

# AN # 192, Application Note

## Getting Started With ATXMEGA and BASCOM-AVR Part 1

---

By MAK3

This Application Note will not cover BASCOM-AVR programming basics so you should know BASCOM-AVR Language Fundamentals.

<http://avrhelp.mcselec.com/>

[http://avrhelp.mcselec.com/language\\_fundamentals.htm](http://avrhelp.mcselec.com/language_fundamentals.htm)

This Application Note will not cover Analog I/O such as Analog Input, Analog Output or Analog Comparator. There will be a Part 2 of Getting Started with ATXMEGA which will cover the Analog Part of XMEGA.

At first to get started with ATXMEGA there are a lot of White Papers and Application Notes available from ATMEL:

White Paper:

[http://www.atmel.com/dyn/resources/prod\\_documents/doc7926.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc7926.pdf)

Application Notes:

[http://www.atmel.com/dyn/products/documents.asp?category\\_id=163&family\\_id=607&subfamily\\_id=1965](http://www.atmel.com/dyn/products/documents.asp?category_id=163&family_id=607&subfamily_id=1965)

<http://www.atmel.com/xmega>

You can find samples for XMEGA in Bascom-AVR folder:

- General samples ..... \BASCOSM-AVR\SAMPLES\XMEGA
- Bootloader samples in ..... \BASCOSM-AVR\SAMPLES\BOOT
- For xm128a1 in ..... \BASCOSM-AVR\SAMPLES\CHIPS

The ATXMEGA work with 3.3V so please do not connect something which output 5V. Use a Level Shifter for that. <http://www.maxim-ic.com/app-notes/index.mvp/id/3007>

The maximum rating for a XMEGA Pin is 3.6 V.

When using the internal 32MHz oscillator you need at least 2.7V Vcc. The maximum CPU Clock Frequency is 12MHz when using the XMEGA with 1.6V Vcc.

All XMEGA 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.

## Table of Contents

Manuals for ATXMEGA: There are 2 manuals available from ATMEL .....	3
What you need to get started with ATXMEGA and BASCOM-A VR .....	3
Supported XMEGA Chips in Bascom-A VR (Status of Bascom-A VR Version 2.0.7.2) .....	3
Supported Functions for XMEGA in Bascom-A VR (Status of Bascom-A VR Version 2.0.7.2) .....	3
The Program and Debug Interface (PDI).....	4
Fuse Bits.....	4
Now you can flash your first Bascom-A VR Program where you toggle an Output: .....	6
The most important parts of an Bascom-A VR Program for XMEGA are: .....	8
Port I/O.....	9
External Pin Interrupt.....	10
System Clock Options (internal and external oscillator options).....	11
ATXMEGA128A1 with EXTERNAL oscillator 16MHz .....	12
USART - Universal Synchronous and Asynchronous Serial Receiver and Transmitter.....	13
Using Interrupts with UARTS:.....	15
Using Dynamic Channels with XMEGA COM Ports .....	16
ATXMEGA and RS-485.....	16
Using a Bootloader with ATXMEGA .....	17
Using EEPROM .....	25
How to initiate a software Reset of ATXMEGA: .....	26
Using the Watchdog of ATXMEGA: .....	26
TC - 16-bit Timer/Counter .....	27
Now we use the RTC - Real Time Counter for generating 1 Second Tick: .....	28
Using config tctx to easy configure a PWM with XMEGA .....	28
Easy Frequency Generation with XMEGA.....	30
XMEGA I2C bzw. TWI in MASTER MODE.....	31
Using a LCD with XMEGA:.....	31
Reduce power consumption by setting Power Reduction Register:.....	34
Using the Event System (first example): .....	34
ATXMEGA as SPI Master.....	35
ATXMEGA as SPI SLAVE.....	37
Using Dynamic Channels with XMEGA SPI .....	39
Power Mode Options .....	39
Virtual Port Registers.....	39
AES Crypto Module .....	40
DMA (Direct Memory Access).....	42

Reading and Writing to A TXMEGA Registers .....	47
Great existing Bascom-AVR A TXMEGA Projects:.....	48

### Manuals for ATXMEGA: 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.

### 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 XMEGA).
2. An evaluation board like the Atmel AVR XMEGA® Xplained evaluation kit or any other XMEGA evaluation board with PDI (Program and Debug Interface)header.
3. A Programmer like AVRISP MKII [http://www.atmel.com/dyn/products/tools\\_card.asp?tool\\_id=3808](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3808) or any other PDI or JTAG programmer which support XMEGA.
4. Latest AVR-Studio 4.X [http://www.atmel.com/dyn/products/tools\\_card.asp?tool\\_id=2725](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725) mainly for setting fuse bits and to flash Bootloader to ATXMEGA.

### Supported XMEGA Chips in Bascom-AVR (Status of Bascom-AVR Version 2.0.7.2)

A TXMEGA16A4, A TXMEGA16D4, A TXMEGA32A4, A TXMEGA32D4, A TXMEGA64A1, A TXMEGA64A3, A TXMEGA64D3, A TXMEGA62D4, A TXMEGA128A1, A TXMEGA128A3, A TXMEGA128D3, A TXMEGA128D4, A TXMEGA192A3, A TXMEGA192D3, A TXMEGA256D4, A TXMEGA256A3, A TXMEGA256A3B, A TXMEGA256D3

### Supported Functions for XMEGA in Bascom-AVR (Status of Bascom-AVR Version 2.0.7.2)

- ADC - Analog to Digital Converter
- DAC - Digital to Analog Converter
- AC - Analog Comparator
- Analog Comparator
- I/O Ports
- System Clock and Clock options
- WDT - Watchdog Timer
- Power Management and Sleep Modes
- System Control and Reset
- Real-time Clock
- EBI - External Bus Interface

- Interrupts and Programmable Multi-level Interrupt Controller
- DMA - Direct Memory Access Controller
- TC - 16-bit Timer/Counter
- RTC - Real Time Counter
- AWeX - Advanced Waveform Extension
- Hi-Res - High Resolution Extension
- USART - Universal Synchronous and Asynchronous Serial Receiver and Transmitter
- SPI - Serial Peripheral Interface
- I2C/ TWI - Two Wire Serial Interface
- AES Crypto Engine
- Memory Programming
- 1-WIRE functions

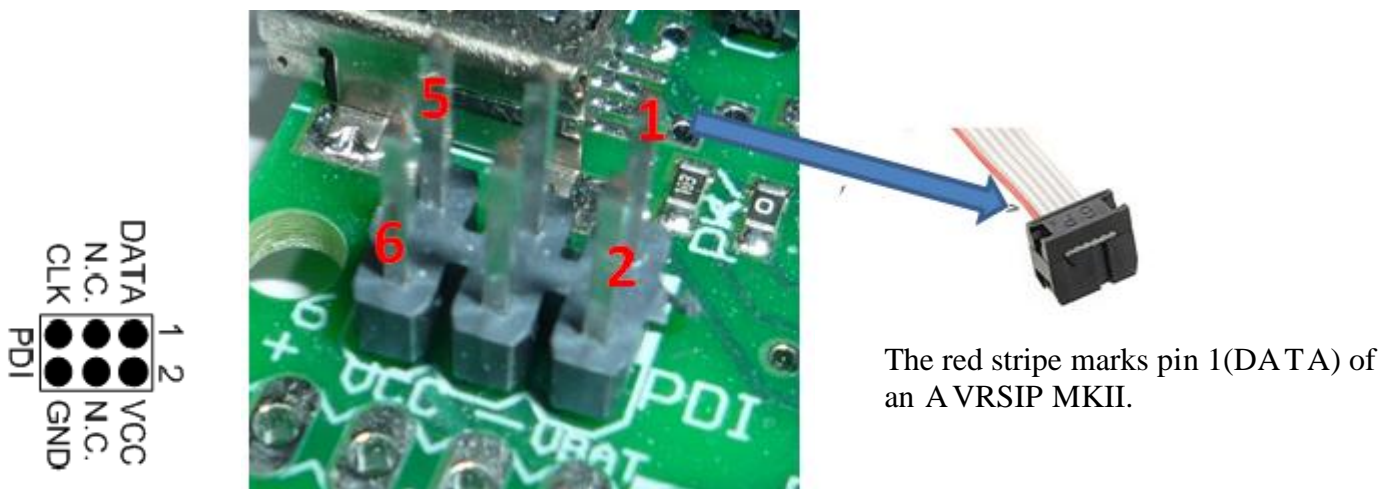
### The Program and Debug Interface (PDI)

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 megaA VR. Nor does it have the debugWIRE interface. These have been replaced by a two wire “Programming and Debugging Interface” (PDI).” [from Atmel App Note A VR1005]

Following the PDI Header Pin out (top view):

Important is also that the AVRISP MKII need 3.3V supply voltage from the Target.

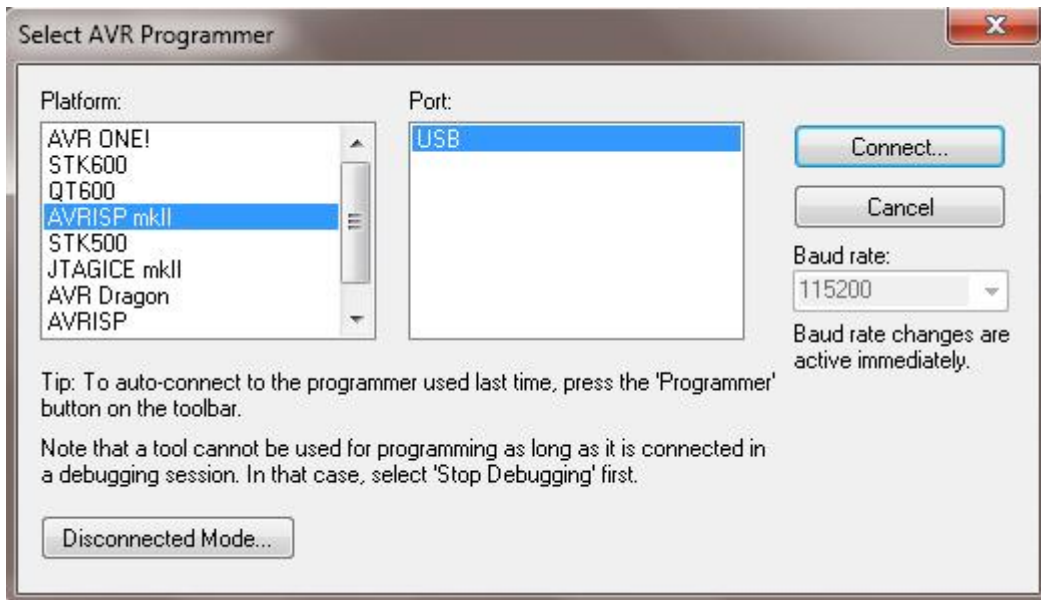


### Fuse Bits

“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.....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]

You do not need to change any Fuse Bit to get started except you want to use a Bootloader.

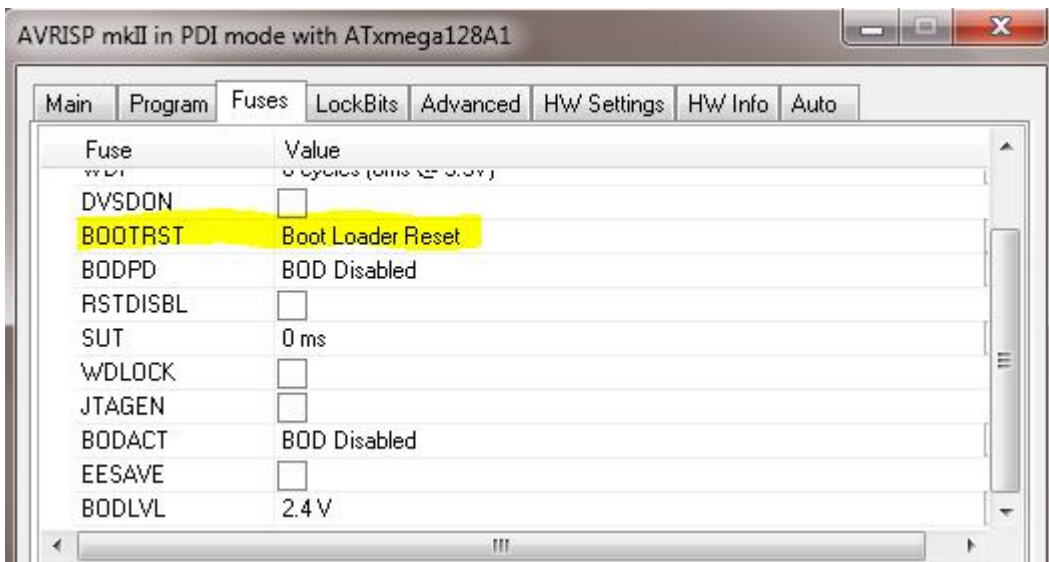
In AVR-Studio select **Tools >>> Program AVR >>> Connect...**  
And choose the programmer:



Then hit **Connect...** Button and select the XMEGA Type:



After this go to Tab Fuses:



Now you can flash your first Bascom-AVR Program where you toggle an Output:

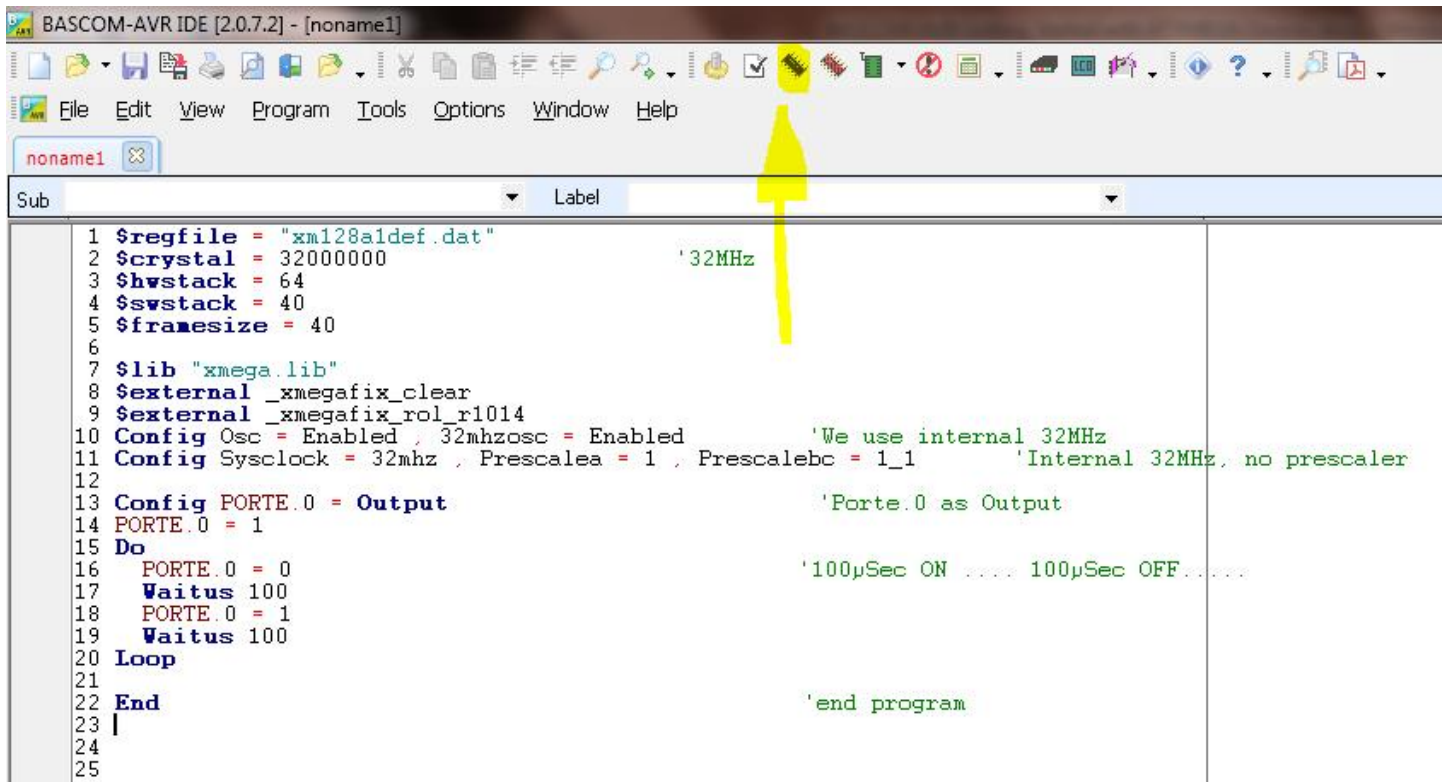
```
$regfile = "xm128a1def.dat"
$crystal = 32000000           ' 32MHz
$hwstack = 64
$swstack = 40
$framesize = 40

$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
Config Osc = Enabled , 32mhzosc = Enabled           'We use internal 32MHz
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1           'Internal 32MHz, no prescaler

Config Porte.0 = Output           'Porte.0 as Output
Porte.0 = 1
Do
  Porte.0 = 0           '100µSec ON .... 100µSec OFF.....
  Waitus 100
  Porte.0 = 1
  Waitus 100
Loop
End           'end program
```

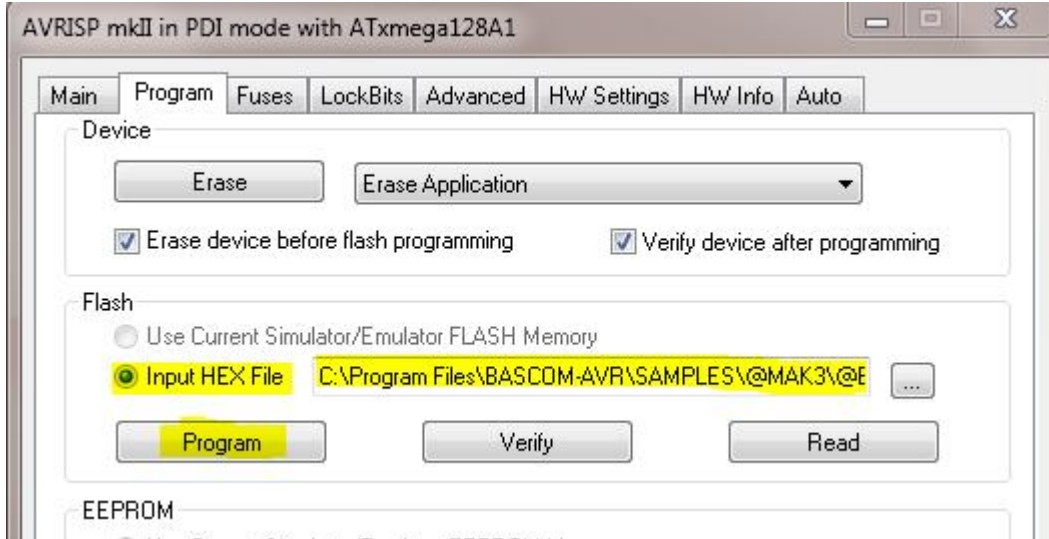


Copy the example in the Bascom-AVR Environment and hit F7 to compile it.



```
1 $regfile = "xm128a1def.dat"
2 $crystal = 32000000          '32MHz
3 $hwstack = 64
4 $swstack = 40
5 $framesize = 40
6
7 $lib "xmega.lib"
8 $external _xmegafix_clear
9 $external _xmegafix_rol_r1014
10 Config Osc = Enabled , 32mhzosc = Enabled          'We use internal 32MHz
11 Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1          'Internal 32MHz, no prescaler
12
13 Config PORTE.0 = Output          'Porte.0 as Output
14 PORTE.0 = 1
15 Do
16     PORTE.0 = 0          '100µSec ON ..... 100µSec OFF.....
17     Waitus 100
18     PORTE.0 = 1
19     Waitus 100
20 Loop
21
22 End          'end program
23 |
24
25
```

Here is the screen shot how to program the HEX File generated by Bascom-AVR to the ATXMEGA:



## The most important parts of an Bascom-AVR Program for XMEGA are:

### 1. The Register file for the Chip, crystal init and Stacks

```
$regfile = "xm128aldef.dat"  
$crystal = 32000000 '32MHz  
$hwstack = 64  
$swstack = 40  
$framesize = 40
```

For more information on \$hwstack , \$swstack and \$framesize please read Application Note:  
AN #183 - "Reveal the secret of Stack" BASCOM-AVR - Part 1  
[http://www.mcselec.com/index.php?option=com\\_content&task=view&id=286&Itemid=57](http://www.mcselec.com/index.php?option=com_content&task=view&id=286&Itemid=57)

### 2. The xmega.lib and fix

The reason for xmega.lib and fix is following:

“In the XMEGA variant, the working register file is not mapped into the data address space; as such, it is not possible to treat any of the XMEGA's working registers as though they were SRAM. Instead, the I/O registers are mapped into the data address space starting at the very beginning of the address space.”  
[from Wikipedia]

When the normal libs cannot be used, the alternative special XMEGA code is placed in xmega.lib  
For example the print and val/str use these external routines.

```
$lib "xmega.lib"  
$external _xmegafix_clear  
$external _xmegafix_rol_r1014
```

### 3. Enable and configure the oscillator of your choice:

```
Config Osc = Enabled , 32mhzosc = Enabled ' enable 2 MHz and 32 MHz internal oscillators
```

4. 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
```

5. 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
```

6. 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
```

7. After this you can add Enable Interrupts and your code in the Main Loop.



---

## Port I/O

The Port pins will be configured set and reset like with ATTINY or ATMEGA devices

```
Config Porte.0 = Output
Config Pine.1 = Input
```

And you read it with for example:

```
if PINE.0 = 1 then
  'This
Else
  'That
end if
```

What's changed is how to configure for example Pull-up or Pull down:

The Pin Configuration register is used for additional I/O pin configuration.

(for example for Porte Pin 0 the register is [Porte\\_pin0ctrl](#))

A pin can be set in a totem-pole, wired-AND, or wired-OR configuration. It is also possible to enable inverted input and output for the pin or slew rate limiting over this register.

In addition to that you can set the pin sense configuration like rising edge, falling edge or both edges ...

**Enable Pull-up:**

```
Porte_pin0ctrl = &B00_011_000 ' Pullup
```

Or

```
Porte_pin0ctrl = Bits(3 , 4) ' Pullup
```

**Enable Pull down:**

```
Porte_pin0ctrl = &B00_010_000 ' Pulldown
```

Or

```
Porte_pin0ctrl = Bits( 4) ' Pulldown
```

In the ATXMEGA Manual search for *PINnCTRL - Pin n Configuration Register*

For a single port you have 8 Register:

```
PORTE_PIN0CTRL
PORTE_PIN1CTRL
PORTE_PIN2CTRL
PORTE_PIN3CTRL
PORTE_PIN4CTRL
PORTE_PIN5CTRL
```

```
PORTE_PIN6CTRL
PORTE_PIN7CTRL
```

To set more or all register with the same setting you can use MPCMASK register before the Pin Control register is written:

(MPCMASK - Multi-pin Configuration Mask Register)

The MPCMASK register enables several pins in a port to be configured at the same time.

```
Portcfg_mpcmask = &B00000111          'Same setting for Pin 0, 1 and 2
```

See following Example which is also more readable by using constants for the single Bits:

```
' Bit's of PINnCTRL Register
Const Isc0 = 0          'Input/Sense Configuration 0
Const Isc1 = 1          'Input/Sense Configuration 1
Const Isc2 = 2          'Input/Sense Configuration 2
Const Opc0 = 3          'Output and Pull Configuration 0
Const Opc1 = 4          'Output and Pull Configuration 1
Const Opc2 = 5          'Output and Pull Configuration 2
Const Inven = 6         'Inverted I/O Enable
Const Srlen = 7         'Slew Rate Limit Enable
Portcfg_mpcmask = &B00000111          'Port F Mask
Portf_pin0ctrl = Bits(opc1 , Opc0 , Isc1)  'opc[0..2] = 011 --> Pull-up (on input)
                                           'isc[0..2] = 010 --> FALLING Sense falling edge
```

## External Pin Interrupt

Example for using Pin 0 of Port E as external Interrupt on falling edge

At first enable the Interrupts:

```
On Porte_int0 Port_e_int0_isr
Enable Porte_int0 , Lo          'Enable this Interrupt as Lo Level Interrupt
Enable Interrupts
```

Configure the Pin as Input:

```
Config Pine.0 = Input          'Set PINE.0 as Input
Porte_pin0ctrl = &B00_011_010  'Enable Pullup and reaction on falling edge
                        '^ ^
                        '^ Reaction on falling edge
                        '^
                        'Enable Pullup
```

Setup the Interrupt Mask:

```
Porte_int0mask = &B0000_0001    'include PIN0 in INT0 Mask
```

After this you only need a Main Loop and of course an Interrupt Service Routine called port\_E\_int0\_isr

```
'Port E INTO Interrupt Service Routine
port_E_int0_isr:
    Set Pine_0_int_flag
return
```

### System Clock Options (internal and external oscillator options)

Up to now we used the internal 32MHz oscillator. Here we use the internal 2MHz and in addition also a Prescaler of 2 ending in a 1MHz Clock for the XMEGA.

Don't forget to set the \$crystal = 1000000 according to the other settings.

```
$regfile = "xml28a1def.dat"
$crystal = 1000000          '1MHz
$hwstack = 64
$swstack = 40
$framesize = 40

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014

Config Osc = Enabled
Config Sysclock = 2mhz , Prescalea = 1 , Prescalebc = 1_2          '1MHz

Config Com5 = 9600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #1
waitms 5

Print #1 ,
Print #1 , "1MHz Test"
Print #1 , "Prescaler Control Register = " ; Bin(clk_psctrl)

Do
    !NOP
Loop
End          'end program
```

After reset, the device will always start up running from the 2 MHz internal oscillator until it will be changed by Config Osc and Config Sysclock.

That the reason why also following works with \$crystal = 2000000 without Config Osc and Config Sysclock.

But it is always recommended to use Config Osc and Config Sysclock.

```
$regfile = "xml28a1def.dat"
$crystal = 2000000          '2MHz
$hwstack = 64
$swstack = 40
$framesize = 40

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014

Config Com5 = 9600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #1
waitms 5

Print #1 ,
```

```
Print #1 , "After reset, the device will always start up running from the 2 MHz internal oscillator"
```

```
Do
  !NOP
Loop
End                                     'end program
```

There are also additional nice features for the clock system:

A calibration feature (DFLL) is available, and can be used for automatic runtime calibration of the internal oscillators.

```
Osc_dflctrl = &B00000000          'The internal 32.768 KHz Osc is used for calibration
Set Dflrc32m_ctrl.0              'Enable DFLL and autocalibration
```

A Crystal Oscillator Failure Monitor can be enabled to issue a Non-Maskable Interrupt and switch to internal oscillator if the external oscillator fails. The External Clock Source Failure Monitor is disabled by default, and it must be enabled from software before it can be used.

For this you need to set the **OSC\_XOSCFAIL** Register and the according interrupt **OSCFAIL**

```
'enable change of protected Registers for following 4 CPU Instruction Cycles
Cpu_ccp = &HD8
Osc_xoscfail.0 = 1
```

See also Manual for: XOSCFAIL - XOSC Failure Detection Register

## ATXMEGA128A1 with EXTERNAL oscillator 16MHz

It is easy to configure with Config Osc and Config Sysclock.

```
$regfile = "xm128a1def.dat"
$hwstack = 64
$swstack = 40
$framesize = 40

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014

$crystal = 1600000          ' 16MHz
'Enable the external oscillator with Range 12MHz...16MHz
Config Osc = Disabled , Extosc = Enabled , Range = 12mhz_16mhz , Startup = Xtal_256clk
Config Sysclock = External , Prescalea = 1 , Prescalebc = 1_1          'EXTERNAL 16MHz

Config Com5 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #5

Dim B As Byte

Print #5 , "We use external 16MHz oscillator"

#####[MAINLOOP]#####
Do
```

```

Incr B
Print #5 , "Test " ; Str(b)
Wait 2
Loop
'#####[MAINLOOP]#####
End          'end program

```

## USART - Universal Synchronous and Asynchronous Serial Receiver and Transmitter

You can use and configure up to 8 UARTS with an ATXMEGA128A1.

```

Config Com1 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM1:" For Binary As #1
Config Com2 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM2:" For Binary As #2
Config Com3 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM3:" For Binary As #3
Config Com4 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM4:" For Binary As #4
Config Com5 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #5
Config Com6 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM6:" For Binary As #6
Config Com7 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM7:" For Binary As #7
Config Com8 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM8:" For Binary As #8

```

You can use the serial buffer function [Config serialin](#) and [Config serialout](#) for COM1 .....COM4. Serial buffer for COM5...COM8 is not supported at the moment.

Don't forget the interface number like(#3) in the UART functions for example:

```

Ischarwaiting(#3)
Inkey(#3)

```

Regarding config Serialin and the BYTEMATCH option see following example:

```

' Using CONFIG SERIALIN and BYTEMATCH with XMEGA

$regfile = "xm32a4def.dat"
$crystal = 32000000          '32MHz
$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

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

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

'Config Interrupts
Config Priority = Static , Vector = Application , Lo = Enabled      'Enable Lo Level Interrupts

```

```

'CONFIG COM1
Config Com1 = 38400 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Serialin = Buffered , Size = 20 , Bytematch = ALL

Enable USARTC0_RXC, lo 'Enable Receive Interrupt for COM1

'CONFIG COM2
Config Com2 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Serialin1 = Buffered , Size = 20 , Bytematch = 66
Open "COM2:" For Binary As #2

Enable USARTC1_RXC, lo 'Enable Receive Interrupt for COM2

Print "-----CONFIG SERIALIN and BYTEMATCH with XMEGA-----"

Enable Interrupts

Dim Buffer_full As Bit
Dim I As Byte

'#####[MAINLOOP]#####
Do
  If Buffer_full = 1 Then
    Reset Buffer_full          'Reset Flag
    Print "This is the buffer content: " ;
    For I = 1 To _rs_bufcountr0
      Print Chr(_rs232inbuf0(i)) ;      'Print the complete Buffer
    Next
    Print
    Clear Serialin          'Clear Buffer
  End If
Loop
'#####[MAINLOOP]#####

'Label called when COM1 received a BYTE

Serial0byterecieved: 'For Bytematch = ALL
  Pushall
  Print Str(_rs_bufcountr0) ; " Byte in Buffer" ;      'Don't use print in Interrupt Service
  Routines. This is only for testing.
  Print " Bufferhead = " ; Str(_rs_head_ptr0) ;
  Print " Buffertail = " ; Str(_rs_tail_ptr0)
  If _rs_bufcountr0 = 20 Then Set buffer_full
  Popall
Return

'Label called when COM2 received a B

Serial1charmatach:
  Pushall
  Print #2 , "we got a B"
  Popall
Return

```

```
End
```

```
Close #2
```

## Using Interrupts with UARTS:

Every of the 8 USART's has for example a Receive Interrupt.

ATXMEGA128A1:

COM1 --> Usartc0\_rxc

COM2 --> Usartc1\_rxc

COM3 --> Usartd0\_rxc

COM4 --> Usartd1\_rxc

COM5 --> Usarte0\_rxc

COM6 --> Usarte1\_rxc

COM7 --> Usartf0\_rxc

COM8 --> Usartf1\_rxc

At first you config the interface

```
Config Com2 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8  
Open "COM2:" For Binary As #2
```

As with other Interrupts with XMEGA you configure the Receive Interrupt this way:

```
'Enable Receive Interrupt for COM1  
On Usartc1_rxc Rxc_isr  
Enable Usartc1_rxc , Io  
Enable Interrupts
```

Then you need the Interrupt Service Routine:

```
Rxc_isr:  
  Rs232 = Inkey(#2)  
  'do something with rs232 data  
Return
```

It is important to know that the function Inkey is also resetting the Interrupt flag. So when you not use Inkey you need to reset the Interrupt flag manual otherwise it fires again and again.



## Using Dynamic Channels with XMEGA COM Ports

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.

```
$regfile = "xm128a1def.dat"
$crystal = 32000000 ' 32MHz
$hwstack = 64
$swstack = 40
$framesize = 40

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014
Config Osc = Disabled , 32mhzosc = Enabled ' 32MHz
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 38400 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Com2 = 19200 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Com3 = 9600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Com4 = 38400 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Com5 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Com6 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Com7 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Config Com8 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8

'But when you want to open it then do it this way
Open "COM5:" For Binary As #buart_channel

'Dynamic COM Port Variable must start with BUART
Dim Buart_channel As Byte '0 = COM1 ..... 7 = COM8
'when using a variable, notice that the index is 0 based !

Buart_channel = 4 'Channel 4 --> COM5
Print #buart_channel , 'New Line
Print #buart_channel , "This use also COM5"

For Buart_channel = 0 To 7
Print #buart_channel , "Hello " ; Buart_channel ; " notice that the index is 0 based" 'print
Hello on every COM Port (with configured Baud rate)
Waitms 10
Next

End 'end program
```

## ATXMEGA and RS-485

CONFIG PRINT0 = pin à for COM1

CONFIG PRINT1 = pin à for COM2

CONFIG PRINT2 = pin à for COM3

CONFIG PRINT3 = pin à for COM4

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.

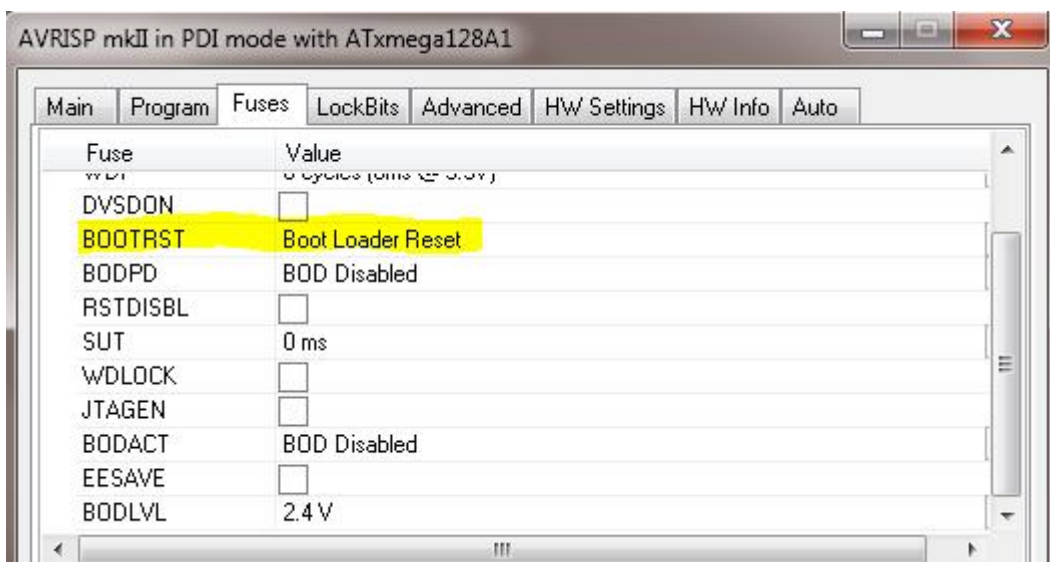
You need to set the direction of the used pin to output mode yourself.

```
'RS-485 on COM2
Config Com2 = 9600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM2:" For Binary As #2

Config Portc.5 = Output
Config Print1 = Portc.5 , Mode = Set 'This will auto-enable/auto-disable the
Transmitter (works for print function)
```

## Using a Bootloader with ATXMEGA

Ensure you have set the Fuse Bit: BOOTRST to Boot Loader Reset



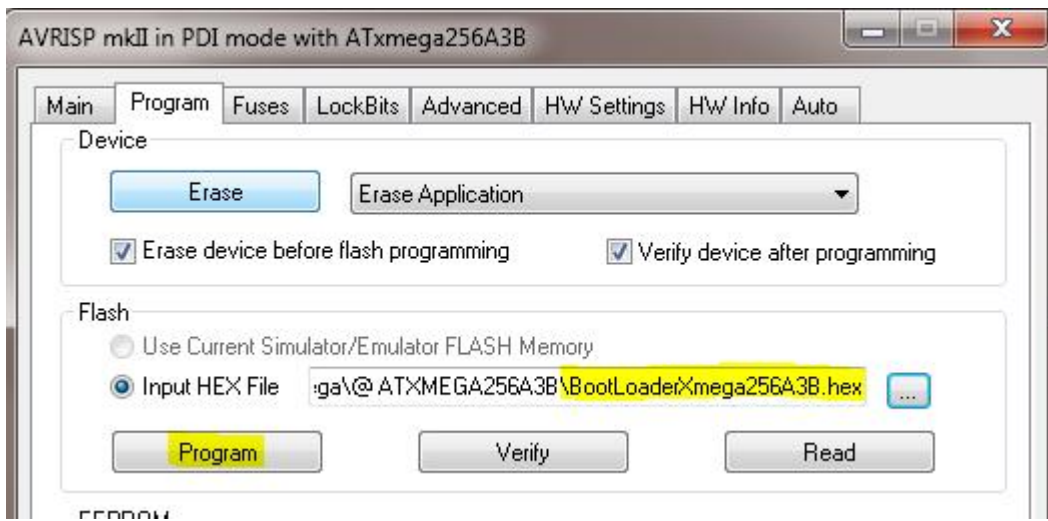
Here we use an Example for A TXMEGA256A3B:

In the datasheet of A TXMEGA256A3B you find under *In-System Programmable Flash Program Memory* Where the Boot Section start  $\&$  \$loader = &H20000

## Word Address

0	Application Section (256 KB)
...	
1EFFF	
1F000	Application Table Section (8 KB)
1FFFF	
20000	Boot Section (8 KB)
20FFF	

Use the following Bootloader and flash it with for example ARVISP MKII



Bootloader for ATXMEGA256A3B:

```
'-----  
'                                     (c) 1995-2009, MCS  
'                                     BootloaderXmega256.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 = "xm256A3Bdef.dat"                           ' add a reference to this lib  
$lib "xmega.lib"  
  
'first enabled the osc of your choice  
Config Osc = Enabled , 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 = &H20000                                     ' bootloader starts after the application
```

```

'this sample uses 57600 baud and we use here COM7
Config Com7 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
'Portf.2 and Portf.3 is COM7
Open "COM7:" For Binary As #1

'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 , Spmc rval As Byte                        ' self program command byte value

Dim Z As Long                                           'this is the Z pointer word
Dim V1 As Byte , Vh As Byte                             ' these bytes are used for the data
values
Dim Wrđ 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(#1)                                  '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 #1 , 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(#1)
    Loop Until Bstatus = 0

    Spmcrcval = &H20 : Gosub Do_spm          ' erase all app pages

Bretries = 10                             'number of retries

Do
  Csum = 0                                 'checksum is 0 when we start
  Print #1 , Chr(nak);                    ' first time send a nack
  Do
    Bstatus = Waitkey(#1)                 '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(#1) : Csum = Csum + Bblock    'get block
        Bcsum1 = Waitkey(#1) : Csum = Csum + Bcsum1    'get checksum first byte
        For J = 1 To 128
          Buf(j) = Waitkey(#1) : Csum = Csum + Buf(j)
        Next
        Bcsum2 = Waitkey(#1)              '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 #1 , Chr(ack);         'acknowledge
          Else                             'no match so send nak
            Print #1 , Chr(nak);
          End If
        Else
          Print #1 , Chr(nak);           'blocks do not match
        End If
      Case 4:                             ' end of transmission , file is
transmitted
        If Wrđ > 0 Then                  'if there was something left in the
page
          Wrđ = 0
          Spmcrcval = &H24 : Gosub Do_spm    'write page
        End If
        Print #1 , 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
    V1 = Buf(j) : Vh = Buf(j + 1)                       'get Low and High bytes
    lds r0, {v1}                                         'store them into r0 and r1 registers
    lds r1, {vh}
    Spmcrrval = &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
      Spmcrrval = &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 = Spmcrrval
  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

'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

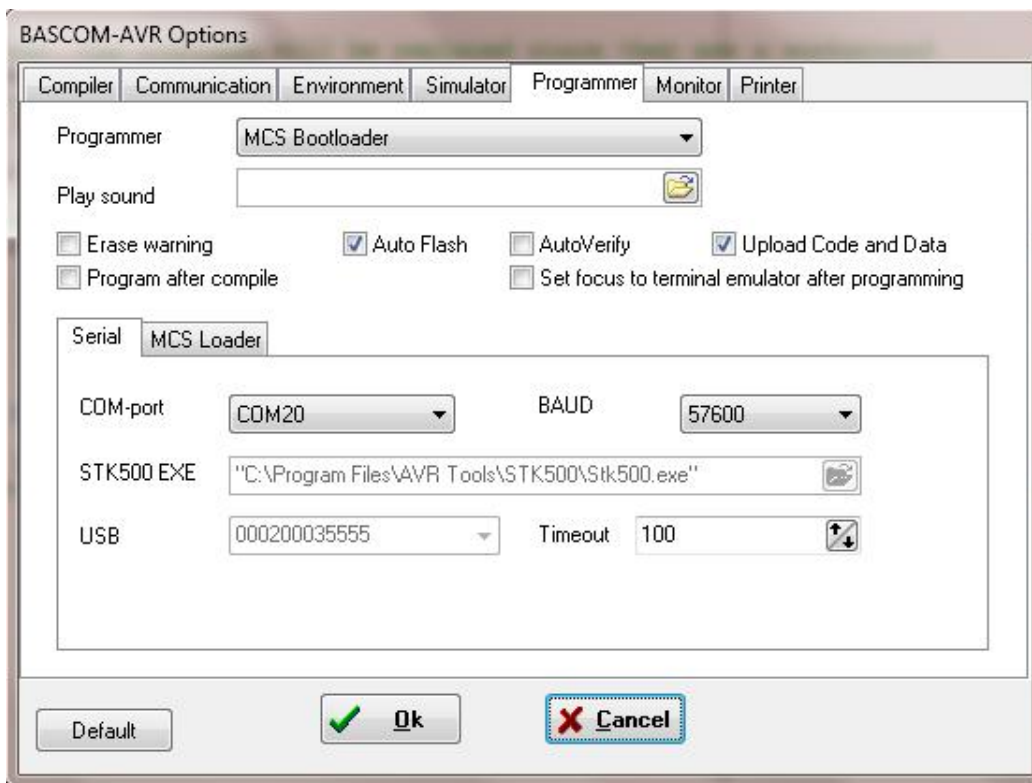
'The GOTO will do the work, you need to specify the correct bootloader address
'this is the same as the $LOADER statement.

```

After this you can use the Bascom-A VR Bootloader:

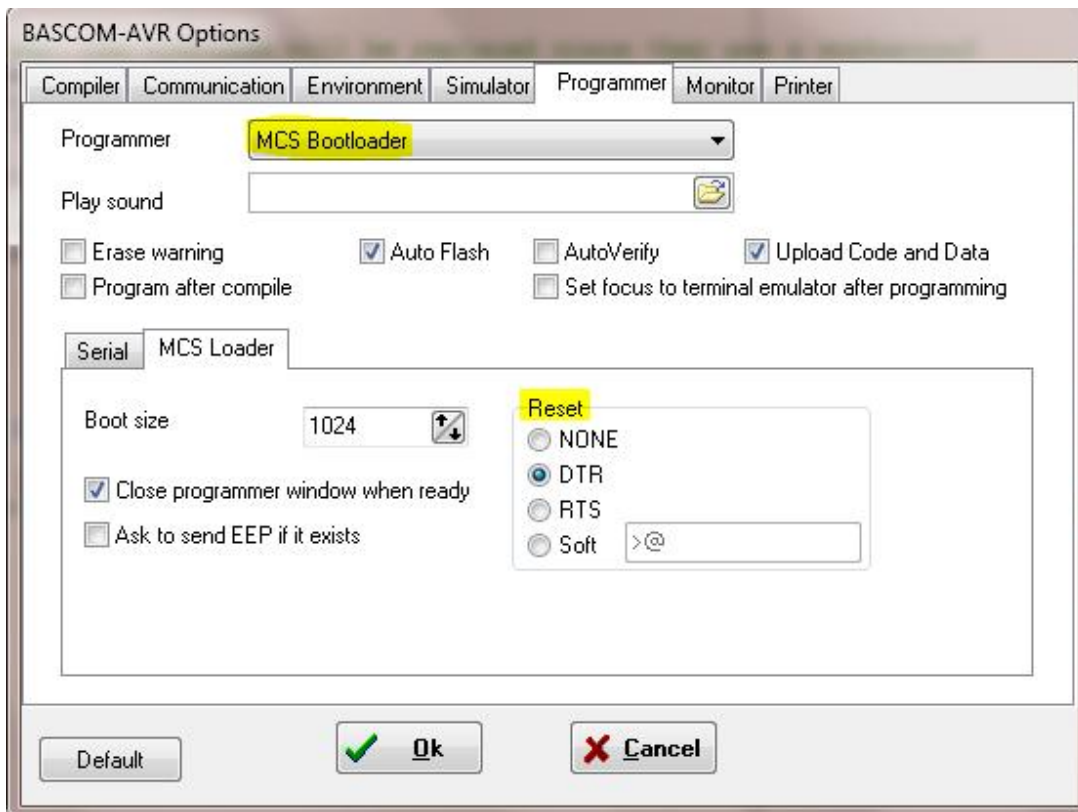
In Bascom-A VR >>> Options >>> Programmer

Select the MCS Bootloader, the COM Port and Baud rate of Bootloader (in this case 57600)



Then goto MCS Loader Tab and choose the Bootsize and Reset





### 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 choose to use the RTS line.

### Soft-Reset:

When your program does not use the boot vector or need 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.

The Soft-Reset needs to be implemented in your regular program so when Bascom-AVR send for example ">@" the ATXMEGA will reset and jump according the BOOTRST Fuse Bit to the beginning of Boot-Section.

This here is an example for an UART Receive Interrupt Service Routine which check for the ">@" to be sent from MCS Bootloader. When the defined 2 character received in a row the ATXMEGA will be reseted by Software Reset.

```
' Receive ISR of COM6
Rxc_e1_isr:
  Com6_in = Inkey(#6)           'Receive data from MCS Bootloader

  'Software Reset when ">@" is received from MCS Bootloader
  If Bootloader_bit = 1 Then    'if ">" was already transmitted
    Reset Bootloader_bit
  If Com6_in = 64 Then          '"@"
    Cpu_ccp = &HD8             'enable change of protected Registers for
following 4 CPU Instruction Cycles
```

```

        Rst_ctrl1.0 = 1                                'Initiate Software Reset by setting Bit0 of
RST_CTRL Register
    End If
End If

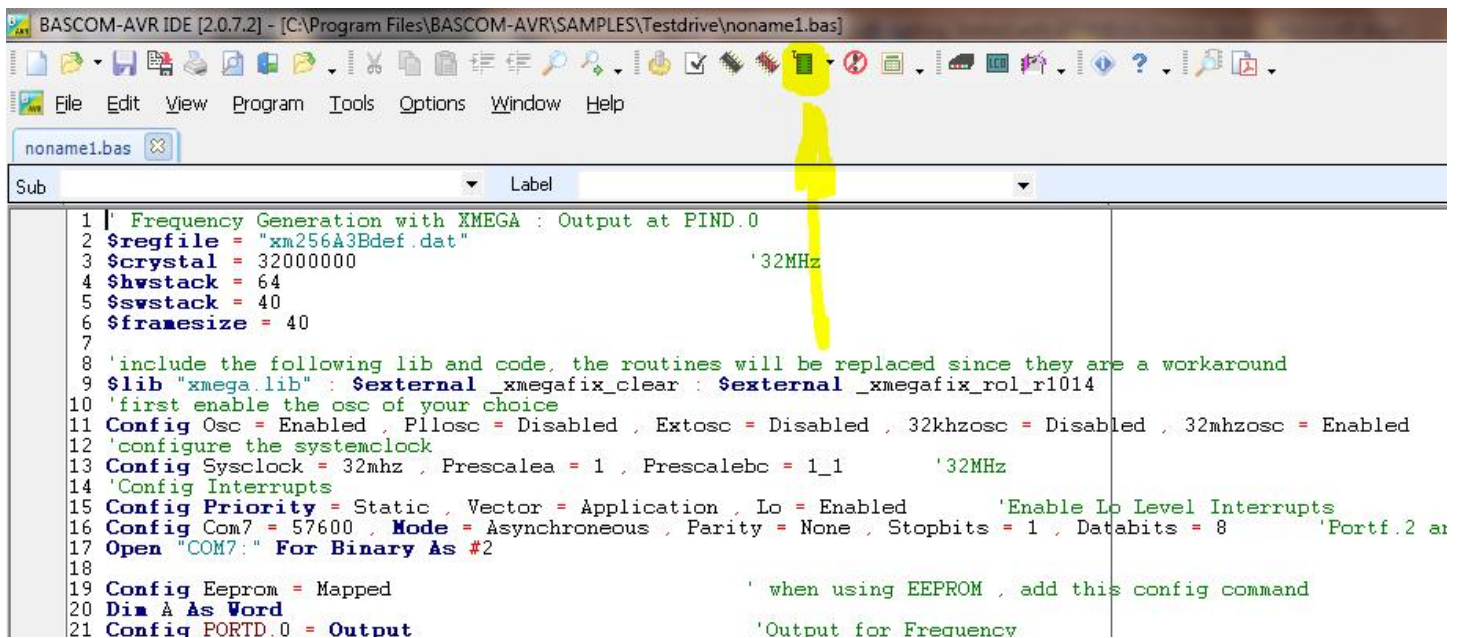
If Com6_in = 62 Then Set Bootloader_bit              '>'

'Debug output
Print #5 , Chr(com6_in) ;                            'print the received data over serial interface to
PC (just for testing)

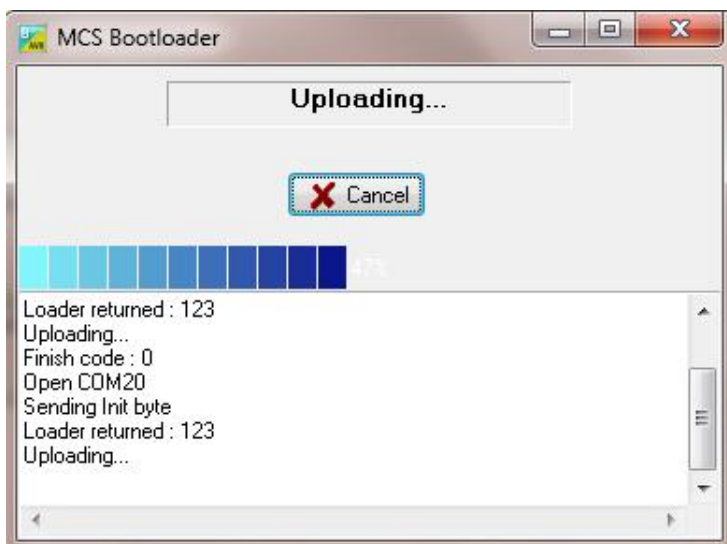
Return

```

Now you can program your A TXMEGA direct from Bascom-A VR by click on Program Chip or F4



Upload Dialog:



## Using EEPROM

After

```
Config Eeprom = Mapped
```

you can use following read and write functions for EEPROM.

Write to EEPROM:

```
My_byte = 11
Writeeeprom My_byte , 0
My_byte = 22
Writeeeprom My_byte , 1
```

Read from EEPROM

```
Readeeprom Read_byte , 0           'Address 0
Readeeprom Read_byte , 1           'Address 1
```

Or you can use ERAM BYTE:

```
$regfile = "xml28aldef.dat"           'ATXMEGA32A4 is used for this example
$crystal = 32000000
$hstack = 100
$wstack = 100
$framesize = 260

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014
Config Osc = Enabled , 32mhzosc = Enabled , 32khzosc = Enabled
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

'Configure UART for communication with PC (USB)
Config Com5 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #1

Config Eeprom = Mapped               ' <<<<<-----

Dim B As Byte
Dim Testbyte As Eram Byte           'EEPROM Testbyte

Print #1 , "----Start----"

Testbyte = 10                       'write to EEPROM
Wait 1
B = Testbyte                         'read from EEPROM
Print #1 , "B = " ; B                'print it

End                                  'end program
```

## How to initiate a software Reset of ATXMEGA:

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
```

## Using the Watchdog of ATXMEGA:

You can use the watchdog between 8ms and 8seconds. The XMEGA watchdog usually need app. 1 $\mu$ A to work. The drawback of this low current watchdog is the accuracy so it is important to reset the watchdog early. There is also a watchdog window mode where you need to reset the watchdog between the specified window times.

```
$regfile = "xml28a1def.dat"
$crystal = 32000000
$hstack = 64
$wstack = 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
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 19200 , Mode = Asynchronous , 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
revisions 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
```

## TC - 16-bit Timer/Counter

The following example show how to use the Timer for Periodic interrupt generation.  
We configure the Timer C0 as NORMAL ( no wave generation) and with Prescaler 1024.

When using 32MHz internal osc the timer runs with  $32\text{MHz}/1024 = 31250$  Hz so every count takes  $32\mu\text{s}$ .  
For the full 16-Bit it would count to  $32\mu\text{s} * 65536 = 2.097152$  Second

For 1 second we choose  $Tcc0\_per = 31250$  so we have  $32\mu\text{s} * 31250 = 1$  Second

```
$regfile = "xm128a1def.dat"
$crystal = 32000000          ' 32MHz
$hstack = 64
$swstack = 40
$framesize = 40

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014

Config Osc = Disabled , 32mhzosc = Enabled          'Init oscillator
Config Sysclock = 32mhz                             '--> 32MHz

Config Priority = Static , Vector = Application , Lo = Enabled          'Init Interrupts

'Serial Interface to PC
Config Com5 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #1

Print #1 , "Second Tick with Timer C 0"

'Timer C0
Config Tcc0 = Normal , Prescale = 1024
Tcc0_per = 31250          ' 32MHz/1024 = 31250 --> One Second Tick

On Tcc0_ovf Tc0_isr          'Setup overflow interrupt of Timer/Counter C0 and
name ISR
Enable Tcc0_ovf , Lo          'Enable overflow interrupt in LOW Priority
Enable Interrupts

'-----[Main Loop]-----
Do
  !nop
Loop
'-----[Main Loop]-----

End          'end program

'-----[Interrupt Service Routines]-----
Tc0_isr:

  Print #1 , "tick"          'Only for testing

Return
```

## Now we use the RTC - Real Time Counter for generating 1 Second Tick:

The clock source option for the RTC are:

- Accurate Internal 32 kHz RC oscillator
- Ultra Low Power Internal 32 kHz RC oscillator
- External 32 kHz crystal oscillator .

And the according Bascom-AVR parameters:

- 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

Here we use the accurate internal 32KHz RC oscillator à 1KHZ\_INT32KHZ\_RCOSC:

```
$regfile = "xm128aldef.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 64
$framesize = 64

$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 = Disabled , 32mhzosc = Enabled , 32khzosc = Enabled
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com5 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #1

Config Clock = Soft , Rtc = 1khz_int32khz_rcosc ' we select the internal 1 KHz clock from the
32KHz internal oscillator

Config Priority = Static , Vector = Application , Lo = Enabled ' the RTC uses LO priority
interrupts so these must be enabled !!!
Enable Interrupts

Print #1 , "Second Tick TEST"

Do

  If Rtc_intflags.1 = 1 Then
    Set Rtc_intflags.1 'Clear the Int Flag

    Print #1 , Time$ ' print the time
  End If

Loop

End 'end program
```

## Using config tcxx to easy configure a PWM with XMEGA

```
' Using PWM with XMEGA : Output at PIND.0
```

```

$regfile = "xm256A3Bdef.dat"
$crystal = 32000000 '32MHz
$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 , Pllosc = Disabled , Extosc = Disabled , 32khzosc = Disabled , 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

'COM2 = Port C Pin 6 and Pin 7
'this sample uses 57600 baud.
Config Com7 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
'Portf.2 and Portf.3 is COM7
Open "COM7:" For Binary As #2

Config Eeprom = Mapped ' when using EEPROM , add this config command

Dim A As Word

Config Portd.0 = Output 'Output for PWM

Config Tcd0 = Pwm , Prescale = 8 , Comparea = Enabled , Resolution = 16
' Tcd0 --> pwm --> pulse width modulation single slope
' Prescale = 8 --> 32MHz/8 = 4MHz
' Comparea = enabled --> Enable COMPARE or CAPTURE A
' Resolution = 16 --> 16-Bit Resolution

' <--TCD0_CCA-->
' +-----+ +
' | | |
'-----+ +-----+
'
' <----->
' Period = 16,38mSec

'SET Resolution of PWM (min. = &H0003 ..... max. = &HFFFF)
Tcd0_per = &HFFFF 'Set Period = FFFF = 65535 --> 65535/4MHz =
16.38mSec

Tcd0_cca = 10000 '10000/4MHz = 2.5mSec

Print #2 , "----Example PWM with XMEGA----"

Do

Waitms 500

'Change TCD0_CCA (Duty Cycle)
Tcd0_cca = 20000 '20000/4MHz = 5ms

Waitms 500

```



```

'Change TCD0_CCA (Duty Cycle)
Tcd0_cca = 10000                                '2.5ms

Wait 2

'Duty Cycle from 0 to 30000 = from 0 to 7.5ms
Do
  Incr A
  Tcd0_cca = A
  Waitus 100
Loop Until A = 30000

A = 0

Wait 2

Loop

End                                                'end program

```

## Easy Frequency Generation with XMEGA

```

' Frequency Generation with XMEGA : Output at PIND.0
$regfile = "xm256A3Bdef.dat"
$crystal = 32000000                                '32MHz
$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 , Pllosc = Disabled , Extosc = Disabled , 32khzosc = Disabled , 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 Com7 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
'Portf.2 and Portf.3 is COM7
Open "COM7:" For Binary As #2

Config Eeprom = Mapped                                ' when using EEPROM , add this config command
Dim A As Word
Config Portd.0 = Output                                'Output for Frequency

Print #2 , "----Example Digital Frequency Generation with XMEGA----"

Config Tcd0 = Freq , Prescale = 2 , Comparea = Enabled , Resolution = 16
'TCD_PER is not used in Frequency (Waveformgeneration) mode, only the TCX_CCX Registers to set the
Period of the Digital Waveform
'Digital Waveform Generation

'
'          +-----+          +
'          |          |          |
'-----+          +-----+
'
'          <----->
'          Period = TCD_CCA Register

' Prescale = 1 --> 32MHz

```

```

' Frequency max with TCD_CCA = 0 --> System Clock/2*Prescaler(CCA + 1) = 32MHz/2*2(0 + 1) =
32MHz/4 = 8MHz
' Frequency min with TCD_CCA = &HFFFF --> System Clock/2*Prescaler(CCA + 1) = 32MHz/2*2(65535 + 1)
= 32MHz/4*65536 = 122 Hz

Tcd0_cca = 0 'F = 8 MHz
Wait 3
Tcd0_cca = &HFFFF 'F = 122 Hz

End 'end program

```

## XMEGA I2C bzw. TWI in MASTER MODE

We use here the I2C/TWI interface of Port C (`twic`).

In addition the ATXMEGA128A1 also feature I2C Interface on Port D, E and F.

You can run the different I2C interfaces also with different clock rates like one with 100KHz and the other one with 400KHz.

Configure the Interface:

```

Dim Twi_start As Byte 'This Variable is used by the I2C functions
Open "twic" For Binary As #2

'Portc.0 'SDA Pin of Port C (ATXMEGA128A1)
'Portc.1 'SCL Pin of Port C (ATXMEGA128A1)
I2cinit #2 'set i2c pins to right state , open collector , pull up activated
Config Twic = 100000 'Set TWI Baud Rate and Enable TWI Master

```

After this you need to add the interface number in this case #2:

```

I2cstart #2
I2cwbyte Schreibadresse , #2
I2cwbyte Speicheradresse , #2
I2cwbyte Byte , #2
I2cstop #2

```

Or you can do it this way:

```

Array_to_write(1) = Address_byte
I2creceive Array_to_write(1) , 1 , 3 , #2 'Send one byte (address) and read 3 byte

```

Or:

```

I2csend Address_byte , Array_to_send(1) , 4 , #2 'send 4 byte

```

## Using a LCD with XMEGA:

**Example: 3.3V EA DOGM162 in 4-Bit Mode (2x16 LCD)**

<http://www.lcd-module.com/products/dog.html>

## Configure the LCD

```
'Hardware Connections according Datasheet 4-Bit Mode 3.3 Volt and R/W --> GND
Config Lcdpin = Pin , Db4 = Porte.5 , Db5 = Porte.4 , Db6 = Porte.3 , Db7 = Porte.2 , E = Porte.6 ,
Rs = Porte.7
Config Lcd = 16 * 2 , Chipset = Dogm162v3
```

Then you can write to the LCD:

```
Cursor Off Noblink
Cls
Locate 1 , 3 : Lcd "ATXMEGA128A1"
Locate 2 , 1 : Lcd "EA-DOGM 162 3.3V"
```

Another example for using LCD with XMEGA is with a DOGM128x64:

```
'-----DOGM128X64 Display init-----
Config Vport0 = D 'Assign Port D to Virtual Port 0 to get smaller
and faster code

$lib "glcdeadogm128x6.lbx" ' specify the used lib
Config Graphlcd = 128 * 64eadogm , Cs1 = Port0.5 , A0 = Port0.3 , Si = Port0.1 , Sclk = Port0.2 ,
Rst = Port0.4

'Connections:
'CS1 = PortD.5
'A0 = PortD.3
'SI = PortD.1
'sclk = PortD.2
'Rst = PortD.4
```

Then you can use LCDAT function to write to the Display:

```
'specify the font we want to use
SetFont Font8x8tt 'We start with a smaller font
Lcdat 1 , 1 , Time$
Lcdat 1 , 64 , Date$
```

Or show a picture with:

```
Showpic 90 , 32 , Home_small 'Now we show a picture
```

The picture and font needs to be included after the END:

```
...
End 'end program

'include the picture data
Home_small:
$bgf "home_small.bgf"

'include used fonts
$include "font8x8TT.font"
```

## ATXMEGA und 128x64 Graphic Display: 64128N SERIES from DisplayTech with ST7565R

```
'-----
'
'          (c) 1995-2011, MCS
'          xml28A1-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 = "xml28a1def.dat"
$crystal = 32000000
$hwhstack = 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 = Asynchronous , 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 something on the LCD, use the LDAT command
'LCDAT Y , COL, value
Lcdat 1 , 1 , "11111111"
Lcdat 2 , 1 , "ABCDEFGHJKLM1234"
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"

```

## Reduce power consumption by setting Power Reduction Register:

```

'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,spic,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

```

## Using the Event System (first example):

The Event System in this example easy show after the 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 ! So it is working direct in hardware.

This example also shows how to manual fire an Event

Notice: when you want to measure the Event on PortC.7 you need to know that an event is only one Clock Cycle so you need an oscilloscope and a trigger to see it !

```

' Using the new CONFIG_EVENT_SYSTEM function

' Bascom Version 2.0.4.0 or higher
'

' PINC.0 (INPUT FOR EVENT CHANNEL 0) ----->>> PINC.7 (OUTPUT FOR EVENT CHANNEL 0)

$regfile = "xm32a4def.dat"
$crystal = 32000000          '32MHz
$hwstack = 64
$swstack = 40
$framesize = 100

$lib "xmega.lib" :$external _xmegafix_clear : $external _xmegafix_rol_r1014

Config Osc = Disabled , 32mhzosc = Enabled          '32MHz
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1          'CPU Clock = 32MHz

Config Com1 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8

Print
Print "-----S T A R T-----"

'Configure PC0 for input, Event triggered on falling edge
Config Portc.0 = Input

```

```

Portc_pin0ctrl = &B00_011_010
                '^  ^
                '^  React on falling edge (010)
                '^
                'enable Pullup

' 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

Config Portc.7 = Output
'Event Channel 0 Ouput Configuration
Portcfg_clkevout = &B0_0_01_0_0_00 'Output on PortC.7 /Clock Out must be disabled

Print "Mainloop -->"

Do

'IMPORTANT: YOU WILL SEE THE PIN CHANGES ALSO DURING WAIT 4 BECAUSE IT USE THE EVENT SYSTEM
Wait 4

'This shows how to manual fire an Event
Set Evsys_strobe.0

Loop

End 'end program

```

## ATXMEGA as SPI Master

This sample demonstrates the Xmega128A1 SPI master mode

There are 4 SPI interfaces on the XMEGA. You need to specify SPIC, SPID, SPIE or SPIF for SPIx. The value must be HARD.

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.

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.

See datasheet for *Alternate Pin Functions*. For example for SPI on Port E the SS Pin is PINE.4

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.

In Master Mode you can set the SPI Clockrate up to 16MHz with Clockdiv = Clk2 and when using the 32MHz Clock.

```
Config Spie = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk2 , Data_order = Msb , Ss = Auto
```

```
' XMEGA SPI MASTER
' xmega_spi_master.bas

' This example should be used with xmega_spi_slave.bas or with xmega_spi_slave_interrupts.bas

$regfile = "xm128aldef.dat"
$crystal = 32000000 '32MHz
$hwstack = 50
$swstack = 50
$framesize = 200

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014

'-----CLOCK CONFIGURATION-----
Config Osc = Disabled , 32mhzosc = Enabled ' , 32khzosc = Enabled '32MHz and 32KHz
enabled
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 'CPU Clock = 32MHz

'Init and enable the DFLL (Digital Frequency Locked Loop) for automatic run-time calibration of the
internal 32MHz Oscillator
Osc_dflctrl = &B00000000 'The internal 32.768 KHz Oscillator is used for
calibration
Set Dfllrc32m_ctrl.0 'Enable DFLL and auto calibration
'-----

Config Priority = Static , Vector = Application , Lo = Enabled 'Interrupt Setup

'-----CONFIGURE TIME, DATE AND RTC FOR SECOND TICK-----
Config Date = Dmy , Separator = .
Config Clock = Soft , Rtc = 1khz_int32khz_rcosc 'Internal 32KHz

'COM Interface to PC (COM5 is PINE.6 and PINE.7)
Config Com5 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #1

config porte.4 = output 'Slave Select Pin
set porte.4

Dim Select_bit As Bit
Reset Select_bit 'We start with 0

'CONFIG SPI MASTER on Port E
Config Spie = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk32 , Data_order = Msb , Ss = None
```



```

Open "SPIE" For Binary As #10

Dim Array(10) As Byte

'Array(1) = incremented
Array(2) = 2
Array(3) = 3
Array(4) = 4
Array(5) = 5
Array(6) = 6
Array(7) = 7
Array(8) = 8
Array(9) = 9
Array(10) = 10

Const Start_of_frame = 111

Do

    'Second Tick from RTC
    If Rtc_intflags.1 = 1 Then
        Set Rtc_intflags.1 'Clear the Flag

        Incr Array(1)

        'SEND 10 BYTE ARRAY TO SLAVE
        Reset Porte.4 'Select Slave
        Print #10 , Start_of_frame ; Array(1) , 10 'SEND Start Byte then ARRAY WITH 10 BYTES
        Set Porte.4 'Deselect Slave

        Print #1 , Array(1) 'Print B to Serial PC connection

    End If

Loop

End 'end program

```

## ATXMEGA as SPI SLAVE

The max. SPI Clock Rate in SLAVE MODE is  $32\text{MHz}/\text{CLK4} = 8\text{MHz}$

```

'SPI SLAVE USING INTERRUPTS

' xmega_spi_slave_interrupt.bas should be used with xmega_spi_master.bas

'This here is the Output of this example (when used with xmega_spi_master.bas):

'(
-----XMEGA AS SPI SLAVE using Interrupts-----
Received Array = 111,212,2,3,4,5,6,7,8,9,10
Received Array = 111,213,2,3,4,5,6,7,8,9,10
Received Array = 111,214,2,3,4,5,6,7,8,9,10
Received Array = 111,215,2,3,4,5,6,7,8,9,10
Received Array = 111,216,2,3,4,5,6,7,8,9,10
Received Array = 111,217,2,3,4,5,6,7,8,9,10
Received Array = 111,218,2,3,4,5,6,7,8,9,10

```

```

...
...

')

$regfile = "xm32a4def.dat"
$crystal = 32000000 '32MHz
$hwstack = 100
$swstack = 100
$framesize = 100

$lib "xmega.lib" : $external _xmegafix_clear : $external _xmegafix_rol_r1014

Config Osc = Enabled , 32mhzosc = Enabled
Config Sysclock = 32mhz '---> 32MHz
Config Priority = Static , Vector = Application , Lo = Enabled

'Serial Interface to PC
Config Com1 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8

' CONFIG SPI SLAVE on Port C
' Clock comes from SPI Master (Maximum clock rate a XMEGA Slave can handle is when MASTER Clock
rate is CLK4)
Config Spic = Hard , Master = No , Mode = 0 , Data_order = Msb 'XMEGA SPI SLAVE
Open "SPIC" For Binary As #10

Print "-----XMEGA AS SPI SLAVE using Interrupts-----"

On Spic_int Spi_data_available
Enable Spic_int , Lo
Enable Interrupts

Dim Spi_data(11) As Byte

Dim B As Byte , I As Byte
Dim Spi_data_ready As Bit

Do

    If Spi_data_ready = 1 Then
        Reset Spi_data_ready

        Print "Received Array = " ;
        For B = 1 To 10
            Print Spi_data(b) ; "," ;
        Next
        Print Spi_data(11)
    End If

Loop

End 'end program

'SPI Port C Interrupt Service Routine
Spi_data_available:
    Incr I
    Spi_data(i) = Spic_data 'Read the SPI Data Register

```

```

If I = 11 Then
  I = 0
  Set Spi_data_ready
End If
Return

```

## Using Dynamic Channels with XMEGA SPI

To use a Variable as Dynamic Channel you must DIM a byte which start with BSPI.

Bspivar = 0 ----> SPI Port C

Bspivar = 1 ----> SPI Port D

Bspivar = 2 ----> SPI Port E

Bspivar = 3 ----> SPI Port F

In case of Bspivar = 1 it is SPI of Port D.

```

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 support PRINT and INPUT

Print #10 , "to spi" ; W
Input #10 , Ar(1) , W
Print #bspivar , W
Input #bspivar , W

```

## Power Mode Options

Sleep modes enables the microcontroller to shut down unused modules to save power. When the device enters sleep mode, program execution is stopped and interrupts or reset is used to wake the device again.

Only Asynchronous Port Interrupts and TWI(I2C) Address match interrupts can wake up an ATXMEGA from PowerDown mode.

Examples:

```
Power powerdown
```

or

```
power PowerSave
```

PowerDown mode with All Functions Disabled at 3.0Volt Vcc need just 0.1µA current consumption.

## Virtual Port Registers

“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 A TXMEGA A Manual]

```
$regfile = "xm128a1def.dat"
$crystal = 32000000
$hstack = 64
$swstack = 40
$framesize = 40

$lib "xmega.lib" :$external _xmegafix_clear :$external _xmegafix_rol_r1014
Config Osc = Enabled , 32mhzosc = Enabled
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Config Com1 = 19200 , Mode = Asynchronous , 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

'Continuously 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
```

## AES Crypto Module

“The AES Crypto Module encrypts and decrypts 128-bit data blocks with the use of a 128-bit key. The key and data must be loaded into the module before encryption/decryption is started. It takes 375 peripheral clock cycles before encrypted/decrypted data can be read out.” [XMEGA A Manual]

This function only works for XMEGA chips that have an AES encryption unit 128 bit encryption is used.

Keydata must hold the 16 Byte key.

Ar(1) is the Array of data which you want to encrypt. This array also hold the encrypted data after encryption. The 32 is the number of bytes which you want to encrypt. It is important that this is a multiple of 16 so only 16, 32, 48..... is valid.

```
AESENCRYPT Keydata , Ar(1) , 32
```

```
AESDECRYPT key, var , size
```

You can also use it as a function. Then the variable targ will contain the encrypted/decrypted data.

```
targ = AESDECRYPT ( key, var , size)
```

```
targ = AESENCRYPT ( key, var , size)
```

```
'-----  
' (c) 1995-2010, MCS  
' xml28-AES.bas  
' This sample demonstrates the Xmega128A1 AES encryption/decryption  
'-----  
  
$regfile = "xml28aldef.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 = Asynchronous , 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) ; "-" ; Arcnc(j)
Next

Print "Decrypt function"
Ar(1) = Aesdecrypt(keydata , Arcnc(1) , 32)

For J = 1 To 32
Print J ; ">" ; Ar(j) ; "-" ; Arcnc(j)
Next

End

Keydata:
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 , 16

```

## DMA (Direct Memory Access)

“The Atmel XMEGA Direct Memory Access Controller (DMAC) is a highly flexible four channel DMA Controller capable of transferring data between memories and peripherals with minimal CPU intervention. While the CPU spends time in low-power sleep modes or performs other tasks, the XMEGA DMA offloads the CPU by taking care of mere data copying from one area to another.” [XMEGA AppNote]

The DMA has four independent channels which can be set over separate control and status registers.

This can be done in one configuration with Config DmachX:

```

'you can configure 4 DMA channels
Config Dmach0 = Enabled , Burstlen = 1 , Chanrpt = Enabled , Tci = Lo , Eil = Lo , Singleshot =
Enabled , _
  Sar = Burst , Sam = Fixed , Dar = Transaction , Dam = Inc , Trigger = &HAB , Btc = Array_size ,
Repeat = 0 , Sadr = Varptr(usartf0_data) , Dadr = Varptr(receive_array(1))

```

Burstlen = 1 à 1 byte burst mode. The DMA transfer start after each byte.

Chanrpt = Enabled à In repeat mode, this bit is cleared by hardware in the beginning of the last block transfer.

Tci = Lo , Eil = Lo à Low Level Interrupts for Channel Transaction Complete and for DMA Channel Error  
The Transaction is in this example is completed when all 8 Bytes are transferred in 1-Byte single shot transfers.

Singleshot = enabled à By enabling Single Shot each trigger will complete one data transfer instead of a block transfer. The channel will do a burst transfer of Burstlen bytes on the transfer trigger (in this example each trigger will generate a transfer of 1 Byte from USART E0 to SRAM (Receive\_Array)).

Sar = Burst à Source Address Reload after each Burst. In this example here after each Byte.

Sam = Fixed → Source Address Mode: The source address remains the same (it is the USART F0 Data)

Dar = transaction → Channel Destination Address Reload: DMA destiny address register is reloaded with initial value at the end of each transaction (it start again with receive\_array(1))

“For applications where data is to be stored in an array, the DMAC can be configured to increase or decrease the destination address after each byte access. Similarly, when data is to be retrieved from an array, the DMAC can be configured to increase or decrease the source address after each byte access.” [ATMEL AVR1304]

Dam = Inc → Destination Address Mode: INC means the address is incremented by one. In this example we want to fill the Receive\_Array byte by byte.

Trigger = &HAB → In the following example the DMA should be started by a Receive Complete trigger of USART F0. To find the value search for *DMA Trigger Sources* in the ATXMEGA A Manual.

There you find the TRIGSRC base V alue in case of USART F0 DMA triggers base value = &HAB

Then you need to add the DMA Trigger sources offset V alue. For the Receive Complete (RXC) it is &H00 so the trigger is **&HAB**.

Btc = Array\_size → Block Transfer Count: The BTC represents the 16-bit value TRFCNT.

“The concept of a Block Transfer in this context refers to the operation of performing all data transfers necessary to transfer the number of bytes given by the block size.” [ATMEL AVR1304]

This 16-Bit Register allows up to 64Kbyte Block Transfer Count. You can extend this 64Kbyte with the Repeat Counter.

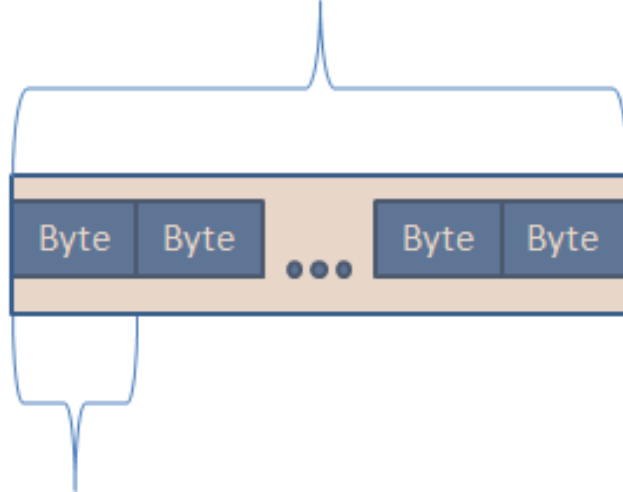
Repeat = 0 → REPCNT counts how many times a block transfer is performed. For each block transfer this register will be decremented.

Sadr = Varptr(usartf0\_data) → Source Address is the address of the DMA source. With VARPTR(X) you get the address of a variable in this case the address of the usartf0\_data register.

Dadr = Varptr(receive\_array(1)) → Destination address. The destination address in this case it the address of Reiceive\_array(1).

**Block Transfer Count (TRFCNT)**

**BTC= Array\_size = 8**



**Burst = 1-Byte (can be 1,2,4 or 8)**

**Burstlen = 1**

**DMA Transaction** is in this case 8 Byte  
(Repeat = 0)

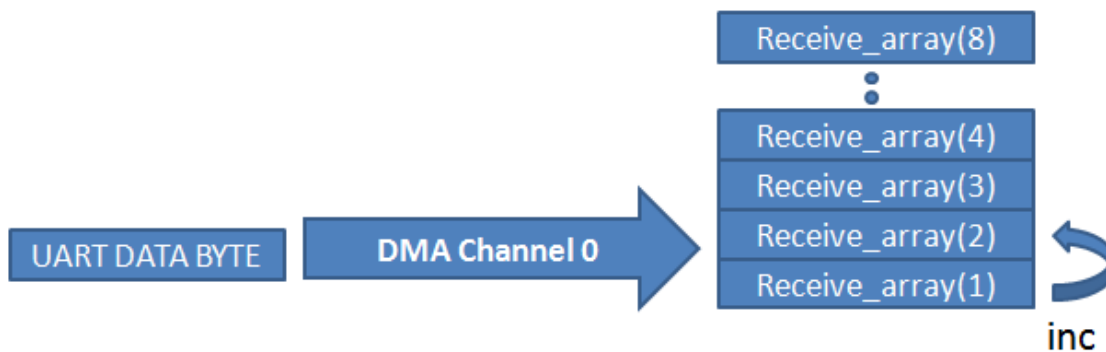
**SOURCE**

USART F0  
Receive Data

**ADDRESSING**

**DESTINATION**

Receive\_Array(1)  
(Byte)



No Source Address increment  
Needed. It is always the same  
address.

Destination Address is incremented





```

Config Osc = Enabled , 32mhzosc = Enabled
Config Sysclock = 32mhz          '--> 32MHz
Config Priority = Static , Vector = Application , Lo = Enabled          'Interrupts

'Serial Interface to PC
Config Com7 = 57600 , Mode = Asynchronous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM7:" For Binary As #1

Print #1 , "UART F0 (COM7) over DMA to SRAM"

const array_size = 8

dim receive_array(array_size) as byte
dim dma_ready as bit
dim dma_Channel_0_error as bit
dim x as byte

' DMA Interrupt
On Dma_ch0 Dma_ch0_int
'Interrupt will be enabled with Tci = XX in Config DMAX

Config Dma = Enabled , Doublebuf = Disabled , Cpm = Ch0rr123          ' enable DMA,

'you can configure 4 DMA channels
Config Dmach0 = Enabled , Burstlen = 1 , Chanrpt = Enabled , Tci = Lo , Eil = Lo , Singleshot =
Enabled , _
Sar = Burst , Sam = Fixed , Dar = Transaction , Dam = Inc , Trigger = &HAB , Btc = Array_size ,
Repeat = 0 , Sadr = Varptr(usartf0_data) , Dadr = Varptr(receive_array(1))

'
'USART F0 DMA triggers base value = &HAB + Receive complete (RXC) &H00 --> &HAB
'Note that unlimited repeat count can be achieved by enabling repeat mode and setting the repeat
count to zero (Chanrpt = Enabled and Repeat = 0)

'Destination Address of Array will be reloaded after each Transaction (Dar = transaction)

Enable Interrupts

'-----[Main Loop]-----
Do

if dma_ready = 1 then
reset dma_ready

print #1 , "-----"
' Do something with the data here.....

' like Print Results back to COM1
for x = 1 to array_size
Print #1 , Chr(receive_array(x)) ; "/" ;
Next
Print #1 ,
end if

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:

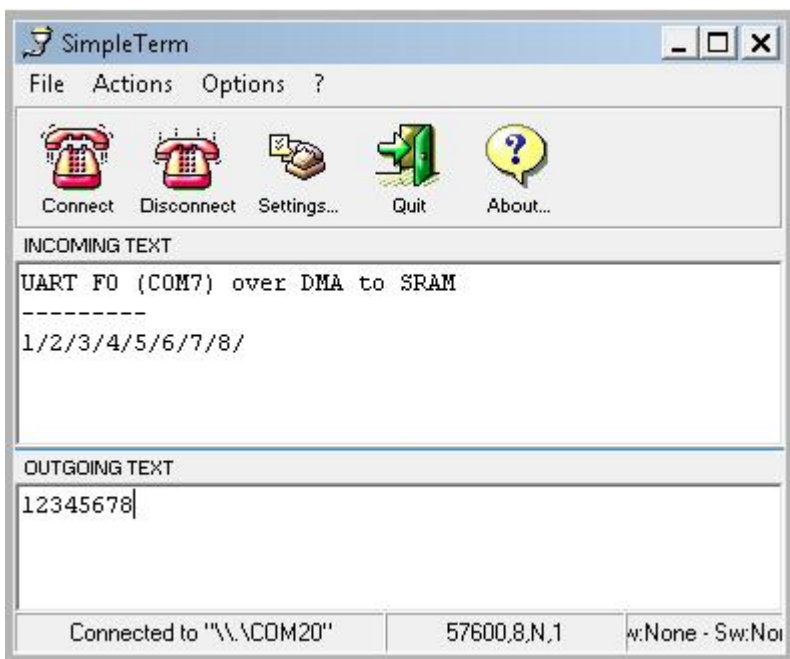
If Dma_intflags.0 = 1 Then      'Channel 0 Transaction Interrupt Flag
    set Dma_intflags.0 'Clear the Channel 0 Transaction Complete flag
    Set Dma_ready
end if

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

```

If you run this example and use a terminal program you can send 8 characters like “12345678” and when you type the “8” the DAMA transaction complete will be set and the characters will be print back separated by “/”



## Reading and Writing to ATXMEGA Registers

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 you find it under MCU – MCU Control

```

;*****
; ** MCU - MCU Control
;*****

MCU_DEVID0 = 144 ; Device ID byte 0
MCU_DEVID1 = 145 ; Device ID byte 1

```

```
MCU_DEVID2 = 146 ; Device ID byte 2
MCU_REVID = 147 ; Revision ID
MCU_JTAGUID = 148 ; JTAG User ID
MCU_MCUCR = 150 ; MCU Control
MCU_EVSYSLOCK = 152 ; Event System Lock
MCU_AWEXLOCK = 153 ; AWEX Lock
```

Then you can read it for example with:

```
Select Case Mcu_revid
  Case 0:
    Print #1 , "Rev = A"
  Case 1:
    Print #1 , "Rev = B"
  Case 2:
    Print #1 , "Rev = C"
  Case 3:
    Print #1 , "Rev = D"
  Case 4:
    Print #1 , "Rev = E"
  Case 5:
    Print #1 , "Rev = F"
  Case 6:
    Print #1 , "Rev = G"
  Case 7:
    Print #1 , "Rev = H"
  Case Else
    Print #1 , "Rev = > H" ; Mcu_revid
End Select
```

Take care with protected registers. Before you can write to this registers you need to release it like with Software Reset:

```
'enable change of protected Registers for following 4 CPU Instruction Cycles

CPU_CCP = &HD8
Rst_ctrl.0 = 1           'When this bit is set a software reset occur
```

---

If you have feedback or a question regarding this App Note go to Bascom-A VR Forum:

[http://www.mcselec.com/index2.php?option=com\\_forum&Itemid=59](http://www.mcselec.com/index2.php?option=com_forum&Itemid=59)

### Great existing Bascom-AVR ATXMEGA Projects:

Following is a very nice ATXMEGA project:

AN #187 - kiXAHRS ONE: Atmel Xmega A1 and Sensors Xplained

[http://www.mcselec.com/index.php?option=com\\_content&task=view&id=295&Itemid=57](http://www.mcselec.com/index.php?option=com_content&task=view&id=295&Itemid=57)

**TFT Display 240x320 65536 Colors with Controller ILI9320 or SPFD5408**

By user SIX1

[http://www.koecher-web.de/media/12-Display\\_320x240.zip](http://www.koecher-web.de/media/12-Display_320x240.zip)

A VI Soft

[http://www.six1.net/bascom/\\_\\_\\_A\\_VI.zip](http://www.six1.net/bascom/___A_VI.zip)

## **A TXMEGA Bootloader with AES Encryption**

By user reinhars

[http://www.mcselec.com/index2.php?option=com\\_forum&Itemid=59&page=viewtopic&p=47902#47902](http://www.mcselec.com/index2.php?option=com_forum&Itemid=59&page=viewtopic&p=47902#47902)

(You will find this also in the Bascom-A VR Samples folder)

List of References:

<http://bascom-forum.de/showthread.php?2198-A-TXMEGA-kleines-einmaleins-Erste-Schritte-mit-A-Txmega>  
[www.amtel.com](http://www.amtel.com)

MAK3

ATMEL Manuals and Application Notes

MWS

[http://www.mcselec.com/index2.php?option=com\\_forum&Itemid=59](http://www.mcselec.com/index2.php?option=com_forum&Itemid=59)