte` wbefore you call the Initfilesystem.

```
Btemp1 = Initfilesystem(1)
```

With Btemp1 = 0 (no error) the **Filesystem is successfully initialized** and you can use all other AVR-DOS functions like open, close, read and write.
Functions like PUT 1402, GET 1262, SEEK-Set 1434 only work when the file is opened in binary mode for example: `Open "test.bin" For Binary As #2`

When you want change (ejecting from the card socket) the SD-card (during the AVR is running other code than AVR-DOS) you need to call DRIVEINIT() 781 and INITFILESYSTEM 792(1) again in order to reset the AVR-Hardware (PORTs, PINs) attached to the Drive,reset the Drive again and initialize the file system again.

 When you include a Constant named SHIELD like : CONST SHIELD=1 , the CS line is kept active which is required for some W5100/W5200 shields.

Requirements:

- Software: appr. 2K-Word Code-Space (4000 Bytes in flash)
- SRAM: 561 Bytes for File system Info and DIR-Handle buffer
- 517 Bytes if FAT is handled in own buffer (for higher speed), otherwise it is handled with the DIR Buffer
- 534 Bytes for each File handle
- This means that a ATMEGA644, ATMEGA128 or ATXMEGA have enough memory for it.
- Even an ATMEGA32 could work but you really need to know what you do and you need to fully understand the settings in Config_AVR-DOS.BAS to reduce the amount of SRAM used by AVR-DOS (which will also affect AVR-DOS performance)

For example by setting `Const Cfilehandles = 1` and handling of FAT- and DIR-Buffer in one SRAM buffer with 561 bytes). You will not have much SRAM left anyway for other tasks in the ATMEGA32 and you can not expect maximum performance. $HWSTACK 648, $SWSTACK 705 and $FRAMESIZE 641 also needs to be set carefully.

```
' Count of file-handles, each file-handle needs 524 Bytes of SRAM
Const Cfilehandles = 1                                          ' [default = 2]

' Handling of FAT-Buffer in SRAM:
' 0 = FAT- and DIR-Buffer is handled in one SRAM buffer with 561 bytes
' 1 = FAT- and DIR-Buffer is handled in separate SRAM buffers with 1078 bytes
' Parameter 1 increased speed of file-handling
Const Csepfathandle = 0                                         ' [default = 1]
```

In the Main.bas you also need a Filename like `Dim File_name As String * 12`

With the above configuration and with the filename there is approximately 500 byte SRAM left in an ATMEGA32 for other tasks. Or in other words AVR-DOS needs at least **1500 Byte SRAM** in this case. To get detailed values compile your AVR-DOS application and open the **Bascom-AVR compiler Report (CTRL+W)** then you see the value with *Space left : 508 Bytes* (then you have 508 Bytes left for other tasks).

Then you can log data with for example:

```
Wait 4

Open File_name For Append As #100
    Print #100 , "This is what I log to SD-Card !"
Close #100
```

When you change now `Const Csepfathandle = 1` then you will get an OUT OF SRAM space message from the compiler with an ATMEGA32 which indicates that this will not work with an ATMEGA32.

- Other chips have too little internal memory. You could use XRAM memory too to extend the RAM.
- SPI Interface for SD and SDHC cards (can be used in hardware and software SPI mode where hardware SPI mode is faster)

TTo get started there are Examples in the ...BASCOM-AVR\SAMPLES\avrdos folder.

Steps to get started with an ATMEGA (and with MMC.lib):

The MMC.lib is for SD-Cards (Standard SD-Cards usually up to 2Gbyte and not for SDHC cards)

1. Open **Test_DOS_Drive.bas**
2. Add $HWSTACK, $SWSTACK and $FRAMESIZE
3. Add the hardware driver you want to use (for example for SD-Card this is `$include "Config_MMC.bas"`)
4. Open the Config_MMC.bas file and configure the SPI interface (hardware or software SPI and which Pin's for example for SPI chip select should be used. Config_MMC.bas will call the **MMC.lib** library which is located in the ...BASCOM-AVR\LIB folder.
5. Then you will find in **Test_DOS_Drive.bas** the Include AVR-DOS Configuration and library (`$include "Config_AVR-DOS.BAS"`). Config_AVR-DOS.BAS can be also found in ... BASCOM-AVR\SAMPLES\avrdos folder.
6. In Config_AVR-DOS.BAS you can change the AVR-DOS user settings like the number of

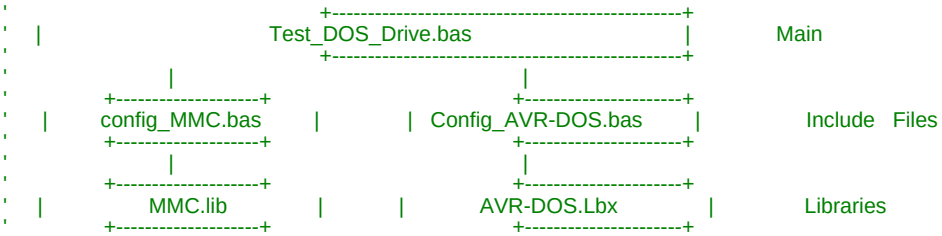
file handles or if AT- and DIR-Buffer is handled in one SRAM buffer or in different SRAM buffer. With this settings you can balance between SRAM space used and speed/performance of AVR-DOS.

File System Configuration in **CONFIG_AVR-DOS.BAS**

cFileHandles:	Count of File handles: for each file opened at same time, a file handle buffer of 534 Bytes is needed
cSepFATHandle:	For higher speed in handling file operations the FAT info can be stored in a own buffer, which needs additional 517 Bytes. Assign Constant cSepFATHandle with 1, if wanted, otherwise with 0.

7. Config_AVR-DOS.BAS will call **AVR-DOS.Lbx** library which is located in the ... BASCOM-AVR\LIB folder.
8. Compile, flash and run **Test_DOS_Drive.bas**

Files used in the **Test_DOS_Drive.bas** example:



Getting started with an ATMEGA and ATXMEGA with MMCSD_HC.LIB:

The **mmcsd_hc.lib** can be found in the ...BASCOM-AVR\LIB folder.

This library support:
- SD-Cards (also known as SDSC Cards = Secure Digital Standard-Capacity usually up to 2 GByte (also microSD)
- SDHC cards (Secure Digital High Capacity) cards start at 2Gbyte up to 32GByte. You can also use micro SDHC cards.
- It works with ATMEGA and ATXMEGA chips.
- See also : MMCSD_HC.LIB 1823

See ATXMEGA example program below.

Memory Usage of DOS – File System:

1. General File System information (need 35 Byte in SRAM)

Variable Name	Type	Usage
gbDOSError	Byte	holds DOS Error of last file handling routine
gbFileSystem	Byte	File System Code from Master Boot Record
glFATFirstSector	Long	Number of first Sector of FAT Area on the Card

gbNumberOfFATs	Byte	Count of FAT copies
gwSectorsPerFat	Word	Count of Sectors per FAT
glRootFirstSector	Long	Number of first Sector of Root Area on the Card
gwRootEntries	Word	Count of Root Entries
glDataFirstSector	Long	Number of first Sector of Data Area on the Card
gbSectorsPerCluster	Byte	Count of Sectors per Cluster
gwMaxClusterNumber	Word	Highest usable Cluster number
gwLastSearchedCluster	Word	Last cluster number found as free
gwFreeDirEntry	Word	Last directory entry number found as free
glFS_Temp1	Long	temporary Long variable for file system
gsTempFileName	String * 11	temporary String for converting file names

2. Directory (need 559 Byte in SRAM)

Variable Name	Type	Usage
gwDirRootEntry	Word	number of last handled root entry
glDirSectorNumber	Long	Number of current loaded Sector
gbDirBufferStatus	Byte	Buffer Status
gbDirBuffer	Byte (512)	Buffer for directory Sector

3. FAT (need 517 Byte in SRAM)
FAT Buffer is only allocated if the constant: cSepFATHandle = 1

Variable Name	Type	Usage
glFATSectorNumber	Long	Number of current loaded FAT sector
gbFATBufferStatus	Byte	Buffer status
gbFATBuffer	Byte(512)	buffer for FAT sector

4. File handling

Each file handle has a block of 534 Bytes in the variable abFileHandle which is a byte-array of size (534 * cFileHandles)

Variable Name	Type	Usage
FileNumber	Byte	File number for identification of the file in I/O operations to the opened file
FileMode	Byte	File open mode
FileRootEntry	Word	Number of root entry
FileFirstCluster	Word	First cluster
FATCluster	Word	cluster of current loaded sector
FileSize	Long	file size in bytes
FilePosition	Long	file pointer (next read/write) 0-based
FileSectorNumber	Long	number of current loaded sector
FileBufferStatus	Byte	buffer Status

FileBuffer	Byte(512)	buffer for the file sector
SectorTerminator	Byte	additional 00 Byte (string terminator) for direct reading ASCII files from the buffer

Error Codes:

Code	Compiler – Alias	Remark
0	cpNoError	No Error
1	cpEndOfFile	Attempt behind End of File
17	cpNoMBR	Sector 0 on Card is not a Master Boot Record
18	cpNoPBR	No Partition Sector
19	cpFileSystemNotSupported	Only FAT16 File system is supported
20	cpSectorSizeNotSupported	Only sector size of 512 Bytes is supported
21	cpSectorsPerClusterNotSupported	Only 1, 2, 4, 8, 16, 32, 64 Sectors per Cluster is supported. This are values of normal formatted partitions. Exotic sizes, which are not power of 2 are not supported
22	Cpcountofclustersnotsupported	The number of clusters is not supported
33	cpNoNextCluster	Error in file cluster chain
34	cpNoFreeCluster	No free cluster to allocate (Disk full)
35	cpClusterError	Error in file cluster chain
49	cpNoFreeDirEntry	Directory full
50	cpFileExist	File exists
51	Cpfiledeletenotallowed	File may not be deleted
52	Cpsubdirectorynotempty	Sub directory not empty.You can not delete sub folders which contain files
53	Cpsubdirectoryerror	Sub directory error
54	Cpnotasubdirectory	
65	cpNoFreeFileNumber	No free file number available, only theoretical error, if 255 file handles in use
66	cpFileNotFound	File not found
67	cpFileNumberNotFound	No file handle with such file number
68	cpFileOpenNoHandle	All file handles occupied
69	cpFileOpenHandleInUse	File handle number in use, can't create a new file handle with same file number
70	cpFileOpenShareConflict	Tried to open a file in read and write modus in two file handles
71	cpFileInUse	Can't delete file, which is in use
72	cpFileReadOnly	Can't open a read only file for writing

73	cpFileNoWildCardAllowed	No wildcard allowed in this function
74	Cpfilenumberinvalid	Invalid file number
97	cpFilePositionError	
98	cpFileAccessError	function not allowed in this file open mode
99	cpInvalidFilePosition	new file position pointer is invalid (minus or 0)
100	cpFileSizeToGreat	File size to great for function BLoad
&HC0	Cpdrivererrorstart	

Buffer Status: Bit definitions of Buffer Status Byte (Directory, FAT and File)

Bit	DIR	FAT	File	Compiler Alias	Remark
0 (LSB)			●	dBOF	Bottom of File (not yet supported)
1			●	dEOF	End of File
2			●	dEOFinSector	End of File in this sector (last sector)
3	●	●	●	dWritePending	Something was written to sector, it must be saved to Card, before loading next sector
4		●		dFATSector	This is an FAT Sector, at writing to Card, Number of FAT copies must be checked and copy updated if necessary
5			●	dFileEmpty	File is empty, no sector (Cluster) is allocated in FAT to this file

Validity of the file I/O operations regarding the opening modes

	Open mode			
Action	**Input**	**Output**	**Append**	**Binary**
Attr	●	●	●	●
Close	●	●	●	●
Put				●
Get				●
LOF	●	●	●	●
LOC	●	●	●	●
EOF	●	1)	1)	●
SEEK	●	●	●	●
SEEK-Set				●
Line Input	●			●

Print		●	●	●
Input	●			●
Write		●	●	●

1) Position pointer is always at End of File

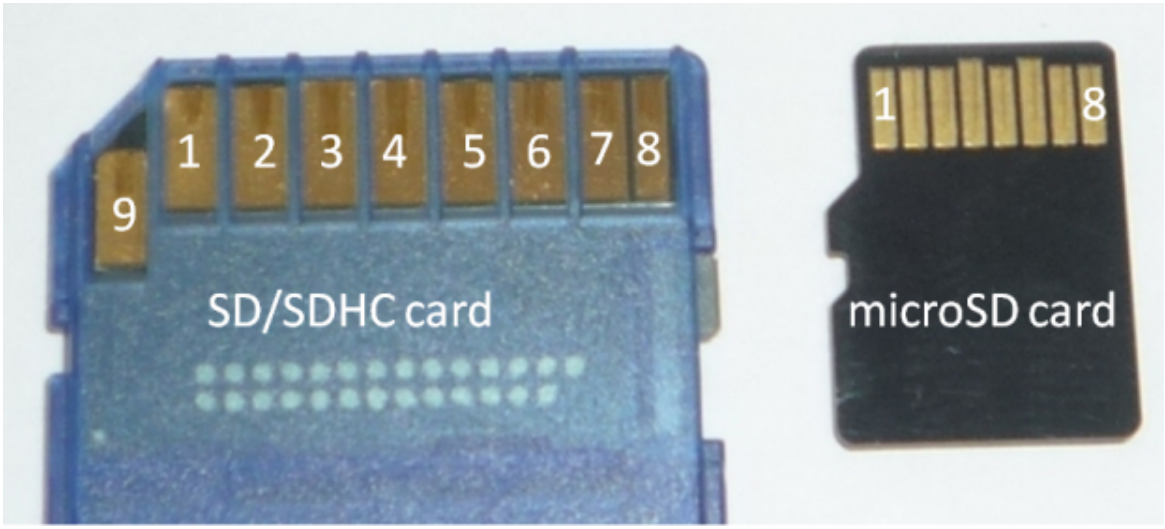
Supported statements and functions:

INITFILESYSTEM 792 , OPEN 1386 , CLOSE 848, FLUSH 790 , PRINT 1501, LINE INPUT 794, LOC 795, LOF 796 , EOF 785 , FREEFILE 791 , FILEATTR 786 , SEEK 1434 , BSAVE 774 , BLOAD 773 , KILL 793 , DISKFREE 778 , DISKSIZE 779 , GET 1262 , PUT 1402 ,FILEDATE 787 , FILETIME 789 , FILEDATETIME 787 , DIR 777 , WRITE 801 , INPUT 1493 , FILELEN 788

SD and SDHC specs and pin-out: (also microSD and microSD pin-out for SPI mode):

SD/SDHC Specs:

- SD and SDHC Cards offer a cost-effective and way to store large amounts of data on a removable memory and is ideal for data logging applications.
- SDHC has a different protocol than SD card with standard Capacity (therefore there was different libraries available at the beginning)
- Standard SD-Cards have a byte addressing. SDHC-Cards have sector-addressing like hard-disks and CF-Cards. One Sector is a portion of 512Bytes. SD cards and SDHC cards also have differences in the protocol at initializing the card, which can be used to check, which kind of card is inserted.
- SD Card operating range: 2.7V...3.6V. So you need a voltage level converter to connect a 5V micro to a SD-card.
- SD cards can be controlled by the six line SD card interface containing the signals: CMD,CLK,DAT0~DAT3 however this is not supported with AVR-DOS.
- AVR-DOS support the SPI interface which can be easily used with the hardware SPI interface of ATMEGA and ATXMEGA. (Software SPI is also supported).
- The SPI mode is active if the CS signal is asserted (negative) during the reception of the reset command (CMD0) which will be automatically handled by AVR-DOS
- The advantage of the SPI mode is reducing the host design in effort.
- With the Chip Select you can also connect several SPI slaves to one SPI interface
- Endurance: Usually SD or SDHC cards can handle typical up to 100,000 writes for each sector. Reading a logical sector is unlimited. Please take care when writing to SD cards in a loop.
- A typical SD Card current consumption should be between 50mA .... 80mA but should not exceed 200mA



Picture: Backside of SD/SDHC card and microSD card

SD/SDHC card pin out:

Pin #	Description for SPI mode	Connect to Pin on ATMEGA128	Connect to Pin on ATXMEGA128A1
1	Chip Select (SS) (Active low)	SS (PortB 0) (Active low)	SS (example for SPIC) PortC 4 (Active low)
2	DI (Data In)	MOSI (PortB 2)	MOSI (example for SPIC) PortC 5
3	GND	GND	GND
4	Vdd (Supply Voltage)	Supply Voltage (2.7V...3.6V)	Supply Voltage (2.7V...3.6V)
5	Clock	SCK (PortB 1)	SCK (example for SPIC) PortC 7
6	GND	GND	GND
7	D0 (Data Out)	MISO (PortB 3)	MISO (example for SPIC) PortC 6
8	Reserved	- - -	- - -
9	Reserved	- - -	- - -

Depending on the used SD-card (or microSD) socket you can also detect if the card is inserted or ejected (for this you need an additional pin on the micro).
In some cases it is best practise to spend another pin able to switch on and off the power to the SD-card socket (e.g. over a transistor or FET). In this case you can cycle power from the AVR when the sd-card controller hangs.
It is also best practise in some cases when you open a file for append, write the data to it and close it right after this so there is no open file where data could be corrupted by an undefined external event.

microSD card pin out (same as microSDHC pin-out):

Pin #	microSD Description for SPI mode
1	Reserved
2	Chip Select (SS)
3	DI (Data In)
4	Vdd (Supply Voltage)
5	Clock
6	GND
7	DO (Data Out)
8	Reserved

Formatting


It turns out that using windows to format a memory card can lead to problems.
It is strongly advised to use the special format tool from sdcard.org !
You may download it here : https://www.sdcard.org/downloads/formatter_4/
It is important to set the 'Overwrite Format' option.

It seems amazing that windows format (quick or full) can give other results but it was extensively tested.

Example 1 for getting started with an ATMEGA and ATXMEGA with MMCSD_HC.LIB:

```
'-----------------------------------------------------------------------------
' Filename:              XMEGA_AVR-DOS_SDHC.BAS
' Library needed:       MMCSD_HC.LIB --> Place MMCSD_HC.LIB in the LIB-Path of BASCOM-AVR
installation
'                            MMCSD_HC.LIB will be called from config_MMCSD_HC.inc
'                          AVR-DOS.Lbx
' Include file:          config_MMCSD_HC.inc     (will be called from XMEGA_AVR-DOS_SDHC.BAS)
' Used ATXMEGA:         ATXMEGA128A1
' Used SPI Port:           Port D (you can also use Software SPI)
'-----------------------------------------------------------------------------
'
'    File      Structure:
'
'                                  +-------------------------------------------------+
'  |                  XMEGA_AVR-DOS_SDHC.BAS                      |         Main
'                                  +-------------------------------------------------+
'                |                                  |
'          +--------------------+                   +----------------------+
'    |  config_MMCSD_HC.inc|           |  Config_AVR-DOS.inc       |             Include Files
'          +--------------------+                   +----------------------+
'                |                                  |
'          +--------------------+                   +----------------------+
'   |       MMCSD_HC.LIB       |        |         AVR-DOS.Lbx          |          Libraries
'          +--------------------+                   +----------------------+
'
'
'  Terminal output of following example (with hardware SPI over Port.D):
'
' Used SD-Card: 4GByte SDHC Card
'
'
' (

---Example for using a SDHC-Card with AVR-DOS and XMEGA---
Starting... SDHC with ATXMEGA....

SD Card Type = SDHC Spec. 2.0 or later

Init File System ... OK --> Btemp1= 0 / Gbdriveerror = 0
Filesystem = 6
FAT Start Sector: 8196
Root Start Sector: 8688
Data First Sector: 8720
Max. Cluster Nummber: 62794
Sectors per Cluster: 128
Root Entries: 512
Sectors per FAT: 246
Number of FATs: 2


Write to file done !
File length = 46
This is my 1 first Text to File with XMEGA !
write to file
Total bytes written: 10200
Write and Readback test done !
Dir function demo
LOGGER.TXT 01\01\01 01:00:00 3120
MY_FILE.TXT 01\01\01 01:00:00 46
```

```
TEST.TXT     01\01\01     01:00:00     10200

Diskfree   =   4018560
Disksize   =   4018752

' )


$regfile = "xm128a1def.dat"
$crystal = 32000000                                          '32MHz
$hwstack = 128
$swstack = 128
$framesize = 128


Config Osc =   Disabled ,  32mhzosc = Enabled                '32MHz
Config   Sysclock = 32mhz                                    '32Mhz
Config Priority =    Static ,   Vector =     Application ,  Lo = Enabled          'config          interrupts
Enable Interrupts

'=====[  Serial  Interface  to  PC  =  COM5  ]=======================================
Config Com5 = 57600 , Mode =   Asynchroneous ,    Parity = None ,    Stopbits = 1 ,    Databits = 8
Open "COM5:" For Binary As #2
Waitms 1

Print #2 ,
Print #2 , "---Example  for  using  a  SDHC-Card  with  AVR-DOS  and  XMEGA---"

'=====[   Global    Vars    ]============================================================
Dim  Btemp1 As Byte                                          '  Needed  for  Fat  Drivers
Dim      Input_string As String * 100
Dim      Output_string As String * 100
Dim     File_handle As Byte
Dim   File_name As String * 14
Dim X As Long



Print #2 , "Starting...    SDHC    with    ATXMEGA...."
Print #2 ,

'-------------------------------------------------------------------------------

'=====[     Includes     ]============================================================


$include "config_MMCSD_HC.inc"

Print #2 , "SD  Card  Type  =  " ;
Select Case   Mmcsd_cardtype
  Case 0 :  Print #2 , "can't   init   the   Card"
  Case 1 :  Print #2 , "MMC"
  Case 2 :  Print #2 , "SDSC  Spec.  1.x  "
  Case 4 :  Print #2 , "SDSC  Spec.  2.0  or  later"
  Case 12 :  Print #2 , "SDHC  Spec.  2.0  or  later"
End Select

Print #2 ,


If     Gbdriveerror = 0 Then                                 'from....      Gbdriveerror      =
Driveinit()
    $include "Config_AVR-DOS.inc"                            '  Include  AVR-DOS
Configuration      and      library


    Print #2 , "Init   File   System   ...   " ;
   Btemp1 = Initfilesystem(1)                                '  Reads  the  Master  boot  record
and  the  partition  boot  record  (Sector)  from  the  flash  card  and  initializes  the  file  system
                                                                        '1   =   Partitionnumber
   If Btemp1 <> 0 Then
      Print #2 , "Error:     " ;  Btemp1 ;  "   at   Init   file   system"
   Else
      Print #2 , "  OK   -->  Btemp1=  " ;  Btemp1 ;  "   /   Gbdriveerror   =   " ;      Gbdriveerror
      Print #2 , "Filesystem   =   " ;      Gbfilesystem
      Print #2 , "FAT   Start   Sector:   " ;          Glfatfirstsector
      Print #2 , "Root   Start   Sector:   " ;          Glrootfirstsector
      Print #2 , "Data   First   Sector:   " ;          Gldatafirstsector
      Print #2 , "Max.   Cluster   Nummber:   " ;      Glmaxclusternumber
      Print #2 , "Sectors   per   Cluster:   " ;       Gbsectorspercluster
      Print #2 , "Root   Entries:   " ;      Gwrootentries
      Print #2 , "Sectors   per   FAT:   " ;         Glsectorsperfat
      Print #2 , "Number   of   FATs:   " ;     Gbnumberoffats
   End If

   Print #2 ,
```

```
    Print #2 ,

      '-----------------------------------------------------------------------
      '  Write  Text  to  file
         File_handle = Freefile( )                          '  get  a  file  handle
        File_name = "My_file.txt"
      Open  File_name For Output As #file_handle            '  open  file  for  output  with
file_handle
      '  If  the  file  exist  already,  the  file  will  be  overwritten  !
      Print #file_handle , "This  is  my  1  first  Text  to  File  with  XMEGA  !"
      Close #file_handle

      Print #2 , "Write  to  file  done  !"

      '-----------------------------------------------------------------------
      'Now  we  want  to  read  back  the  text  we  wrote  to  file  and  print  it  over  Serial
Interface
         File_handle = Freefile( )
      Open "My_file.txt" For Input As #file_handle          '  we  can  use  a  constant  for
the  file  too
      Print #2 , "File  length  =  " ; Lof( #file_handle)
      Line Input #file_handle , Input_string                '  read  a  line
      Print #2 , Input_string                               'print  the  line
      Close #file_handle



        'WRITE  TO  FILE
      Print #2 , "write  to  file"
       File_name = "Test.txt"
         Input_string =
"12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
01234567890"


      Open  File_name For Output As #10

        While X < 10000                                     '10000  *  102  Byte  /  100  =
10200  Byte
           Print #10 ,  Input_string
           X = X + 100
        Wend

      Close #10


       X = Filelen(file_name)
       Print #2 , "Total  bytes  written:  " ; X


       'READ  FROM  FILE

       Open  File_name For Input As #10
        While Eof(#10) = 0
          Line Input #10 , Output_string                    '  read  a  line
          If  Input_string <> Output_string Then
             Print #2 , "Buffer  Error!  near  byte:  " ; X ; "  " ; " [ " ;  Output_string ; " ] "
             Waitms 2000
          End If
        Wend
      Close #10


      Print #2 , "Write  and  Readback  test  done  !"


     '-----------------------------------------------------------------------
      'Print  the  file  name  which  was  created  before
      Print #2 , "Dir  function  demo"
          Input_string = Dir( "*.*")
      'The  first  call  to  the  DIR()  function  must  contain  a  file  mask  The  *  means
everything.
      '  Print  File  Names
      While Len(input_string) > 0                           '  if  there  was  a  file  found
         Print #2 ,  Input_string ; "  " ; Filedate( ) ; "  " ; Filetime( ) ; "  " ; Filelen
( )
          '  print  file  ,  the  date  the  fime  was  created/changed  ,  the  time  and  the  size  of
the  file
         Input_string = Dir( )                              '  get  next
      Wend

      '-----------------------------------------------------------------------
```

```
      Print #2 ,
      Print #2 , "Diskfree   =   " ; Diskfree( )
      Print #2 , "Disksize   =   " ; Disksize( )

 End If                                                    'If   Gbdriveerror   =   0   Then

End                                                        'end   program
```

Example 1: Following the Config_MMCSD_HC.INC which is included in the main example program:

```
$nocompile

'-----------------------------------------------------------------------------
'                              Config_MMCSD_HC.INC
'                        Config File for MMC/SD/SDHC Flash Cards Driver
'                 (c) 1995-2025 , MCS Electronics / Vögel Franz Josef
'-----------------------------------------------------------------------------
' Place MMCSD_HC.LIB in the LIB-Path of BASCOM-AVR installation
'
' you can vary MMC_CS on HW-SPI and all pins on SOFT-SPI, check settings
'
' ========== Start of user definable range ====================================
'
' Declare here you SPI-Mode
' using HW-SPI:          cMMC_Soft = 0
Const    Hardware_spi = 0
' not using HW_SPI:  cMMC_Soft = 1
Const     Software_spi = 1

Const  Cmmc_soft =    Hardware_spi

#if  Cmmc_soft = 0

'      ---------      Start      of      Section      for      HW-SPI      ---------------------------------------

   'Port D of ATXMEGA is used in this example as SPI Interface    to SD-Card

        Portd_pin6ctrl = &B00_011_000                          'Enable Pullup for MISO Pin

   ' Define here Slave Slect (SS) Pin of Hardware SPI
   Config Pind.4 = Output                                      ' define here Pin for CS of
MMC/SD Card
   Mmc_cs Alias Portd.4
   Set Mmc_cs

   ' Define here Slave Slect (SS) Pin of Hardware SPI
   Config Pind.4 = Output                                      ' define here Pin of SPI SS
    Spi_ss Alias Portd.4
   Set Spi_ss                                                  ' Set SPI-SS to Output and
High por Proper work of


   'FOR XMEGA DEVICES
   #if _xmega = 1
       'SPI Configuration for XMEGA
       'Used Library = $LIB "MMCSD_HC.LIB"


       'Portd.4    SS      --> SD-Card    Slave Select
       'Portd.5    MOSI  --> SD-Card    MISO
       'Portd.6    MISO  --> SD-Card    MOSI
       'Portd.7    CLK    --> SD-Card    Clock

       Config Spid = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk2 , Data_order = Msb
       Open "SPID" For Binary As #14
       Const _mmc_spi =    Spid_ctrl
   #else
```

```
    ' HW-SPI is configured to highest Speed
    Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes , Polarity = High
, Phase = 1 , Clockrate = 4 , Noss = 1
'   Spsr = 1                                        ' Double speed on ATMega128
    Spiinit
    #endif

'       ---------       End       of       Section       for       HW-SPI       ----------------------------------------

#else                                                   ' Config here SPI pins, if not
using HW SPI

'       ---------       Start       of       Section       for       Soft-SPI       ----------------------------------------

    ' Chip Select Pin   => Pin 1 of MMC/SD
    Config Pind.4 = Output
    Mmc_cs Alias Portd.4
    Set Mmc_cs

    ' MOSI - Pin   => Pin 2 of MMC/SD
    Config Pind.5 = Output
    Set Pind.5
     Mmc_portmosi Alias Portd
    Bmmc_mosi Alias 5

    ' MISO - Pin   => Pin 7 of MMC/SD
    Config Pind.6 = Input
     Mmc_portmiso Alias Pind
    Bmmc_miso Alias 6

    ' SCK - Pin   => Pin 1 of MMC/SD
    Config Pind.7 = Output
    Set Pind.7
     Mmc_portsck Alias Portd
    Bmmc_sck Alias 7

'       ---------       End       of       Section       for       Soft-SPI       ----------------------------------------

#endif

'  ==========  End  of  user  definable  range  ======================================


'====  Variables  For  Application  ================================================
 Dim  Mmcsd_cardtype As Byte                            ' Information about the type
of the Card
'       0  can't  init  the  Card
'    1  MMC
'     2  SDSC  Spec.  1.x
'      4  SDSC  Spec.  2.0  or  later
'    12  SDHC  Spec.  2.0  or  later

Dim     Gbdriveerror As Byte                            ' General Driver Error
register
'  Values  see  Error-Codes
'=============================================================================


'  ====  Variables  for  Debug  ===============================================
'  You  can  remove  remarks(')  if  you  want  check  this  variables  in  your  application
Dim     Gbdrivestatusreg As Byte                        ' Driver save here Card
response
'  Dim  gbDriveErrorReg  as  Byte  at  GbdriveStatusReg  overlay          '
' Dim gbDriveLastCommand as Byte                          ' Driver save here Last
Command to Card
Dim   Gbdrivedebug As Byte
' Dim MMCSD_Try As Byte                                  ' how often driver tried to
initialized the card
'=============================================================================


'====  Driver  internal  variables  ================================================
'  You  can  remove  remarks(')  if  you  want  check  this  variables  in  your  application
'  Dim  _mmcsd_timer1  As  Word
'  Dim  _mmcsd_timer2  As  Word
'=============================================================================


'     Error-Codes
Const       Cperrdrivenotpresent = &HE0
Const       Cperrdrivenotsupported = &HE1
Const          Cperrdrivenotinitialized = &HE2

Const       Cperrdrivecmdnotaccepted = &HE6
```

```
Const      Cperrdrivenodata = &HE7

Const       Cperrdriveinit1 = &HE9
Const       Cperrdriveinit2 = &HEA
Const       Cperrdriveinit3 = &HEB
Const       Cperrdriveinit4 = &HEC
Const       Cperrdriveinit5 = &HED
Const       Cperrdriveinit6 = &HEE

Const      Cperrdriveread1 = &HF1
Const      Cperrdriveread2 = &HF2

Const       Cperrdrivewrite1 = &HF5
Const       Cperrdrivewrite2 = &HF6
Const       Cperrdrivewrite3 = &HF7
Const       Cperrdrivewrite4 = &HF8


$lib "MMCSD_HC.LIB"
$external _mmc
'   Init   the   Card
Gbdriveerror = Driveinit( )


'   you   can   remark/remove   following   two   Code-lines,   if   you   dont't   use   MMCSD_GetSize()
$external    Mmcsd_getsize
Declare Function    Mmcsd_getsize( ) As Long


'   you   can   remark/remove   following   two   Code-lines,   if   you   dont't   use   MMCSD_GetCSD()
'   write   result   of   function   to   an   array   of   16   Bytes
$external   Mmcsd_getcsd
Declare Function   Mmcsd_getcsd( ) As Byte


'   you   can   remark/remove   following   two   Code-lines,   if   you   dont't   use   MMCSD_GetCID()
'   write   result   of   function   to   an   array   of   16   Bytes
$external    Mmcsd_getcid
Declare Function    Mmcsd_getcid( ) As Byte


'   you   can   remark/remove   following   two   Code-lines,   if   you   dont't   use   MMCSD_GetOCR()
'   write   result   of   function   to   an   array   of   4   Bytes
$external   Mmcsd_getocr
Declare Function   Mmcsd_getocr( ) As Byte


'   you   can   remark/remove   following   two   Code-lines,   if   you   dont't   use   MMCSD_GetSDStat
'   write   result   of   function   to   an   array   of   64   Bytes
$external     Sd_getsd_status
Declare Function     Sd_getsd_status( ) As Byte

'   check   the   usage   of   the   above   functions   in   the   sample   MMCSD_Analysis.bas
'   check   also   the   MMC   and   SD   Specification   for   the   content   of   the   registers   CSD,   CID,   OCR
and   SDStat
```

Example 1: Following the Config_AVR-DOS.inc which is included in the main example program:

```
$nocompile
'    Config    File-System    for    Version    5.5:

'   ===   User   Settings   ============================================================

'   Count   of   file-handles,   each   file-handle   needs   524   Bytes   of   SRAM
Const      Cfilehandles = 2                                                    '   [default   =   2]

'   Handling   of   FAT-Buffer   in   SRAM:
'   0   =   FAT-   and   DIR-Buffer   is   handled   in   one   SRAM   buffer   with   561   bytes
'   1   =   FAT-   and   DIR-Buffer   is   handled   in   separate   SRAM   buffers   with   1078   bytes
'    Parameter   1   increased   speed   of   file-handling
Const      Csepfathandle = 1                                                   '   [default   =   1]
```

```
' Handling of pending FAT and Directory information of open files
' 0 = FAT and Directory Information is updated every time a data sector of the file is
updated
' 1 = FAT and Directory Information is only updated at FLUSH and SAVE command
' Parameter 1 increases writing speed of data significantly
Const Cfatdirsaveatend = 1 ' [default = 1]


' Surrounding String with Quotation Marks at the Command WRITE
' 0 = No Surrounding of strings with quotation.marks
' 1 = Surrounding of strings with quotation.marks (f.E. "Text")
Const Ctextquotationmarks = 1 ' [default = 1]


' Write second FAT. Windows accepts a not updated second FAT
' PC-Command: chkdsk /f corrects the second FAT, it overwrites the
' second FAT with the first FAT
' set this parameter to 0 for high speed continuing saving data
' 0 = Second FAT is not updated
' 1 = Second FAT is updated if exist
Const Cfatsecondupdate = 1 ' [default = 1]


' Character to separate ASCII Values in WRITE - statement (and INPUT)
' Normally a comma (,) is used. but it can be changed to other values, f.E.
' to TAB (ASCII-Code 9) if EXCEL Files with Tab separated values should be
' written or read. This parameter works for WRITE and INPUT
' Parameter value is the ASSCII-Code of the separator
' 44 = comma [default]
' 9 = TAB ' [default = 44]
Const Cvariableseparator = 44



' === End of User Setting ==========================================================


' === Variables for AVR-DOS ========================================================

' FileSystem Basis Informationen
Dim Gldrivesectors As Long
Dim Gbdoserror As Byte

' Master Boot Record
Dim Gbfilesystem As Byte
' Partition Boot Record
Dim Gbfilesystemstatus As Byte
Dim Glfatfirstsector As Long
Dim Gbnumberoffats As Byte
Dim Glsectorsperfat As Long
Dim Glrootfirstsector As Long
Dim Gwrootentries As Word
Dim Gldatafirstsector As Long
Dim Gbsectorspercluster As Byte
Dim Glmaxclusternumber As Long
Dim Gllastsearchedcluster As Long

' Additional info
Dim Glfs_temp1 As Long

' Block für Directory Handling

Dim Gldirfirstsectornumber As Long

Dim Gwfreedirentry As Word
Dim Glfreedirsectornumber As Long

Dim Gsdir0tempfilename As String * 11
Dim Gwdir0entry As Word ' Keep together with next,
otherwise change _DIR
Dim Gldir0sectornumber As Long

Dim Gstempfilename As String * 11
Dim Gwdirentry As Word
Dim Gldirsectornumber As Long
Dim Gbdirbufferstatus As Byte
Dim Gbdirbuffer(512) As Byte
Const C_filesystemsramsize1 = 594
#if Csepfathandle = 1
Dim Glfatsectornumber As Long
Dim Gbfatbufferstatus As Byte
Dim Gbfatbuffer(512) As Byte
Const C_filesystemsramsize2 = 517
#else
```

```
Const       C_filesystemsramsize2 = 0
#endif

'  File  Handle  Block
Const    Co_filenumber = 0
Const    Co_filemode = 1
Const       Co_filedirentry = 2 : Const       Co_filedirentry_2 = 3
Const        Co_filedirsectornumber = 4
Const          Co_filefirstcluster = 8
Const      Co_filesize = 12
Const       Co_fileposition = 16
Const       Co_filesectornumber = 20
Const         Co_filebufferstatus = 24
Const       Co_filebuffer = 25
Const       C_filehandlesize =       Co_filebuffer + 513                  '  incl.  one  Additional  Byte
for  00  as  string  terminator
                                                                          '  for  direct  text  reading
from       File-buffer
Const       C_filehandlesize_m = 65536 -  C_filehandlesize               '  for  use  with  add  immediate
word  with  subi,  sbci
                                                                          '  =  minus  c_FileHandleSize  in
Word-Format

Const       C_filehandlessize =       C_filehandlesize *       Cfilehandles


Dim      Abfilehandles( c_filehandlessize)  As Byte
Const       C_filesystemsramsize =       C_filesystemsramsize1 +       C_filesystemsramsize2 +
C_filehandlessize


'  End  of  variables  for  AVR-DOS  ================================================

'  Definitions  of  Constants  ======================================================

'  Bit  definiton  for  FileSystemStatus

Dfilesystemstatusfat Alias 0 : Const       Dfilesystemstatusfat = 0          ' 0 = FAT16, 1 = FAT32
Dfilesystemsubdir Alias 1 : Const       Dfilesystemsubdir = 1       '  0  =  Root-Directory,  1  =
Sub-Directory
Const       Dmfilesystemsubdir = (2 ^       Dfilesystemsubdir)              '  not  used  yet
Const       Dmfilesystemdirincluster = (2 ^       Dfilesystemstatusfat + 2 ^       Dfilesystemsubdir)       '
not  used  yet
Dfatsecondupdate Alias 7 : Const    Dfatsecondupdate = 7          '  Bit-position  for  parameter
o f
                                                                   '  Update  second  FAT  in
gbFileSystemStatus


'  Bit  Definitions  for  BufferStatus  (FAT,  DIR,  File)

Deof Alias 1 : Const Deof = 1 : Const Dmeof = (2 ^  Deof)
Deofinsector Alias 2 : Const    Deofinsector = 2 : Const    Dmeofinsector = (2 ^     Deofinsector)
Dwritepending Alias 3 : Const    Dwritepending = 3 : Const    Dmwritepending = (2 ^     Dwritepending
)
Dfatsector Alias 4 : Const    Dfatsector = 4 : Const    Dmfatsector = (2 ^     Dfatsector)          '
For  Writing  Sector  back  (FATNumber  times)
Dfileempty Alias 5 : Const    Dfileempty = 5 : Const    Dmfileempty = (2 ^     Dfileempty)

'  New  feature  for  reduce  saving
Dfatdirwritepending Alias 6 : Const       Dfatdirwritepending = 6 : Const       Dmfatdirwritepending
= (2 ^       Dfatdirwritepending)
Dfatdirsaveatend Alias 7 : Const     Dfatdirsaveatend = 7 : Const     Dmfatdirsaveatend = (2 ^
Dfatdirsaveatend)
Dfatdirsaveanyway Alias 0 : Const     Dfatdirsaveanyway = 0 : Const     Dmfatdirsaveanyway = (2 ^
Dfatdirsaveanyway)



Const    Dmeofall = (2 ^  Deof + 2 ^     Deofinsector)
Const  Dmeof_empty = (2 ^  Deof + 2 ^     Deofinsector + 2 ^     Dfileempty)


Const          Cp_fatbufferinitstatus = (2 ^     Dfatsector)
Const          Cp_dirbufferinitstatus = 0


#if     Cfatdirsaveatend = 1
Const          Cp_filebufferinitstatus = (2 ^      Dfatdirsaveatend)
#else
Const          Cp_filebufferinitstatus = 0
#endif
```

```
#if Cfatsecondupdate = 0
  Const Cp_fatsecondupdate = (2 ^ Dfatsecondupdate)
#else
  Const Cp_fatsecondupdate = 0
#endif


' Bit definitions for FileMode (Similar to DOS File Attribut)
Dreadonly Alias 0 : Const Dreadonly = 0
'Const cpFileReadOnly = &H21                ' Archiv and read-only Bit set
Const Cpfilewrite = &H20                    ' Archiv Bit set


' Error Codes

' Group Number is upper nibble of Error-Code
' Group 0 (0-15): No Error or File End Information
Const Cpnoerror = 0
Const Cpendoffile = 1

' Group 1 (17-31): File System Init
Const Cpnombr = 17
Const Cpnopbr = 18
Const Cpfilesystemnotsupported = 19
Const Cpsectorsizenotsupported = 20
Const Cpsectorsperclusternotsupported = 21
Const Cpcountofclustersnotsupported = 22

' Group 2 (32-47): FAT - Error
Const Cpnonextcluster = 33
Const Cpnofreecluster = 34
Const Cpclustererror = 35
' Group 3 (49-63): Directory Error
Const Cpnofreedirentry = 49
Const Cpfileexists = 50
Const Cpfiledeletenotallowed = 51
Const Cpsubdirectorynotempty = 52
Const Cpsubdirectoryerror = 53
Const Cpnotasubdirectory = 54
' Group 4 (65-79): File Handle
Const Cpnofreefilenumber = 65
Const Cpfilenotfound = 66
Const Cpfilenumbernotfound = 67
Const Cpfileopennohandle = 68
Const Cpfileopenhandleinuse = 69
Const Cpfileopenshareconflict = 70
Const Cpfileinuse = 71
Const Cpfilereadonly = 72
Const Cpfilenowildcardallowed = 73
Const Cpfilenumberinvalid = 74              ' Zero is not allowed

' Group 7 (97-127): other errors
Const Cpfilepositionerror = 97
Const Cpfileaccesserror = 98
Const Cpinvalidfileposition = 99
Const Cpfilesizetogreat = 100

Const Cpdrivererrorstart = &HC0


' Range 224 to 255 is reserved for Driver

' Other Constants
' File Open Mode / stored in File-handle return-value of Fileattr(FN#, [1])
Const Cpfileopeninput = 1                   ' Read
Const Cpfileopenoutput = 2                  ' Write sequential
'Const cpFileOpenRandom = 4                 ' not in use yet
Const Cpfileopenappend = 8                  ' Write sequential; first set
Pointer to end
Const Cpfileopenbinary = 32                 ' Read and Write; Pointer can
be changed by user


' permission Masks for file access routine regarding to the file open mode
Const Cfilewrite_mode = &B00101010          ' Binary, Append, Output
Const Cfileread_mode = &B00100001           ' Binary, Input
Const Cfileseekset_mode = &B00100000        ' Binary
Const Cfileinputline = &B00100001           ' Binary, Input
Const Cfileput_mode = &B00100000            ' Binary
Const Cfileget_mode = &B00100000            ' Binary

' Directory attributs in FAT16/32
Const Cpfileopenallowed = &B00100001        ' Read Only and Archiv may be
set
Const Cpfiledeleteallowed = &B00100000
Const Cpfilesearchallowed = &B00111101      ' Do no search hidden Files
```

```
' Bit 0 = Read Only
' Bit 1 = Hidden
' Bit 2 = System
' Bit 3 = Volume ID
' Bit 4 = Directory
' Bit 5 = Archiv
' Long File name has Bit 0+1+2+3 set
Dim Lastdosmem As Byte


$lib "AVR-DOS.Lbx"
```

- - - END of EXAMPLE 1 - - -

Example 2: SD and SDHC Card Analysis Example Demo program (Show the Card Capacity and the Card-Register CSD, CID, OCR and SD_Status):

This example uses: $include "Config_MMCSD_HC.bas" which calls following Libary: $lib "MMCSD_HC.LIB"
This example is written for ATMEGA but is also working for ATXMEGA devices.

```
'-----------------------------------------------------------------------------
'                                MMCSD_Analysis.BAS
'                               Test MMC / SD Card
'               (c) 1995-2025 , MCS Electronics / Vögel Franz Josef
'-----------------------------------------------------------------------------
' Test MMC / SD Card
' Show the Card Capacity and the Card-Register CSD, CID, OCR and SD_Status
' First you have to init the Card in the File Config_MMCSD_HC.bas with
'  $Include "Config_MMCSD_HC.bas"
' All Card registers are written with the MSB first to the Byte-array
' f.E. CSD(1) contains then MSB (Bit 120-127) of the CSD-Register

$regfile = "M644pdef.dat"
$crystal = 16000000

$hwstack = 100
$swstack = 100
$framesize = 100


$baud = 57600


Config Serialin = Buffered , Size = 20
Config Clock = Soft

Enable Interrupts

Config Date = Dmy , Separator = .
Print "Test_Dos_Drive compiled at " ; Version( )
$include "Config_MMCSD_HC.bas"




Dim Xc As Byte                                              ' for Print - counter
Dim Xd As Byte                                              ' for Print - Counter

Print "Start of Card Analysis"
Print "Last Drive-Error-Code = " ; Gbdriveerror
Print "Gbdrivestatusreg =" ; Gbdrivestatusreg

' Check detected Card Type
Select Case Mmcsd_cardtype
   Case 1
      Print "MMC-Card detected"
   Case 2
      Print "SD-Card Spec. 1.x detected"
   Case 4
      Print "SD-Card Spec. 2.0 detected"
   Case 12
```

```
      Print "SD-Card Spec. 2.0 High Capacity detected"
   Case Else
      Print "No Card detected"
End Select

If Mmcsd_cardtype > 0 Then

' check the CSD Register

   Dim Csd(16) As Byte
   Print "Get CSD"
   Csd(1) = Mmcsd_getcsd()
   If Gbdriveerror <> 0 Then
      Print "Error at reading CSD"
   Else
      For Xc = 1 To 16
         Print Hex(csd(xc)) ; " " ;
      Next
      Print " "
   End If

' Get the Card Capacity from the CSD Register

   Dim Mmcsd_size As Long
   Print "Get Card Capacity [KB]"
    Mmcsd_size = Mmcsd_getsize()
   If Gbdriveerror <> 0 Then
      Print "Error at reading CSD"
   Else
      Print "Card Capacity = ; " ; Mmcsd_size ; "kb (1KB=1024 Bytes)"
   End If

' Get the CID Register

   Dim Cid(16) As Byte
   Print "Get CID"
    Cid(1) = Mmcsd_getcid()
   If Gbdriveerror <> 0 Then
      Print "Error at reading CID"
   Else
      For Xc = 1 To 16
         Print Hex(cid(xc)) ; " " ;
      Next
      Print " "
   End If

' Get the OCR Register

   Dim Ocr(4) As Byte
   Print "Get OCR"
    Ocr(1) = Mmcsd_getocr()
   If Gbdriveerror <> 0 Then
      Print "Error at reading OCR"
   Else
      For Xc = 1 To 4
         Print Hex(ocr(xc)) ; " " ;
      Next
      Print " "
   End If

   If Mmcsd_cardtype > 1 Then

' Get the SD_Status Register on SD-Cards

      Dim Sd_status(64) As Byte
      Print "Get SD_Status"
        Sd_status(1) = Sd_getsd_status()
      If Gbdriveerror <> 0 Then
         Print "Error at reading SD_Status"
      Else
        For Xc = 1 To 64
           Print Hex(sd_status(xc)) ; " " ;
           Xd = Xc Mod 8
           If Xd = 0 Then
              Print " "
           End If
        Next
     End If
   End If
End If


Print "End of Card Analysis"

End
```

XTINY EXAMPLE

```
'-----------------------------------------------------------------
---------------
'name                     : FlashCard-Demo-XTINY.BAS
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : demonstrates AVR-DOS on XYTINY
platform
'micro                    : avr128da28
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------
---------------
'    This sample file shows how the AVR-DOS library from Josef
Franz Vögel
'    can be used.
'-----------------------------------------------------------------
---------------
$regfile = "AVRX128da28.dat"

$crystal = 24000000
'speed
$hwstack = 128
'stack
$swstack = 128
$framesize = 128

'The AVRX series have more oscillator options
Config Osc = Enabled , Frequency = 24mhz

'set the system clock and prescaler
Config Sysclock = Int_osc , Prescale = 1

'set up the COM por/USART connected to PA0 and PA1
Config Com1 = 115200 , Mode = Asynchroneous , Parity = None ,
Databits = 8 , Stopbits = 1

$external Waitms                                            ' one
of the libs needs the waitms code

Declare Sub Directorylist(pstr1 As String , Byval Pdays As Word)
     'declare sub

Config Date = Dmy , Separator = DOT                         ' the
file system uses date and times
Config Clock = Soft                                         '
real clock installed
Enable Interrupts
```

```
Time$ = "12:00:00" : Date$ = "16.04.23"                    ' set
the time and date
'when you have a clock, the date and time of the clock will be
used

Print "Lib version : " ; Ver()

'include the driver configuration file this file will setup the
SPI
$include "config_MMCSD_HC_XTINY.inc"
'include specific AVR-DOS settings
$include "CONFIG_AVR-DOS.inc"                              '
some constants

' Init Port and Card
Print "Setup Port and Reset Card ... ";

If Drivecheck() = 0 Then                                   ' we
have a card detected
   Print "OK"
   _temp1 = Driveinit()                                    '
init the drive
Else
   Print "Card not inserted, check Card!"
   End                                                     ' end
program
End If

' The card is now setup and the low level card driver routine
could be used
' such as ReadSector, WriteSector etc.
' We use the AVD-DOS Filesystem which is compatible with
DOS/Windows
' Make sure your CF card is 32MB or bigger. It needs to be
formatted with FAT 16
' Smaller cards can only be formatted with FAT12
' of course you can also use FAT32

Print "Init File System ...  ";

Dim Gbtemp1 As Byte                                        '
scratch byte
Gbtemp1 = Initfilesystem(1)                                ' we
must init the filesystem once
If Gbtemp1 > 0 Then
   Print "Error " ; Gbtemp1
Else
   Print "OK"
```

```
   Print "Disksize : " ; Disksize()                                    '
show disk size in bytes
   Print "Disk free: " ; Diskfree()                                    '
show free space too
End If


'dim some test variables
Dim S As String * 60 , Fl As String * 12 , Ff As Byte
Dim Sdatetime As String * 18
Fl = "test.txt"
S = "test this"

'Now we are getting to it
'We can specify a file handle with #1  or #2 etc. or we can ask
for a free
'file handle with the FreeFile function. It will return a free
handle if there is one.

Ff = Freefile()                                                      ' get
a file handle

'With this file handle we refer to a file
Open Fl For Output As #ff                                            '
open fikle for output
'  we need to open a file  before we can use the file commands
'  we open it for OUTPUT, INPUT , APPEND or BINARY
'  In this case we open it for OUTPUT because we want to write to
the file.
'  If the file existed, the file would be overwritten.
Print #ff , S                                                        '
print some data
Print #ff , S
Print #ff , S
Print #ff , "A constant" ; S
Close #ff                                                            '
close file

'A file opened if OUTPUT mode is convenient to write string data
too
'The next beta will support WRITE too

'We now created a file that contains 3 lines of text.
'We want to append some data to it
S = "this is appended"
Open Fl For Append As #150                                           ' we
specify the file number now
Print #150 , S
Close #150
```

```
'Ok we want to check if the file contains the written lines
Ff = Freefile()                                             ' get
file handle
Open "test.txt" For Input As #ff                            ' we
can use a constant for the file too
Print Lof(#ff) ; "  length of file"
Print Fileattr(#ff) ; " file mode"                          '
should be 1 for input
Do
   LineInput #ff , S                                        '
read a line
  ' line input is used to read a line of text from a file
   Print S                                                   '
print on terminal emulator
Loop Until Eof(ff) <> 0
'The EOF() function returns a non-zero number when the end of the
file is reached
'This way we know that there is no more data we can read
Close #ff

Ddemo:
'Lets have a look at the file we created
   Print "Dir function demo"
   S = Dir( "*.*")
'The first call to the DIR() function must contain a file mask
' The * means everything.
'
   While Len(s) > 0                                         '
if there was a file found
      Print S ; "   " ; Filedate() ; "   " ; Filetime() ; "   " ;
Filelen()
   '   print file , the date the fime was created/changed , the
time and the size of the file
      S = Dir()                                             '
get next
   Wend
'Wait 3

'We can use the KILL statement to delete a file.
'A file mask is not supported
   Print "Kill (delete) file demo"
   Kill "test.txt"

   S = Dir( "*.TXT")                                        '
check for TXT files
   While Len(s) > 0
      Print S ; "   " ; Filedate() ; "   " ; Filetime() ; "   " ;
```

```
Filelen()
      S = Dir()                                                    '
get next
   ' the next call to the DIR function must not specify the mask
so we get the next file
   Wend
'Wait 3


'for the binary file demo we need some variables of different
types
   Dim B As Byte , W As Word , L As Long , Sn As Single , Ltemp
As Long
   Dim Stxt As String * 10
   B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt = "test"

'open the file in BINARY mode
   Open "test.biN" For Binary As #2
   Put #2 , B                                                   '
write a byte
   Put #2 , W                                                   '
write a word
   Put #2 , L                                                   '
write a long
   Ltemp = Loc(#2) + 1                                          '
get the position of the next byte
   Print Ltemp ; " LOC"                                         '
store the location of the file pointer
   Print Lof(#2) ; "  length of file"
   Print Seek(#2) ; " file pointer"                             '
now you understand difference between loc and seek function
   Print Fileattr(#2) ; " file mode"                            '
should be 32 for binary
   Put #2 , Sn                                                  '
write a single
   Put #2 , Stxt                                                '
write a string

   Flush #2                                                     '
flush to disk
   Close #2

'now open the file again and write only the single
   Open "test.bin" For Binary As #2
   Seek #2 , Ltemp                                              '
set the filepointer
   Sn = 1.23                                                    '
change the single value so we can check it better
   Put #2 , Sn
```

```
   L = 1
'specify the file position
   B = Seek(#2 , L)                                          '
reset is the same as using SEEK #2,L
   Get #2 , B                                                '
get the byte
   Get #2 , W                                                '
get the word
   Get #2 , L                                                '
get the long
   Get #2 , Sn                                               '
get the single
   Get #2 , Stxt                                             '
get the string
   Close #2


'now we send to the RS-232 port the values we read back from the
file
   Print B                                                   '
byte must be 1
   Print W                                                   '
word must be 50000
   Print L                                                   '
long must be 12345678
   Print Sn                                                  '
single must be 1.23
   Print Stxt                                                '
string must be test


'we can print to a file like we print to the RS-232
   Open "file.tst" For Output As #3
   Print #3 , "This is a test" ; B ; " " ; W ; " " ; L ; " " ; Sn
   Close #3

'read back the data
   Open "file.tst" For Input As #3
   LineInput #3 , S
   Print #1 , S
'send to terminal emulator
   Close #3


'now the good old bsave and bload
   Dim Ar(100) As Byte , I As Byte
   For I = 1 To 100
      Ar(i) = I                                              '
```

```
fill the array
   Next

   Wait 2

   W = Varptr(ar(1))
   Bsave "josef.img" , W , 100
   For I = 1 To 100
      Ar(i) = 0                                                    '
reset the array
   Next

   Bload "josef.img" , W                                           '
Josef you are amazing !

   For I = 1 To 10
      Print Ar(i) ; " " ;
   Next
   Print

   Print "File demo"
   Print Filelen( "josef.img") ; " length"                         '
length of file
   Print Filetime( "josef.img") ; " time"                          '
time file was changed
   Print Filedate( "josef.img") ; " date"                          '
file date

   Flush                                                           '
flush all open files
   Print "gbDOSerror " ; Gbdoserror
   Print "Finally an advanced dir demo"
   S = "*.*"                                                       '
return anything
   Directorylist S , 2                                             '
show all from the last 2 days

   S = "write"
   Open "write.dmo" For Output As #2
   Write #2 , S , W , L , Sn                                       '
write is also supported
   Close #2

   Open "write.dmo" For Input As #2
   Input #2 , S , W , L , Sn                                       '
write is also supported
   Close #2
   Print S ; "  " ; W ; " " ; L ; "  " ; Sn
```

```
' For singles take in mind that the result might differ a bit
because of conversion
   Sn = 123.45
   S = Str(sn)                                                '
create a string
   Print S                                                    '
show result of conversion
   Sn = Val(s)                                                '
convert from string to single
   Print Sn                                                   '
show result
'When storing singles, better use GET/PUT

   End



' Read and print Directory and show Filename, Date, Time, Size
' for all files matching pStr1 and create/update younger than
pDays
Sub Directorylist(pstr1 As String , Byval Pdays As Word)
   Local Lfilename As String * 12                             '
hold file name for print
   Local Lwcounter As Word , Lfilesizesum As Long             ' for
summary
   Local Lwnow As Word , Lwdays As Word
   Local Lsec As Byte , Lmin As Byte , Lhour As Byte , Lday As
Byte , Lmonth As Byte , Lyear As Byte
   Print "Listing of all Files matching " ; Pstr1 ; " and
create/last update date within " ; Pdays ; " days"
   Lwnow = Sysday()
   Lwcounter = 0 : Lfilesizesum = 0
   Lfilename = Dir(pstr1)
   While Lfilename <> ""
      Lsec = Filedatetime()
      Lwdays = Lwnow - Sysday(lday)                           '
Days between Now and last File Update; uses lDay, lMonth, lYear
      If Lwdays <= Pdays Then                                 '
days smaller than desired with parameter
         Print Lfilename ; "   " ; Filedate() ; " " ; Filetime() ;
 " " ; Filelen()
         Incr Lwcounter : Lfilesizesum = Filelen() + Lfilesizesum
      End If
      Lfilename = Dir()
   Wend
   Print Lwcounter ; " File(s) found with " ; Lfilesizesum ; "
Byte(s)"
End Sub
```

Some notes. As you can see all included files have the now preferred INC extension.
Besides the $regfile and code to setup the oscillator and system clock, the code is the same as for other processors.
The used include file for Xtiny platform differs because the SPI is named different.
You best have a look at the Config_MMCSD_HC_XTINY.inc file.
It can be used for the old AVR, the XMEGA and the Xtiny, MegaX and AVRX.

8.23.2 MMCSD_HC.LIB

The MMCSD_HC.LIB is an MMC SD-HC card driver library.
See the AVR-DOS topic for an example.

There is an optional constant you can set in your code :

CONST _CS_EXTENDED_PORT=1

You need to set this constant when using a normal AVR chip with the CS pin connected to an extended port. We recommend to use a normal port which allows the CBI/SBI instructions but some times it is required to use an extended port like PORTF on an MEGA2560. Since the extended port needs a register to read-alter-write a bit, the register R23 need to be saved in the lib. When you define the constant and give it a value of 1, the register is preserved.

You can always set this directive, it will only create unneeded code when using normal ports.

In version **2086** the LIB has been modified and this constant is no longer required or used.

Using MMC/SD card on a SPI bus with multiple devices

The MMC specification requires that clock pulses are sent to the MMC card with the CS line disabled !!!
It means that when another device is active, the clock pulses can confuse or even corrupt the card.
The solution is to use either a bus driver with tri-state , or to use a dedicated pin for the clock lines.
With XMega there are multiple SPI buses possible.
With normal AVR you can use HW SPI for the MMC, and use SHIFTIN/SHIFTOUT for the other SPI devices.

8.24 CF Card

8.24.1 Compact FlashCard Driver

The compact flash card driver library is written by Josef Franz Vögel. He can be contacted via the BASCOM user list.

Josef has put a lot of effort in writing and especially testing the routines.
Josef nor MCS Electronics can be held responsible for any damage or data loss of your CF-cards.

Compact flash cards are very small cards that are compatible with IDE drives. They work at 3.3V or 5V and have a huge storage capacity.

The Flash Card Driver provides the functions to access a Compact Flash Card.

At the moment there are six functions:
DriveCheck 780, DriveReset 782 , DriveInit 781 , DriveGetIdentity 780 , DriveWriteSector 784 , DriveReadSector 782

The Driver can be used to access the Card directly and to read and write each sector of the card or the driver can be used in combination with a file-system with basic drive access functions.

Because the file system is separated from the driver you can write your own driver.

This way you could use the file system with a serial EEPROM for example.

For a file system at least the functions for reading (DriveReadSector / _DriveReadSector) and writing (DriveWriteSector / _DriveWriteSector) must be provided. The preceding under slash _ is the label of the according asm-routine. The other functions can, if possible implemented as a NOP – Function, which only returns a No-Error (0) or a Not Supported (224) Code, depending, what makes more sense.

For writing your own Driver to the AVR-DOS File system, check the ASM-part of the functions-description.

Error Codes:

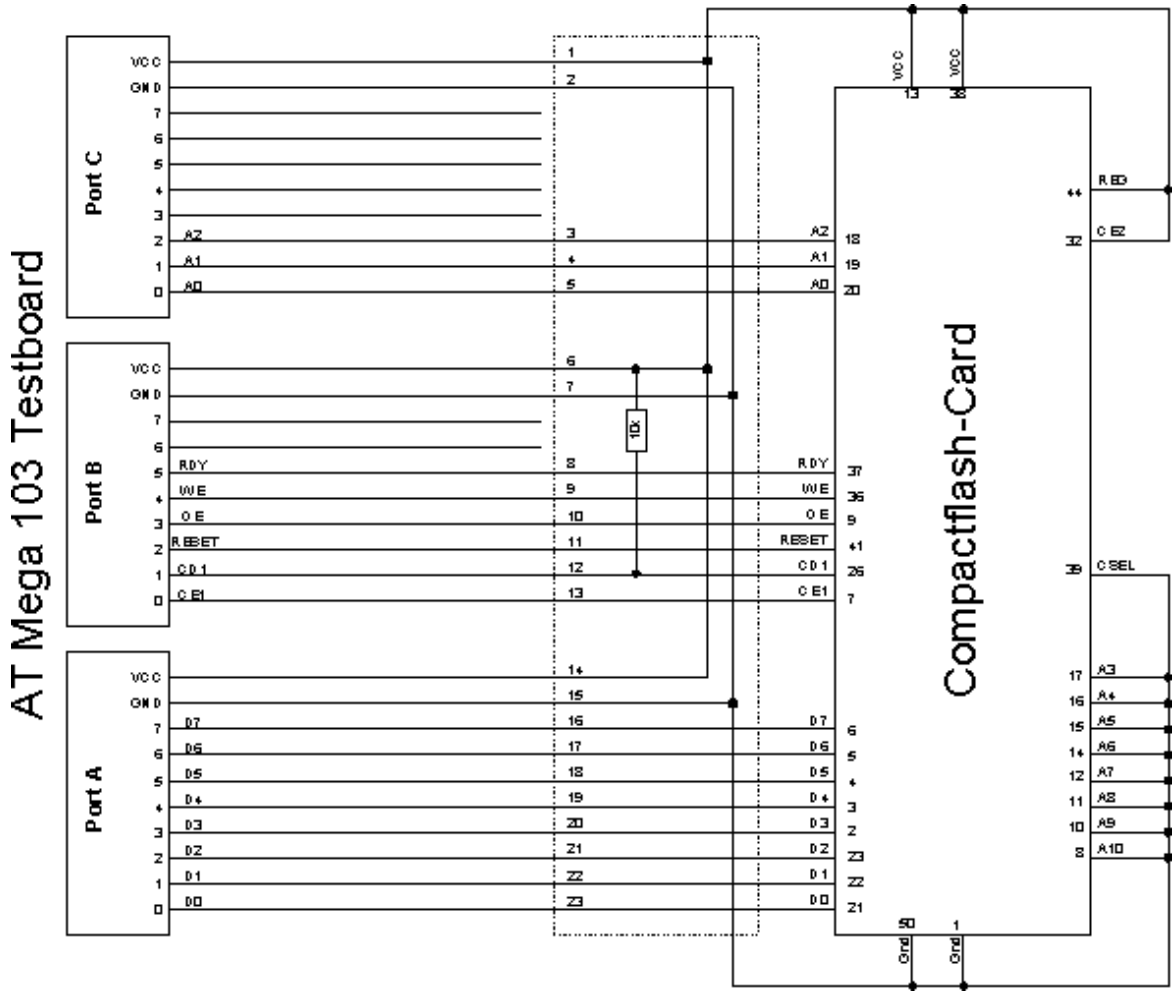
Code	Compiler – Alias	Remark
0	CpErrDriveNoError	No Error
224	cpErrDriveFunctionNotSupported	This driver does not supports this function
225	cpErrDriveNotPresent	No Drive is attached
226	cpErrDriveTimeOut	During Reading or writing a time out occurred
227	cpErrDriveWriteError	Error during writing
228	cpErrDriveReadError	Error during reading

At the MCS Web AN section you can find the application note 123.

More info about Compact Flash you can find at :

http://www.sandisk.com/download/Product%20Manuals/cf_r7.pdf

A typical connection to the micro is shown below.



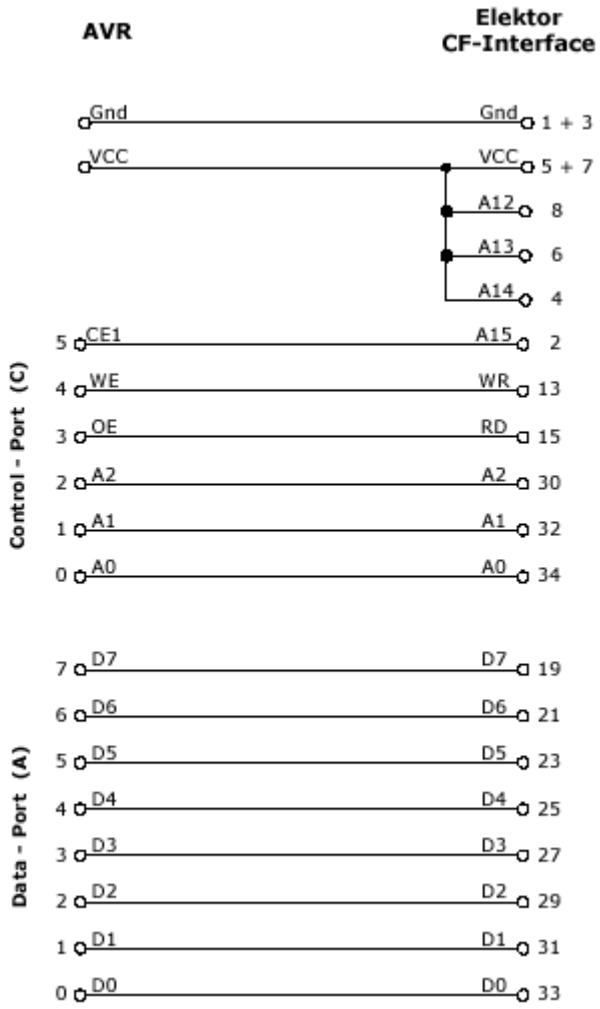
8.24.2 Elektor CF-Interface

The popular Electronics magazine Elektor, published an article about a CF-card interface. This interface was connected to an 89S8252. This interface can be used and will use little pins of the micro.

Note that because of the FAT buffer requirement, it is not possible to use a 8051 micro.,

At this moment, only the Mega128 and the Mega103 AVR micro's are good chips to use with AVR-DOS.

You can use external memory with other chips like the Mega162.



Changes of the hardware pins is possible in the file Config_FlashCardDrive_EL_PIN.bas.
The default library is FlashCardDrive.lib but this interface uses the library FlashCardDrive_EL_PIN.lib.

See also: AVR-DOS File System 1794

8.24.3 XRAM CF-Interface for simulation

The XRAM CF-Card interface is created for the purpose of testing the File System routines without hardware.

You can use an external RAM chip (XRAM) for the CF-interface but of course it is not practical in a real world application unless you backup the power with a battery.

For tests with the simulator it is ideal.

Just specify the Config_XRAMDrive.bas file and select a micro that can address external memory such as the M128. Then specify that the system is equipped with 64KB of external RAM.

You can now simulate the flashdisk.bas sample program !

In order to simulate Flashdisk.bas, set the constant XRAMDRIVE to 1. Then select 64KB of external RAM and compile.

8.24.4 New CF-Card Drivers

New CF-Card drivers can be made relatively simple.

Have a look at the supplied drivers.

There are always a few files needed :

- A config file in the format : CONFIG_XXX.bas
- FlashCardDrive_XXX.LIB
- FlashCardDrive_XXX.lbx is derived from the LIB file

XXX stands for the name of your driver.

At the AVR-DOS web you can find more drivers.

See also: AVR-DOS File System 1794

8.25 Floating Point

8.25.1 FP_TRIG

The FP_TRIG library is written by Josef Franz Vögel.

All trig functions are stored in fp_trig.lib library.
The fp_trig.lbx contains the compiled object code and is used by BASCOM.

This sample demonstrates all the functions from the library:

```
'-----------------------------------------------------------------------------------------
'name                                : test_fptrig2.bas
'copyright                           : (c) 1995-2025, MCS Electronics
'purpose                             : demonstates FP trig library from Josef Franz Vögel
'micro                               : Mega8515
'suited for demo                     : no
'commercial addon needed             : no
'-----------------------------------------------------------------------------------------

$regfile = "m8515.dat"                                      ' specify the used micro
$crystal = 4000000                                          ' used crystal frequency
$baud = 19200                                               ' use baud rate
$hwstack = 32                                               ' default use 32 for the
hardware stack
$swstack = 10                                               ' default use 10 for the SW
stack
$framesize = 40                                             ' default use 40 for the frame
space


Dim S1 As Single , S2 As Single , S3 As Single , S4 As Single , S5 As Single , S6 As
Single
Dim Vcos As Single , Vsin As Single , Vtan As Single , Vatan As Single , S7 As Single
Dim Wi As Single , B1 As Byte
Dim Ms1 As Single



Const Pi =  3.14159265358979

'calculate PI
Ms1 = Atn(1) * 4


Testing_power:
```

```
Print "Testing Power X ^ Y"
Print "X                Y              x^Y"
For S1 =  0.25 To 14 Step  0.25
    S2 = S1 \ 2
    S3 = Power(s1 , S2)
    Print S1 ; " ^ " ; S2 ; " = " ; S3
Next
Print : Print : Print

Testing_exp_log:

Print "Testing EXP and LOG"
Print "x          exp(x)                log([exp(x)])          Error-abs          Error-rel"
Print "Error is for calculating exp and back with log together"
For S1 = -88 To 88
    S2 = Exp(s1)
    S3 = Log(s2)
    S4 = S3 - S1
    S5 = S4 \ S1
    Print S1 ; "       " ; S2 ; "       " ; S3 ; "       " ; S4 ; "       " ; S5 ; "   " ;
    Print
Next
Print : Print : Print


Testing_trig:
Print "Testing COS, SIN and TAN"
Print "Angle Degree    Angle Radiant              Cos              Sin              Tan"
For Wi = -48 To 48
    S1 = Wi * 15
    S2 = Deg2rad(s1)
    Vcos = Cos(s2)
    Vsin = Sin(s2)
    Vtan = Tan(s2)
    Print S1 ; "       " ; S2 ; "      " ; Vcos ; "      " ; Vsin ; "      " ; Vtan
Next
Print : Print : Print


Testing_atan:
Print "Testing      Arctan"
Print "X        atan in Radiant,        Degree"
S1 = 1 / 1024
Do
    S2 = Atn(s1)
    S3 = Rad2deg(s2)
    Print S1 ; "          " ; S2 ; "          " ; S3
    S1 = S1 * 2
    If S1 > 1000000 Then
       Exit Do
    End If
Loop

Print : Print : Print

Testing_int_fract:
Print "Testing Int und Fract of Single"
Print "Value          Int              Frac"
S2 = Pi \ 10
For S1 = 1 To 8
    S3 = Int(s2)
    S4 = Frac(s2)
    Print S2 ; "      " ; S3 ; "      " ; S4
    S2 = S2 * 10
Next

Print : Print : Print

Print "Testing degree - radiant - degree converting"
Print "Degree      Radiant      Degree      Diff-abs      rel"

For S1 = 0 To 90
  S2 = Deg2rad(s1)
  S3 = Rad2deg(s2)
  S4 = S3 - S1
  S5 = S4 \ S1
  Print S1 ; "      " ; S2 ; "      " ; S3 ; "      " ; S4 ; "      " ; S5
Next

Testing_hyperbolicus:
Print : Print : Print
Print "Testing SINH, COSH and TANH"
```

```
Print "X              sinh(x)              cosh(x)              tanh(x)"
For S1 = -20 To 20
  S3 = Sinh(s1)
  S2 = Cosh(s1)
  S4 = Tanh(s1)
  Print S1 ; "     " ; S3 ; "     " ; S2 ; "     " ; S4
Next
Print : Print : Print

Testing_log10:
Print "Testing    LOG10"
Print "X              log10(x)"
S1 =  0.01
S2 = Log10(s1)
Print S1 ; "     " ; S2
S1 =  0.1
S2 = Log10(s1)
Print S1 ; "     " ; S2
For S1 = 1 To 100
  S2 = Log10(s1)
  Print S1 ; "     " ; S2
Next

Print : Print : Print


'test  MOD  on  FP
S1 =  10000
S2 = 3
S3 = S1 Mod S2
Print S3

Print "Testing_SQR-Single"
For S1 = -1 To 4 Step  0.0625
   S2 = Sqr(s1)
   Print S1 ; "   " ; S2
Next
Print
For S1 =  1000000 To  1000100
   S2 = Sqr(s1)
   Print S1 ; "   " ; S2
Next

Testing_atn2:
Print "Testing  Sin  /  Cos  /  ATN2  /  Deg2Rad  /  Rad2Deg  /  Round"
Print "X[deg]        X[Rad]          Sin(x)          Cos(x)          Atn2          Deg  of  Atn2        Rounded"
For S1 = -180 To 180 Step 5
   S2 = Deg2rad(s1)
   S3 = Sin(s2)
   S4 = Cos(s2)
   S5 = Atn2(s3 , S4)
   S6 = Rad2deg(s5)
   S7 = Round(s6)
   Print S1 ; "   " ; S2 ; "   " ; S3 ; "   " ; S4 ; "   " ; S5 ; "   " ; S6 ; "   " ; S7
Next
Print "note:   -180°   is   equivalent   to   +180°"
Print
Testing_asin_acos:
Print "Testing   ASIN,   ACOS"
Print "X              asin(x)              acos(x)"
   For S1 = -1.125 To  1.125 Step  0.0625
   S2 = Asin(s1)
   S3 = Acos(s1)
   Print S1 ; "   " ; S2 ; "   " ; S3
Next
Print "Note:  >  1.0  and  <  -1.0  are  invalid  and  shown  here  for  error  handling"


Testing_shift:
S1 = 12
For B1 = 1 To 20
   S2 = S1 : S3 = S1
   Shift S2 , Left , B1
   Shift S3 , Right , B1
   Print S1 ; "   " ; S2 ; "   " ; S3
Next

Print "End   of   testing"

End
```

Back

8.25.2 DOUBLE

The double.lbx (lib) is written by Josef Franz Vögel. The library supports the basic operations :

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Val() , INPUT
- Str() , PRINT
- Int()
- Frac()
- Fix()
- Round()
- Conversion from double to single and long
- Conversion from single and long to double

The double library uses special Mega instructions not available in all AVR chips. But as the old chips are not manufactured anymore, this should not be a problem.
In version 1.11.9.8 a software multiplication is performed so the trig functions can be used on any chip that has enough internal memory.
In the report file you can find out if your micro supports the HWMUL. the _HWMUL conststant is set to 1 in that case.
When software multiplication is used, the multiply routine needs more processor cycles. A number of trig functions depend on the multiplication code and as a result, they become more slow too.

All Trig() functions are supported by the double too!

8.26 I2C SLAVE

8.26.1 CONFIG I2CSLAVE

The I2C Slave Add-on started with a software emulation for TWI slave using an interrupts and timer. It supported a number of early processors.
When TWI was added to some of the processors, an additional TWI slave lib was added to the package.

With Xmega having up to 4 TWI/I2C interfaces, TWI slave support for Xmega has been added to the package in version 2077 build 3
Most tiny processors do not support TWI but USI. USI support is added as well in 2077 build 3.
So the add on comes with a number of I2C slave libraries.

See CONFIG I2CSLAVE 978 , CONFIG USI 1138, CONFIG TWISLAVE 1123 , CONFIG TWIXSLAVE 1128

8.26.2 I2C TWI Slave

The I2C-Slave library is intended to create I2C slave chips. This is an add-on library that is not included in Bascom-AVR by default. It is a commercial add on library. It is available from MCS Electronics
The I2C Slave add on can turn some chips into a I2C slave device. You can start your own chip plant this way.

Most new AVR chips have a so called TWI/I2C interface. As a customer of the I2C slave lib, you can get both libs.
The **i2cslave.lib** works in interrupt mode and is the best way as it adds less overhead and also less system resources.

With this add-on library you get both libraries:

- **i2cslave.lib** and **i2cslave.lbx** : This library is used for AVR's which have no hardware TWI/I2C interface like for example ATTINY2313 or ATTINY13. In this case TIMER0 and INT0 is used for SDA and SCL (Timer0 Pin = SCL, INT0 Pin = SDA). Only AVR' with TIMER0 and INT0 on the same port can use this library like for example ATTINY2313 or ATTINY13. The i2cslave.lbx is the compiled library version of i2cslave.lib. See also Config I2CISLAVE 978

- **i2c_TWI-slave.LBX** : This library can be used when an AVR have an TWI/I2C hardware interface like for example ATMEGA8, ATMEGA644P or ATMEGA128. In this case the hardware SDA and SCL pin's of the AVR will be used (with ATMEGA8: SCL is PORTC.5 and SDA is PORTC.4). This library will be used when USERACK = OFF. When USERACK =ON then **i2c_TWI-slave-acknack.LBX** will be used. See also Config TWISLAVE 1123

See also: Using the I2C protocol 297

8.27 SPI

8.27.1 SPISLAVE

SPISLAVE.LIB (LBX) is a library that can be used to create a SPI slave chip when the chip does not have a hardware SPI interface.
Although most AVR chips have an ISP interface to program the chip, the 2313 for example does not have a SPI interface.

When you want to control various micro's with the SPI protocol you can use the SPISLAVE library.

The SPI-softslave.bas sample from the samples directory shows how you can use the SPISLAVE library.
Also look at the spi-slave.bas sample that is intended to be used with hardware SPI.

The sendspi.bas sample from the samples directory shows how you can use the SPI hardware interface for the master controller chip.

```
'-------------------------------------------------------------------------
-----------------
'name                    : spi-softslave.bas
'copyright               : (c) 1995-2025, MCS Electronics
'purpose                 : shows how to implement a SPI SLAVE with
software
'micro                   : AT90S2313
'suited for demo         : yes
```

```
'commercial addon needed  : no
'-------------------------------------------------------------------------
-----------------

$regfile = "2313def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

'Some atmel chips like the 2313 do not have a SPI port.
'The BASCOM SPI routines are all master mode routines
'This example show how to create a slave using the 2313
'ISP slave code


'define the constants used by the SPI slave
Const _softslavespi_port = Portd                            ' we used
portD
Const _softslavespi_pin = Pind                              'we use the
PIND register for reading
Const _softslavespi_ddr = Ddrd                              ' data
direction of port D

Const _softslavespi_clock = 5                               'pD.5 is
used for the CLOCK
Const _softslavespi_miso = 3                                'pD.3 is
MISO
Const _softslavespi_mosi = 4                                'pd.4 is
MOSI
Const _softslavespi_ss = 2                                  ' pd.2 is SS
'while you may choose all pins you must use the INT0 pin for the SS
'for the 2313 this is pin 2

'PD.3(7),  MISO  must be output
'PD.4(8),  MOSI
'Pd.5(9) , Clock
'PD.2(6),  SS /INT0

'define the spi slave lib
$lib "spislave.lbx"
'sepcify wich routine to use
$external _spisoftslave

'we use the int0 interrupt to detect that our slave is addressed
On Int0 Isr_sspi Nosave
'we enable the int0 interrupt
Enable Int0
'we configure the INT0 interrupt to trigger when a falling edge is
detected
Config Int0 = Falling
'finally we enabled interrupts
Enable Interrupts

'
Dim _ssspdr As Byte                                         ' this is
out SPI SLAVE SPDR register
```

```
Dim _ssspif As Bit                                          ' SPI
interrupt revceive bit
Dim Bsend As Byte , I As Byte , B As Byte                   ' some other
demo variables

_ssspdr = 0                                                 ' we send a
0 the first time the master sends data
Do
   If _ssspif = 1 Then
   Print "received: " ; _ssspdr
   Reset _ssspif
   _ssspdr = _ssspdr + 1                                    ' we send
this the next time
   End If
Loop
```

When the chip has a SPI interface, you can also use the following example:

```
'-------------------------------------------------------------------------
------------------
'name                     : spi-slave.bas
'copyright                : (c) 1995-2025, MCS Electronics
'purpose                  : shows how to create a SPI SLAVE
'micro                    : AT90S8515
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------
------------------

$regfile = "8515def.dat"                                    ' specify
the used micro
$crystal = 3680000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 32                                               ' default
use 32 for the hardware stack
$swstack = 10                                               ' default
use 10 for the SW stack
$framesize = 40                                             ' default
use 40 for the frame space

' use together with sendspi.bas
'-----------------------------------------------------------------
' Tested on the STK500. The STK200 will NOT work.
' Use the STK500 or another circuit



Dim B As Byte , Rbit As Bit , Bsend As Byte

'First configure the MISO pin
Config Pinb.6 = Output                                      ' MISO

'Then configure the SPI hardware SPCR register
Config Spi = Hard , Interrupt = On , Data Order = Msb , Master = No ,
Polarity = Low , Phase = 0 , Clockrate = 128

'Then init the SPI pins directly after the CONFIG SPI statement.
Spiinit
```

```
'specify the SPI interrupt
On Spi Spi_isr Nosave

'enable global interrupts
Enable Interrupts

'show that we started
Print "start"
Spdr = 0                                                    ' start with
sending 0 the first time
Do
  If Rbit = 1 Then
    Print "received : " ; B
    Reset Rbit
    Bsend = Bsend + 1 : Spdr = Bsend                        'increase
SPDR
  End If
  ' your code goes here
Loop



'Interrupt routine
'since we used NOSAVE, we must save and restore the registers ourself
'when this ISR is called it will send the content from SPDR to the
master
'the first time this is 0
Spi_isr:
  push r24    ; save used register
  in r24,sreg ; save sreg
  push r24
  B = Spdr
  Set Rbit                                                  ' we
received something
  pop r24
  !out sreg,r24 ; restore sreg
  pop r24        ; and the used register
Return                                                      ' this will
generate a reti
```

8.28 DATE TIME

8.28.1 EUROTIMEDATE

The CONFIG CLOCK statement for using the asynchrony timer of the 8535, M163, M103 or M128 (and others) allows you to use a software based clock. See TIME$ 1208 and DATE$ 1191.

By default the date format is in MM/DD/YY.

By specifying:
$LIB 665 "EURODATETIME.LBX"

The DATE$ will work in European format : DD-MM-YY

Note that the eurotimedate library should not be used anymore. It is replaced by the DATETIME 1835 library which offers many more features.

8.28.2 DATETIME

The DateTime library is written by Josef Franz Vögel. It extends the clock routines with date and time calculation.

The following functions are available:

DayOfWeek 1181	Returns the day of the week
DayOfYear 1190	Returns the day of the year
SecOfDay 1203	Returns the second of the day
SecElapsed 1202	Returns the elapsed Seconds to a former assigned time-stamp
SysDay 1206	Returns a number, which represents the System Day
SysSec 1204	Returns a Number, which represents the System Second
SysSecElapsed 1206	Returns the elapsed Seconds to a earlier assigned system-time-stamp
Time 1209	Returns a time-value (String or 3 Byte for Second, Minute and Hour) depending of the Type of the Target
Date 1193	Returns a date-value (String or 3 Bytes for Day, Month and Year) depending of the Type of the Target

 Date and time not to be confused with Date$ and Time$!

The date starts at 1.1.2000 and valid from 2000 to 2099
If you wish to convert to NTP which starts at 1.1.1970, which is 30 years earlier, you need to subtract a value of 946684800
BASCOM DATE_TIME = NTP - 946684800

 Most of the Date and Time functions accept variables which must be in sequential memory order. Like bSec, bMin, bHour
When using DIM like this : Dim bSec As Byte, bMin As Byte, bHour As byte , the variables will be in sequential order, but this might change in the future.
Better would be to be explicit : Dim bSec as byte , bMin as byte at bSec + 1 , bHour as byte at bMin + 1
This will ensure that the bytes will be mapped in the right order.

 It is important that you use the CONFIG CLOCK 895 option since this will include the date time library.

See also

config clock 895, config date 925

8.29 PS2-AT Mouse and Keyboard Emulation

8.29.1 AT_EMULATOR

The PS2 AT Keyboard emulator library is an optional add on library you can purchase.

The library allows you to emulate an AT PS/2 keyboard or mouse.
The following statements become available:

CONFIG ATEMU 883
SENDSCANKBD 1443

8.29.2 PS2MOUSE_EMULATOR

The PS2 Mouse emulator library is an optional addon library you can purchase.
The library allows you to emulate an AT PS/2 mouse.
The following statements become available:

CONFIG PS2EMU 1030
PS2MOUSEXY 1399
SENDSCAN 1441

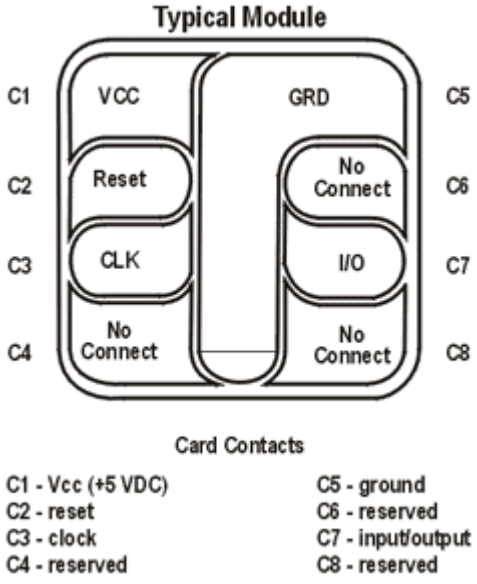
8.30 BCCARD

8.30.1 BCCARD

BCCARD.LIB is a commercial addon library that is available separately from MCS Electronics.
With the BCCARD library you can interface with the BasicCards from www.basiccard.com
BasicCards are also available from MCS Electronics

A BasicCard is a smart card that can be programmed in BASIC.

The chip on the card looks like this :



To interface it you need a smart card connector.

In the provided example the connections are made as following:

Smart Card PIN	Connect to
C1	+5 Volt
C2	PORTD.4 , RESET
C3	PIN 4 of 2313 , CLOCK
C5	GND
C7	PORTD.5 , I/O

The microprocessor must be clocked with a 3579545 crystal since that is the frequency the Smart Card is working on. The output clock of the microprocessor is connected to the clock pin of the Smart card.

Some global variables are needed by the library. They are dimensioned automatic by the compiler when you use the CONFIG BCCARD statement.

These variables are:

_Bc_pcb : a byte needed by the communication protocol.
Sw1 and SW2 : both bytes that correspondent to the BasicCard variables SW1 and SW2

The following statements are especially for the BasicCard:

CONFIG BCCARD 887 to init the library
BCRESET 1844 to reset the card
BCDEF 1837 to define your function in the card
BCCALL 1838 to call the function in the card

Encryption is not supported by the library yet.

8.30.2 BCDEF

Action

Defines a subroutine name and it’s parameters in BASCOM so it can be called in the BasicCard.

Syntax

BCDEF name([param1 , paramn])

Remarks

name	The name of the procedure. It may be different than the name of the procedure in the BasicCard but it is advised to use the same names.
Param1	Optional you might want to pass parameters. For each parameter you pass, you must specify the data type. Supported data types are byte, Integer, Word, Long, Single and String

 This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

BCDEF Calc(string)

Would define a name 'Calc' with one string parameter.
When you use strings, it must be the last parameter passed.

BCDEF name(byte,string)

BCDEF does not generate any code. It only informs the compiler about the data types of the passed parameters.

See Also
CONFIG BCCARD 887 , BCCALL 1838 , BCRESET 1844

Partial Example
```
Bcdef Calc(string)
```

8.30.3 BCCALL

Action
Calls a subroutine or procedure in the BasicCard.

Syntax
BCCALL name(nad , cla, ins, p1, p2 [param1 , paramn])

Remarks

name	The name of the procedure to all in the BasicCard. It must be defined first with BCDEF. The name used with BCDEF and BCCALL do not need to be the same as the procedure in the BasicCard but it is advised to use the same names.
NAD	Node address byte. The BasicCard responds to all node address values. Use 0 for default.
CLA	Class byte. First byte of two byte CLA-INS command. Must match the value in the BasicCard procedure.
INS	Instruction byte. Second byte of two byte CLA-INS command. Must match the value in the BasicCard procedure.
P1	Parameter 1 of CLA–INS header.
P2	Parameter 2 of CLA-INS header

 This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

When in your BasicCard basic program you use:
```
'test of passing parameters
Command &hf6 &h01 ParamTest( b as byte, w as integer,l as long)
b=b+1
w=w+1
l=l+1
end command
```

You need to use &HF6 for CLA and 1 for INS when you call the program:

```
Bccall Paramtest(0 , &HF6 , 1 , 0 , 0 , B , W , L)
                                  ^ NAD

                                ^CLA

                              ^INS

                            ^P1

                         ^P2
```

When you use BCCALL, the NAD, CLA, INS, P1 and P2 are sent to the BasicCard. The parameter values are also sent to the BasicCard. The BasicCard will execute the command defined with CLA and INS and will return the result in SW1 and SW2.

The parameter values altered by the BasicCard are also sent by the BasicCard.

You can not sent constant values. Only variables may be sent. This because a constant can not be changed.

See Also

CONFIG BCCARD 887 , BCDEF 1837 , BCRESET 1844

Example

```
'-----------------------------------------------------------------------------
'                             BCCARD.BAS
'  This  AN  shows  how  to  use  the  BasicCard  from  Zeitcontrol
'                               www.basiccard.com
'-----------------------------------------------------------------------------
'connections:
' C1 = +5V
' C2 = PORTD.4 - RESET
' C3 = PIN 4   - CLOCK
' C5 = GND
' C7 = PORTD.5 - I/O

'                    /--------------------------------\
'  |                                                  |
'  |      C1   C5                                     |
'  |      C2   C6                                     |
'  |      C3   C7                                     |
'  |      C4   C8                                     |
'  |                                                  |
'                    \--------------------------------/
'
'


'----------  configure  the  pins  we  use  ------------
Config  Bccard = D ,  Io = 5 ,  Reset = 4
'                                       ^ PORTD.4
'                                  ^------------ PORTD.5
'                             ^--------------------- PORT  D

'Load  the  sample  calc.bas  into  the  basiccard


'  Now  define  the  procedure  in  BASCOM
'  We  pass  a  string  and  also  receive  a  string
Bcdef  Calc(string)

'We  need  to  dim  the  following  variables
'SW1  and  SW2  are  returned  by  the  BasicCard
'BC_PCB  must  be  set  to  0  before  you  start  a  session
```

```
'Our program uses a string to pass the data so DIM it
Dim S As String * 15

'Baudrate might be changed
$baud = 9600
' Crystal used must be 3579545 since it is connected to the Card too
$crystal = 3579545


'Perform an ATR
Bcreset


'Now we call the procedure in the BasicCard
'bccall funcname(nad,cla,ins,p1,p2,PRM as TYPE,PRM as TYPE)
S = "1+1+3"                                                 ' we want to calculate the
result of this expression

Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
'                                                   ^--- variable to pass that holds the expression
'                                              ^------- P2
'                                         ^----------- P1
'                                    ^---------------- INS
'                               ^--------------------- CLA
'                          ^---------------------------- NAD
'For info about NAD, CLA, INS, P1 and P2 see your BasicCard manual
'if an error occurs ERR is set
' The BCCALL returns also the variables SW1 and SW2
Print "Result of calc : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb)     ' for test you can see that it toggles between 0 and 40
Print "Error : " ; Err

'You can call this or another function again in this session

S = "2+2"
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Result of calc : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb)     ' for test you can see that it toggles between 0 and 40
Print "Error : " ; Err


'perform another ATR
Bcreset
Input "expression " , S
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Answer : " ; S


'----and now perform an ATR as a function
Dim Buf(25) As Byte , I As Byte
Buf(1) = Bcreset()
For I = 1 To 25
  Print I ; "  " ; Hex(buf(i))
Next
'typical returns :
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81    T=1 indication
'TD2 = 31    TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20   IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45    BWT blocl waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
'          B  a  s  i  c  C  a  r  d     Z  C  1  2  3


'and another test
'define the procedure in the BasicCard program
Bcdef Paramtest(byte , Word , Long )


'dim some variables
```

```
Dim B As Byte , W As Word , L As Long

'assign the variables
B = 1 : W = &H1234 : L = &H12345678

Bccall Paramtest( 0 , &HF6 , 1 , 0 , 0 , B , W , L)
Print Hex( sw1) ; Spc( 3) ; Hex( sw2)
'and see that the variables are changed by the BasicCard !
Print B ; Spc( 3) ; Hex( w) ; "  " ; Hex( l )


'try the echotest command
Bcdef Echotest( byte)
Bccall Echotest( 0 , &HC0 , &H14 , 1 , 0 , B)
Print B
End                                                    'end program
```

```
Rem BasicCard Sample Source Code
Rem  ---------------------------------------------------------------
Rem Copyright (C) 1997-2001 ZeitControl GmbH
Rem You have a royalty-free right to use, modify, reproduce and
Rem distribute the Sample Application Files (and/or any modified
Rem version) in any way you find useful, provided that you agree
Rem that ZeitControl GmbH has no warranty, obligations or liability
Rem for any Sample Application Files.
Rem  ---------------------------------------------------------------

#Include CALCKEYS.BAS

Declare ApplicationID = "BasicCard Mini-Calculator"

Rem  This BasicCard program contains recursive procedure calls, so the
Rem  compiler will allocate all available RAM to the P-Code stack unless
Rem  otherwise advised. This slows execution, because all strings have to
Rem  be allocated from EEPROM. So we specify a stack size here:

#Stack 120

' Calculator Command (CLA = &H20, INS = &H01)
'
' Input: an ASCII expression involving integers, and these operators:
'
'    * / % + - & ^ |
'
' (Parentheses are also allowed.)
'
' Output: the value of the expression, in ASCII.
'
' P1 = 0: all numbers are decimal
' P1 <> 0: all numbers are hex

' Constants
Const SyntaxError = &H81
Const ParenthesisMismatch = &H82
Const InvalidNumber = &H83
Const BadOperator = &H84

' Forward references
Declare Function EvaluateExpression (S$, Precedence) As Long
Declare Function EvaluateTerm (S$) As Long
Declare Sub Error (Code@)
```

```
'test for passing a string
Command &H20 &H01 Calculator (S$)

   Private X As Long
   S$ = Trim$ (S$)
   X = EvaluateExpression (S$, 0)
   If Len (Trim$ (S$)) <> 0 Then Call Error (SyntaxError)
   If P1 = 0 Then S$ = Str$ (X) : Else S$ = Hex$ (X)

End Command


'test of passing parameters
Command &hf6 &h01 ParamTest( b as byte, w as integer,l as long)
   b=b+1
   w=w+1
   l=l+1
end command



Function EvaluateExpression (S$, Precedence) As Long

   EvaluateExpression = EvaluateTerm (S$)

   Do
      S$ = LTrim$ (S$)
      If Len (S$) = 0 Then Exit Function

      Select Case S$(1)

      Case "*"
         If Precedence > 5 Then Exit Function
         S$ = Mid$ (S$, 2)
         EvaluateExpression = EvaluateExpression * _
            EvaluateExpression (S$, 6)
      Case "/"
         If Precedence > 5 Then Exit Function
         S$ = Mid$ (S$, 2)
         EvaluateExpression = EvaluateExpression / _
            EvaluateExpression (S$, 6)
      Case "%"
         If Precedence > 5 Then Exit Function
         S$ = Mid$ (S$, 2)
         EvaluateExpression = EvaluateExpression Mod _
            EvaluateExpression (S$, 6)
      Case "+"
         If Precedence > 4 Then Exit Function
         S$ = Mid$ (S$, 2)
         EvaluateExpression = EvaluateExpression + _
            EvaluateExpression (S$, 5)
      Case "-"
         If Precedence > 4 Then Exit Function
         S$ = Mid$ (S$, 2)
         EvaluateExpression = EvaluateExpression - _
            EvaluateExpression (S$, 5)
      Case "&"
```

```
            If Precedence > 3 Then Exit Function
            S$ = Mid$ (S$, 2)
            EvaluateExpression = EvaluateExpression And _
               EvaluateExpression (S$, 4)
         Case "^"
            If Precedence > 2 Then Exit Function
            S$ = Mid$ (S$, 2)
            EvaluateExpression = EvaluateExpression Xor _
               EvaluateExpression (S$, 3)
         Case "|"
            If Precedence > 1 Then Exit Function
            S$ = Mid$ (S$, 2)
            EvaluateExpression = EvaluateExpression Or _
               EvaluateExpression (S$, 2)
         Case Else
            Exit Function
         End Select

      Loop

   End Function

   Function EvaluateTerm (S$) As Long

      Do                            ' Ignore unary plus
         S$ = LTrim$ (S$)
         If Len (S$) = 0 Then Call Error (SyntaxError)
         If S$(1) <> "+" Then Exit Do
         S$ = Mid$ (S$, 2)
      Loop

      If S$(1) = "(" Then       ' Expression in parentheses
         S$ = Mid$ (S$, 2)
         EvaluateTerm = EvaluateExpression (S$, 0)
         S$ = LTrim$ (S$)
         If S$(1) <> ")" Then Call Error (ParenthesisMismatch)
         S$ = Mid$ (S$, 2)
         Exit Function

      ElseIf S$(1) = "-" Then   ' Unary minus
         S$ = Mid$ (S$, 2)
         EvaluateTerm = -EvaluateTerm (S$)
         Exit Function

      Else                          ' Must be a number
         If P1 = 0 Then               ' If decimal
            EvaluateTerm = Val& (S$, L@)
         Else
            EvaluateTerm = ValH (S$, L@)
         End If
         If L@ = 0 Then Call Error (InvalidNumber)
         S$ = Mid$ (S$, L@ + 1)
      End If

   End Function

   Sub Error (Code@)
      SW1 = &H64
```

```
   SW2 = Code@
   Exit
End Sub
```

8.30.4 BCRESET

Action
Resets the BasicCard by performing an ATR.

Syntax
BCRESET
Array(1) = **BCRESET**()

Remarks

Array(1)	When BCRESET is used as a function it returns the result of the ATR to the array named array(1). The array must be big enough to hold the result. Dim it as a byte array of 25.

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

An example of the returned output when used as a function:

```
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81 T=1 indication
'TD2 = 31 TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT block waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00

' B a s i c C a r d Z C 1 2 3
```

See the BasicCard manual for more information
When you do not need the result you can also use the BCRESET statement.

See Also
CONFIG BCCARD 887 , BCDEF 1837 , BCCALL 1838

Partial Example (no init code shown)
```
'----and now perform an ATR as a function
Dim Buf(25)AsByte, I AsByte
Buf(1)=Bcreset()
For I = 1 To 25
Print I ;" ";Hex(buf(i))
Next
'typical returns :
```

```
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81 T=1 indication
'TD2 = 31 TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT blocl waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
' B a s i c C a r d Z C 1 2 3
```

8.31 USB

8.31.1 USB Add On

The USB Add On is a commercial add on which is available from the MCS Electronics Web Shop.
The CONFIG USB statement needs this add on. The add on is written in BASCOM BASIC mixed with assembler. Since the examples from Atmel were not really consistent, it took some effort to create reusable code. At a later stage, a number of routines will be moved to an assembler library.
The advantage of the BASCOM code is that it is similar to the C-code examples.

 Please read this entire topic first before you start with experiments.

The Add On only supports the device mode. There is no support for host mode yet. In fact the add on is just the first step into USB support.

To use the USB Add on, unzip all the files to the SAMPLES\USB directory.
You will find three samples :

- hid_generic-162.bas
- virtcom-162.bas
- hid_keyboard-162.bas

The same samples are also provided for the USB1287.

And you will find the include file : **usbinc.bas**. It is not allowed to distribute any of the files.

Further, you will find a subdirectory named VB which contains a simple VB generic HID sample that uses the HIDX.OCX from the OCX subdirectory.

The PDF directory contains a PDF with a translation between PS2 scan codes and USB key codes.

The TOOLS directory contains the USBDEVIEW.EXE which can be used to display all USB devices,

The CDC-Driver directory contains the INF file you need for the CDC/Virtual COM port example.

The USB162 has a boot loader which can be programmed by USB using FLIP.
BASCOM will also support this USB boot loader in version 1.11.9.2.
It is great for development but of course the boot loader uses some space which you probably need. The chip is also programmable via the normal way with the ISP protocol. when you do not use FLIP, and you erase the chip, the boot loader from Atmel is erased too! You can always reprogram the Atmel boot loader. But not using

FLIP which depends on the boot loader.

For USB to work properly the chip needs a good oscillator. The internal oscillator is not good enough. For that reason, the USB162 module from MCS has a 8 MHz crystal. Your hardware should use a crystal or crystal oscillator too.

It is not the intention of MCS or the documentation to learn you everything about USB. There is a lot of information available from various sources. It is the goal of MCS to make it easy to use USB with your AVR micro. When there is enough demand for it, a special Wizard will be created to be able to generate HID applications.

HID Keyboard

Let's begin with a simple program. Load the hid_keyboard-162.bas sample and compile it. Use either FLIP or a different programmer to program the chip. Each program has some important settings.

```
Const Mdbg = 1                  ' add print to see what is happening
Const Chiddevice = 1            ' this is a HID device
```

MDBG is a constant that can be set to 0 since all the print statements will use flash code. When you are new to USB and want to look at the events, it is good to have it turned on. You can view all events from the program.
cHIDdevice need to be set to 1 when the application is a HID device. Most of your own devices will be HID devices. But the virtual COM example uses a different USB class and in that program, the constant is set to 0.

These constants are used in the add on to keep all code generic for all applications. Since not all USB chips have the same options, the code also checks which microprocessor is used.
The USB1287 is a kind of M128 with USB support. It supports host and device mode.
The USB162 is a cheap host chip. It does not support the HOST mode and it does not have all registers found in the USB1287. It also can not detect when a device is plugged/unplugged.
Atmel solved this in the STK526 in a simple way that we recommend too : A voltage divider is connected to PORTC.4 which serves as a simple way to detect plug/unplug.
In the USB_TASK() routine you will find this code :

```
If Usb_connected = 0 And Pinc.4 = 1 Then          ' portc.4 is used as vbus detection on stk526
```

This is used with the STK526. If you want to use a different pin, you have to change PINC.4.
When you use the USB1287 this is not needed since the 1287 has a Usbsta register which can determine if a device is plugged or removed.

The USB program structure is always the same :
1. constants are defined that describe the end points, interfaces, vendor ID, product ID
2. you call a subroutine that initializes your variables
3. In a loop you call :
4. the generic USB_TASK routine so that the USB communication with the PC is executed
5. the specific task is called
6. your other code is called

This is clear in the keyboard sample :

```
Print "init usb task"
Usb_task_init
Do
  Usb_task
  Kbd_task
  'call your other code here
Loop
```

While the word Task might give you the idea that multi task switching is used, this is not the case! The USB_Task must be called by your code in order to process pending USB events. It will also find out if a device is plugged or unplugged. Events are handled in the background by the **Usb_gen_int** interrupt.
In the example the KBD_TASK is a user routine which is called in regular intervals. There is always the normal USB_TASK and there is an additional task specific to the program. In the generic-hid example this is the hid_task routine.

HID classes are simple to use since they do not require additional drivers. FTDI chips need additional drivers. But the Atmel USB chips do not need additional drivers since they use standard implemented HID classes.

When you compile the program and program it into a chip you are ready to test it. When you use FLIP you need to switch to application mode so your device can be recognized by windows. Windows will show some info that your device is found. And after installing the driver, it will report that your device is ready to be used.
On the terminal emulator, press a space, and set the focus to notepad or the bascom editor. The text data from the **keys**: label is send as if it was typed on a keyboard! You in fact created a HID-keyboard, or USB keyboard. The document translatePS2-HID.pdf contains HID key codes which are different then PS2 key scan codes.

When you do not have a terminal emulator connected you can also modify the program and connect a push button. Which makes more sense for a keyboard :-)
So modify the code into : If Inkey() = 32 Or Pinb.0 = 0 Then 'if you press SPACE BAR or make PINB.0 low
Now you can test the code without the terminal emulator.

All USB programs are similar. You specify the number of end points , the interfaces and the class. There is a lot of information available at
http://www.usb.org/home
Atmel has a number of samples and you will find tools and info at various places. MCS will publish some convenient tools too.

FLIP

The USB chips are programmed with a boot loader. This is very convenient since you do not need any hardware to program the chip. FLIP can be downloaded from the Atmel site.
URL : http://www.atmel.com/dyn/resources/prod_documents/Flip%20Installer%20-%203.3.1.exe
The FLIP website you can find at :
http://www.atmel.com/dyn/products/tools_card.asp?family_id=604&family_name=8051+Architecture&tool_id=3886

FLIP is a Java application. The BASCOM-IDE can use the FLIP software to program the chip too. But in order to use the FLIP programmer, you need to install FLIP first.
When FLIP is working, you can select FLIP from Options, Programmer, in order to program quickly without the FLIP executable.
On Vista there is a problem with loading some of the FLIP DLL's. In case you get an error, copy

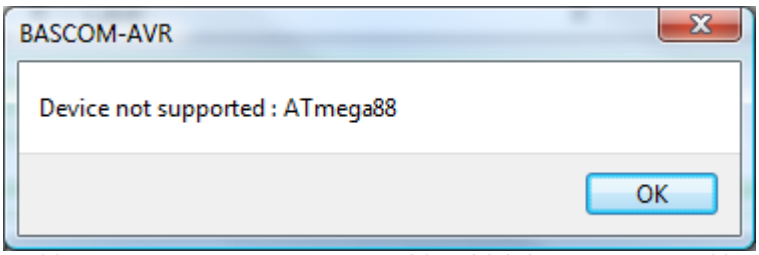
the FLIP DLL's to the BASCOM application directory.
You need to copy the following files :

- atjniisp.dll
- AtLibUsbDfu.dll
- msvcp60.dll
- msvcrt.dll

You can run the **flipDLLcopy.cmd** file from the BASCOM application directory to copy these files.
The content of the command file :
copy "c:\program files\atmel\flip 3.3.1\bin\atjniisp.dll" .
copy "c:\program files\atmel\flip 3.3.1\bin\AtLibUsbDfu.dll" .
copy "c:\program files\atmel\flip 3.3.1\bin\msvcp60.dll" .
copy "c:\program files\atmel\flip 3.3.1\bin\msvcrt.dll" .
pause

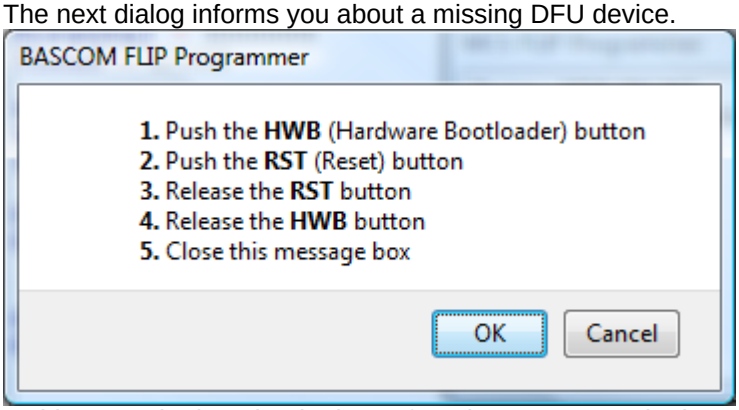
The last line pauses so you can view the result. Notice the . (dot) that will copy the file to the current directory, which is the reason that you need to run this file from the BASCOM application directory.

As with other programmers, you press F4 to program the HEX file into the chip. A small window will become visible.
A number of dialogs are possible:



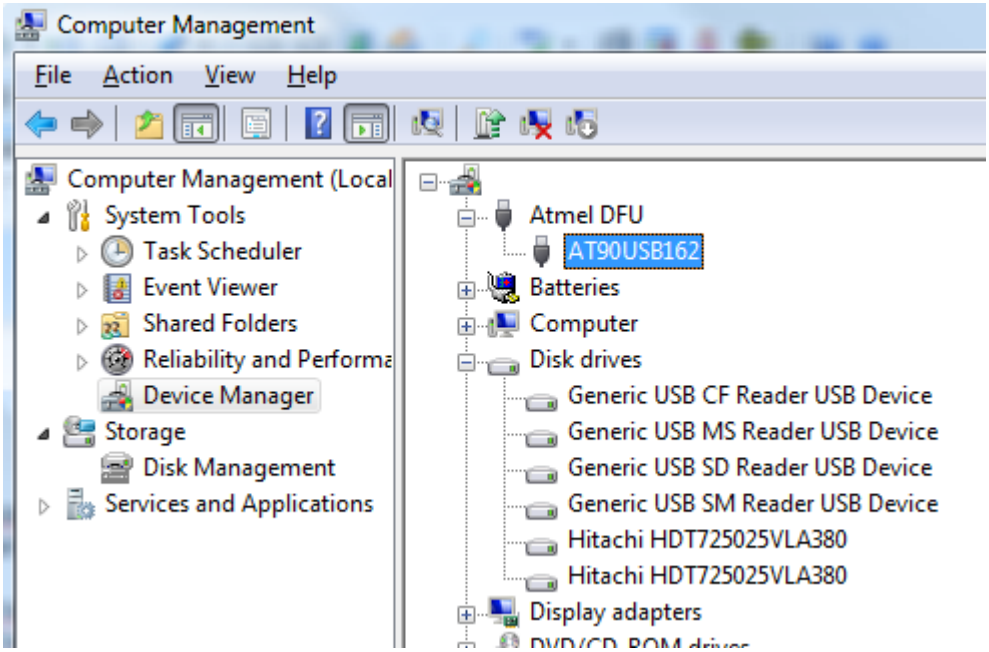
In this case, you try to program a chip which is not supported by FLIP. The Mega88 is not an USB chip so the error makes sense.

The next dialog informs you about a missing DFU device.



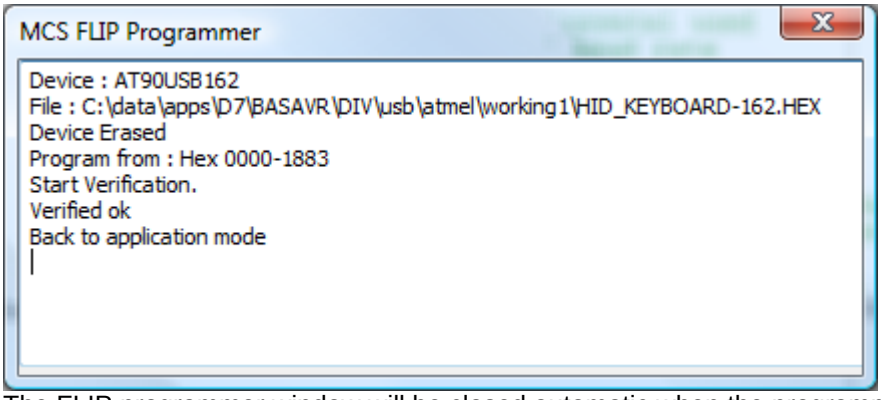
In this case, the boot loader is not found. You can run the boot loader by following the sequence from the dialog box.
In order to make this work, the HWB and RST input both need a small switch to ground.
When HWB is pressed(low) during a reset, the boot loader will be executed.

In the device manager you will find the USB device :



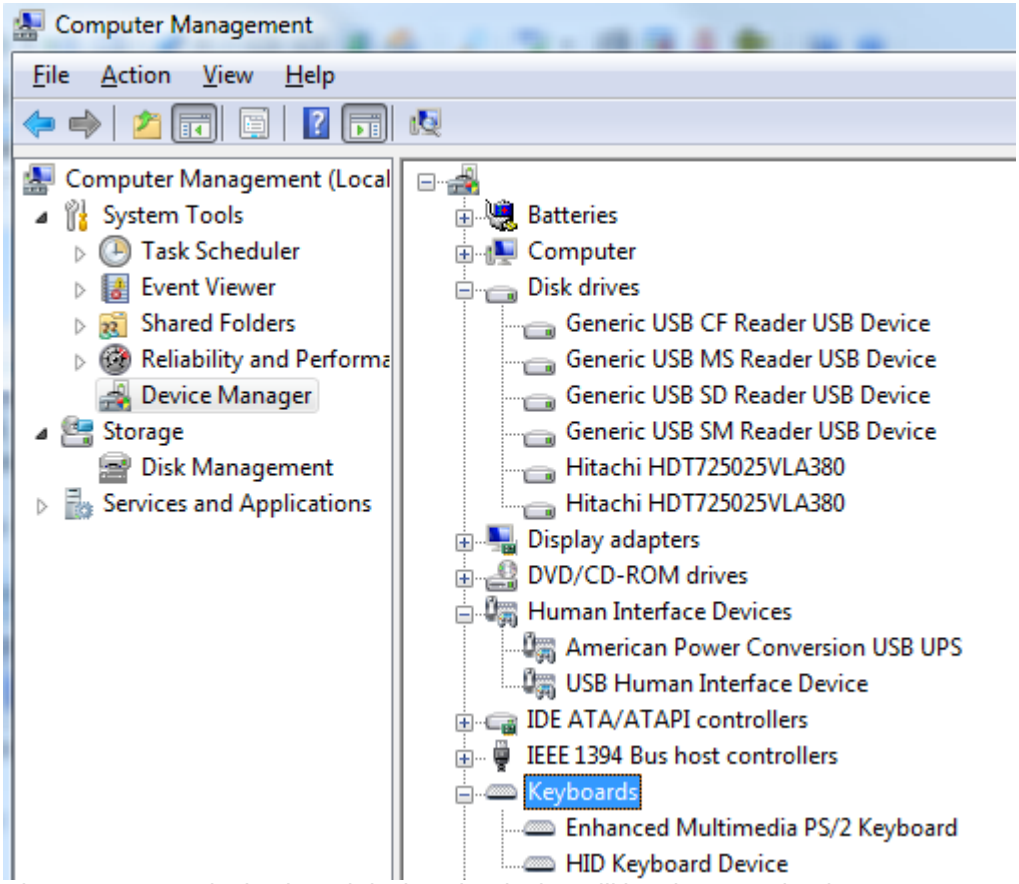
When you have a different chip, a different device will be shown !

When the programming succeeds, and there is no verify error, the application mode will be selected. This will disconnect the DFU and will connect your USB device !



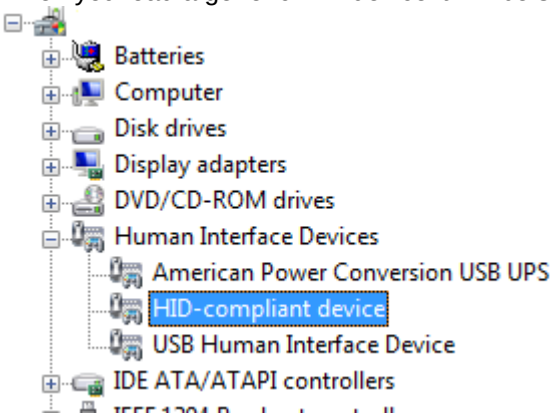
The FLIP programmer window will be closed automatic when the programming succeeds.

The USB device will be shown :



Since you created a keyboard device, the device will be shown under the KEYBOARDS node.

When you load a generic HID device it will be shown under HUMAN INTERFACE DEVICES



HID Generic

The generic HID class is the class that is well suited for transferring bytes between the PC and the micro processor.
As with any USB application, you specify the number of end points, The example just transfers 8 bytes in and 8 bytes out.
You need to change the Ep_in_length_1 , Ep_out_length, Length_of_report_in and Length_of_report_out constants when you want to transfer a different amount of bytes.
You also need to take into account the maximum data size which will depend on the used chip.

The **Usb_user_endpoint_init** sub routine also need to be adjusted. The size_8 constant specifies how many bytes are used by the endpoint.

```
'init the user endpoints
Sub Usb_user_endpoint_init(byval Nm As Byte)
```

```
  Call Usb_configure_endpoint(ep_hid_in , Type_interrupt , Direction_in , Size_8 , One_bank ,
Nyet_enabled)
  Call Usb_configure_endpoint(ep_hid_out , Type_interrupt , Direction_out , Size_8 , One_bank ,
Nyet_enabled)
End Sub
```

As with all USB program, we first initialize the USB task and the HID task. Then we call the tasks in a loop ;

```
Usb_task_init                              ' init the usb task
Hid_task_init                              ' init the USB task
Do
  Usb_task                                 'call this subroutine once in a while
  Hid_task                                 'call this subroutine once in a while
  'you can call your sub program here
Loop
```

The Hid_task itself is very simple :

```
Sub Hid_task()
   If Usb_connected = 1 Then                        ' Check USB HID is enumerated
     Usb_select_endpoint Ep_hid_out                 ' Get Data Repport From Host
     If Ueintx.rxouti = 1 Then                     ' Is_usb_receive_out())
       Dummy1 = Uedatx : Print "Got : " ; Dummy1        ' it is important that you read the same
amount of bytes here as were sent by the host !
       Dummy2 = Uedatx : Print "Got : " ; Dummy2
       Dummy = Uedatx : Print "Got : " ; Dummy
       Dummy = Uedatx : Print "Got : " ; Dummy
       Dummy = Uedatx : Print "Got : " ; Dummy
       Dummy = Uedatx : Print "Got : " ; Dummy
       Dummy = Uedatx : Print "Got : " ; Dummy
       Dummy = Uedatx : Print "Got : " ; Dummy
       Usb_ack_receive_out
     End If

     If Dummy1 = &H55 And Dummy2 = &HAA Then             ' Check if we received DFU mode
command from host
       Usb_detach                                ' Detach Actual Generic Hid Application
       Waitms 500
       Goto &H1800                               'goto bootloader
       'here you could call the bootloader then
     End If

     Usb_select_endpoint Ep_hid_in                    ' Ready to send these information to the host
application
     If Ueintx.txini = 1 Then                        ' Is_usb_in_ready())
       Uedatx = 1
       Uedatx = 2
       Uedatx = 3
       Uedatx = 4
       Uedatx = 5
       Uedatx = 6
       Uedatx = 7
       Uedatx = 8
       Usb_ack_fifocon                           ' Send data over the USB
     End If
   End If
End Sub
```

We first check if the device is connected to the USB bus. Then we use **Usb_select_endpoint** with the number of the end point, to select the end point.
When we want to communicate with an end point, we always have to select this end point using

the **Usb_select_endpoint** procedure.
In the sample, we first select the EP_HID_OUT end point. We check the UEINTX.RXOUTI flag to determine if we received an interrupt with data. If that is the case, we read the UEDATX register to read the data byte.
The UEDATX register is the USB data register. When you read it, you read data from the USB bus. When you write it, you write data to the USB bus.
After reading the bytes you MUST acknowledge with the **Usb_ack_receive_out** macro.

The sample also shows how to run the boot loader from your code. In order to run the boot loader you must detach the current device from the USB bus. Then there is some delay to have windows process it. Finally the GOTO jumps to the boot loader address of the USB162.

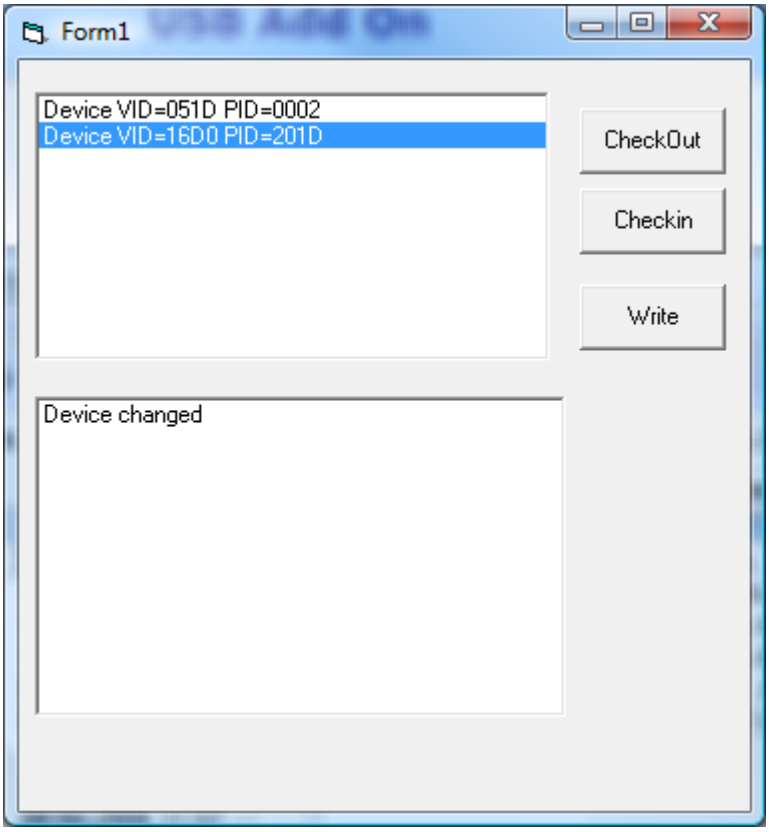
If you want to write some data back, you need to select the end point, and check if you may send data. If that is the case, you assign the data to the UEDATX register and finally, you MUST acknowledge with the USB_ACK_FIFOCON macro.

Finally, you will find in the report data the length of the end points specified : Data &H75 , &H08
You need to adjust these values when you want to send/receive more data.

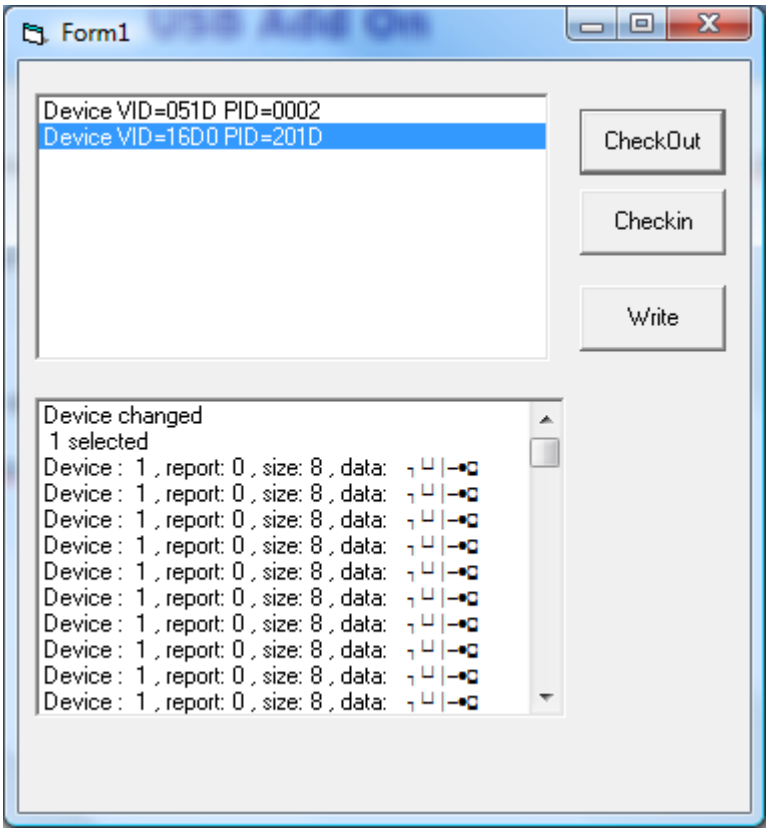
HIDX.OCX

There are plenty of examples on the internet that show how to communicate with HID devices using the windows API.
The HIDX.OCX is an OCX control that can be used for simple communication.
Like all OCX controls, you must register it first with REGSVR32 : regsvr32 hidx.ocx
After it has been registered you can run the VB test application named HIDdemo.exe.

The application will list all HID devices :



Our device is the device with VID 16D0 and PID 201D.
There can only be one application/process at the time that communicates with an USB device.
You must click the checkout-button the device to start communication. This will call the **SelectDevice** method of the OCX.
As soon as you do this, you will notice that the **OnDataRead** event will receive data.



The event has the following parameters :
(ByVal Device As Long, ByVal ReportID As Long, ByVal Data As String, ByVal Size As Long)

The device is a number with the index of all HID devices. The first device will have number 0. The report number is passed in ReportID. The data is passed as a string.
You can use MID to access this data : firstByte= Asc(Mid(data,1,1))

To write to the device, you can use the **WriteDevice** method. The same parameters are used as with the OnDataRead event.
Example : WriteDevice curdev, 0, s, 8
Curdev is the index of the device. 0 is the report ID and s contains the data. You must specify the length of the data to send.

To stop communication you can click the Checkin-button.This will call the **ReleaseDevice** method.

When the device changes, or will be removed or inserted, you will receive a notification.
In the sample program, all these events will result in a release of the device. This is done since the curdev variable can change when a new device is added. The index will not correspond to the existing index then anymore. The sample is very simple. In an application you could add a function or procedure that will examine the new list of devices and return the index of our device. When our device is found we could open it automatic again.

Notice that you can not add too much lines to a listbox in VB. Since data arrives at a very high rate, it will not take long before VB/Windows will give some error.

Property	Description
NumCheckedInDevices	Number of available devices
NumCheckedOutDevices	Number of devices that are checked out and communicating.
NumUnpluggedDevices	

DevThreadSleepTime	The time in mS that the HID thread will sleep. You can see this as a timer interval. The lower the interval the more process time it will take. 100 mS is a good value for most applications.
Version	The version of the control
DeviceCount	The number of devices.
Methods	
SelectDevice	Parameters • Device : LONG that specifies the index of the device to select. The index starts at 0.
ReleaseDevice	Parameters • Device : LONG that specifies the index of the device to release. The index starts at 0.
WriteDevice	Parameters • Device : LONG that specifies the index of the device to write to. The index starts at 0. • Report : LONG that specifies the report number. This would be 0 in most cases. • Data : string that contains the data to send. • Size : the length of the data to send.
Events	
OnDeviceChange	Parameters • none. This event fires when a device changes. This can be because a new device is added, or a device is removed.
OnDeviceArrival	Parameters • Device : LONG that specifies the index of the device that arrived. The index starts at 0. This event fires when a device is inserted. When a device is added or removed, the index that was used previously, does not need to match the new index anymore. For this reason you have to checkout the device again.
OnDeviceRemoval	Parameters • Device : LONG that specifies the index of the device that has been removed. The index starts at 0. This event fires when a device is removed. When a device is added or removed, the index that was used previously, does not need to match the new index anymore. For this reason you have to checkout the device again.
OnDataRead	Parameters • Device : LONG that specifies the index of the device that sent data. The index starts at 0. • ReportID : LONG with the report ID of the device that sent the data. • Data : string that contains the data. This string might contain 0-bytes. • Size : LONG that contains the length of the received data. When data is received you can read it in this event. For example : dim ar(8) as Byte

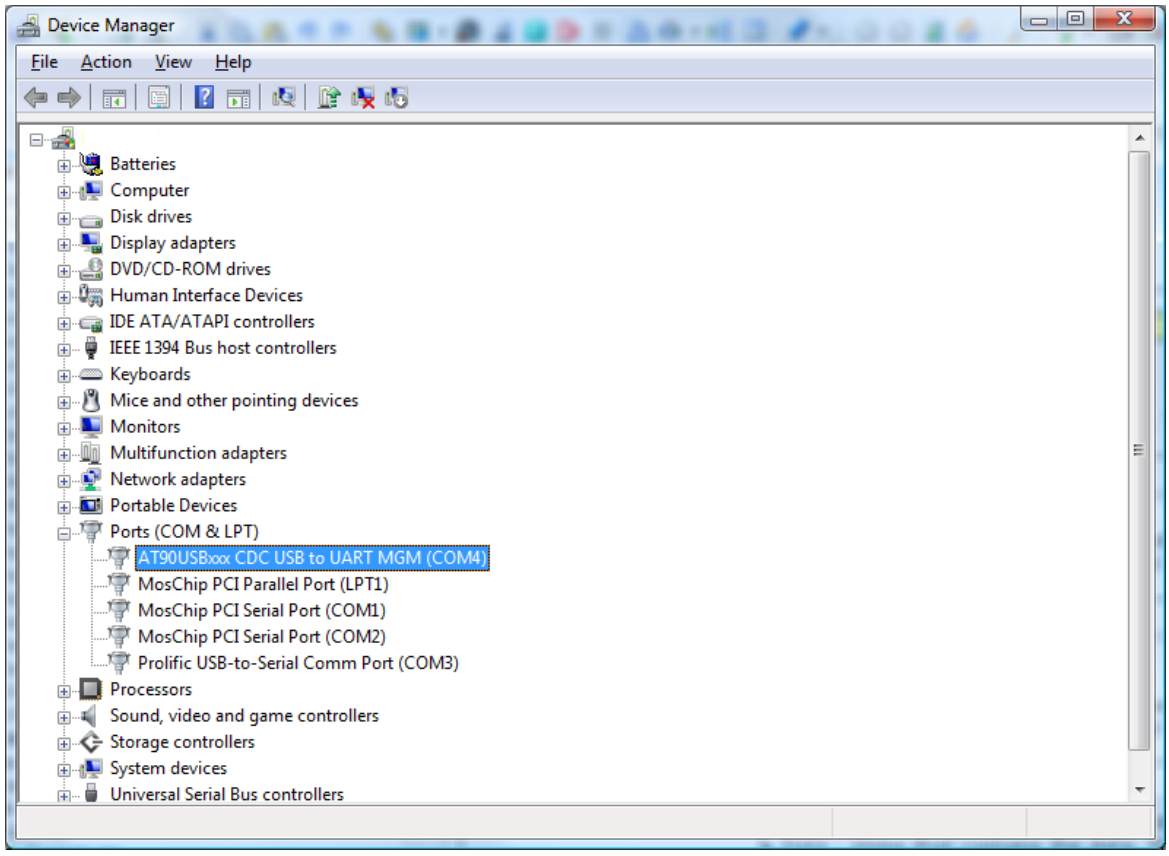
	For J=1 to Size ar(j) = ASC(Mid(data,J,1)) ' fill the array Next

The OCX can be used with all programming languages that can host OCX controls. The OCX was tested with Delphi and VB.
Your windows must support USB in order to use the OCX. So it will not work on Windows 95.

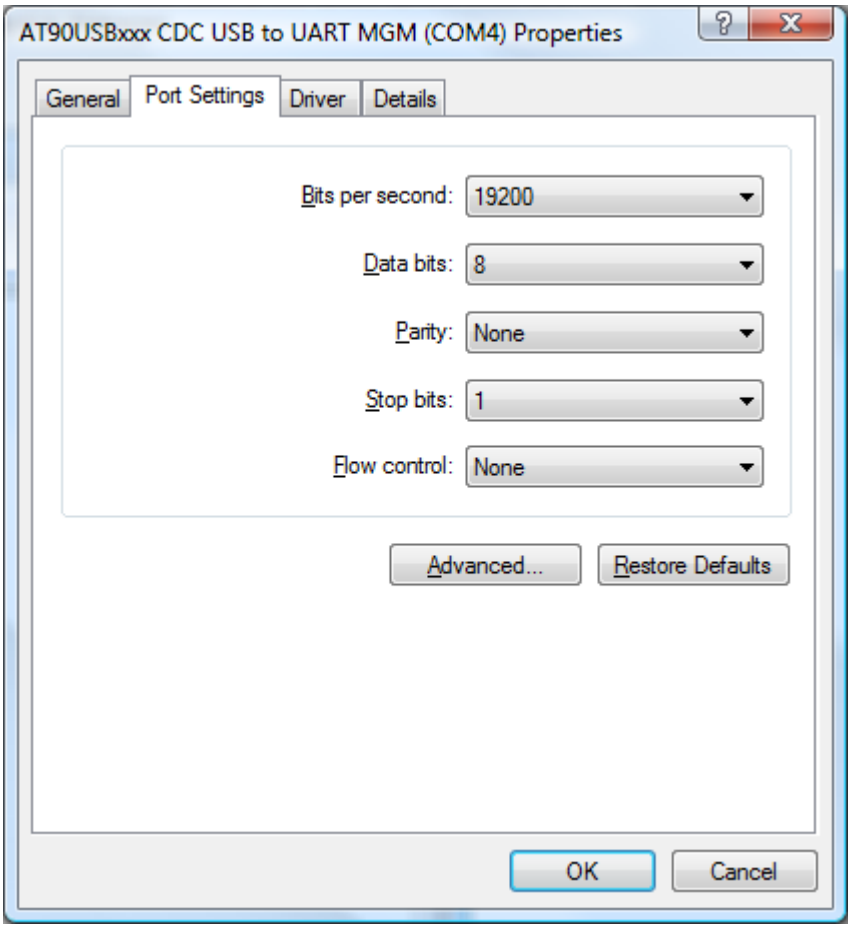
Virtual COM sample

The virtual COM demo shows how to implement an USB device with a virtual COM port. The Demo will echo data sent to the UART to the USB and vise versa.
When you compile and program the sample, you will notice that you find a new COM port in the device manager.


When you press CTRL+D, BASCOM will launch the device manager.



As you can see, the CDC class is used for the virtual COM port. As with most virtual COM devices, you can change the settings :



In the BASCOM application the procedure Cdc_get_line_coding is called when the PC need to know the settings.
The Cdc_set_line_coding is called when the settings are changed by the user. You need to change the settings according to the received parameters.
Notice that these settings are virtual too : for the USB it does not matter how the baud rate is set ! Only for a real UART this is important. For an USB-RS232 converter for example it is very convenient to be able to change the baud rate and other settings. But when you just use the USB port for communication, and choose to use the COM port in your program as a way for communication, then you do not really need the settings.

When you want to send date to the USB/COM you can use the **Uart_usb_putchar** procedure. Like any USB routine, it will select the proper end point. After the end point for sending data is selected it will wait if it may send data, and finally it will send this data.
The **Uart_usb_getchar**() function can be used to receive data from the USB/COM.

When you create your own device, the virtual COM port has the advantage that the PC application is simple. In most cases you already have the experience to read/write data to the PC COM port. The disadvantage is that it requires mode code. It also need an INF file. This INF file you can change to suite your own needs.
When you create your own device, the HID device is the simplest way to go.

CDC INF file

The CDC INF file looks like this. The bold parts need to be changed if you want to customize with your own text and VID/PID.

; Windows 2000, XP & Vista setup File for AT90USBxx2 demo

```
[Version]
Signature="$Windows NT$"
Class=Ports
ClassGuid={4D36E978-E325-11CE-BFC1-08002BE10318}

Provider=%ATMEL%
LayoutFile=layout.inf
DriverVer=10/15/1999,5.0.2153.1

[Manufacturer]
%ATMEL%=ATMEL

[ATMEL]
%ATMEL_CDC%=Reader, USB\VID_03EB&PID_2018


[Reader_Install.NTx86]
;Windows2000

[DestinationDirs]
DefaultDestDir=12
Reader.NT.Copy=12

[Reader.NT]
include=mdmcpq.inf
CopyFiles=Reader.NT.Copy
AddReg=Reader.NT.AddReg

[Reader.NT.Copy]
usbser.sys

[Reader.NT.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,usbser.sys
HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"

[Reader.NT.Services]
AddService = usbser, 0x00000002, Service_Inst

[Service_Inst]
DisplayName = %Serial.SvcDesc%
ServiceType = 1 ; SERVICE_KERNEL_DRIVER
StartType = 3 ; SERVICE_DEMAND_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
ServiceBinary = %12%\usbser.sys
LoadOrderGroup = Base

[Strings]
ATMEL = "ATMEL, Inc."
ATMEL_CDC = "AT90USBxxx CDC USB to UART MGM"
Serial.SvcDesc = "USB Serial emulation driver"

;---- END OF INF FILE
```

You can also change the key names.

8.32 MODBUS Slave/Server

The MODBUS protocol is used a lot in the industry. With the MODBUS add-on, you can create a slave or server.
This add-on is a MODBUS server-RTU that implements function 03,06 and 16. (decimal)

We use the term master and slave to indicate that there is at least one master, and that there is at least one slave device that will respond.
A slave could be a master too. Another term is client/server. The server is the MODBUS device that will respond to the client. It is the same as master/slave and thus slave=server and master=client.
Like a web server, the server does not initiate the communication. It simply waits for data and when it is addressed, it will respond.
When it is not addressed, it should not respond. When it is addressed, it should process the data and send a response.

A client sends the following data : server address, function, data, checksum
The server address is a byte , the function code is a byte too. The data depends on the function and the checksum is a 16 bit CRC checksum.
MODBUS uses the term registers for the data. A register is 16 bit width. You can pass words or integers with a single register.
In order to send a long, single, double or string, you need to send multiple registers.

There are a lot of functions defined in the MODBUS protocol. The add-on implements the functions that are most suited for an own MODBUS server device.
These functions are :
- 03 : read (multiple) register(s)
- 06 : write a single register
- 16 : write multiple registers

If needed you can add other functions yourself. The implemented functions should be sufficient however.

Constants

There are a few constants that you might need to change.
Registersize : this constant defines how many registers can be processed. For example if a client asks to return 10 registers with function 03, you should set this constant to 10.
The reason for the constant is that RAM space is limited. And each register need storage space (2 bytes for each register) thus we do not want to take more bytes then needed.

Mdbg : this can be used for debugging. The add-on uses a Mega162 since it has 2 UARTS. One UART can be used for debugging. You need to set mdbg to a non-zero value to enable debugging to the serial port.

RS232-RS485

The protocol can be used with RS-232 and RS-485 and TCP/IP, etc. The add-on can be used with RS-232 and RS-485.
RS-485 half duplex needs a data direction pin. It is defined in the source like this :

```
Rs485dir Alias Portb.1
Config Rs485dir = Output
```

```
Rs485dir = 0
'Config Print1 = Portb.1 , Mode = Set
```

You can remark or remove the mark depending on the mode you need.
For testing, RS-232 is most simple.

TIMER

A timer is used to detect the start of a frame. With RTU (binary data) a silence of 3.5 characters is needed between frames. A frame is a complete MODBUS message.
A timer is used to detect such a silence. The statement : GENRELOAD , is used to generate the proper timer divisor and timer reload values. GENRELOAD will only work on TIMER0 and TIMER1. You pass the names of the constants which are free to chose, and in the sample are named _RL and _TS, and these constant values will be calculated and assigned to constants by the compiler.
The TM_FRAME constant is the time of 4 characters. When the timer reaches this value it will overflow and execute the ISR_TMR0 interrupt. The interrupt routine will set the start state since now the server can expect an address.
In the TM_FRAME calculation the baud rate value is used. In the add on this is 9600. When you use a different value, you need to change the constant here as well.

Server Address

The server address need to be set. The MBSLAVE variable need to be set by you. Optional, you could change the variable into a constant.
But when you use a DIP switch for example to set the address, it is better to use a variable.

Event mode

The MODBUS handeling is coded into a state machine and executed as a task. You can call the Modbustask() in your code yourself in the main program loop, or you can have it called in the interrupt of the buffered serial input routine.
The sample uses the last option :

```
Config Serialin1 = Buffered , Size = 50 , Bytematch = All
```

Notice that BYTEMATCH = ALL is used so the Serial1bytereceived routine is called for every received byte. If the state is right, the modbustask code is executed and otherwise, the data is read to remove it from the buffer. Since there can be multiple slaves, the data will keep coming and we may only handle the data when we are addressed.

Functions

Each function that is requested will call a sub routine.
Function 03 (read registers) : Sub Modbus03(addr3 As Word , Idx3 As Byte , Wval3 As Word)
addr3 contains the address that was passed by the client.
Idx3 contains an index in case multiple registers are read. It is 1 for the first register, 2 for the second, etc.
With these 2 values you can fill the wval3 value.
In the sample, a select case is used to send different values.

You should NOT change the addr3 and idx3 values ! There variables are passed by reference and changes will corrupt the data.

Notice that the function is called for each register. When the client want to read 2 word registers, the sub routine is called twice.

Function 06(write register) Sub Modbus06(addr3 As Word , Wval3 As Word)
Addr3 contains the address that was passed by the client.
wval3 contains a word value passed by the client.
You can use the address to change some variable in your code.

Function 16 (write multiple registers) Sub Modbus16w(addr3 As Word , Idx As Byte , Bw As Word)
Addr3 contains the address send by the client.
Idx contain the index to a word register.
Bw contains the value that was send.

Notice that the sub routine is called for each register. You can use the address and index to alter the proper variable in your code.

For functions that are not implemented, an error response will be sent.

Part

9 Tools

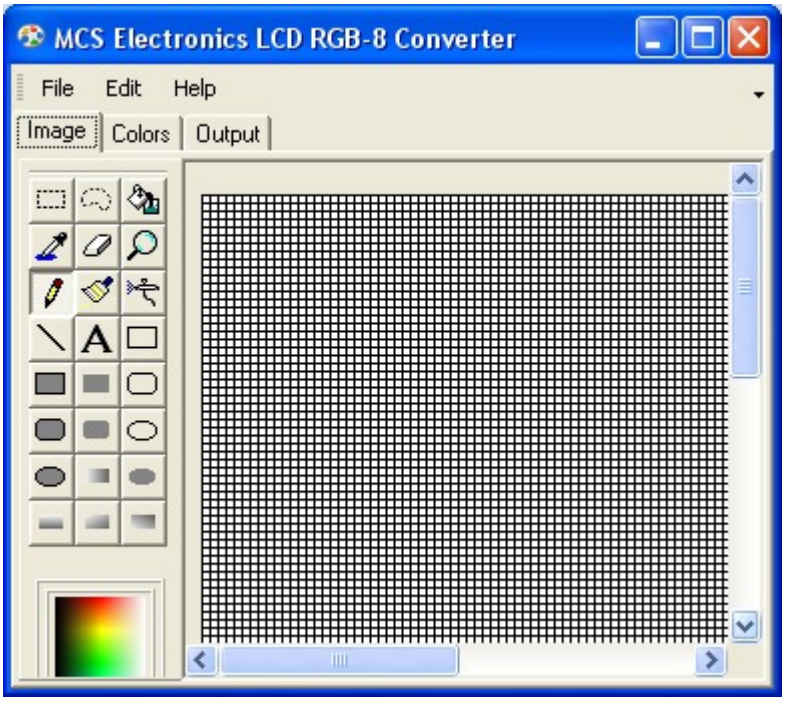
9.1 LCD RGB-8 Converter

Action

This tool is intended to convert normal bitmaps into BGC files.
The BGC format is the **B**ascom **G**raphic **C**olor Format.
This is a special RLE compressed format to save space.

The SHOWPIC statement can display graphic bitmaps.
The color display uses a special RGB8 format.
The LCD converter has the task to convert a normal windows bitmap into a 256-color RGB8 coded format.

When you run the tool you will see the following window :



You can use File , Open, to load an image from disk.
Or you can use Edit, Paste, to paste an image from the clipboard.

Option	Description
File, Open	Open a graphical file from disk.
File, Save, Image	Save the file as a windows graphical file
File, Save, Binary	Save the BGC file, the file you need with SHOWPIC
File, Save , Data Lines	Save the file as data lines into a text file
File, Convert	Converts the bitmap into a RGB8 bitmap
Edit, Bitmap height	height of the image. Change it to make the image smaller or larger
Edit, Bitmap width	width of the image. Change it to make the image wider.
Edit, Select All	Select entire image
Edit, Copy	Copy selection to the clipboard
Edit, Paste	Paste clipboard to the selection. You must have an area

	selected !
Edit, Delete	Delete the selected area

The Output TAB, has an option : Save as RLE. This must be checked. By default it is checked.
When you do not want the image to be RLE encoded, you can uncheck this option.

The bottom area is used to store the DATA lines.

The Color TAB shows the effect on the table inside the color display.
When a picture uses a lot of different red colors, you can put the most used into the table.
It is well explained in the manuals from display3000.

By clicking on the color , you can view which colors are used by the picture.
You can match them with the color table.

You can download the LCD Converter tool from :
http://www.mcselec.com/index.php?option=com_docman&task=doc_download&gid=168&Itemid=54

9.2 BASCOMP

BASCOMP.EXE is a command line compiler utility.
It can be called from your own favorite editor when using linux. Or when compiling projects from a batch file.

The **bascomp**.exe utility must be placed in the same folder as bascavr.exe.
It depends on the bascom-avr license dll and the basc-avr.dll compiler DLL.
It also depends on the DAT files and the LIB folder with the libraries.
For this reason the files is best placed into the bascom-avr application folder.

Change in 2085

The utility used to work by passing the chip ID. But since some processors are binary compatible and share the same ID this could result in problems.
bascomp requires only one parameter : the source file you want to compile. When the path contains spaces you need to enclose it in double quotes.
Example : **bascomp** "c:\some folder\mycode example.bas"

bascomp will read the source and extract info about the dat file and stack.
Specifically it will look for $regfile, $hwstack,$swstack and $framesize
If these settings are not found you need to provide them using SS for $softstack, HW for $hwstack and FR for $framesize
Example : **bascomp** "myfile.bas" HW=64
This would use a $hwstack of 64 bytes. Note that even if your source would contain a different value, a value of 64 would be used.

In order to override the processor dat file, you need to use the new DEVICE parameter. It accepts only the official processor name.
Example : **bascomp** "myfile.bas" device=atmega88

When you specify a device name that does not exist, a list will be shown with all device names.

The command line utility will set the DOS errorcode to 0 when no error is found, or to

a non zero value when there is a problem.

Parameters

HW	Hardware stack ($hwstack)
FR	Frame size ($framesize)
SS	Soft stack ($swstack)
DEVICE	Device name. Stored in the DAT files with the name device= The name is not case sensitive.

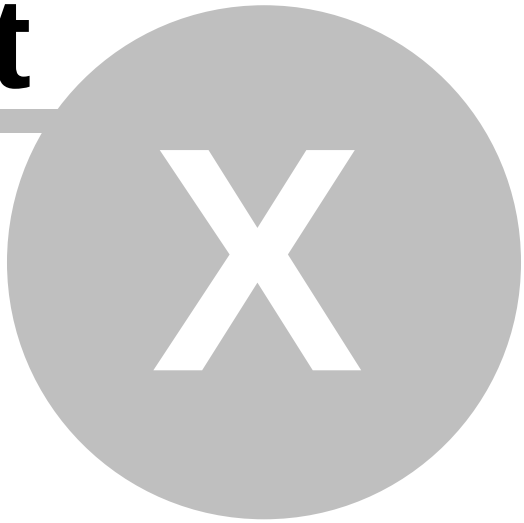
This is a list of device names which will change when new DAT files are added.
available devices:
AT90S1200
AT90S2313
AT90S2323
AT90S2333
AT90S2343
AT90S4414
AT90S4433
AT90S4434
AT90S8515
AT90S8535
AT86RF401
AT90PWM1
AT90PWM216
AT90PWM3
ATtiny12
ATtiny13
ATtiny13A
ATtiny15
ATtiny1634
ATtiny167
ATtiny20
ATtiny22
ATtiny2313
ATtiny2313A
ATtiny24
ATtiny24A
ATtiny25
ATtiny26
ATtiny261
ATtiny4313
ATtiny43U
ATtiny44
ATtiny441
ATtiny45
ATtiny461
ATtiny48
ATtiny828
ATtiny84
ATtiny841
ATtiny85
ATtiny861
ATtiny87
ATtiny88
ATtiny1604
ATtiny1606
ATtiny1607

ATtiny1614
ATtiny1616
ATtiny1617
ATtiny202
ATtiny204
ATtiny212
ATtiny214
ATtiny3216
ATtiny3217
ATtiny402
ATtiny404
ATtiny406
ATtiny412
ATtiny414
ATtiny1416
ATtiny417
ATtiny804
ATtiny806
ATtiny807
ATtiny814
ATtiny816
ATtiny817
AVR128DB28
AVR128DB32
AVR128DB64
AVR64DB32
ATMega103
ATMega1280
ATMega128
ATMega1281
ATMEGA1284
ATMEGA1284P
AT90CAN128
ATMega128
ATMega128RFA1
ATMEGA161
ATmega162
ATMEGA163
ATMEGA164A
ATMEGA164PA
ATMEGA164P
ATMega165A
ATMega165
ATmega168
ATmega168PA
ATmega168PB
ATmega168P
ATmega169A
ATmega169
ATmega169PA
ATmega169P
ATmega16A
ATmega16
ATMEGA16M1
ATMEGA16U2
ATMEGA16U4
ATMega2560
ATMega2561

ATMEGA323
ATMEGA324A
ATMEGA324PA
ATMEGA324PB
ATMEGA324P
ATMEGA3250A
ATMEGA3250PA
ATMEGA3250P
ATMEGA325
ATmega328
ATmega328PB
ATmega328P
ATmega329
ATMEGA32A
ATMEGA32C1
AT90CAN32
ATMEGA32
ATMEGA32M1
ATMEGA32U2
ATMEGA32U4
ATmega406
ATmega48
ATmega48PA
ATmega48PB
ATmega48P
ATmega603
ATMega640
ATMEGA644A
ATMEGA644
ATMEGA644PA
ATMEGA644P
ATMEGA6450P
ATMEGA645
ATmega6490
ATmega649A
ATmega649
ATmega649P
ATMEGA64C1
AT90CAN64
atmega64
ATMEGA64M1
ATmega8515
ATmega8535
ATmega88A
ATmega88
ATmega88PA
ATmega88PB
ATmega88P
ATmega8A
ATmega8
ATMEGA8U2
ATmega4808
ATmega4809
AT90USB1286
AT90USB1287
AT90USB162
AT90USB646
AT90USB82

ATXMega128A1
ATXMega128A1U
ATXMega128A3
ATXMega128A3U
ATxmega128A4U
ATxmega128B1
ATxmega128B3
ATXMega128C3
ATXMega128D3
ATxmega128D4
ATxmega16A4
ATxmega16D4
ATxmega16E5
ATXMega192A3
ATXMega192A3U
ATXMega192D3
ATXMega256A3B
ATXMega256A3BU
ATXMega256A3
ATXMega256A3U
ATXMega256D3
ATxmega32A4
ATxmega32A4U
ATXMega32C4
ATxmega32D4
ATxmega32E5
ATXMega384C3
ATXMega64A1
ATXMega64A3
ATXMega64A3U
ATxmega64A4U
ATXMega64D3
ATxmega64D4
ATxmega8E5

Part



10 International Resellers

10.1 International Resellers

Since the resellers list changes so now and then, it is not printed in this help. You can best look at the list at the MCS website.
See MCS Electronics web.

There is always a reseller near you. A reseller can help you in your own language and you are in the same time zone.
Sometimes there are multiple resellers in your country. All resellers have their own unique expertise. For example : industrial, robotics, educational, etc.

Index

- # -

#AUTOCODE 604
#ELSE 604
#ELSEIF 604
#ENDIF 604
#IF 604

- $ -

$AESKEY 606
$ASM 606
$BAUD 607
$BAUD1 608
$BGF 609
$BIGSTRINGS 611
$BOOT 612
$BOOTVECTOR 613
$CRYPT 624
$CRYSTAL 625
$DATA 626
$DBG 628
$DEFAULT 630
$EEPLEAVE 630
$EEPROM 631
$EEPROMHEX 632
$EEPROMSIZE 633
$END ASM 606
$EXTERNAL 634
$FILE 635
$FORECESOFTI2C 635
$FRAMECHECK 651
$FRAMEPROTECT 637
$FRAMESIZE 267, 641
$HWCHECK 651
$HWSTACK 267, 648
$INC 653
$INCLUDE 654
$INITMICRO 656
$LCD 657
$LCDPUTCTRL 659
$LCDPUTDATA 661
$LCDRS 662
$LCDVFO 664
$LIB 665
$LOADER 667
$LOADERSIZE 683
$MAP 684
$NOCOMPILE 685
$NOFRAMEPROTECT 637, 685
$NOINIT 686
$NORAMCLEAR 686
$NORAMPZ 686
$NOTRANSFORM 687
$NOTYPECHECK 688
$PROG 689
$PROGRAMMER 690
$PROJECTTIME 688
$REDUCEIVR 692
$REGFILE 694
$RESOURCE 694
$ROMSTART 697
$SERIALINPUT 698
$SERIALINPUT1 700
$SERIALINPUT2LCD 701
$SERIALOUTPUT 701
$SERIALOUTPUT1 702
$SIM 703
$SOFTCHECK 651
$STACKDUMP 703
$SWSTACK 267, 705
$TIMEOUT 708
$TINY 710
$TYPECHECK 710
$USER 711
$VERSION 712
$WAITSTATE 712
$XA 713
$XRAMSIZE 713
$XRAMSTART 714
$XTEAKEY 715

- 1 -

1WIRECOUNT 716
1WIRESEARCHNEXT 725
1WREAD 720
1WRESET 718
1WSEARCHFIRST 723
1WVERIFY 727
1WWRITE 729

- 2 -

2081 42
2082 41
2083 39
2084 38
2085 37
2086 34
2087 33

- 4 -

4808 497
4809 499

- A -

Abotu Help 51
ABS 736
ACOS 737
Add SPI 1708
Adding SRAM 4-port Non Multiplexed 257
Adding XRAM 251
Additional Hardware 242
ADR 731
ADR2 731
AESDECRYPT 1290
AESENCRYPT 1288
Alert 99
ALIAS 735
AlphaFunc 1625
AND 560, 738
ARDUINO 197
ARRAY 560
ASC 819
ASCII chart 600
ASIN 741
Assembler mnemonics 578
AT_EMULATOR 1836
AT86RF401 336
AT90CAN128 337
AT90CAN32 336
AT90PWM216 347
AT90PWM2-3 346
AT90S1200 338
AT90S2313 339
AT90S2323 339
AT90S2333 340
AT90S2343 340
AT90S4414 342
AT90S4433 342
AT90S4434 344
AT90S8515 345
AT90S8535 345
AT90US82 348
AT90USB1286 351
AT90USB1287 352
AT90USB162 349
AT90USB646 350
ATMEGA103 386
ATMEGA128 388
ATMEGA1280 416
ATMEGA1281 417
ATMEGA1284 418
ATMEGA1284P 418
ATMEGA128RFA1 389
ATMEGA16 368
ATMEGA1608 493
ATMEGA161 390
ATMEGA162 390
ATMEGA163 391
ATMEGA164P 392
ATMEGA164PA 392
ATMEGA165 394
ATMEGA165A 394
ATMEGA168 396
ATMEGA168P 396
ATMEGA168PB 397
ATMEGA169 397
ATMEGA169P 398
ATMEGA169PA 399
ATMEGA16A 369
ATMEGA16M1 372
ATMEGA16U2 366, 369
ATMEGA16U4 371
ATMEGA2560 420
ATMEGA2561 421
ATMEGA32 373
ATMEGA3208 495
ATMEGA323 400
ATMEGA324A 401
ATMEGA324P 402
ATMEGA324PA 402
ATMEGA324PB 403
ATMEGA325 404
ATMEGA3250P 422
ATMEGA328 405

ATMEGA328P 405
ATMEGA328PB 406
ATMEGA329 407
ATMEGA32C1 375
ATMEGA32M1 372, 376
ATMEGA32U2 366, 376
ATMEGA32U4 378
ATMEGA406 407
ATMEGA48 379
ATMEGA4808 497
ATMEGA4809 499
ATMEGA48P 379
ATMEGA48PA 379
ATMEGA48PB 380
ATMEGA603 408
ATMEGA64 381
ATMEGA640 410
ATMEGA644 411
ATMEGA644P 411
ATMEGA644PA 412
ATMEGA645 413
ATMEGA6450P 423
ATMEGA649 414
ATMEGA649PA 414
ATMEGA64C1 382
ATMEGA64M1 372, 383
ATMEGA8 366
ATMEGA808 491
ATMEGA8515 424
ATMEGA8535 424
ATMEGA88 383
ATMEGA88A 384
ATMEGA88P 385
ATMEGA88PA 385
ATMEGA88PB 386
ATMEGA8A 366
ATMEGA8U2 366
ATMEGAX 490
ATN 742
ATN2 743
Attaching an LCD Display 266
ATtiny12 352
ATtiny13 353
ATtiny13A 353
ATtiny15 353
ATTINY1604 482
ATTINY1606 483
ATTINY1607 484
ATTINY1614 485
ATTINY1616 486
ATTINY1617 487
ATTINY1634 363
ATTINY167 359
ATTINY20 353
ATTINY202 465
ATTINY204 466
ATTINY212 467
ATTINY214 468
ATtiny22 354
ATtiny2313 363
ATTINY2313A 364
ATtiny24 354
ATtiny25 355
ATtiny26 355
ATtiny261 360
ATTINY3216 488
ATTINY3217 489
ATTINY402 469
ATTINY404 470
ATTINY406 471
ATTINY412 472
ATTINY414 473
ATTINY416 474
ATTINY417 475
ATTINY4213A 364
ATtiny4313 365
ATTINY4313A 365
ATTINY43U 356
ATtiny44 356
ATTINY441 360
ATtiny45 356
ATtiny461 361
ATtiny48 357
ATTINY804 476
ATTINY806 477
ATTINY807 478
ATTINY814 479
ATTINY816 480
ATTINY817 481
ATTINY828 361
ATtiny84 358
ATTINY841 362
ATtiny85 358
ATtiny861 363
ATTINY87 358
ATtiny88 359
ATXMEGA 425
ATXMEGA128A1 445

ATXMEGA128A3 447
ATXMEGA128A4U 448
ATXMEGA128B1 448
ATXMEGA128C3 450
ATXMEGA128D3 451
ATXMEGA128D4 452
ATXMEGA16A4 434
ATXMEGA16D4 435
ATXMEGA16E5 436
ATXMEGA192A3 453
ATXMEGA192D3 454
ATXMEGA256A3 455
ATXMEGA256A3B 456
ATXMEGA256A3BU 458
ATXMEGA256D3 458
ATXMEGA32A4 437
ATXMEGA32A4U 439
ATXMEGA32D4 439
ATXMEGA32E5 440
ATXMEGA384C3 460
ATXMEGA64A1 441
ATXMEGA64A3 443
ATXMEGA64D3 443
ATXMEGA64D4 444
ATXMEGA8E5 433
ATXTINY1604 482
ATXTINY1606 483
ATXTINY1607 484
ATXTINY1614 485
ATXTINY1616 486
ATXTINY1617 487
ATXTINY202 465
ATXTINY204 466
ATXTINY212 467
ATXTINY214 468
ATXTINY3216 488
ATXTINY3217 489
ATXTINY402 469
ATXTINY404 470
ATXTINY406 471
ATXTINY412 472
ATXTINY414 473
ATXTINY416 474
ATXTINY417 475
ATXTINY804 476
ATXTINY806 477
ATXTINY807 478
ATXTINY814 479
ATXTINY816 480
ATXTINY817 481
AVR Internal Hardware 242
AVR Internal Hardware Port B 248
AVR Internal Hardware Port D 250
AVR Internal Hardware Watchdog timer 248
AVR Internal Registers 243
AVR ISP Programmer 175
AVR128DA28 549
AVR128DA32 550
AVR128DA48 551
AVR128DA64 552
AVR128DB 503
AVR128DB28 553
AVR128DB32 554
AVR128DB48 555
AVR128DB64 556
AVR16DD14 506
AVR16DD32 510
AVR16EA28 511
AVR16EA32 513
AVR16EA48 514
AVR32DA28 515
AVR32DA32 516
AVR32DA48 517
AVR32DB28 518
AVR32DB32 519
AVR32DB48 520
AVR32DD14 521
AVR32DD20 522
AVR32DD28 524
AVR32DD32 526
AVR32EA28 527
AVR32EA32 529
AVR32EA48 530
AVR64DA28 531
AVR64DA32 532
AVR64DA48 533
AVR64DA64 534
AVR64DB28 535
AVR64DB32 536
AVR64DB48 537
AVR64DB64 538
AVR64DD14 539
AVR64DD20 540
AVR64DD28 542
AVR64DD32 544
AVR64EA28 545
AVR64EA32 547
AVR64EA48 548

AVRD16DD28 508
AVR-DOS 773
AVR-DOS File I/O 773
AVR-DOS File System 1794
AVRX 503

- B -

BASCOM Editor Keys 237
BASCOMP 1863
BASE64DEC 1547
BASE64ENC 1549
BAUD 1488
BAUD1 1489
BCCALL 1838
BCCARD 1836
BCD 822
BCDEF 1837
BCINIT 887
BCRESET 1844
Begin_G 1625
BIN 824
BIN2GRAY 825
BINVAL 825
BIT 560
BitmapHandle 1627
BitmapLayout 1627
BitmapSize 1628
BitmapSource 1630
BitmapTransform 1631
BITS 804
BITWAIT 803
BlendFunc 1633
BLOAD 773
BOOT 697
BOOT LOADER 285
BOOTLOADER 285, 697
BOX 1317
BOXFILL 1319
BREAK 805
BSAVE 774
BUFSPACE 1491
BYTE 560
BYVAL 805

- C -

CALL 806, 1079
Call_C 1634
CAN 840
CANBAUD 840
CANCLEARALLMOBS 842
CANCLEARMOB 843
CANGETINTS 841
CANID 842
CANRECEIVE 843
CANRESET 844
CANSELPAGE 844
CANSEND 845
CASE 1437
Cell 1635
Changes 2078-2079 44
Changes compared to BASCOM-8051 559
Changes in 2080 43
CHARPOS 1519
CHECKFLOAT 744
CHECKSUM 808
CHECKSUMXOR 808
CHR 827
CIRCLE 1319
CLEAR 846
Clear_B 1635
CLEARATTR 776
ClearColorA 1636
ClearColorRGB 1637
ClearColorRGBdw 1638
ClearScreen 1640
ClearStencil 1639
ClearTag 1639
CLOCKDIVISION 847
CLOSE 848
CLS 1322
CMD16 1641
CMD32 1641
CMD8 1640
CmdAppend 1642
CmdBgColor2 1642
CmdButton 1643
CmdCalibrate 1644
CmdCalibratex 1645
CmdClock 1645
CmdColdStart 1648
CmdDial 1649
CmdDlStart 1650
CmdFgColor 1650
CMDFTSTACK 1651
CmdGauge 1652
CmdGetMatrix 1655

CmdGetPtr 1655
CmdGradColor 1656
CmdGradient 1657
CmdInflate 1658
CmdInterrupt 1659
CmdKeys 1659
CmdLoadIdentity 1661
CmdLoadImage 1662
CmdLogo 1662
CmdMemCpy 1663
CmdMemCrc 1663
CmdMemSet 1664
CmdMemWrite 1664
CmdMemZero 1665
CmdNumber 1665
CmdProgress 1667
CmdRegRead 1668
CmdRotate 1669
CmdRotateA 1670
CmdScale 1671
CmdScreenSaver 1672
CmdScrollBar 1672
CmdSetFont 1674
CmdSetMatrix 1674
CmdSketch 1674
CmdSlider 1675
CmdSnapshot 1677
CmdSpinner 1677
CmdStop 1680
CmdSwap 1680
CmdText 1680
CmdToggle 1682
CmdTrack 1684
CmdTranslate 1686
CmdTranslateP 1687
Color_A 1688
ColorMask 1688
ColorRGB 1689
ColorRGBdw 1690
Compact FlashCard Driver 1823
COMPARE 851
Compiler directives 606
CONFIG 853
CONFIG 1WIRE 857
CONFIG AC0 861
CONFIG AC1 861
CONFIG AC2 861
CONFIG ACA0 862
CONFIG ACA1 862
CONFIG ACB0 862
CONFIG ACB1 862
CONFIG ACI 861
CONFIG ACX 861
CONFIG ADC 864
CONFIG ADC0 880
CONFIG ADCA 867
CONFIG ADCB 867
CONFIG ADCx 880
CONFIG ATEMU 883
CONFIG BASE 886
CONFIG CANBUSMODE 889
CONFIG CANMOB 893
CONFIG CLOCK 895
CONFIG CLOCKDIV 909
CONFIG COM1 909
CONFIG COM2 911
CONFIG COMx 913
CONFIG DAC 918
CONFIG DAC0 923
CONFIG DACA 918
CONFIG DACB 918
CONFIG DATE 925
CONFIG DCF77 927
CONFIG DEBOUNCE 935
CONFIG DMA 936, 937
CONFIG DMACHx 937
CONFIG DMXSLAVE 949
CONFIG DP 951
CONFIG EDMA 944
CONFIG EDMAx 945
CONFIG EEPROM 952
CONFIG ERROR 953
CONFIG EVENT SYSTEM XTINY 956
CONFIG EVENT_SYSTEM 953
CONFIG EXTENDED_PORT 958
CONFIG FUSES 962
CONFIG GRAPHLCD 964
CONFIG HITAG 970
CONFIG I2CBUS 974
CONFIG I2CDELAY 975
CONFIG I2CSLAVE 978
CONFIG INPUT 983
CONFIG INPUTBIN 984
CONFIG INTVECTORSELECTION 987
CONFIG INTx 985
CONFIG KBD 988
CONFIG KEYBOARD 989
CONFIG LCD 992

CONFIG LCDBUS 998
CONFIG LCDMODE 1001
CONFIG LCDPIN 1001
CONFIG MODBUS 1005
CONFIG OPAMP 1005
CONFIG OSC 1008, 1009
CONFIG OSC XMEGA 1008
CONFIG OSC XTINY 1009
CONFIG PORT 1011
CONFIG PORT_MUX 1014
CONFIG POWER_REDUCTION 1023
CONFIG POWERMODE 1017
CONFIG PRINT 1028
CONFIG PRINT0 1028
CONFIG PRINT1 1028
CONFIG PRINT2 1028
CONFIG PRINT3 1028
CONFIG PRINTBIN 1029
CONFIG PRIORITY 1026
CONFIG PRIORITY XTINY 1027
CONFIG PS2EMU 1030
CONFIG RAINBOW 1033
CONFIG RC5 1037
CONFIG RND 1044
CONFIG SCL 1049
CONFIG SDA 1046
CONFIG SERIALIN 1051
CONFIG SERIALOUT 1057
CONFIG SERVOS 1071
CONFIG SHIFTIN 1060
CONFIG SINGLE 1059
CONFIG SPI 1061
CONFIG SPI0 1065
CONFIG SPI1 1061
CONFIG SPIC, SPID, SPIE, SPIF 1068
CONFIG STRCHECK 1075
CONFIG SUBMODE 1079
CONFIG SYSCLOCK 1081
CONFIG TC 1094
CONFIG TCA0 1084
CONFIG TCB0 1087
CONFIG TCB1 1087
CONFIG TCD0 1090
CONFIG TCPIP 1098
CONFIG TIMER0 1109
CONFIG TIMER1 1112
CONFIG TIMER2 1115
CONFIG TWI 1116
CONFIG TWI1 1116
CONFIG TWISLAVE 1123
CONFIG TWIXSLAVE 1128
CONFIG USB 1132
CONFIG USI 1138
CONFIG VARPTRMODE 1142
CONFIG VPORT 1144
CONFIG VREF 1147
CONFIG VREGPWR 1148
CONFIG WAITSUART 1149
CONFIG WATCHDOG 1149
CONFIG X10 1156
CONFIG XPIN 1158
CONFIG XRAM 1161
CONFIG ZCDx 1167
CONFIG_PRIORITY_XTINY 1027
CONFIG_TCA0 1084
CONFIG_TCB0-TCB1 1087
CONFIG_TCD0 1090
CONFIG_VREF 1147
CONST 1170
Constants 267
CONTINUE 1168, 1422
COS 746
COSH 747
COUNTER0 and COUNTER1 1172
CPEEK 1173
CPEEKH 1174
CRC16 813
CRC16UNI 815
CRC32 817
CRC8 809
CRC8UNI 811
CRCMB 818
CRYSTAL 1176
CURSOR 1325
custom design 50
Custom Shortcuts 150

- D -

DATA 1177
DATE 1193, 1209
DATE and Time 1180
DATE$ 1191, 1208
DATETIME 1835
DAYOFWEEK 1181
DAYOFYEAR 1190
DBG 1210
DCF77TIMEZONE 1210

Dead Code 91
DEBOUNCE 1212
DEBUG 1211
DECLARE 1079
DECLARE FUNCTION 1215
DECLARE SUB 1221
DECR 1214
DEFBIT 1227
DEFINT 1227
DEFLCDCHAR 1328
DEFLNG 1227
DEFSNG 1227
DEFWORD 1227
DEFxxx 1227
DEG2RAD 747
DELAY 1227
DELCHAR 1520
DELCHARS 1521
DESDECRYPT 1294
DESENCRYPT 1292
DIM 1228
DIR 777
DISABLE 1240
DISKFREE 778
DISKSIZE 779
DISPLAY 1329
Display_E 1690
DMACH0 937
DMACH1 937
DMACH2 937
DMACH3 937
DO 1243
DOUBLE 1830
DOWNTO 1260
DriveCheck 780
DriveGetIdentity 780
DriveInit 781
DriveReadSector 782
DriveReset 782
DriveWriteSector 784
DTMFOUT 1244

- E -

ECHO 1247
Edit Copy 81
Edit Cut 81
Edit Find 81
Edit Find Next 84
Edit Fold All Subs and Functions 87, 88
Edit Goto 86
Edit Goto Bookmark 86
Edit Indent Block 86
Edit Insert ASCII 87
Edit Paste 81
Edit Proper Indent 89
Edit Redo 81
Edit Remark Block 86
Edit Replace 85
Edit Show Dead Code 91
Edit Show Excluded Code 90
Edit Toggle Bookmark 86
Edit Undo 81
Edit Unfold All Code 88
Edit Unindent Block 86
EDMA 945
EDMAx 945
Elektor CF-Interface 1825
ELSE 1248, 1313
EM4095 RFID Reader 326
ENABLE 1250
ENCODER 1254
ENCRYPT 88
END 1256
END IF 1313
END SELECT 1437
END TYPE 1597
End_G 1691
EOF 785
ERAM 267
Error Codes 583
EUROTIMEDATE 1834
EXIT 1257
EXP 748
EXTENDED I2C 1741

- F -

File Close 76
File Exit 81
File New 76
File Open 76
File Print 77
File Print Preview 77
File Project 77
File Save 77
File Save As 77
File ZIP project files 80

FILEATTR 786
FILEDATE 787
FILEDATETIME 787
FILELEN 788
FILETIME 789
FIX 749
FLIP 187, 1259
FLUSH 790
FM24C16 1743
FM24C256 1745
FM24C64 1745
FM24C64_256-XMEGA 1746
FM25C256 1748
Font Editor 238
FOR 1260
FORMAT 828
FOR-NEXT 1260
FOURTHLINE 1332
FP_TRIG 1827
FRAC 750
FREEFILE 791
FT800 1619, 1707, 1708
FT800 Commands 1621
FT800 Demos 1710
FUNCTION 1215
FUSING 830

- G -

GET 1262
GETADC 1265
GETATKBD 1270
GETATKBDRAW 1274
GETATTR 791
GETDSTIP 1551
GETDSTPORT 1551
GETKBD 1275
GETRC 1277
GETRC5 1278
GETREG 1284
GETSOCKET 1553
GETTCPREGS 1555
Getting Started 1707
GLCD 1778
GLCDCMD 1333
GLCDDATA 1333
GLCDdSSD1306-I2C 1788
GLCDEADOGMXL240-7-I2C 1786
glcdR7565R 1784
GLCDSED 1778
GlcdSSD1325_96x64 1785
GOSUB 1284
GOTO 1286
GRAY2BIN 831

- H -

HBYTE
TYPE 583
Help About 230
Help Credits 234
Help Index 231
Help Knowledge Base 233
Help MCS Forum 231
Help MCS Shop 232
Help Support 233
Help Update 235
HEX 833
HEXVAL 832, 1753
HIGH 1287
HIGHW 1288
HOME 1334

- I -

I/O 773
I2C 978, 1300, 1756, 1757, 1763
I2C TWI Slave 1831
I2C_MULTIBUS 1753
I2C_TWI-MULTI 1754
I2C_USI 1756
I2C_USI_SLAVE 1757
I2CINIT 1300
I2CRBYTE 1306
I2CRECEIVE 1303
I2CREPSTART 1306
I2CSEND 1305
I2CSLAVE 978, 1830
I2CSTART 1306
I2CSTOP 1306
I2CSTOP: I2CRBYTE: I2CWBYTE 1306
I2CWBYTE 1306
I2START 1306
IDLE 1017, 1313
IF 1313
IF-THEN-ELSE-END IF 1313
INCR 1314
Index 28

INITFILESYSTEM 792
INITLCD 1334
INKEY 1492
INP 1315
INPUT 1493
INPUTBIN 1497
INPUTHEX 1496
INSERTCHAR 1522
Install on multiple computers 67
Installation of BASCOM 53
INSTR 1526
INT 751
INTEGER 560
IP 1547
IP2STR 1555
ISCHARWAITING 1498
ISP programmer 165

- J -

JOIN 1524
Jump 1691

- K -

Keyword Reference 29
KILL 793
KITSRUS Programmer 167

- L -

Language Fundamentals 560
Lawicel BootLoader 174
LBYTE 583
LCASE 1527
LCD 1335, 1775
LCD RGB-8 Converter 1862
LCD_DOGS104a_I2C 1780
LCD_I2C_PCF8574 1790
LCD4.LIB 1776
lcd4_anypin_oled_RS0010 1775
LCD4BUSY 1774
LCD4E2 1777
LCDAT 1339
LCDAUTODIM 1338
LCDCMD 1341
LCDCONTRAST 1342
LCDDATA 1341
LCD-EPSON 1780
LCDFONT 1342
LEFT 1528
LEN 1529
Lib Manager 132
LibManager 132
LIBUSB 216
LINE 1344
LINE_INPUT 794
LINEINPUT 794
LineWidth 1692
LOAD 1358
LOADADR 1358
LOADLABEL 1359
LOADWORDADR 1359
LOC 795
LOCAL 1360
LOCATE 1347
LOF 796
LOG 752
LOG10 752
LONG 560
LOOKDOWN 1363
LOOKUP 1365
LOOKUPSTR 1366
LOOP 1243
LOW 1367
LOWERLINE 1347
LTRIM 1529

- M -

M128-1wire-PortF 1766
MACRO 1368
Macro_R 1693
MAKEBCD 1369
MAKEDEC 1369
MAKEINT 1370
MAKEMODBUS 1499
MAKETCP 1556
MANCHESTERDEC 834
MANCHESTERENC 835
MAX 1370
MCS Bootloader 184
MCS EDBG Programmer 207
MCS Universal Interface Programmer 168
MCSBYTE 1764
MCSBYTEINT 1764
MEGAX 490
MEMCOPY 1372

MEMFILL 1374
Memory usage 267
MID 1530
MIN 1375
Mixing ASM and BASIC 573
MMCSD_HC.LIB 1823
MOD 1376
MODBUS Slave Server 1858
MyAVR/MK2/AVR910 programmer 174
mySmartUSB Light 200

- N -

NAME 797
NBITS 1377
new 33, 34
New CF-Card Drivers 1827
Newbie problems 589
NEXT 1260
NOP 1378
NOSAVE 1379
NOT 753

- O -

ON INTERRUPT 1379
ON VALUE 1383
OPEN 1386
Options Communication 149
Options Compiler 147
Options Compiler 1WIRE 147
Options Compiler Chip 143
Options Compiler Communication 146
Options Compiler I2C 147
Options Compiler LCD 148
Options Compiler Output 145
Options Compiler SPI 147
Options Environment 150
Options Monitor 225
Options Printer 225
Options Programmer 162
Options Select Settings File 226
Options Simulator 161
OR 560, 755
OUT 1394

- P -

PCF8533 1778
PEEK 1395
PG302 programmer 166
PointSize 1693
POKE 1396
POPALL 1397
POWER 758, 1017
POWER MODE 1397
Power Up 325
POWERDOWN 1398
POWERSAVE 1399
PRINT 1501
PRINTBIN 1504
PROGGY 186
Program Compile 108
Program Development Order 238
Program Reset Chip 128
Program Send to Chip 124
Program Show Result 110
Program Simulate 111
Program Syntax Check 109
Project Close 77
Project New 77
Project Open 77
Project Save 77
PS2MOUSE_EMULATOR 1836
PS2MOUSEXY 1399
PSET 1348
PULSEIN 1400
PULSEIN.LIB 1764
PULSEOUT 1401
PUSHALL 1402
PUT 1402

- Q -

QCOS 764
QSIN 760
QUOTE 1532

- R -

RAD2DEG 764
RB_ADDCOLOR 1468
RB_ANDCOLOR 1468
RB_CHANGEPIN 1465
RB_CLEARCOLORS 1472
RB_CLEARSTRIPE 1471
RB_COLOR 1484
RB_COPY 1486

RB_FILL 1472
RB_FILLCOLORS 1473
RB_FILLSTRIPE 1475
RB_GETCOLOR 1483
RB_LOOKUPCOLOR 1484
RB_ORCOLOR 1470
RB_ROTATELEFT 1476
RB_ROTATERIGHT 1477
RB_SELECTCHANNEL 1462
RB_SEND 1464
RB_SETCOLOR 1463
RB_SETTABLECOLOR 1479
RB_SHIFTLEFT 1478
RB_SHIFTRIGHT 1479
RB_SUBCOLOR 1470
RB_SWAPCOLOR 1475
RC5SEND 1405
RC5SENDEXT 1409
RC6SEND 1411
RD16 1694
RD32 1695
RD8 1694
READ 1413
READEEPROM 1415
READHITAG 1417
READMAGCARD 1420
READSIG 1424
READSIGNATURE 1424
READUSERSIG 1427
REDO 1168, 1422
Registration 67
REM 1427
REPLACECHARS 1530
Resellers 1869
Reserved Words 583
RESET 1428
RESTORE 1429
RestoreContext 1696
RETURN 1430
Return_C 1696
RGB8TO16 1350
RIGHT 1531
RND 1431
ROTATE 1432
ROUND 765
RS0010 1775
RTRIM 1532
Running BASCOM-AVR 69

- S -

Sample Electronics cable programmer 166
SAVE 1379
SAVEALL 1379
SaveContext 1697
ScissorSize 1697
ScissorXY 1698
SCREEN CAPTURE 1709
SECELAPSED 1202
SECOFDAY 1203
SEEK 1434
SELECT 1437
SELECT-CASE-END SELECT 1437
SENDSCAN 1441
SENDSCANKBD 1443
SERIN 1506, 1765
SEROUT 1509
SET 1438
SETATTR 798
SETFONT 1351
SETIPPROTOCOL 1556
SETREG 1441
SETTCP 1559
SETTCPREGS 1560
SGN 766
SHIFT 1447
SHIFTCURSOR 1353
SHIFTIN 1449
SHIFTLCD 1354
SHIFTOUT 1453
SHOWPIC 1355
SHOWPICE 1355
SIGNATURE 1424
SIN 769
SINGLE 560
SINH 769
SIZEOF 1457
SNTP 1562
SOCKETCLOSE 1564
SOCKETCONNECT 1567
SOCKETDISCONNECT 1570
SOCKETLISTEN 1571
SOCKETSTAT 1571
Software_vs_Hardware_I2C_or_TWI 1753
SONYSEND 1454
SORT 1458
SOUND 1460

SPACE 1533
SPC 1508
SPI 1513, 1708
SPI1IN 1513, 1519
SPI1INIT 1519
SPI1MOVE 1515, 1519
SPI1OUT 1518, 1519
SPIIN 1513
SPIINIT 1514
SPIMOVE 1515
SPIOUT 1518
SPISLAVE 1831
SPLIT 1534
SQR 768
Stack 267
START 1538
Statements and Hardware Resources 278
STCHECK 1540
StencilFunc 1699
StencilMask 1700
StencilOp 1700
STEP 1260
STK500 Programmer 170
STK600 194
STOP 1544
STR 836
STR2DIGITS 837
STRING 1536
SUB 1079, 1545
Supported Programmers 164
SWAP 1545
SYSDAY 1206
SYSSEC 1204
SYSSECELAPSED 1206

- T -

Tag 1701
TagMask 1702
TAN 767
TANH 767
TCC0 1094
TCC1 1094
TCD0 1094
TCD1 1094
TCP 1547
TCP/IP 1547
TCPCHECKSUM 1574
TCPIP 1765
TCPREAD 1576
TCPWRITE 1578
TCPWRITESTR 1579
THEN 1313
THIRDLINE 1357
TIME 1180, 1209
TIME$ 1208
TIMER 1094
TIMER0 245
TIMER1 246
Tips and tricks 595
TOGGLE 1595
Tools Batch Compile 135
Tools Font Editor 141
Tools Graphic Converter 133
Tools LCD Designer 131
Tools LIB Manager 132
Tools PDF Update 138
Tools Plugin Manager 135
Tools Resource Editor 140
Tools Stack Analyzer 134
Tools Terminal Emulator 128
TRIM 1536
TVOUT 1766
TWI 1300
TWI_START 1116
TWICSLAVE 1128
TWIDSLAVE 1128
TWIESLAVE 1128
TWIFSLAVE 1128
TWISLAVE 1128
TYPE 1597

- U -

UCASE 1537
UDP 1547
UDPREAD 1582
UDPREADHEADER 1585
UDPWRITE 1588
UDPWRITESTR 1589
UpdateScreen 1704
UPDI programmer 202
UPPERLINE 1357
URL2IP 1592
USB Addon 1845
USB-ISP Programmer 175
USI 321, 1756, 1757
Using RS485 295

Using the 1 WIRE protocol 310
Using the I2C protocol 297
Using the SPI protocol 314
USING the UART 278

- V -

VAL 839
VAREXIST 604
VARPTR 1142, 1604
VER 1605
VERSION 1606
Vertex2f 1702
Vertex2ii 1703
View Code Explorer 101
View Error Panel 99
View PDF viewer 97
View PinOut 93
View Project Files 100
View Show Alert Window 99

- W -

WAIT 1607
WaitCmdFifoEmpty 1705
WAITKEY 1511
WAITMS 1607
WAITUS 1608
WEND 1609
what is new in 2085 37
WHILE 1609
WHILE-WEND 1609
Window Arrange Icons 229
Window Maximize All 229
Window Minimize All 229
Window Tile 228
Window Tile Vertically 229
Windows Cascade 227
WORD 560
WR16 1706
WR32 1706
WR8 1705
WRITE 801
WRITEDAC 1610
WRITEEEPROM 1611

- X -

X10DETECT 1613
X10SEND 1615
XOR 560, 770
XRAM 267
XRAM CF-Interface for simulation 1826
XTEADECODE 1298
XTEAENCODE 1297
XTINY 460

www.mcselec.com

Making BASIC and Micro's Easy
Since 1995 !