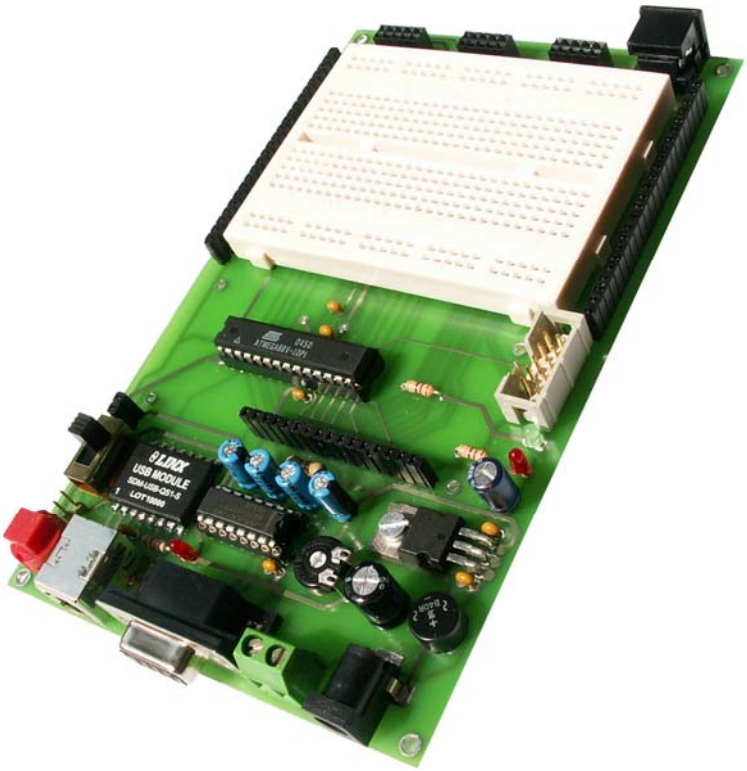
Version

0.3

MCS ELECTRONICS

Making Things Easy



Educational Development Board Features:

- LCD
- PS/2 keyboard/mouse connector
- Breadboard
- USB interface
- RS-232 interface
- ISP connection

- Comes with extensive "getting started" manual

Educational Development Board Manual

UPDATES AVAILABLE VISIT

www.mcselec.com/edb

MCS ELECTRONICS

Educational Development Board Guide

Barry de Graaff

www.mcselec.com

Table of Contents

Table of Contents *(continued)*

Introduction

In the late 80's of the past century microcontrollers where relatively expensive and could only be found in computers and consumer electronics (TV's and stereo sets). Due to the large demand for easy and flexible electronics, microcontrollers became more and more affordable. Nowadays microcontrollers are available for everyone with an interest in electronic development.

The Educational Development Board (EDB) brings you an introduction to the world of electronics and microcontrollers. The board along with this manual will teach you the elements of electronics and microcontrollers. You will learn this elements by carrying out experiments. The experiments follow up in a logical, constructive order which will keep you encouraged to continue to the next experiment.

In the experiments the emphasis is laid on mixing ordinary components such as resistors and LED's with a microcontroller. Where needed a simple explanation of the components and the theory of the microcontroller will be given.

Don't panic if you don't understand the microcontroller, in the first couple of experiments we will consider the microcontroller as a "black box". Further on we will give you some theory on the controller's basics and links to the Internet.

We also like to mention the EDB-CD that is provided with this manual, on the CD you can find the source-code for the experiments, solutions for the exercises as well as Visual Basic examples, datasheets and application notes. You can find an overview of the entire CD in the "Contents of CD.pdf" file in the root of the CD.

We at MCS Electronics wish you much fun while experimenting and welcome all your comments on this and our other products. For additional information or to contact us please visit our website at www.mcselec.com.

Chapter

1

1. Experience

This chapter describes which skills are preferred if you wish to start with the Educational Development Board.

The Educational Development is meant for people who have little or no experience with the use of microcontrollers. But you should have some experience with assembling and soldering "electronic kits" , so you can build your own Educational Development Board. Basic knowledge of computer usage is needed.

1.1 Hardware requirements

To run Bascom AVR you will need a PC running Windows 95, 98, NT, 2000 or XP. In this manual we will use the serial port to program the chip. If you don't have a serial port you can use an STK200/300 dongle to program the chip or the Wiazania USB ISP programmer. You can find how to use the STK200/300 dongle and the Wiazania USB ISP programmer in chapter 4.

1.2 Beware of un-programmed chips!

The ATMega88 microcontroller supplied by MCS Electronics has already been programmed to enable serial programming. If you (accidentally) erase the chip or if you buy a new one, you will need to (re)program it with the STK200/300 dongle *or* the Wiazania USB ISP programmer you can find the procedure to do this in chapter 4.

Chapter

2

2. Getting started

This chapter explains how to get started with your EDB.

Please take the paragraphs in chapter 2 step by step. Doing so will give you a smooth way of starting with the experiments of chapter 3 without any unpleasant surprises.

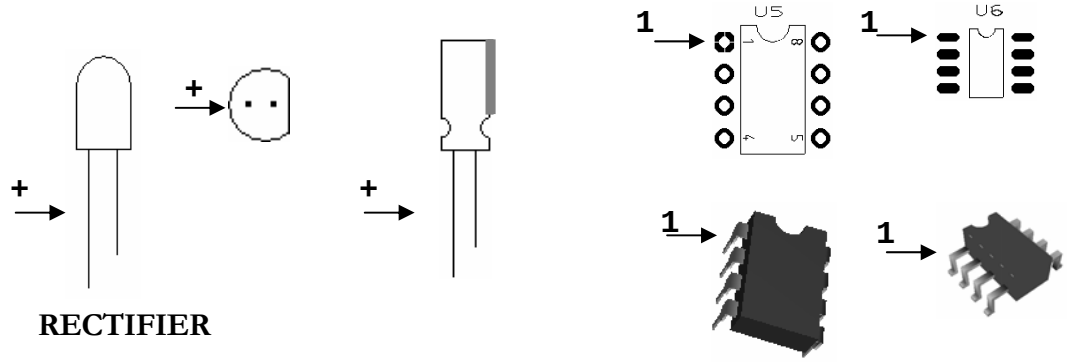
2.1 Assembling your PCB

Chapter 2.1 will take you thru the assembling of your EDB. Soldering the components can be done best in the order of the part list found on page 8.

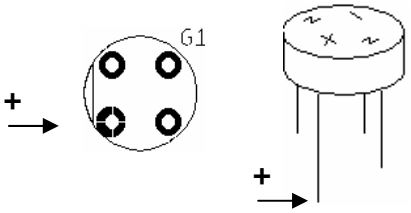
On the PCB you will find pads like this: 

When soldering IC's, the square pad marks pin 1. For LED's and Capacitors the square pad marks the positive (+) pole. Here are some drawings that may help you understand the position of the components:

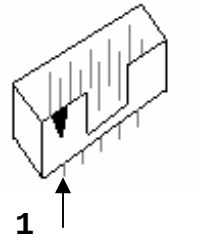
LED **CAPACITOR** **IC**



RECTIFIER



BOXHEADER

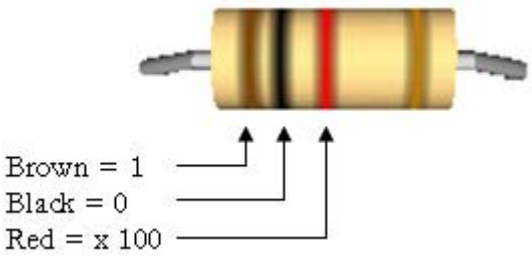


RESISTORS

Resistors don't have poles so it does not matter which pin is connected to which pad. However you need to place the right value resistor at the right position on the PCB. To find the value of a resistor you may want to use an OHM-meter or you can read the color code that is printed on the resistor.

Color codes for resistors:	Value Ring one and two	Value Ring three multiplier
Black	0	x1
Brown	1	x10
Red	2	x100
Orange	3	x1000
Yellow	4	
Green	5	
Blue	6	
Violet	7	
Grey	8	
White	9	

An example how to read the color code:



Thus 1000 OHM = 1kOHM

The fourth ring is the tolerance ring, gold means that the resistor has a 5% tolerance.

PARTLIST

Component	Description	Value
R1,R5	RESISTOR 1/4 W	330E
R4	RESISTOR 1/4 W	220E
C3,C4,C9,C10, C11,C12,C13	CERAMIC CAPACITOR	100N
R2	RESISTOR 2W	47E
U1	IC SOCKET DIP28	for U1
U3	IC SOCKET DIP16	for U3
U4	VOLTAGE REGULATOR	7805
R3	POTENTIOMETER	10K
G1	BRIDGE RECTIFIER	
D1,D2,D3	LED	3mm
X15	SW. UART FEM. HEADER	1x2
X5,X6, X7	FEMALE HEADER	HDR2X5
X3,X4	FEMALE HEADER	HDR1X24
X12	FEMALE HEADER LCD	HDR1X16
X13	USB CONNECTOR	B TYPE FEMALE
X9	ISP HEADER	BOXHDR 10
X1	POWER	POWER CONNECTOR
C5,C6,C7,C8	ELCO RADIAL	1U
C2	ELCO RADIAL	100U
C1	ELCO RADIAL	220U
RESET	RESET SWITCH	SW_SPST
X16	DB9 CONNECTOR FEMALE	DB9FL
X2	POWER	SCREW HEADER
X10	PS2 KEYBOARD	MINIDIN6
S1	USB/UART SWITCH	SWITCH
-	BREAD BOARD	7x7 CM

Do not solder these yet:
1x ATMega88 (U1) + 1x USB MODULE SDM-USB-QS1-S (U2)

ADVISEMENT

We advise you to use IC sockets rather than soldering the IC's directly to the PCB. You may want to use low quality IC sockets for the ATMega88, since it is easier to (re-) insert IC's into low quality sockets then into high quality (turned) sockets.

The USB module can only be soldered directly, **do not solder it yet.**
Also do not insert the IC's and display at this time.

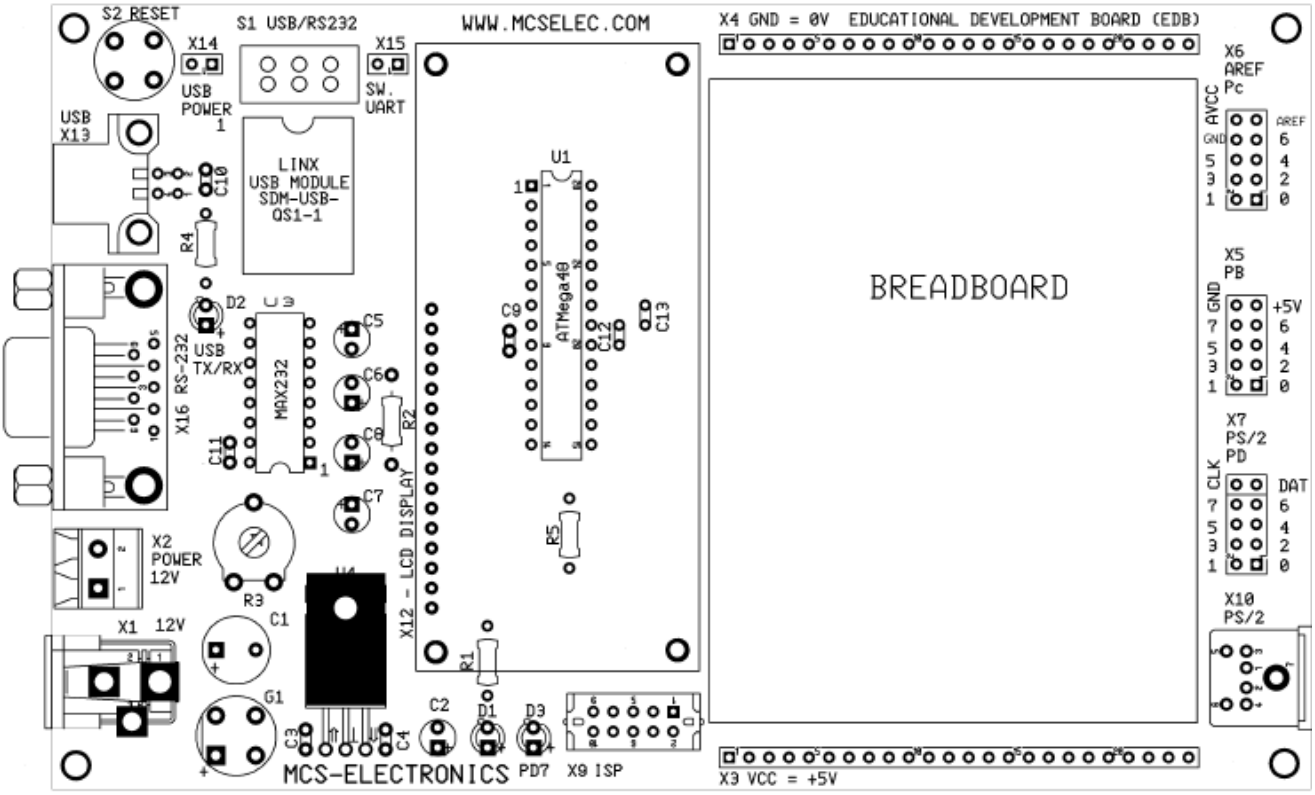
POWER OPTIONS

There are two power options, you can apply an external power supply on X2 or an adapter to X1. You can also use the EDB USB-BUS powered, you then have to solder a wire into X14 USB power. BEWARE if you use USB-BUS power you may not place U4 LM7805 and you may not apply power to X1 and/or X2.

Beware to draw no to much amps from the USB-BUS. (MCS Electronics cannot be held responsible for damage to your computer due to USB-BUS power issues.)

BOARD LAYOUT

To find the right position of the components you may want to reference this drawing that shows the board layout. It is also printed on the PCB itself and available (enlarged) in annex 4.



BEFORE YOU CONNECT THE POWER

Once you have soldered all parts, check the PCB for small solder dots between tracks and remove them.

Measure the resistance between the following pins.
X2.1 and 2.2 should be around 18M OHM
X3 and X4 should be around 2k6.

If you measure a resistance of around 0 OHM you have a short somewhere. Do not connect the power! Check again for solder dots and check if you placed U4 and G1 right

If you measure infinite ohms you probably have not soldered the connector X2, X3 or X4 sufficiently. Re-solder the connectors.

It is possible that the values above are a bit different on your EDB, just make sure you are in a limit of ± 10% of the values mentioned above.

If that worked out fine connect a power supply (or USB power). The (external) power must be in the range from 9-15 Volt DC. The polarity is not important since a diode bridge is used.

The power LED should light now. Measure the voltage between pin 2 and 3 of U4. The measured voltage must be 5V.

If the voltage is not right disconnect the power and check the board.

If the voltage is OK, disconnect the power supply and insert the MAX232 serial buffer chip. Also insert the ATMega88 and finally solder the USB module

Solder a single male header into the LCD module... you can place the LCD into connector X12 (just connect the LCD with male header into X12).

2.2 Installing Bascom AVR

To program the ATMega88 microcontroller on the Educational Development Board you need Bascom AVR. You can choose between a free demo version of Bascom AVR that can compile up to 4kB of code or you can obtain a full version. Both versions are available from www.mcselec.com.

Installation procedure for the full version

The full version comes on a CD. Normally it will start automatically if you insert the disc, if it does not just navigate to your CD-ROM drive and double click setup.exe. You can now proceed to page 11.

Installation procedure for the demo version

- Download the installation files (bcavrd.zip) from www.mcselec.com.

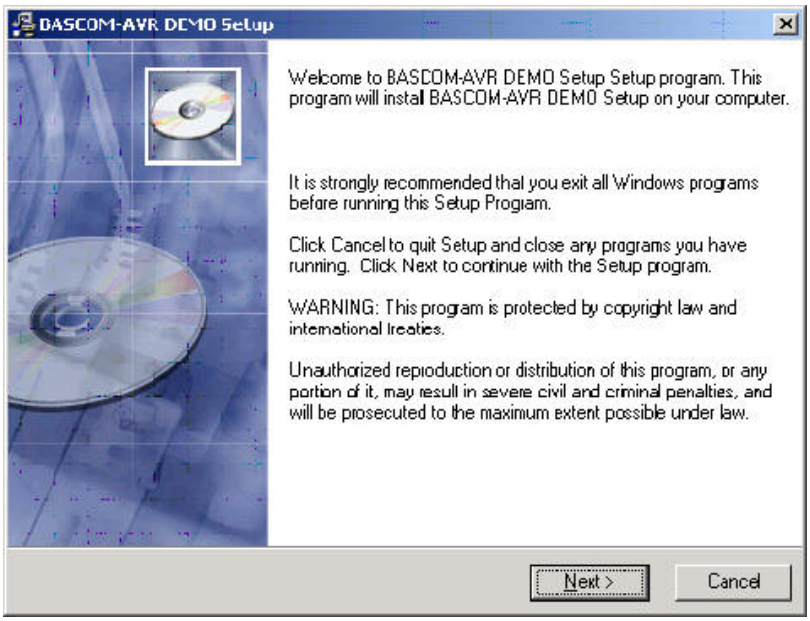
- In Windows XP you can open ZIP files by default. Make sure you extract all the files to the same directory. If your cannot open the ZIP files you might want to install WinZip. (Download at www.winzip.com)
  Do not run setup.exe from the ZIP file!

- After extraction you will have a folder containing the setup.exe file.

- Run setup.exe to start the installation.

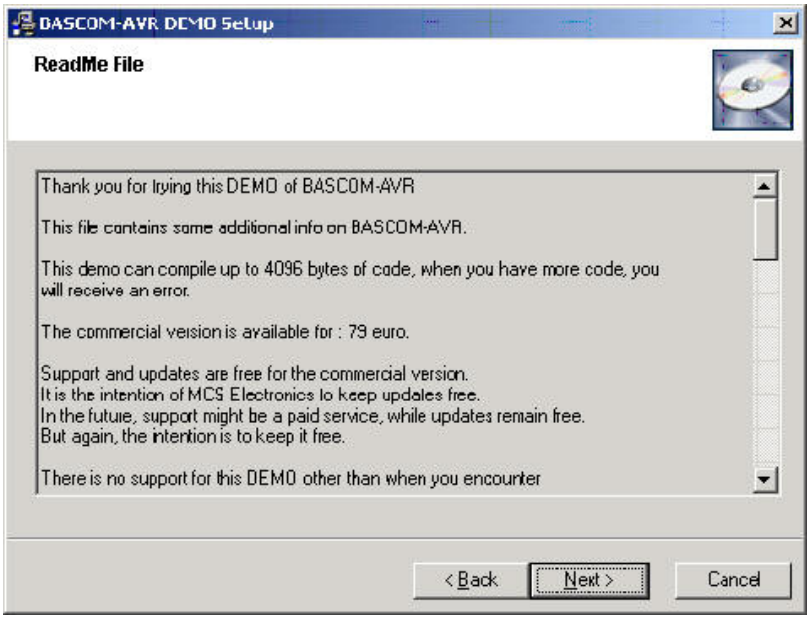
Proceed to the next page.

The following window will appear:



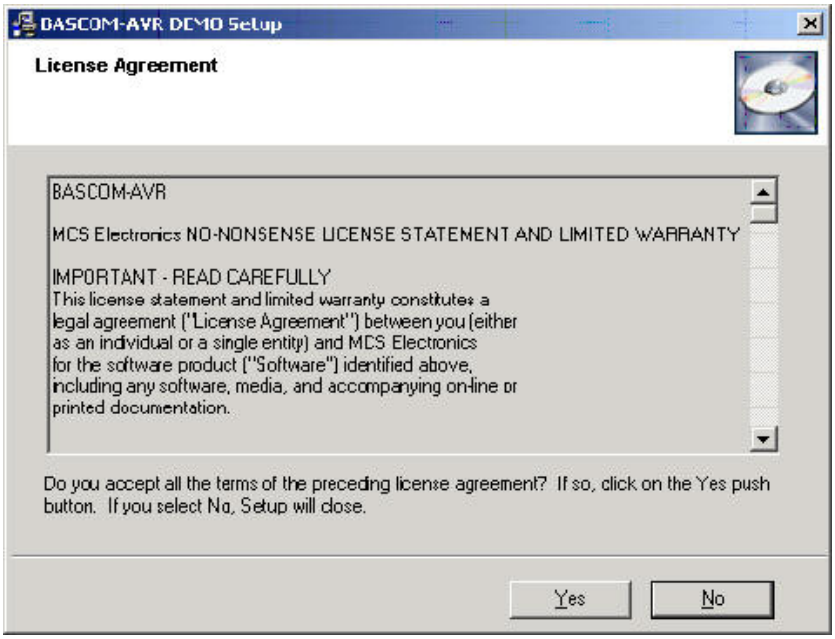
Press the Next button.

The following window will appear:



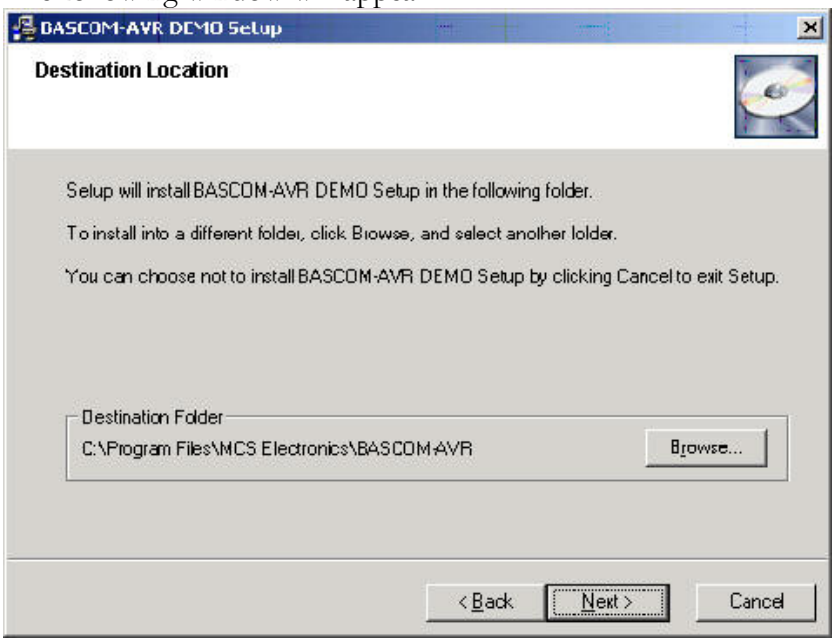
Press the Next button.

The following window will appear:



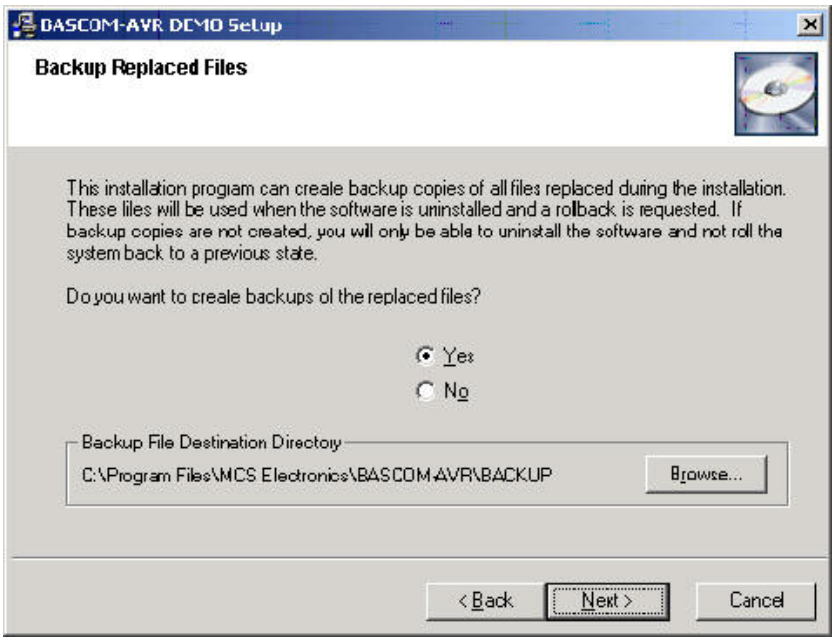
Press the YES button

The following window will appear:



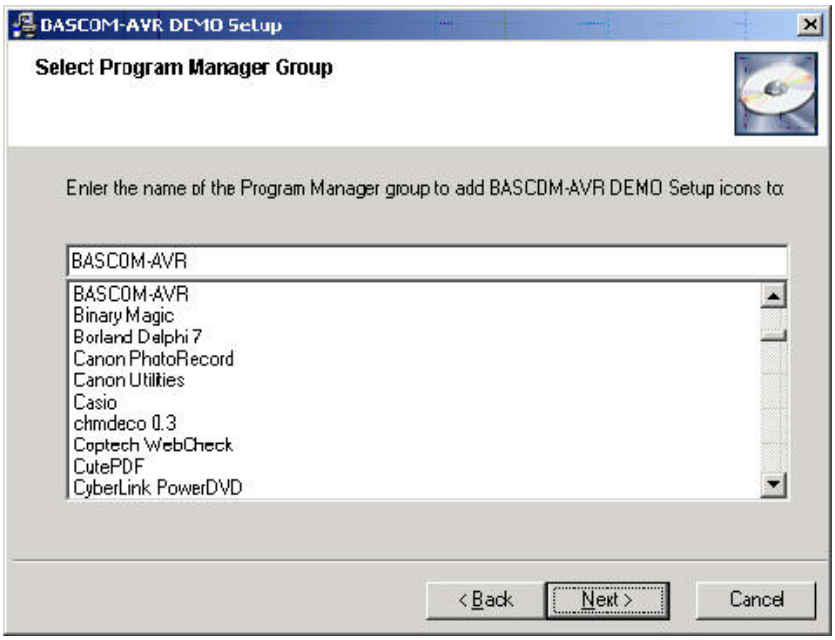
You can select a drive and directory. You can also press the Next button for the default drive and directory. Press the Next button after you have selected the target installation directory.

The following window will appear:



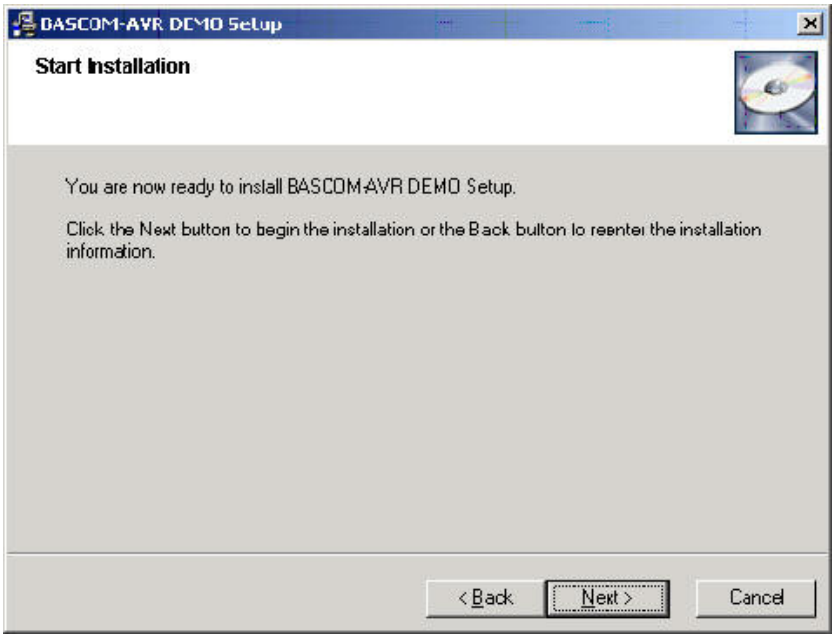
In this windows you can select if you would like to make a back up of all files that will be replaced. Normally there will no files be replaced since they do not exist yet. But select Yes and click the Next button.

This will make the following window to appear:

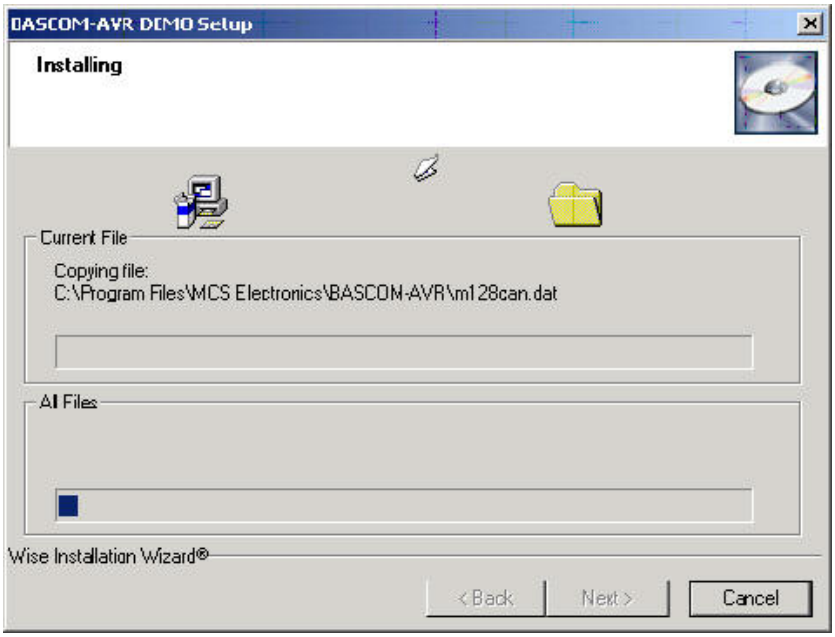


You can select in which program group BASCOM will be installed.
Press the Next button after you have selected or entered a program group.

The following window will appear:



The setup program now knows everything to perform the installation.
Press the Next-button to start the installation.



During installation the window above is shown.

BASCOM-AVR is now ready to use.

When you install BASCOM on Windows NT, Windows 2000 or XP, you will need Administrator rights during setup. You also need administrator rights the first time you run BASCOM. After that you can use BASCOM as any user.

2.3 Programming the controller

thru the serial port (Boot loader procedure)

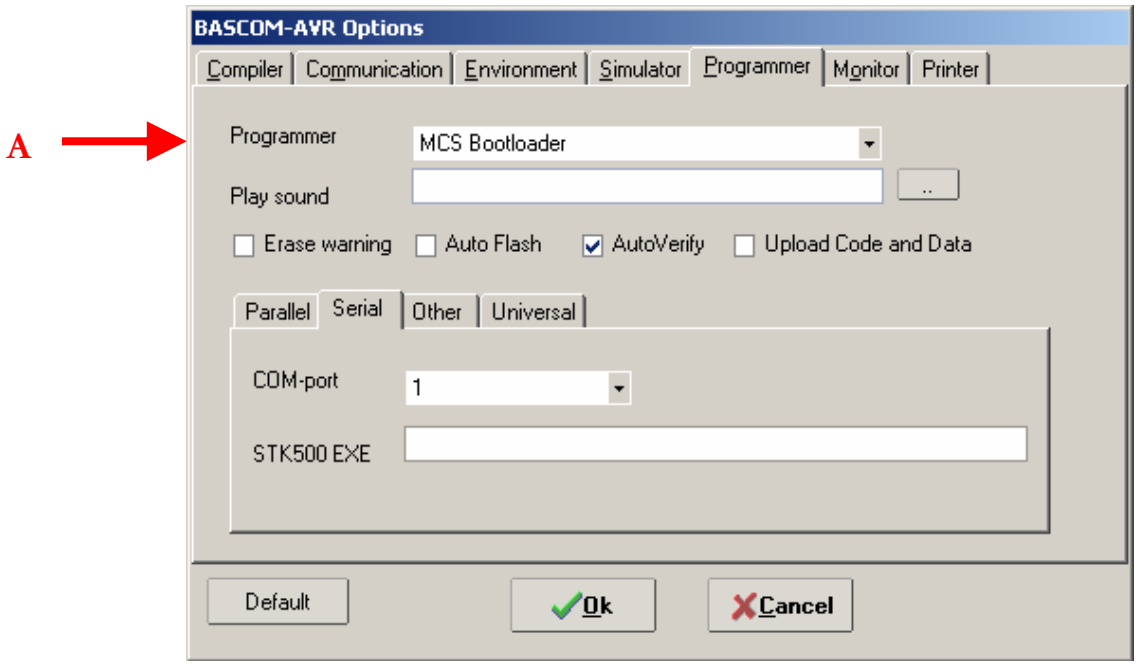
This chapter describes how you can configure Bascom to program the microcontroller thru the serial port.

> The method that programs the controller thru the serial port is called "Boot loader". A boot loader is a piece of software that loads your application into the controller's memory. (The bootloader itself is also placed in the controllers memory, but MCS Electronics already took care of that.)

For programming the EDB thru the serial port you can use the serial cable that is supplied with the KIT. But you may use any straight cable that connects pin 2->2, pin 3->3, pin4->4 and pin 5->5.

Connect the serial cable to the serial port of the computer and connect the other end to connector X16 (DB-9) marked "RS-232" on the Educational Development Board.

In Bascom click on "Options" and "Programmer" now you should see this screen:



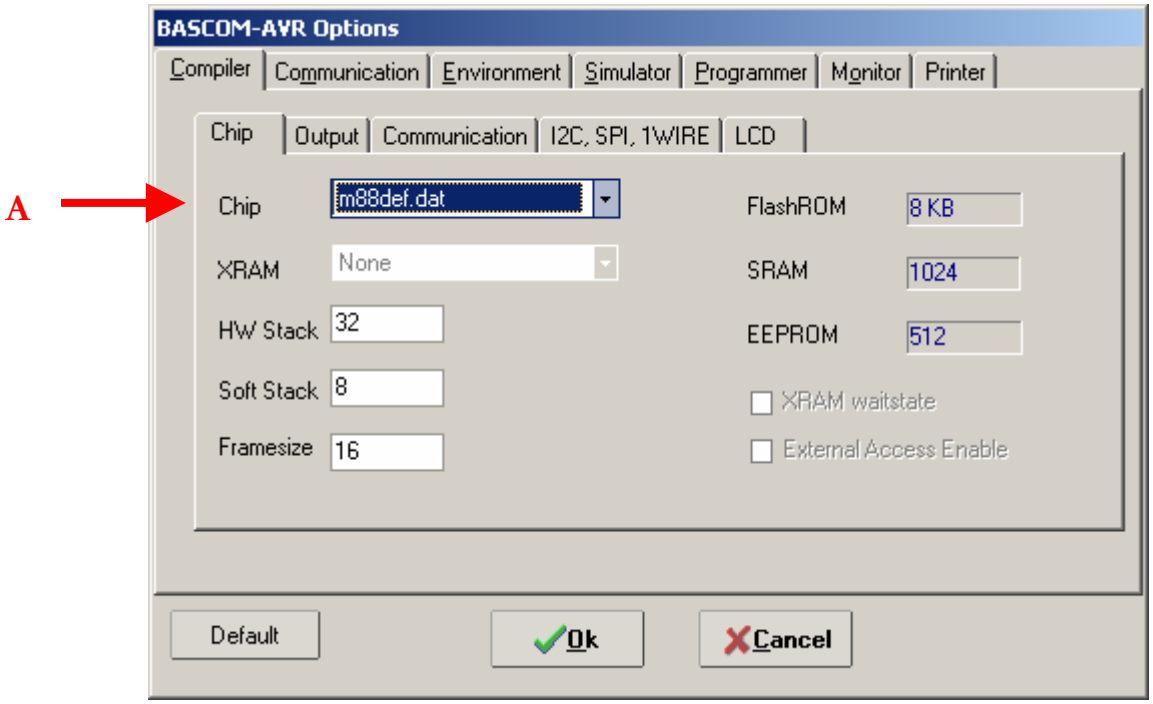
- Select the "MCS Bootloader" from the drop down box (A).
- Select the right COM-port and press the OK-button.

"Now let's program the flash"

Before you can program the chip you first need to open and compile the test program. You can find the test program on the EDB-CD. Open the EDBtest.bas file in Bascom AVR. (Click "File" > "Open")

Now open the chip options menu ("Options" > "Compiler" > "Chip")

You should see this menu:



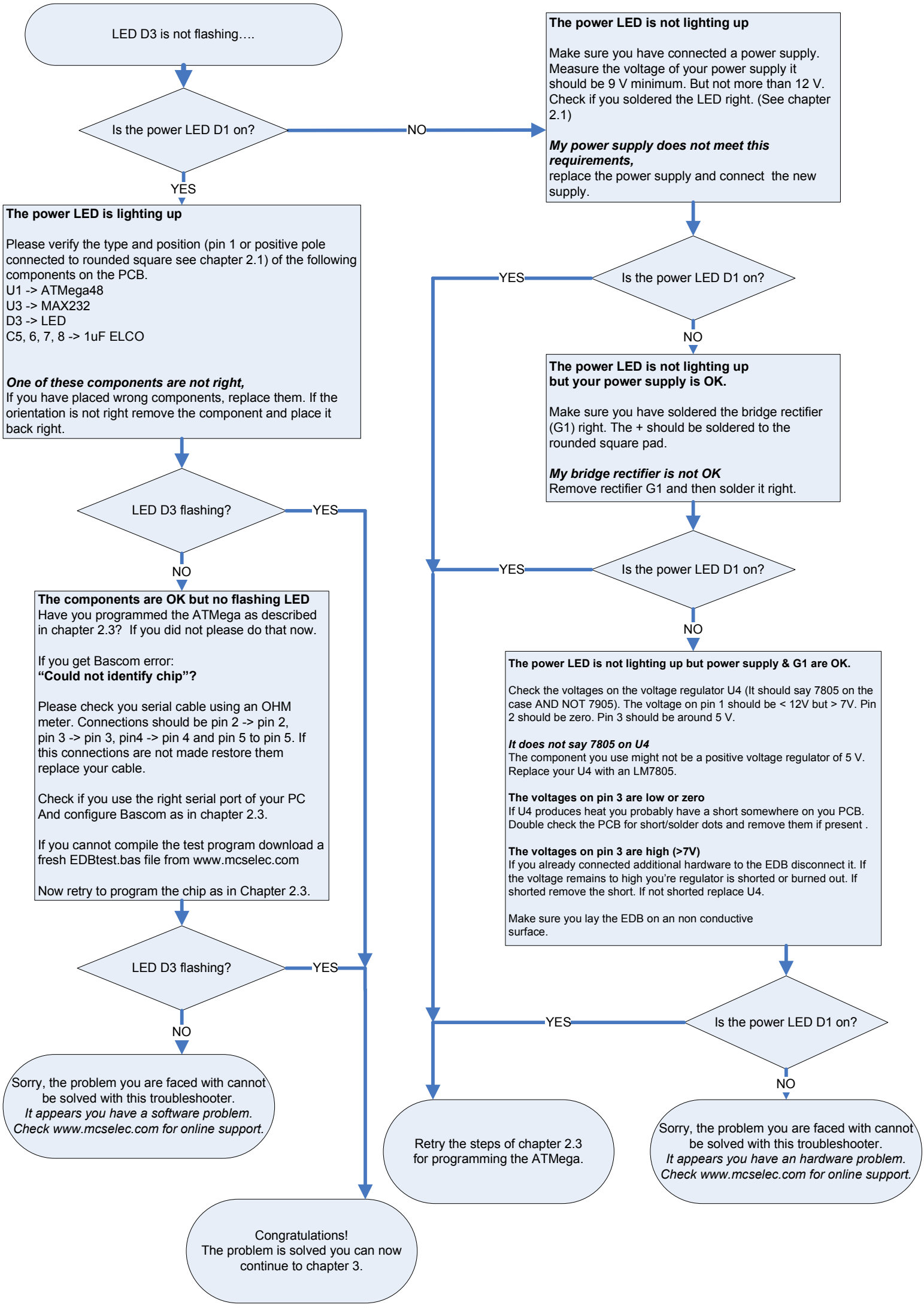
- Select m88def.dat from the "Chip" drop down menu(A).
- Do not change the other options and press OK
- Now press F7 to compile the test program
- Press F4 to auto program the chip.

If everything went good you should now see a flashing LED on the EDB board.

If you use an ATMega88 that is not provided by MCS Electronics or if you have changed the fuse bits please read chapter 4.3 before contacting support.

● Do you see D3 (PD7) LED flashing?
If the answer is YES, congratulations! Continue to Chapter 3
If the answer is No, don't worry... continue to next page.

2.4 Troubleshooting



Chapter

3

3. Experiments

This chapter will teach you the elements of electronics and microcontrollers.

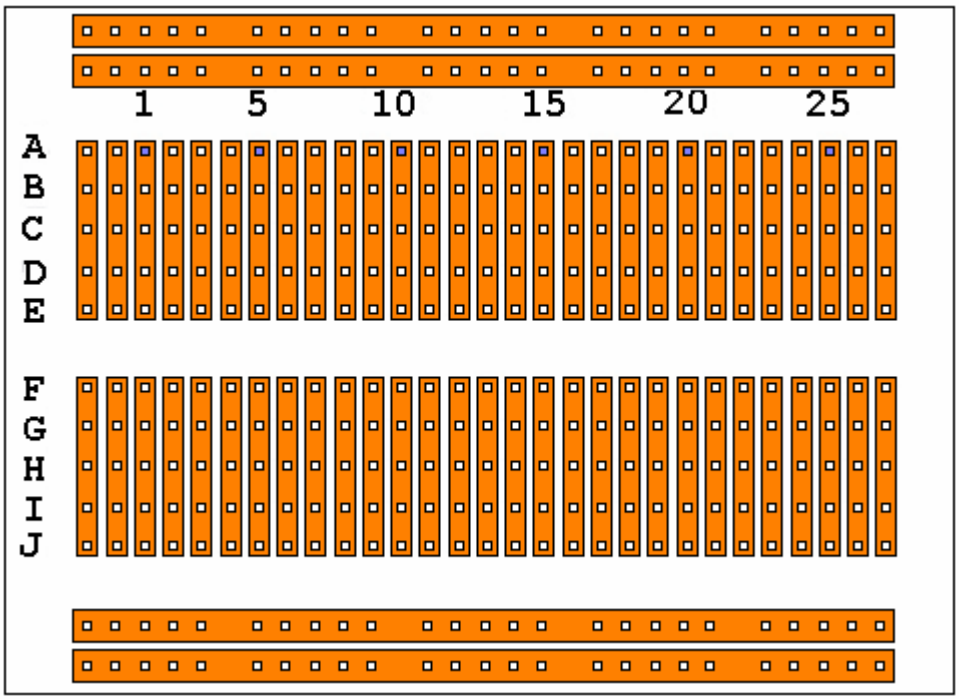
Now you have assembled and tested your Educational Development Board, you can start experimenting. The experiments follow up in a logical, constructive order which will keep you encouraged to continue to the next experiment.

The experiments are placed in these subcategories,

- Basic I/O
- Serial communication with the UART
- Display output (LCD, 7 segment)
- Keyboard input (PS/2, Matrix, RC5 remote control)
- Advanced I/O
- A/D conversion
- Other controller features
- Other interfaces (I^2C, USB)
- Motors

BREAD BOARD

A bread board is used to make solder less connections between components and wires in an experimental stage. You can use it to build the experiments. If you haven't used one before, take a look at this drawing that shows the internal connections of the bread board.



The **copper colored** areas show the internal connections of the board. You can insert all components with a 2,54 mm spacing between the pins. Use wires between 0,3 and 0,8 mm in diameter. Never force wires or components into the board.

DANGER! RISK OF SERIOUS INGURIES OR DEATH

The bread board is not suitable for connections on/to the electricity net.

Never connect the electricity net to the bread board

3.1 Basic I/O Experiments

I/O means "input and output". Inputs are usually connected to input pins of a microcontroller while output usually is connected to output pins. (A bidirectional pin of you're microcontroller can be configured as input or output.) I/O can present itself in many ways the easiest forms are a LED output and a switch input.

3.1.1 LED output, switch input

Experiment 1

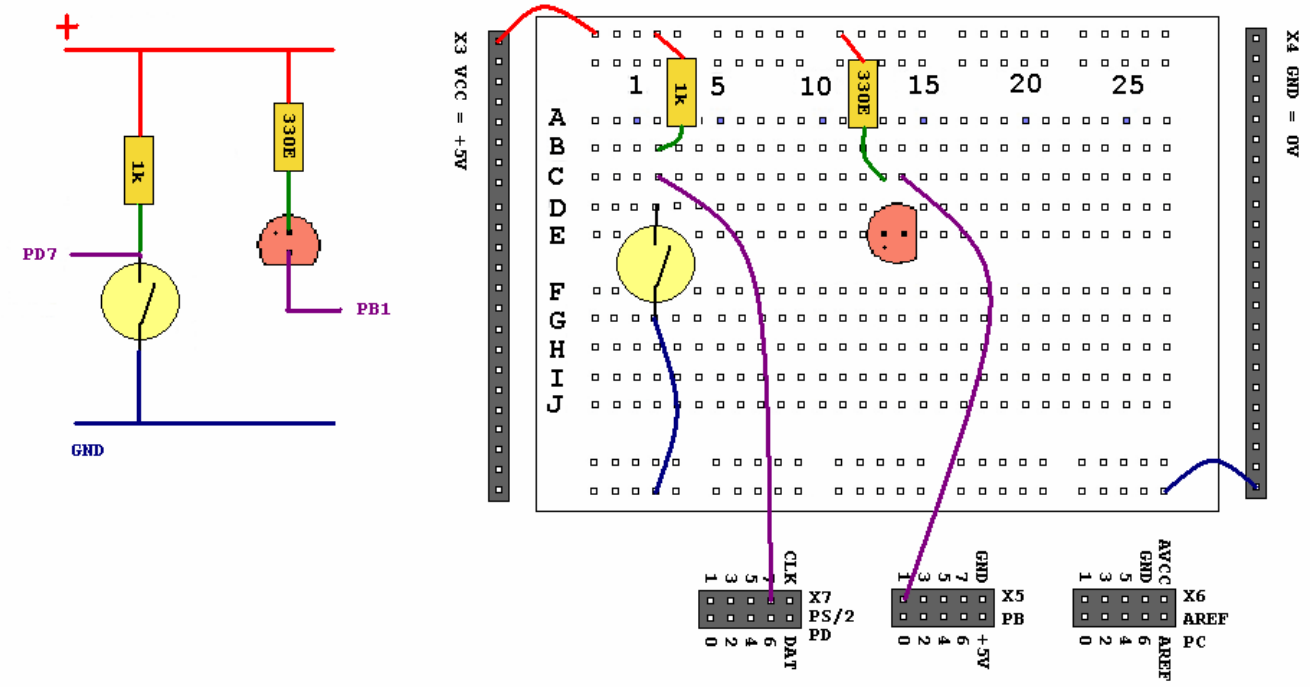
Part list

1x Resistor 1k
1x Resistor 330E
1x LED
1x Switch or a wire
6x Bread board wires

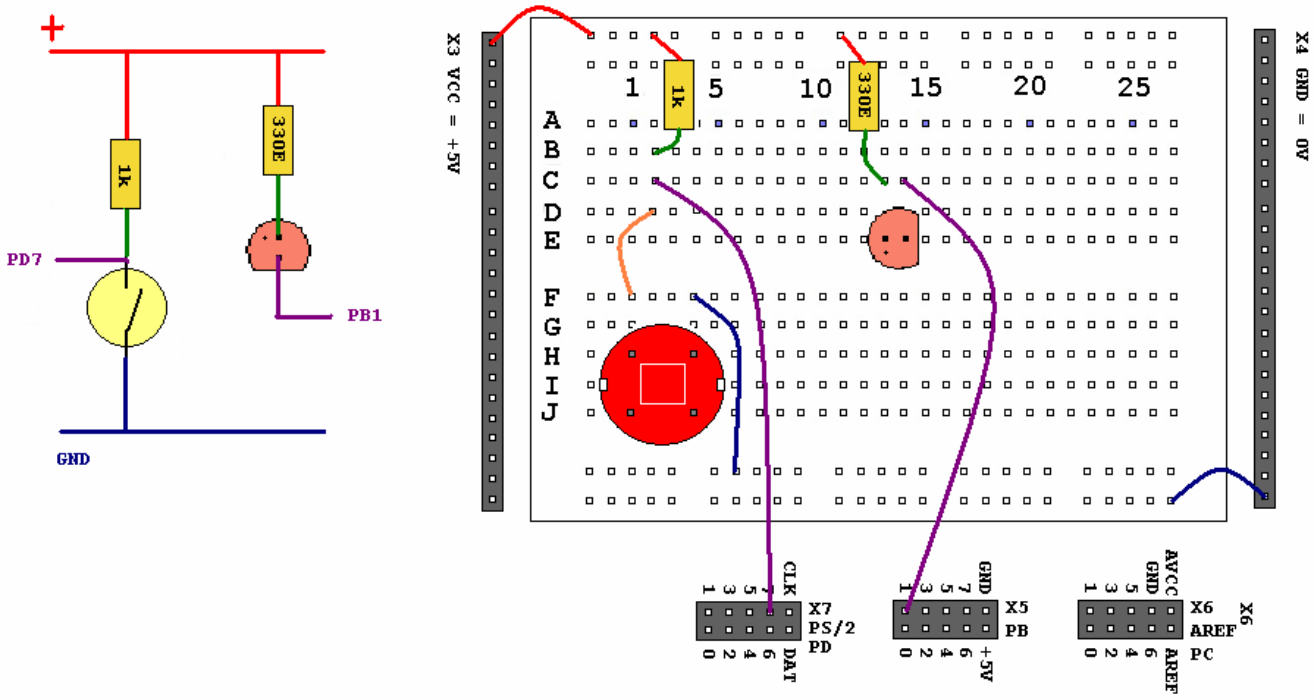
Goals

- Understanding basic input and output (I/O)

Now let's build the circuit shown below. It shows how to connect a switch and a LED to your microcontroller. If you don't have a switch you can also use a plain wire. There is also a suggested breadboard set up.



You can also use a switch supplied by MCS Electronics the board would then look like this:



Now open the EDBexperiment1.bas file from the EDB-CD.
It looks like this:

```
'-----------------------------------------------------------------
'                       EDBexperiment1.bas
'       Experiment 1 for the Educational Development Board
'                 (c) 1995-2005, MCS Electronics
'                       Fileversion 1.0
'-----------------------------------------------------------------
'
'Purpose:
'This program assigns the value of Pind.7 to Pinb.1.
'
'Conclusions:
'You should be able to switch the led with a key press.

$regfile = "m88def.dat"                                    'To tell Bascom which chip we use

Config Pind.7 = Input                                      'Configure Pin D7 as input
Config Pinb.1 = Output                                     'Configure Pin B1 as output


Do                                                         'Do...loop will loop forever
   Portb.1 = Pind.7                                        'Assign the output with the input value
Loop

End
```

Program this code in your controller just as you did with the test program from chapter 2.3 *(Under "Now let's program the flash")*. Test the functionality of this experiment by pressing the switch. (The led should light up if you keep the switch down)

Experiment

2

3.1.2 Shifting LED's

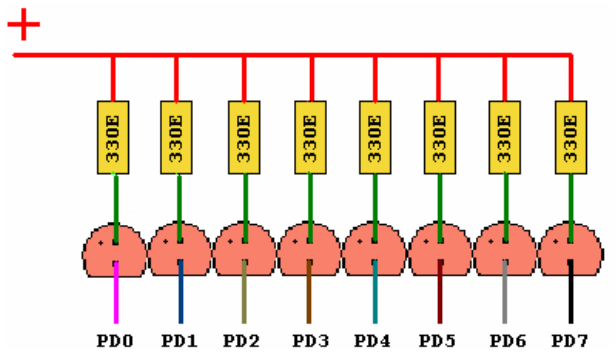
Part list

8x Resistor 330E
8x LED
9x Bread board wires

Goals

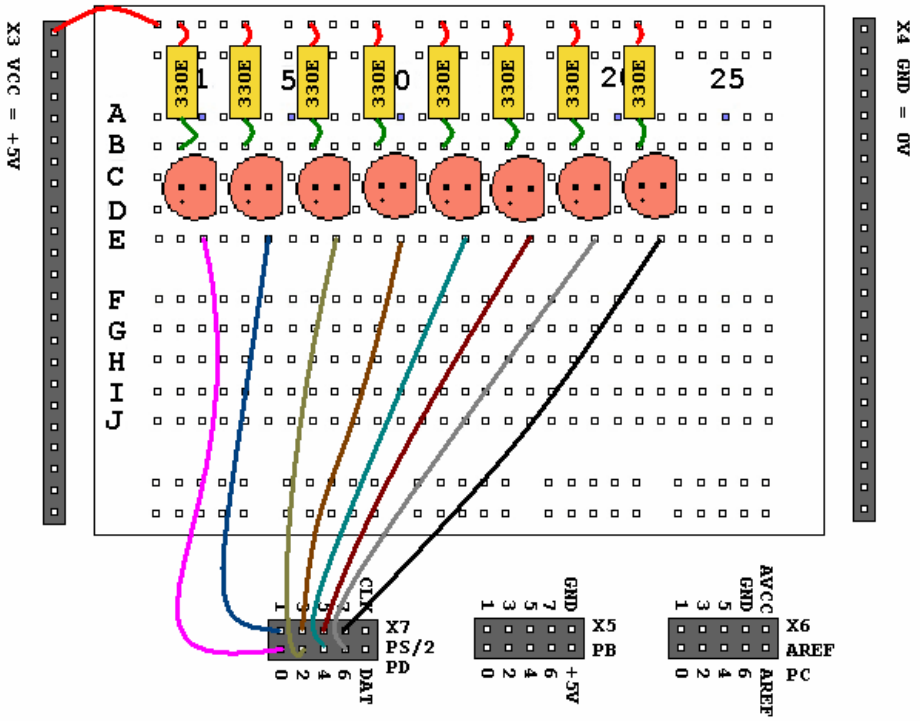
- Understanding the rotate statement,
- Understanding basic (I/O)

Another example of basic I/O, a rotate operation.



You don't have to place any connections to ground. The ground will be supplied thru the controller.

A suggested bread board set up:



Now open the EDBexperiment2.bas file from the EDB-CD.
It looks like this:

```
'-----------------------------------------------------------------
'                       EDBexperiment2.bas
'       Experiment 2 for the Educational Development Board
'                  (c) 1995-2005, MCS Electronics
'                       Fileversion 1.0
'-----------------------------------------------------------------
'
'Purpose:
'This program 'rotates' the LED's on port D.
'
'Conclusions:
'You should be able to see a moving LED effect.

$regfile = "m88def.dat"                                    'To tell Bascom which chip we use

Config Portd = Output                                      'Configure Pin D7 as output

Portd = &B1111110                                         'We start by making pd0 low (led on)

Do
   Rotate Portd , Right , 1                                'Move the 'bits' one place to the right
                                                           'Now PD1 will light and PD0 goes out.
   Waitms 100                                              'Wait otherwise we cannot see te rotate
Loop

End
```

Program this code in your controller just as you did with the test program from chapter 2.3 *(Under "Now let's program the flash")*. You now see one led moving from the left to the right over your bread board.

Change the line:

```
Rotate Portd , Right , 1
```
into:
```
Rotate Portd , Right , 2
```

Can you describe the difference?

Now find out yourself how to change the direction of the rotating LED's from right to left. You can find help by moving the cursor to the rotate statement and pressing F1.

Experiment
3a

3.1.3a Output with transistor or FET

Part list
1x BC547 NPN transistor
1x LED
5x Bread board wires
1x Resistor 330, 47k ohm

Goals
- Basic understanding of the electrical properties of the microcontroller
- Basic transistor and FET usage

Theory

Experiments 1 and 2 use "direct I/O" that means we directly connect the switch/LED to the controller. A simple solution but there are disadvantages. First of all you cannot draw much current from the microcontroller port pins. Secondly direct I/O brings risks of damaging the controller by electro static discharge (ESD). (ESD is discussed briefly in chapter 3.1.5.

The current drawn from a microcontroller pin may not exceed specific values. These values can be found in the ATMega88 datasheet available from www.atmel.com. The currents can be found under "**Electrical Characteristics**". An example of the Electrical Characteristics can be found on the next page.

First note the table titled "**Absolute Maximum Ratings**" the values in this table are as said Absolute Maximum, that means that if you stay below this values the device will not be damaged. It does *not* say anything about the functionality of your application. (So if you draw 40 milliamps from one port pin, Atmel guarantees that you will not destroy the controller. Atmel does *not* guarantee the functionality of the device while you draw that 40 milliamps)

What currents can you draw?
If you draw current from a pin that is at 0V level:
1. The sum of all pins for ports C0 - C5, ADC7, ADC6 should not exceed 100 mA.
2. The sum of all pins, for ports B0 - B5, D5 - D7, XTAL1, XTAL2 should not exceed 100 mA.
3. The sum of all pins, for ports D0 - D4, RESET should not exceed 100 mA.

If you draw current from a pin that is at 5V level:
1. The sum of all pins, for ports C0 - C5, D0- D4, ADC7, RESET should not exceed 150 mA.
2. The sum of all pins, for ports B0 - B5, D5 - D7, ADC6, XTAL1, XTAL2 should not exceed 150 mA.

All this with a total current of 200mA for the entire device (Absolute Max.)

Please refer to the latest datasheet for the full characteristics. And also since the values mentioned above were derived from a preliminary datasheet.

27. Electrical Characteristics

27.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

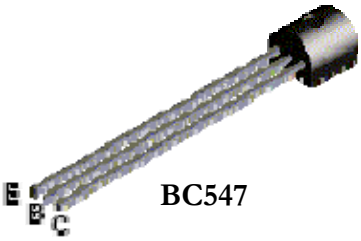
*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

27.2 DC Characteristics ATmega48/88/168*

* Note: The following table contains preliminary data for the ATmega88 and ATmega168.

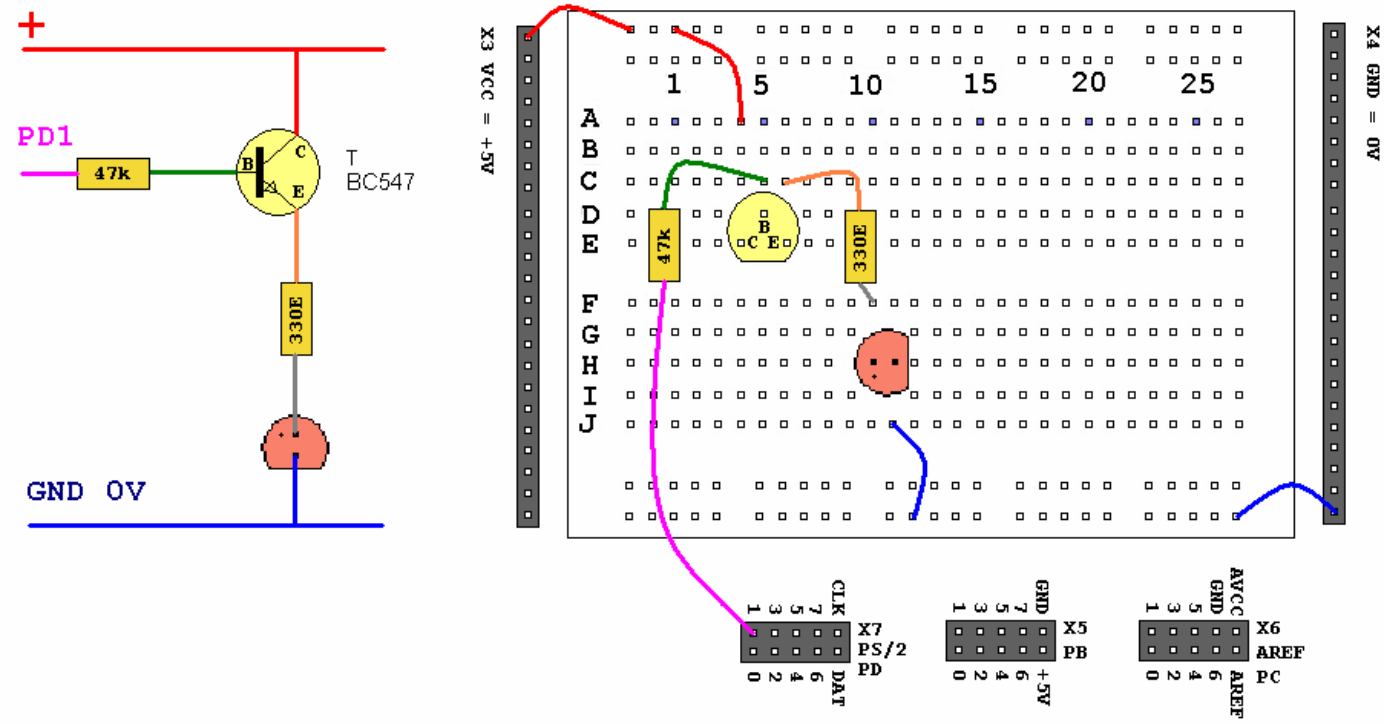
T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.(5)	Typ.	Max.(5)	Units
V_{IL}	Input Low Voltage, except XTAL1 and $\overline{\text{RESET}}$ pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IH}	Input High Voltage, except XTAL1 and $\overline{\text{RESET}}$ pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1)	V
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$(2) $0.7V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL2}	Input Low Voltage, $\overline{\text{RESET}}$ pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.2V_{CC}$(1)	V
V_{IH2}	Input High Voltage, $\overline{\text{RESET}}$ pin	V_{CC} = 1.8V - 5.5V	$0.9V_{CC}$(2)		V_{CC} + 0.5	V
V_{IL3}	Input Low Voltage, $\overline{\text{RESET}}$ pin as I/O	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IH3}	Input High Voltage, RESET pin as I/O	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{OL}	Output Low Voltage(3), $\overline{\text{RESET}}$ pin as I/O	I_{OL} = 20mA, V_{CC} = 5V I_{OL} = 6mA, V_{CC} = 3V			0.7 0.5	V
V_{OH}	Output High Voltage(4), $\overline{\text{RESET}}$ pin as I/O	I_{OH} = -20mA, V_{CC} = 5V I_{OH} = -10mA, V_{CC} = 3V	4.2 2.3			V
V_{OL3}	Output Low Voltage(3), $\overline{\text{RESET}}$ pin as I/O	TBD			TBD	V
V_{OH3}	Output High Voltage(4), $\overline{\text{RESET}}$ pin as I/O	TBD	TBD			V



BC547

You can use a transistor if you want to draw more current than can be supplied by the controller. The drawing below shows how to use a transistor as switch.



Open the EDBexperiment3.bas file from the EDB-CD and program it into the ATMega88.

Online resources:

BC547: http://www.fairchildsemi.com/pf/BC/BC547.html

ATMega88: http://www.atmel.com/dyn/products/product_card.asp?part_id=3302

Experiment

3b

3.1.3b Output with FET

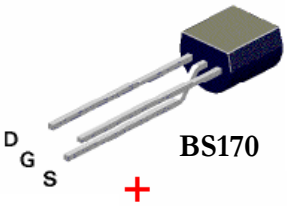
Part list

1x BS170 N-channel enh. FET
1x LED
5x Bread board wires
1x Resistor 330, 47k ohm

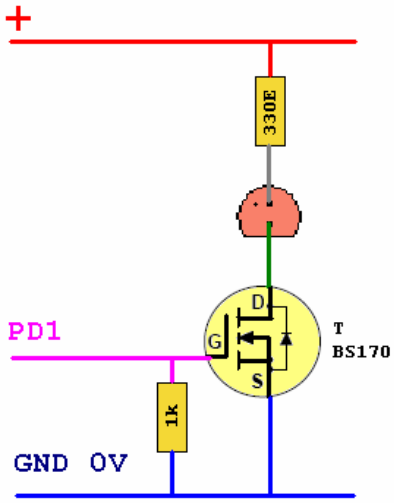
Goals

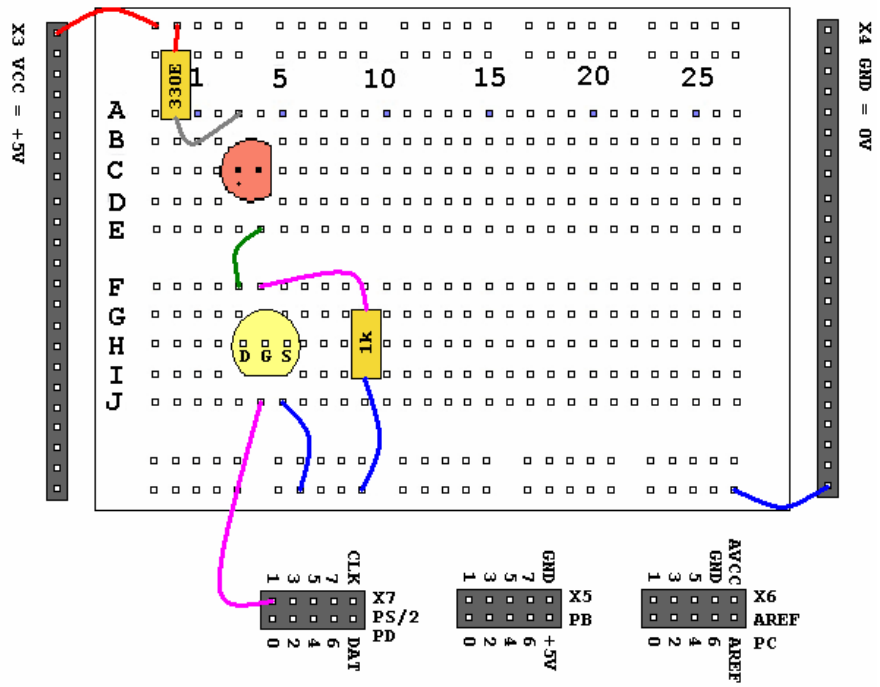
- Basic FET usage

Using a MOSFET instead of a "classic" BC547 transistor has some advantages. A MOSFET can give more power and has a gate current close to 0 mA.



This drawing shows a way to connect a FET to your microcontroller.





If you programmed the ATMega88 in experiment 3a you don't have to reprogram it. If you did not, open EDBexperiment3.bas from the EDB-CD and program it into the ATMega88. Do not remove the components from the board.

Online resources:

BS170: http://www.fairchildsemi.com/pf/BS/BS170.html

Experiment

3c

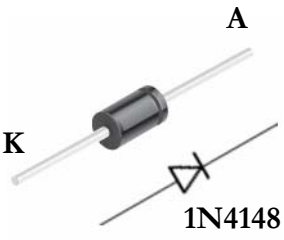
3.1.3c Output with relay

Part list

1x BS170 N-channel enh. FET
1x Relay (Schrack PE014-005)
7x Bread board wires
1x Resistor 1k ohm
1x Diode 1N4148

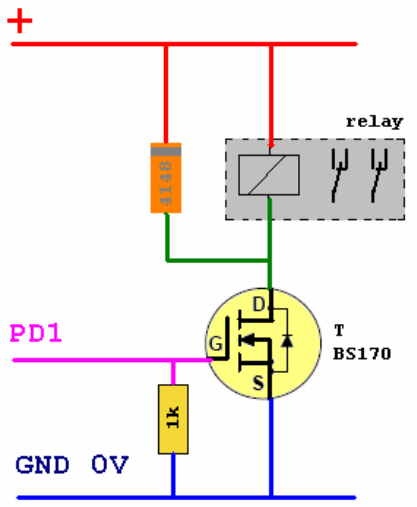
Goals

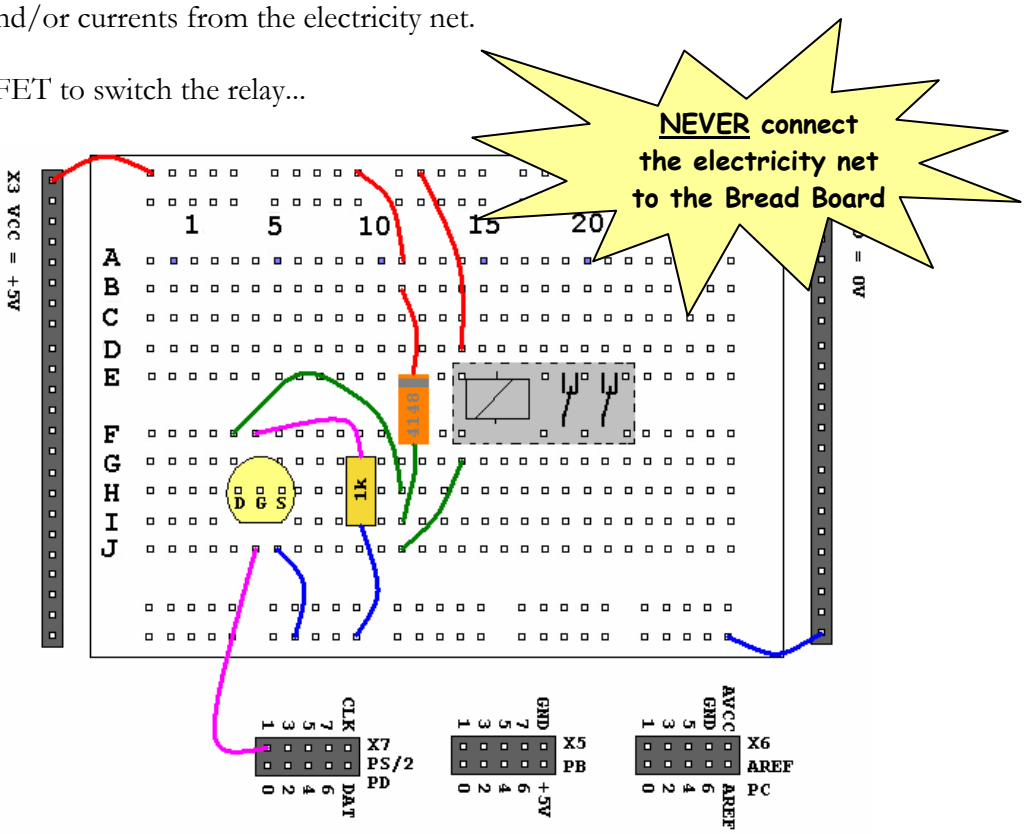
- Learning how/why to use a relay



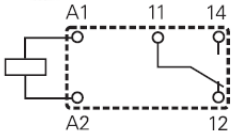
MOSFET's and transistors are both semiconductors. The ones you have used in the passed experiments can only switch DC. You can use a relay if you want a potential free contact. That also gives you the possibility to switch AC and/or currents from the electricity net.

You need a FET to switch the relay...





Example of a Relay
But beware there are many different types!



If you programmed the ATMega88 in experiment 3a or 3b you don't have to reprogram it. If you did not, open EDBexperiment3.bas from the EDB-CD and program it into the ATMega88.

Online resources:
Relay: http://relays.tycoelectronics.com/

Experiment
4

3.1.4 Output with a ULN2803

Part list

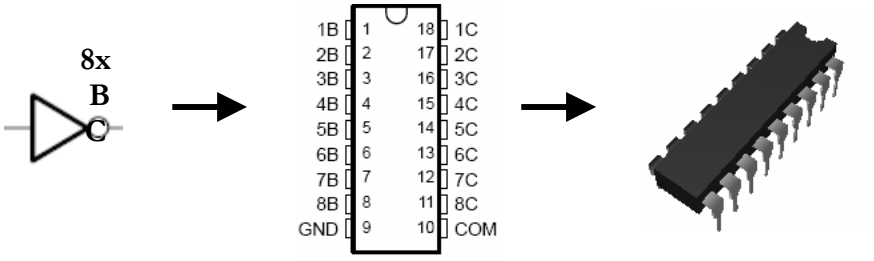
1x ULN2803 or equivalent
2x LED
7x Bread board wires
2x Resistor 330E

Goals

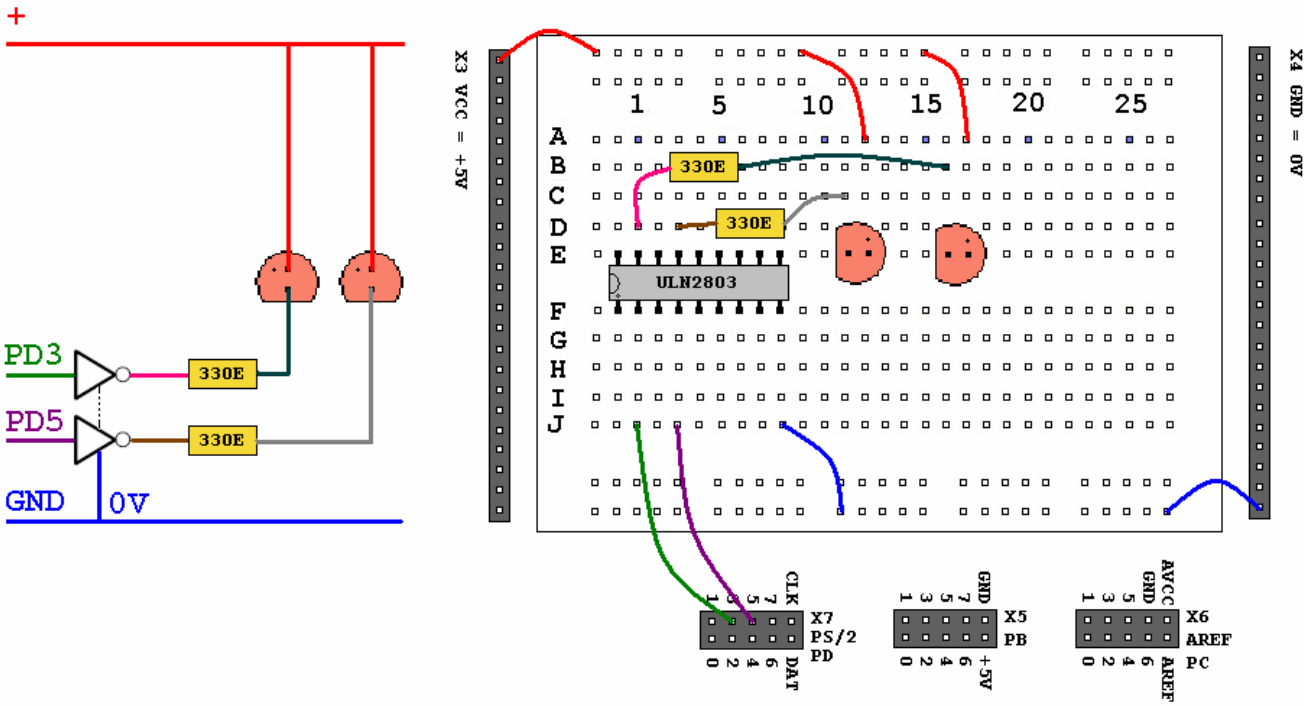
- Learning how to use an ULN2803

In a lot of applications just one output doesn't do the job. Imagine the bunch of components you would need if you wanted 7 or 8 transistor outputs! (See chapter 3.1.3a) Not only would that be expensive it would also be unreliable.

To solve this issue you can buy transistor arrays in DIL package. The ULN2803 has 8 build in "ports" that you can use for currents up to 30 mA per pin.



Build the drawing shown below:



Open the EDBexperiment4.bas file from the EDB-CD and program it into the ATMega88.

In this experiment we have only connected 2 ports of the ULN2803. Of course it is possible to connect 8 LED's. You could also connect relays, make sure not to draw to much amps. Refer to the datasheet for more information.

Online resources:
ULN2803: http://focus.ti.com/docs/prod/folders/print/uln2803a.htm

Experiment

5

3.1.5 I/O with Opto coupler

Part list

1x PC817 or SFH601 or equivalent
1x LED
7x Bread board wires
1x Resistor 330E, 820E

Goals

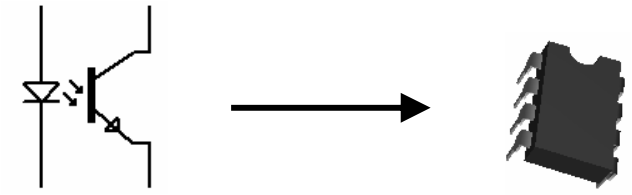
- Learning how/why to use an Opto-Coupler
- Knowing the existence of ESD

Theory ESD

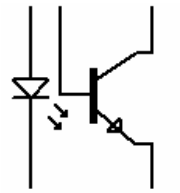
If you ever got a spark when touching something or someone, you have experienced electro static discharge(ESD). Semiconductors are very sensitive for damage due to ESD. ESD is not only formed by human beings and it cannot always be seen or felt.

Designing ESD-proof electronics is not easy and it lays far beyond the scope of this manual. However we will show you how to obtain ***galvanic isolation*** with an opto-coupler.

An opto-coupler is a special transistor that has a build in LED. The transistor does not need a BASE pin. If a current flows thru the LED, the light emitted by the LED will create the BASE current that normally is applied thru the BASE pin of a normal transistor. Creating a situation that can be seen here:

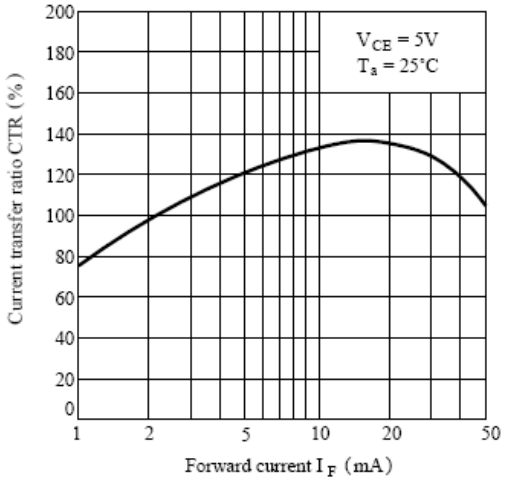


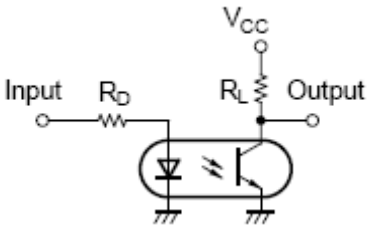
There are also types that do have a BASE pin, in that case you can use the transistor using the LED and/or the BASE pin, like this:



For this experiment you can use the PC817 or the SFH601 or another opto-coupler. Let's say you use the PC817 take a look a the datasheet that can be found on the EDB-CD. Check figure 4, it shows which current needs to be applied.

Fig. 4 Current Transfer Ratio vs. Forward Current

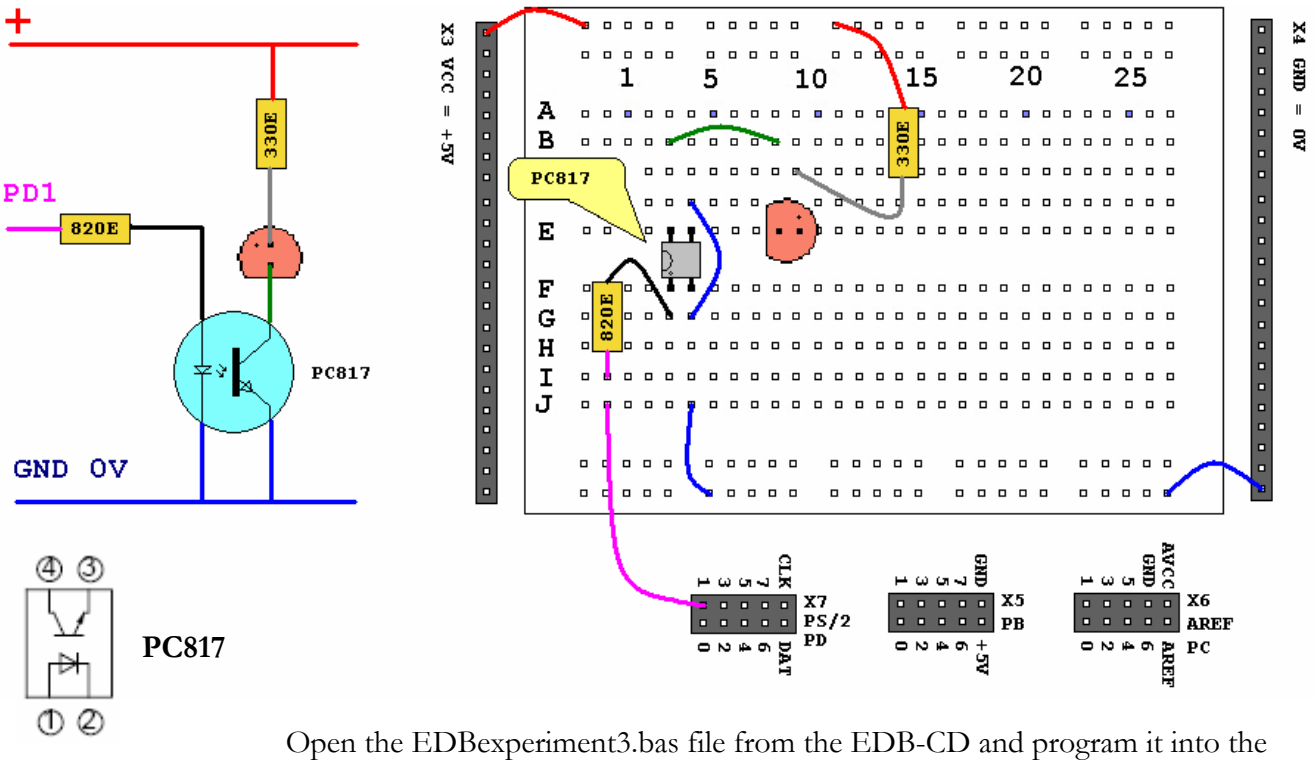




To be safe you better apply a current of about 5mA. Let's say that the voltage falling over the LED is 1,3V (Forward Voltage) and the VCC level is 5V the voltage over Rd should be 5-1,3= 3,7V. Then Rd should be Rd = 3,7V / 5mA = 740 OHM.

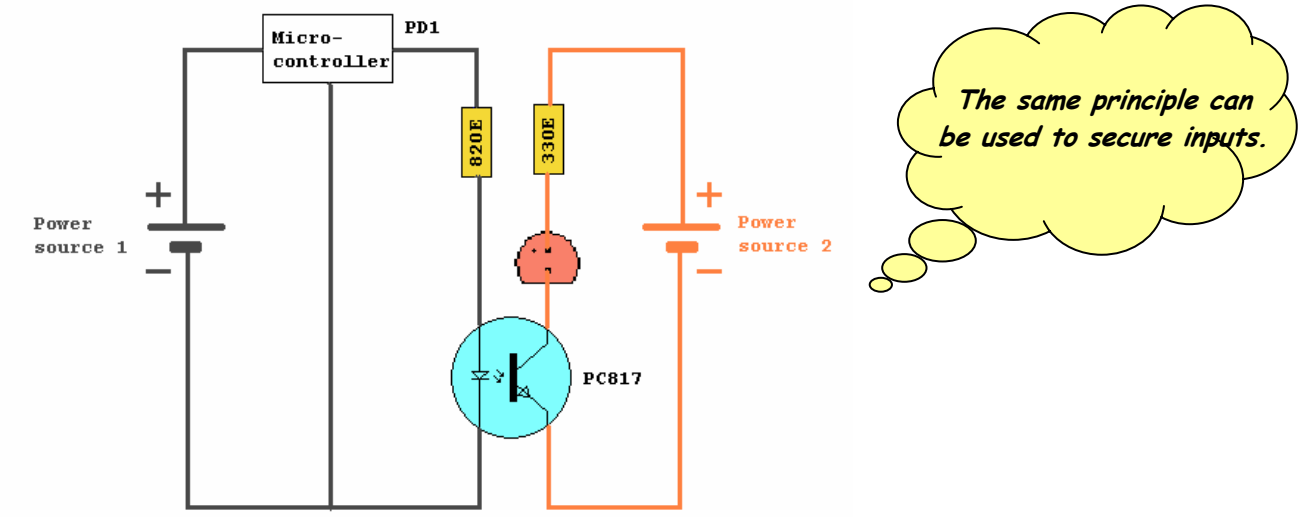
740 OHM is not a value that can be found in the E24 resistor array. The nearest value is 820. That would give us a current of 4,5 mA which is 120% thus OK.

Online resources:
SFH601: http://www.vishay.com/optocouplers/list/product-83663/



Open the EDBexperiment3.bas file from the EDB-CD and program it into the ATMega88.

As you can see the LED flashes again. ESD can occur thru all sorts of *conducting* material. So in the drawing above the PD1 pin is secure. But ESD can also come thru the GND and + lines and destroy the controller anyway. To prevent this you have to make full galvanic isolation like this:



'Just take a look at it, you don't really have to build this now.

3.2 UART Data communication

We have now seen Basic I/O that's solves a lot of simple problems for us. We can let a user press a key and confirm a key press with a LED. But what if we would like a user to enter his name and then display his name?

This chapter tells you the most popular way of data communication in the world of microcontrollers.

3.2.1 The UART

Experiment **6a**

Part list
1x RS232 cable
1x Computer with COM port

Goals
- Learning how to use the serial connection

Theory

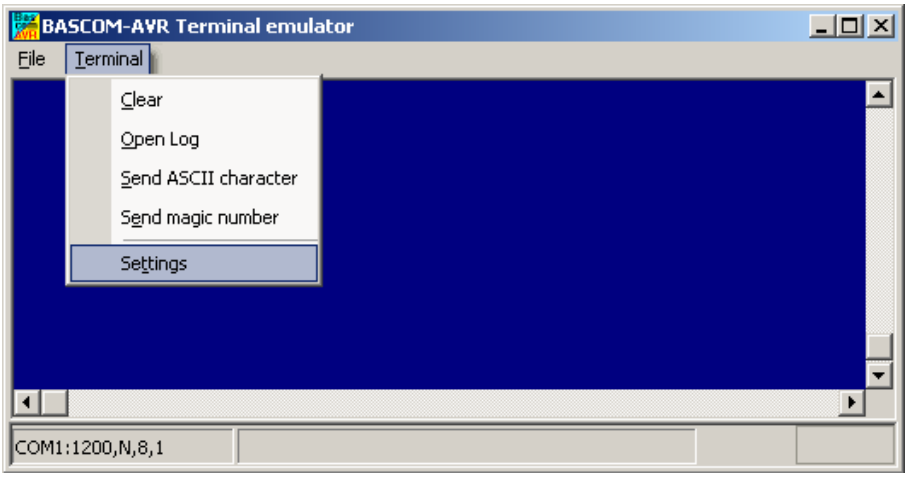
UART means Universal Asynchronous Receiver and Transmitter. You can use a UART to send and receive data between your PC and your EDB.
A UART looks a lot like the COM-port on the back of your computer. The big difference is that your COM-port uses RS232/V24 signal levels (+15 and -15V) while the UART uses TTL levels (5V) or LVTTL (3V or less). To make the UART work with the PC we use a *MAX232 level shifter (U3)*. The MAX232 is an IC that generates voltages used to communicate with the computers COM-port.

For this experiment you need to connect a straight serial cable between X16 (the DB-9 connector marked "RS-232") on the EDB and the COM-port on your PC.

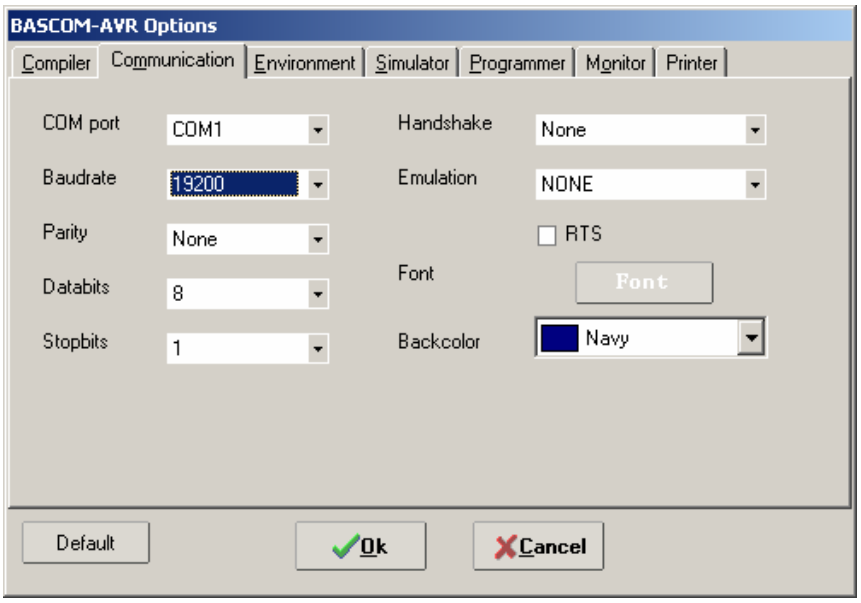
At least the cable should connect
pin2 to pin2,
pin3 to pin3
pin4 to pin4 and
pin5 to pin5.
Make sure switch "S1 USB/RS232" on the EDB is set to RS232.

After you have connected the cable start Bascom AVR and open the Bascom build-in Terminal emulator by clicking on  .

You should see this:



Click "Terminal" and "Settings"



Select the COM port you have used on your PC and select 19200 for baud rate.

If you have a PC without a COM port you can buy an USB to Serial adaptor and use that as a virtual com port.

Close the terminal emulator window.

As said, we use a UART to send and receive data between the on board microcontroller and the PC. Let's define de direction of data as follows:

Seen from the EDB:
Send data means -> data from the EDB to the PC and
Receive data means -> data from the PC to the EDB.

Basically the data looks the same, but we need to be specific so you can understand what we mean. The data can be variables, register values, constants (Text or numbers) or results of calculations. You can also let the controller print a line to show what routines of your program-code is being executed in debug-phase.

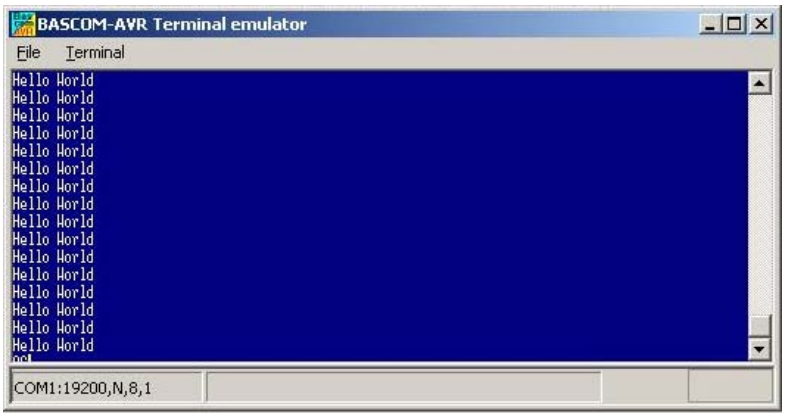
Data can also been send from keyboards allowing an end user to enter messages or use a menu. We will explain later.

Experiment 6a

Open the EDBexperiment6a.bas file from the EDB-CD and program it into the ATMega88. Please read the remarks that can be found in the EDBexperiment6a.bas file they tell you how to initialize and use the UART.

After you have programmed the controller and connected the serial cable, open the terminal emulator by clicking on  in Bascom.

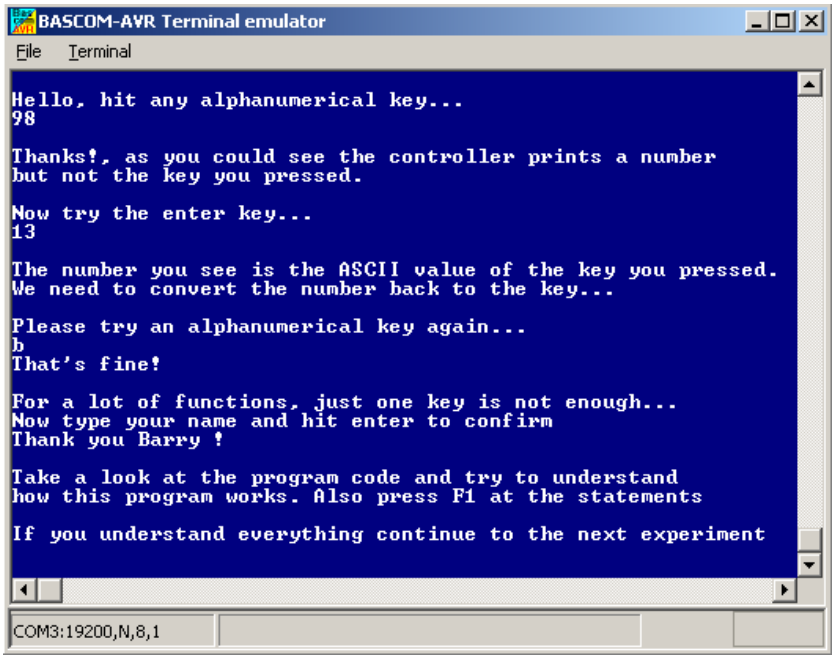
You should see the EDBexperiment6a.bas program printing "Hello World" to the screen of your PC like this:



Experiment 6b

Make sure switch "S1 USB/RS232" on the EDB is set to RS232.
Open the EDBexperiment6b.bas file from the EDB-CD and program it into the ATMega88. After you have programmed the controller and you connected the serial cable, open the terminal emulator by clicking on  in Bascom.

Follow the instructions that are given on the screen by the EDBexperiment6b.bas program:



Please note that with an RS232 connection you can also connect PC's to PC's and microcontrollers to microcontrollers.

Please read some more theory on the next page.

Online resources:
UART Hardware
MAX232: http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1798

Theory
continued

ASCII

As you could have seen in the EDBexperiment6b.bas file 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 in Annex 2.

CARRIAGE RETURN (CR) AND LINE FEED (LF)

As you could have seen in the EDBexperiment6b.bas file we use the PRINT statement to send something to the UART. 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.

Experiment

6c

3.2.2 The Software UART

Part list

1x RS232 cable
1x Computer with COM port

Goals

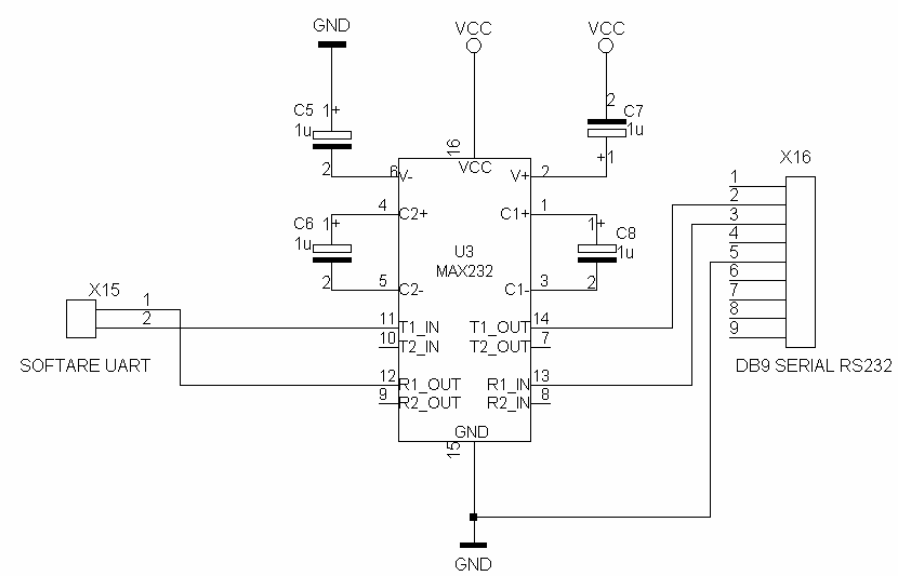
- Learning how to use the software UART

The previous examples used the hardware UART. That means the compiler uses the internal UART registers and internal hardware (portd.0 and portd.1) of the ATMega88. Sometimes we want to use more than one UART or we wish to use both USB and RS232. (USB is explained in chapter 3.8.2 and further.)

The Bascom compiler makes it easy to "create" additional UART's. Bascom creates software UART's that can be created on virtually every port pin.

Make sure switch "S1 USB/RS232" on the EDB is set to USB.
We will not use an USB interface now, but if the switch is set to USB the hardware UART pins portd.0 and portd.1 are disconnected from the MAX232 level shifter.

With S1 set to USB the level shifter is situated like this:



What we now have to do is connect the port pins we like to use as software UART to X15.

Connect the X15 pin that is closest to S1 to portc.1.
Connect the other pin of X15 to portc.2.

Note: if you use the bootloader to program your EDB, you will have to set switch S1 (USB/RS232) to RS232 while programming and you will have to set it back to USB to test the software UART. (Press the reset button after you switched to USB, that way you will not miss messages in the terminal window)

Open the EDBexperiment6c.bas file from the EDB-CD and program it into the ATMega88. Please read the remarks that can be found in the EDBexperiment6c.bas file they tell you how to initialize and use the software UART.

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 EDBexperiment6c.bas program asking for an alphanumerical input, and it should print the input back to the terminal.

3.3 Display output

Not always we want to use a PC to display something. Let's for instance say you just wanted to print only a couple of lines. For that purpose we have a way better solution, an LCD display.

3.3.1 HD44780 LCD

Experiment 7

Part list
1x HD44780 LCD
1x Computer with COM port

Goals
- Learning how to use an LCD display

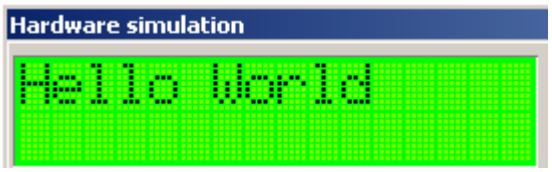
Open the EDBexperiment7.bas file from the EDB-CD and program it into the ATMega88. Please read the remarks that can be found in the EDBexperiment7.bas file they tell you how to initialize and use the LCD.

Make sure you have placed the LCD to connector X12 on the EDB.

After you placed and programmed your chip you should see the "Hello World" message again on your LCD display.

Also try the real hardware simulator for this program. Make sure you still have opened the EDBexperiment7.bas file. Click on  in Bascom.

Then click  in the AVR simulator window. Now click the play and "step-into-code" buttons. (a couple of times) You should the see:



Online resources:
Hitachi Semi is now Renesas
LCD manufacturer: http://www.renesas.com/
LCD background: http://home.iae.nl/users/pouweha/lcd/lcd.shtml

Exercise
1

Exercise

The best way of learning controllers is to do it yourself. Create a program that ask for "Your name" via the UART but then display the name on the LCD instead of on the computer screen. (Solution can be found on the EDB-CD Exercise1.bas)

Experiment
8

3.3.2 7 segment display

Part list

1x 7 Segment display
(Common Anode SL119-OS516HWA)
Conrad 146536 or equiv
8x Bread board wires

Goals

- Learning how to use an 7-segment display

There are 2 disadvantages if you use an LCD, first they can be expensive and second they are not as robust as the good old LED. If you only need to display one character. You can use a 7 segment display.

A 7 segment display is nothing more then some LED's, molded in to a plastic case in a way the LED's can form a character if you switch them on or off. They come in 2 types generally. Common anode and common cathode.

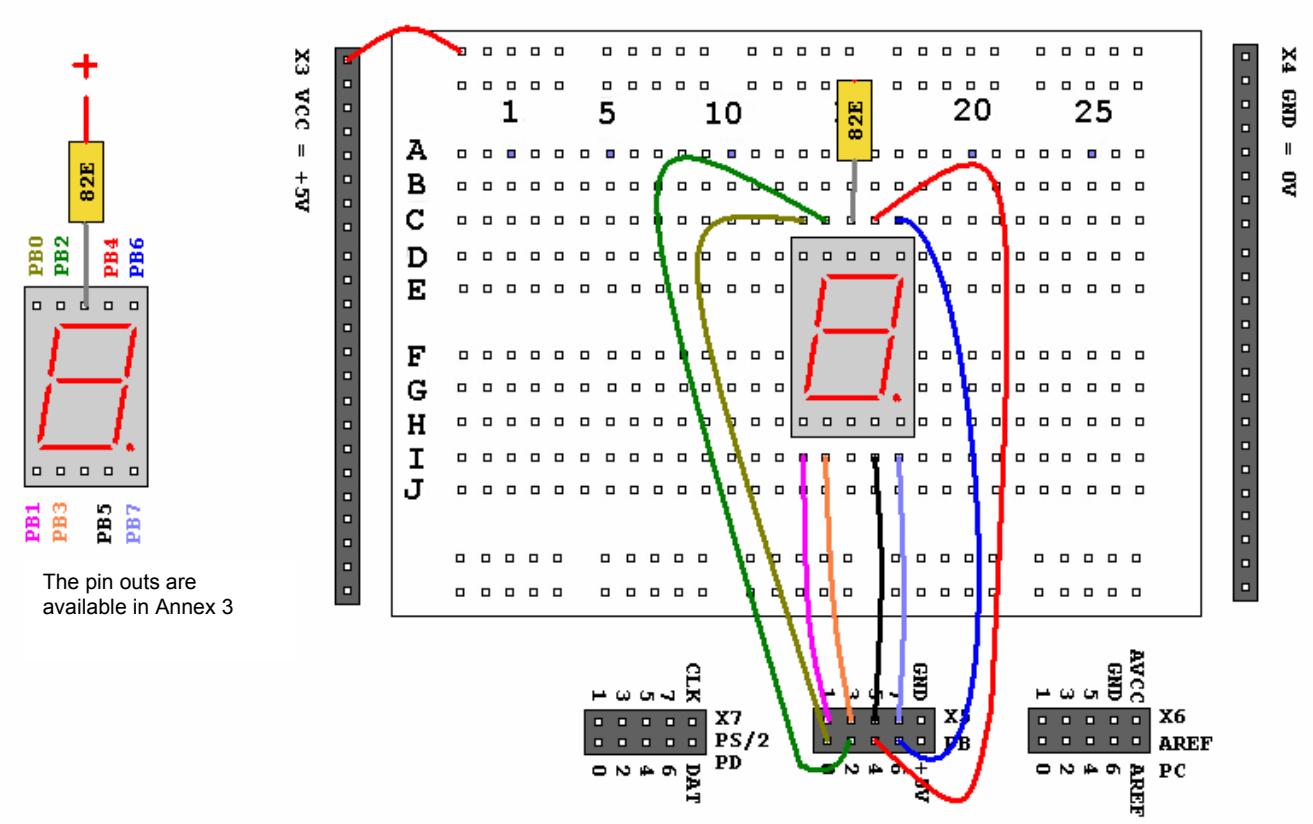
Common anode displays have a common POSITIVE pole, that means you will have to supply VCC to the common pin. And your controller needs to apply a 0V/GND signal to light a LED.

Common cathode displays have a common NEGATIVE pole, that means you will have to supply 0V/GND to the common pin. And your controller needs to apply a +5V/VCC signal to light a LED.

Common anode is used most because microcontrollers can switch more current to the ground then to VCC.

Online resources:
Application note for multiple 7 segment displays:
http://www.maxim-ic.com/appnotes.cfm/appnote_number/3210

Build the drawing below:



The pin outs are available in Annex 3

Open the EDBexperiment8.bas file from the EDB-CD and program it into the ATMega88. After programming you should see a hexadecimal counter on the display. (0....9 A....F)

Please read the remarks that can be found in the EDBexperiment8.bas file. Check if understand the Data and Read statements.

3.4 Keyboard input

It would be nice if we could connect a PS2 or AT keyboard directly to our microcontroller instead of connecting it to our PC. (and then read the keys thru the UART.) In this chapter you will see that interfacing a keyboard is easy with the EDB and Bascom.

3.4.1 PS2 or AT Keyboard

Experiment 9

Part list

1x HD44780 LCD
1x PS2 or AT keyboard
1x Optional a PC with COM port

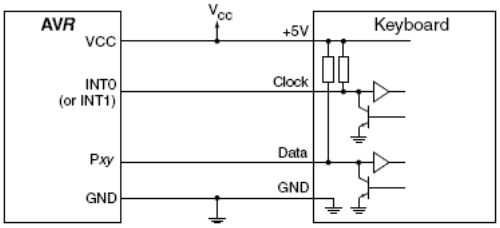
Goals

- Learning how to connect a PS2 or AT Keyboard

The EDB has already been provided with a standard PS2 interface. That means you can directly connect a standard PS2 keyboard to X10 (marked PS2) on the EDB. You can also connect an older AT keyboard, but if you decide to do so, you will need an adapter that converts your DIN42524 connector to mini DIN PS2.

The drawing below shows you the interface and the pin-outs of the DIN connectors that can be found on your keyboard.

Keyboard interface



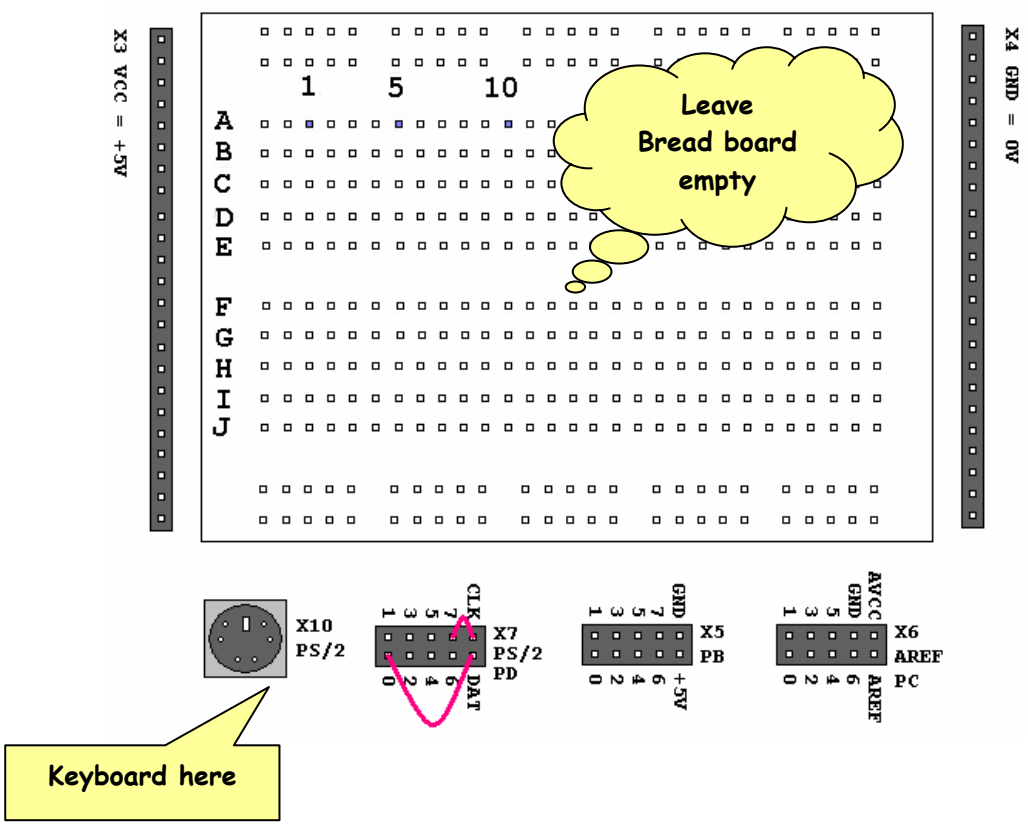
Pin-outs

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

Source: Atmel Application Note "AVR313: Interfacing the PC AT Keyboard"

Online resources:

Atmel Application notes:
http://www.atmel.com/dyn/products/app_notes.asp?family_id=607



Connect your keyboard, see the drawing above and connect the CLK to PD.7 and DAT to PD.0. Then open and program the EDBexperiment9.bas file.

Now type some text on your keyboard and press enter, your text should now show on the LCD.

Experiment
10

3.4.2 Matrix keyboard

Part list

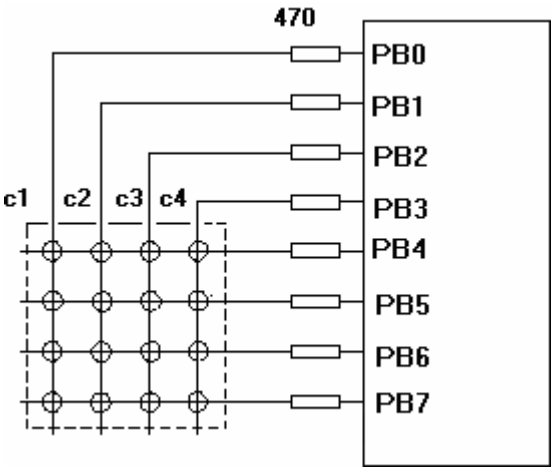
1x Matrix Keyboard
Conrad 709840
1x Computer with COM port
8x Resistor 470E

Goals

- Learning how to connect a matrix keyboard

Theory

The ATmega88 AVR is (at the time of writing this manual) on of the most recent devices in the AVR family. In addition to the regular AVR features it has more interrupt sources than its predecessors. As you will see throughout this experiment a matrix keyboard is a keyboard with 16 or 9 switches. Connected to the AVR like this:



Source: Bascom Help

As you can see there are only 8 I/O lines to detect 16 keys. There are 2 ways of checking whether a user has pressed a key. The first way is polling, that means we continuously let the software check for a key press. (A do...loop structure) The problem with this option is, that your controller can't perform much additional tasks while checking the matrix..

The second way is using interrupts. As said almost every pin on the ATMega88 can be selected as interrupt source. When using interrupts the "matrix read' routine is only executed after a user has pressed a key. (Typically the "matrix read' routine inside the interrupt routine is the same as when we used polling, only with the use of interrupts we will only execute the "read matrix" routine once per interrupt/key press.)

Brief general explanation of interrupts on the next page.

INTERRUPTS

An interrupt is an event that stops the execution of your program and jumps to a certain interrupt routine. Interrupts can be triggered from I/O pins but also the UART RxD pin and the Timers/Counters can generate interrupts.

After an interrupt is handled your program will continue at the statement that should have been executed if no interrupt was generated.

You can use interrupts if you wish to add priority to certain tasks in your system.

We will not give a suggested bread board set up for this experiment, because the matrix does not fit on the bread board. You will have to make the connections as you could see on the previous page.

Connect Pin1 of the matrix with a resistor of 470 Ohms to portb.0,
Connect Pin2 of the matrix with a resistor of 470 Ohms to portb.1, etc, etc...

Now program the ATMega88 with EDBexperiment10.bas and connect the serial cable between your EDB and your computer.

You should be able to read the value 16 in your terminal, if you press a key you should see the value change to a value < 16.

Read the remarks in the EDBexperiment10.bas file and see if you understand the Getkbd statement and the way we initialize and use the interrupts.

This experiment can only be carried out if you use Bascom version 1.11.8.1 or higher.

See also the EDBexperiment10polling.bas that also reads the matrix keyboard, but without interrupts. Check if you understand the difference between both ways.

If you use an older version of Bascom you can download a more recent demo version from www.mcselec.com or you can use the polling matrix example EDBexperiment10polling.bas.

Experiment
11

3.4.3 Remote control with RC5

Part list

1x TSOP1736 or equivalent
1x Computer with COM port
1x RC5 Remote control

Goals

- Learning how to use an RC5 remote control

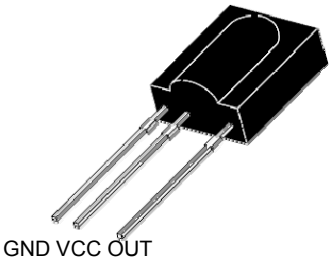
It can be discussed whether a "remote control" chapter belongs inside the "keyboard input" section. Anyway we can use a remote control as input device.

Nowadays every household has dozens of remote controls. Some of these remote controls use the RC5 protocol. The RC5 protocol is integrated in Bascom AVR. Therefore remote control functionality can easily be added to the EDB.

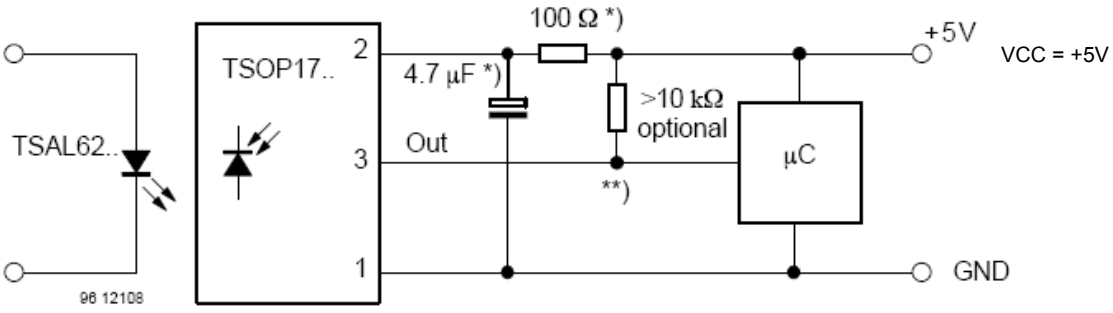
If you like to know more about the RC5 protocol, check this application note provided by Philips:

http://www.semiconductors.philips.com/acrobat_download/applicationnotes/AN10210_2.pdf

The TSOP1736 module is an infra red receiver module, this drawing shows a standard interface for the TSOP1736.



TSOP1736



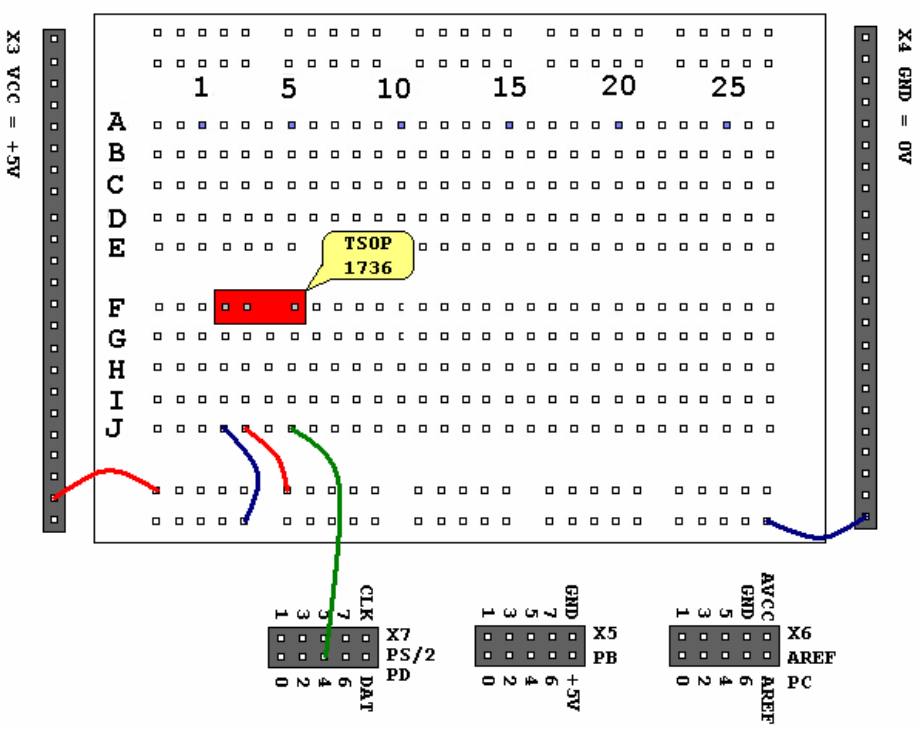
*) recommended to suppress power supply disturbances
**) The output voltage should not be hold continuously at a voltage below 3.3V by the external circuit.

Source: Vishay Semiconductors

(The TSAL diode is inside the remote control)

You can also use a different infra red receiver module (such as the SFH506 or the SFH 5110-36). If you use another one make sure to check the pin outs before connecting the module to the EDB.

A suggested bread board lay out for this experiment



What you now have to do is search for an RC5 remote control. Most Philips remote controls use the RC5 protocol but also most universal remote controls can use the RC5 protocol.

After you found an RC5 remote control or if you are in doubt, open en program the EDBexperiment11.bas file. Connect the serial cable between your EDB and your PC. Then aim with your remote control at the spherical side of the Infra Red receiver module. If you press a key on your remote, you should be able to see numeric values on your computer screen. (terminal window)

This experiment can only be carried out if you use Bascom version 1.11.8.1 or higher.

Read the remarks in the EDBexperiment11.bas file and see if you understand what happens.

Online resources:
Background on RC5: http://www.clearwater.com.au/rc5/
http://www.epanorama.net/links/irremote.html

3.5 Advanced I/O

We will now discuss some more advanced I/O questions that cannot easily been understood if we did not use the UART. (Hence the order of the chapters.) In chapter 3.5.1 we will first see that switches are not ideal, further on we will see how to detect the wiper position of a potentiometer and how to use an LDR to detect the dark. We will end this chapter generating tones with a speaker.

3.5.1 Counter with anti bounce input

Experiment 12

Part list

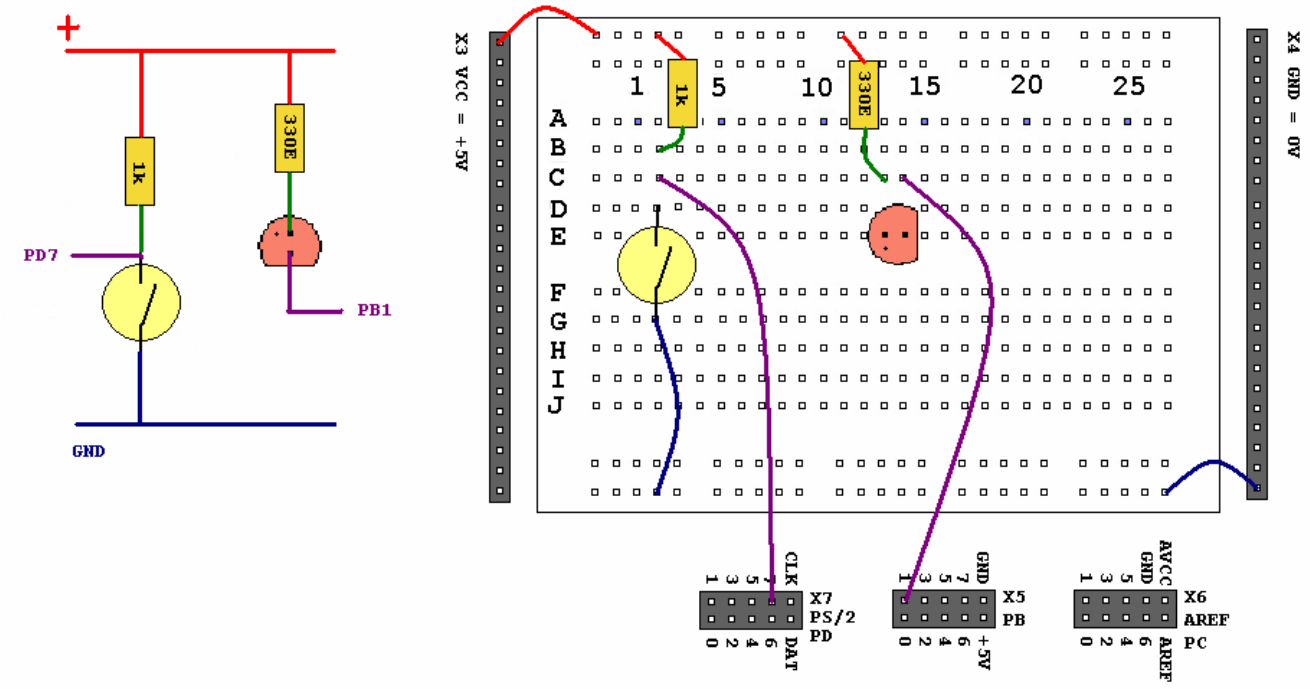
1x LED
1x Switch or a wire
6x Bread board wires
1x Resistor 330E, 1k

Goals

- Learning about non-ideal switches
- Anti bounce by use of software

Switches are not ideal. That means that if you connect a switch to a microcontroller as you did in experiment 1... you could get a non accurate program. That is not a big problem if you connect a LED, but it can be a problem if you want to make a counter.

Let's rebuild the schematic of experiment 1:

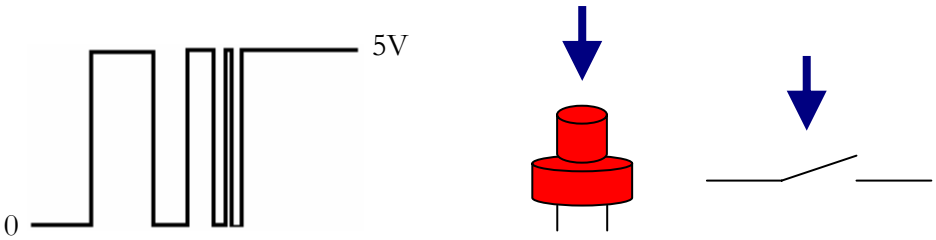


Program the ATMega88 with EDBexperiment12a.bas and connect the serial cable between your EDB and your computer.

The EDBexperiment12a.bas program "counts" the number of times you make contact with the switch. You can see the count in your terminal window. If you are lucky you will see the program count once every time you make contact.

Now replace the switch with a plain wire and then close and open the contact (wire) a couple of times and observe the counter. As you can see this gives an inadequate count. That is it counts correct but sometimes it counts multiple times for each time you make contact.

What happens is this; every time you make contact the contact will not directly be stable. In other words it ***bounces*** like this:



This is called ***bounce effect***. The contact will become stable in milliseconds, but the microcontroller runs on about 8 MHz giving it the possibility to detect falling and rising edges in times of microseconds. If you press the switch in the situation above, the controller would count 4 instead of one.

The solution is to create a ***dead time*** in which the controller does not notice changes in the voltage level. Creating dead time can easily be done by adding a "Waitms 100" just after the line where you ask the controller for the input.

You can now program the EDBExperiment12b.bas file into the microcontroller, check the remarks and take notice of the "Waitms 100" line. If your controller still counts inadequate you can change the value of the Waitms statement in 250.

Adding the "Waitms" statement to provide accurate input counts is called "software anti bounce protection"

Experiment
13

3.5.2 The GetRC Statement

Part list

1x Ceramic capacitor 100nF
1x Potentiometer 10k
7x Bread board wires

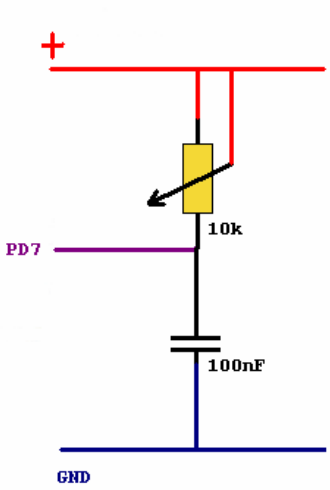
Goals

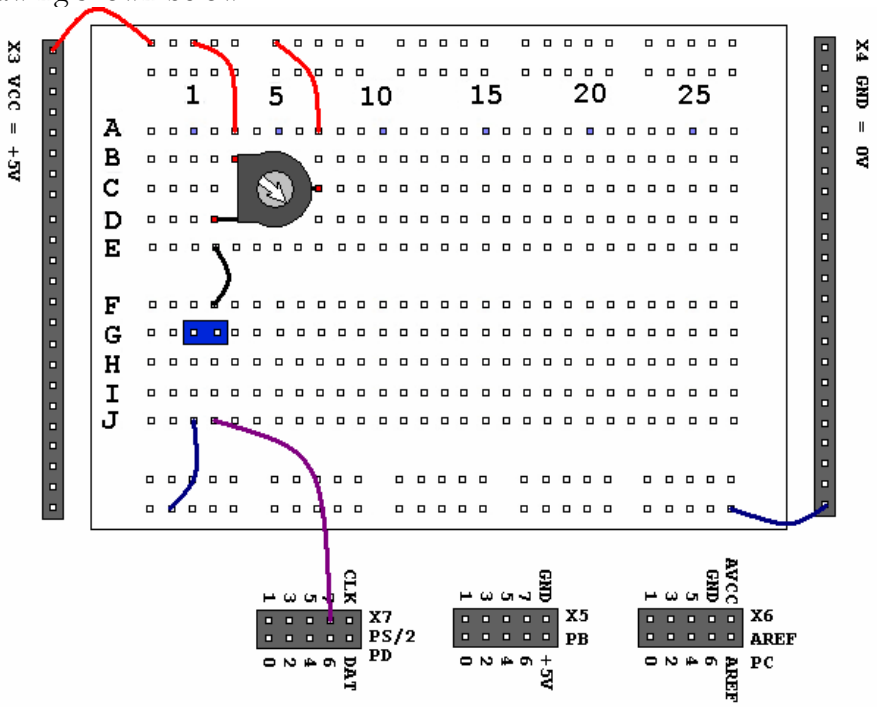
- Learning the GetRC statement

100 nF CAP
"104"

The GetRC statement in Bascom gives us the possibility to read a potentiometers wiper position. It can only be used if no accuracy is needed.

Build the drawing shown below:





The text on the capacitor shows the value in pico Farad. So 100 nF = 100000 pF. The last digit found on the capacitor is the number of '0' 's. (So 104 on the capacitor = 100000 pF = 100 nF)

The potentiometer does not fit on some bread boards, if it does not, you will have to solder 0,5mm wires to the potentiometer before you are able to connect it.

Program the ATMega88 with EDBexperiment13.bas and connect the serial cable between your EDB and your computer. You should be able to read values in your terminal, you should be able to change the values by turning the potentiometer. Read the remarks in the EDBexperiment13.bas file and see if you understand what happens.

Experiment
14

3.5.3 Siren with the SOUND statement

Part list

1x Speaker RS267-6968 or equiv.
1x Elco ≈ 100uF
1x Resistor 120E
7x Bread board wires

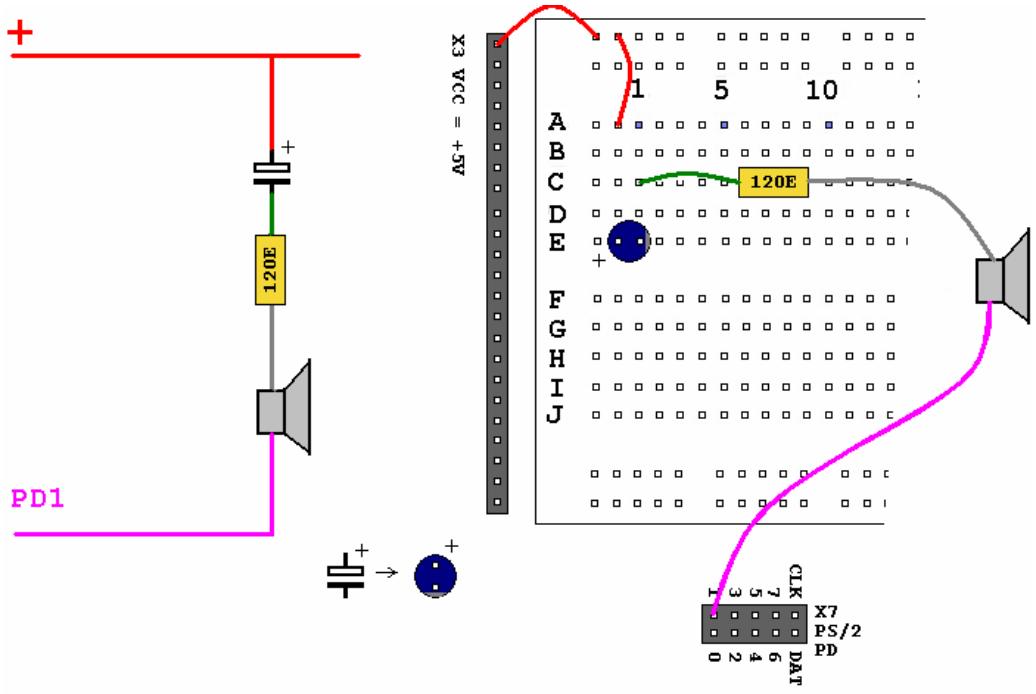
Goals

- Learning the SOUND statement

The SOUND statement gives you the possibility to generate various tones with a speaker. It's ideal if you have an application in which you want to prompt the user specific situations. (OK or Error beep etc.) The SOUND statement is not meant to generate accurate frequencies, use a timer (Chapter 3.7.3) if you wish to do that.

For this experiment you can use an ELCO of about 100uF (but 47uF, 220uF will work as well) You can use an old PC speaker or you can order RS 267-6968. In most cases the speaker cannot be placed on the bread board so you will have to solder a wire to the speaker. Use a massive copper wire of about 0,5 mm.

Build the drawing shown below:



Program the ATMega88 with EDBexperiment14.bas and see if you understand the program code. You should here a siren alarm.

3.6 AD and DA conversion

Although microcontrollers are digital, they can perform analog to digital conversion and digital to analog conversion. This gives you the possibility of measuring analog signals. But also to create "analog" outputs. This chapter learns you how to convert and create analog signals.

Experiment
15a

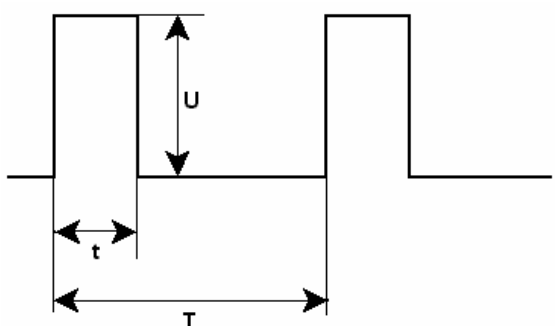
3.6.1 PWM output

Part list
1x MOSFET IRF520 or equiv.
1x Light bulb 6V
6x Bread board wires

Goals
- Learning about PWM

PWM is a form of digital output, however we can use it as analog output.

PWM signals are digital DC signals that look like this:



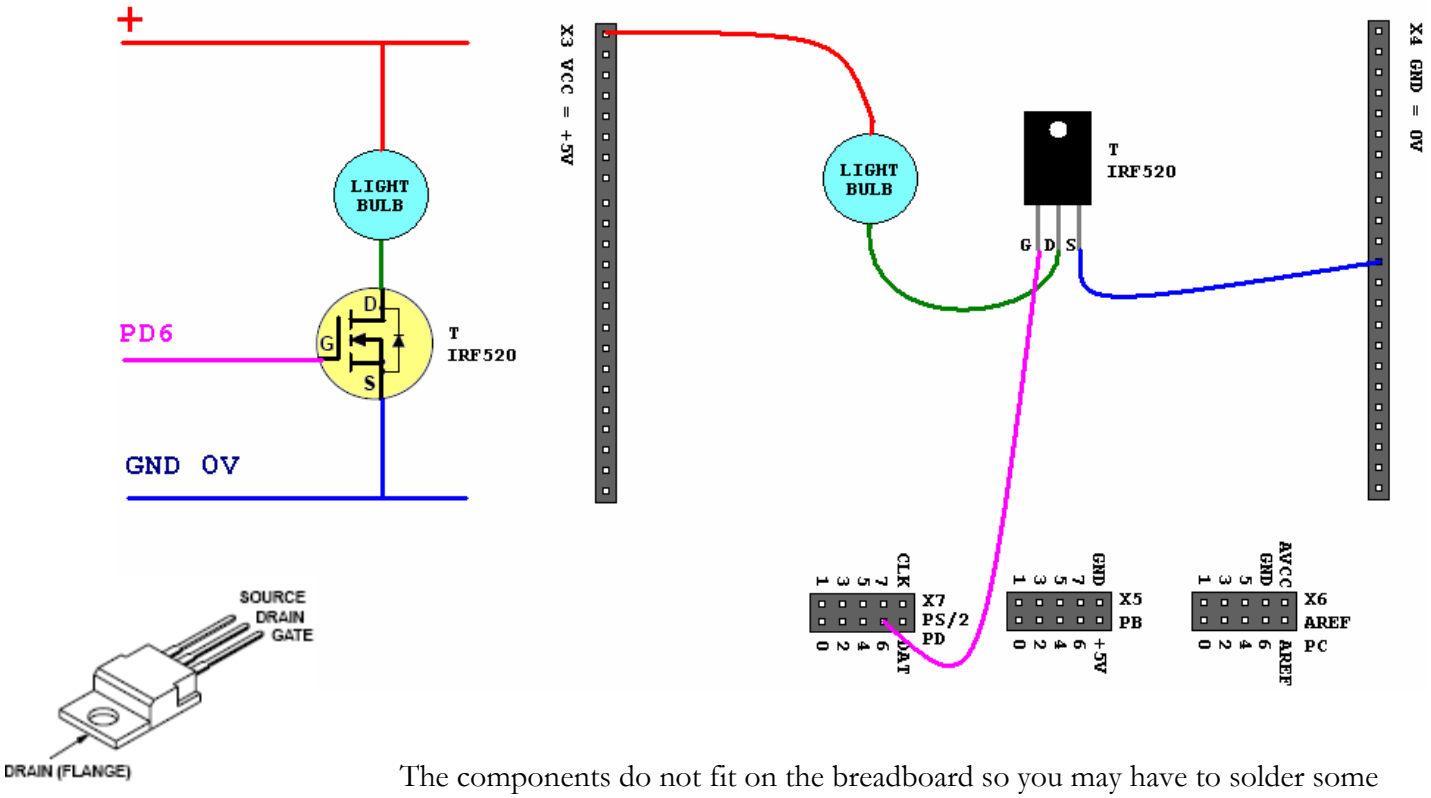
Imagine that we apply a signal like this to a light bulb. If we now increase time t the average voltage on the bulb will also increase. We choose time T so that we cannot see the light flashing. So it looks like if the bulb was controlled analog.

If you use something else then a light bulb, adding a capacitor to the PWM output is usually enough to create the analog signal.

The system described above is called Pulse Width Modulation (PWM). PWM is integrated on most AVR devices and it is supported by Bascom.

This experiment will show an example of PWM.

Build the drawing shown below:



The components do not fit on the breadboard so you may have to solder some wires.

Program the ATMega88 with EDBexperiment15a.bas and connect the serial cable between your EDB and your computer.

The light produced by the light bulb should vary and you should be able to read the PWM values in your terminal.

Read the remarks in the EDBexperiment15a.bas file and see if you understand what happens. The EDBexperiment15a.bas program uses a timer, we will tell you more about timers in chapter 3.7.3.

Online resources:

PWM Background: http://www.netrino.com/Publications/Glossary/PWM.html
http://www.4qdtec.com/pwm-01.html

IRF520: http://ec.irf.com/v6/en/US/adirect/ir?cmd=catProductDetailFrame&productID=IRF520

Experiment
15b

3.6.2 AD conversion with LDR

Part list

1x LDR NSL19-MS51
RS596-141
1x Computer with COM port
2x Resistor 2k2

Goals

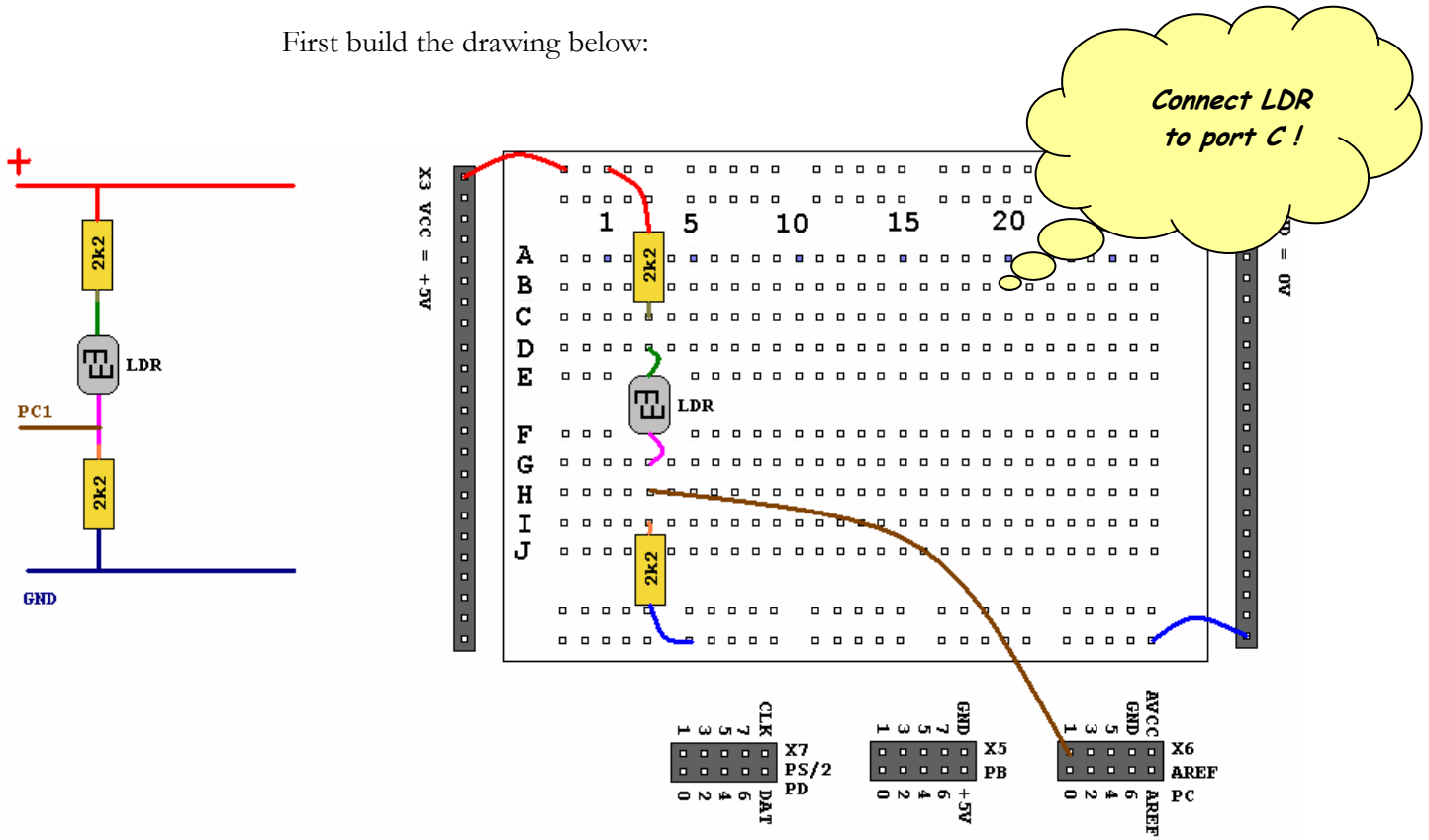
- Learning how to use the AD converter
- Learning about LDR's

This chapter shows an AD conversion application, and prepares you for exercise 2 "the LDR light switch".

LDR's are Light Dependant Resistors, that means that the resistor value of an LDR is influenced by the amount of light falling on the LDR. For this experiment most LDR's can be used.

The LDR NSL19-MS51 has a resistance of 5k in daylight (without bright sun). A resistance of 1M2 in twilight and a resistance of >35M in complete darkness. For this experiment most types of LDR's can be used. If you use another type you will read different values in your terminal.

First build the drawing below:



Program the ATMega88 with EDBexperiment15b.bas and connect the serial cable between your EDB and your computer.

You should be able to read values in your terminal, you should be able to change the values by holding you hand above the LDR or shine on it with a flashlight.

Read the remarks in the EDBexperiment15b.bas file and see if you understand what happens.

> Connect PC1 to ground and observe, then connect PC1 to VCC = +5V and observe.

Online resources:
LDR Background: http://www.radio-electronics.com/info/data/resistor/ldr/light_dependent_resistor.php

http://www.technologystudent.com/elec1/ldr1.htm

Exercise
2

Exercise 2, LDR light switch

Test your knowledge of the previous chapters by building this exercise.

Build an LDR Light switch,

- Connect a LED or a light-bulb (such as you can find in your bicycle) to your microcontroller. Make sure **not to connect it directly,** use a BS170 to switch the relative large current.

- If you have connected your bulb check if you can switch it on and off with a simple program such as you have seen in experiment 1.

- Now use the LDR from the previous chapter, and switch the light bulb on when it gets dark and switch it off when it is not dark. If you succeeded, congratulations!! If not, you may want to take a look at the solution of this exercise . It can be found on the EDB-CD filename: **Exercise2.bas**.

Experiment
16

3.6.3 Low cost voltmeter

Part list

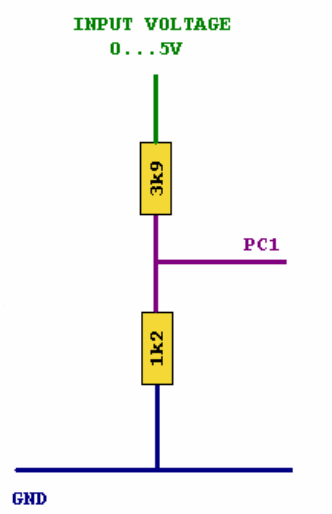
1x Computer with COM port
1x Resistor 1k2, 3k9
1x Potentiometer 10k
1x Voltmeter

Goals

- Learning how to use the AD converter
- Learning an AD Application

Theory

If you observed the AD values in your terminal in experiment 15, you could have found that a voltage of 0V gives an AD value of around 0 and a voltage of 5V gives a value of around 1024. Actually AD value 1024 is already reached at voltage level 1,1V.



Now let's build a voltmeter for voltages between 0 and about 5,0V with a 0,1V resolution.

Since we can only measure 1,1V we need to add a couple of resistors as voltage dividers. This way we let a voltage of 1V measured by the controller represent a 5V input signal.

The drawing on the left side of this page, shows which resistors you could use for the voltage divider.

The conclusion is that we have to divide 1024 thru 11 (1024/11) this equals 93,09. It is preferred to work with integers because microcontrollers cannot divide to well. (That is controllers can divide, but with remainder, 1024:11 = 93 remainder 1)

So we are going to round 93,09 off to 93. When we now *calculate* the AD values for each volt, we can find the translation table that you see on the right.

V	AD Value
1,1	1024
1	931
0,9	838
0,8	745
0,7	652
0,6	559
0,5	465
0,4	372
0,3	279
0,2	186
0,1	93
0	0

Now let's say that we connect a voltage to the port C.1 pin and the AD value is 190. 190 is not a value that is in the table... so we have to round off somewhere.

Let's say that all values below 93 are 0,0V.
Values from 93 to 185 are 0,1V,
values from 186 to 278 are 0,2V, etc, etc...

The only thing we then have to do is write a program that tests whether the AD value lays within a specific range.... like this:

```
Test_value = 93     '(The initial value "11" because everything below 11 is 0,0V)
AD_value = 190
Voltage = 00

Not_ready:
   If AD_value ≤ Test_value then goto ready
   Test_value = Test_value + 93
   Voltage = Voltage + 1
goto not_ready

Ready:
Print Voltage
```

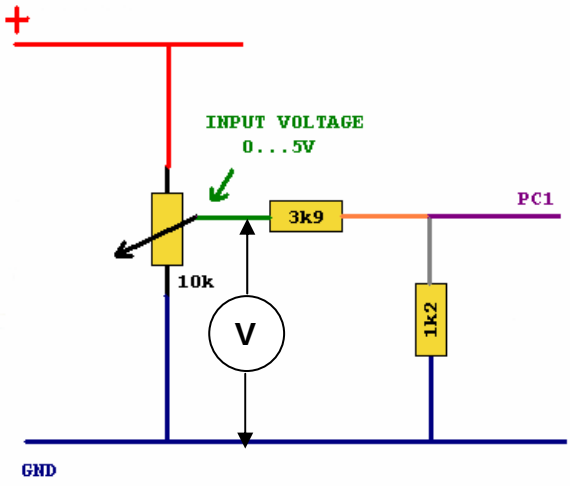
So in our example the AD_value was 190, the initial value for Test_value was 93. So the IF-statement is FALSE... so we add 93 to test value and 1 to the voltage. (We have now concluded that the measured voltage is not 0,0V.)

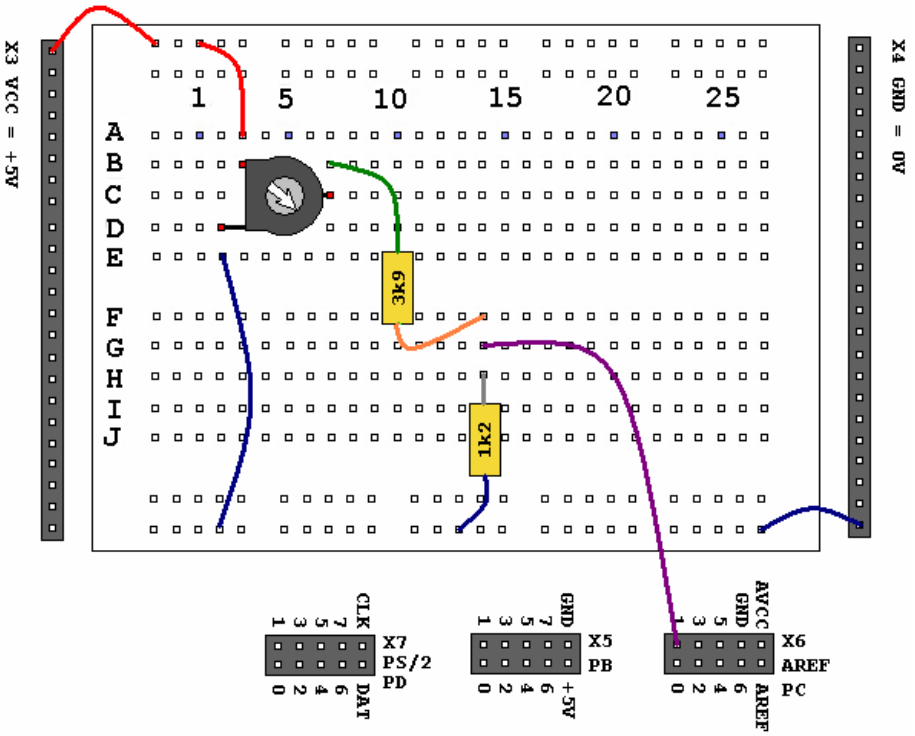
The Test_value is now 186 the IF is still false so the measure voltage is not 0,1V. So again we add 93 to test value and 1 to the voltage.

The Test_value is now 279 the IF is TRUE! We now jump to the Ready tag. If we Print the Voltage variable we would see 0,2V. We know have successfully interpreted the AD value without using division.

If we used the program as mentioned above, the readout would be 0 to 1,1 V. Dividing all values by 4 would give us a readout to the full scale.

Program EDBexperiment16.bas into the controller, connect the serial cable and build the drawing below. **Verify** if the controller works according to the theory above by turning the wiper of the potentiometer and measuring the voltage between GND and PC1 with a voltmeter.





Exercise
3

Exercise 3, Voltmeter with bar graph read out

Test your knowledge of the previous chapters by building this exercise.

Build a bar graph readout for the voltmeter of the previous experiment.

- Connect 8 LED's to Port B. If voltage = 5V all the LED's should be on, if the voltage is 2,5V the LED's on PD0...4 should be on etc, etc.

Hint:

```
Select Case AD_value
  Case Is < 128 : Portb = &B11111110
  Case 128 To 256 : Portb = &B11111100
  Case 256 To 384 : Portb = &B11111000
  Case 384 To 512 : Portb = &B11110000
  Case 512 To 640 : Portb = &B11100000
  Case 640 To 768 : Portb = &B11000000
  Case 768 To 896 : Portb = &B10000000
  Case Is > 896 : Portb = &B00000000
End Select
```

- Also connect an LCD and display the voltage.

If you succeeded, congratulations!! If not, you may want to take a look at the solution of this exercise . It can be found on the EDB-CD filename: **Exercise3.bas**.

Exercise
4

Exercise 4, A variable light source

Test your knowledge of the previous chapters by building this exercise.

Build a variable light source... more specific: you should be able to change the light produced by the bulb by turning the wiper of the potentiometer.

- Connect a light-bulb (such as you can find in your bicycle) to your microcontroller. Make sure **not to connect it directly,** use an IRF520 to switch the relative large current.

- If you have connected your bulb check if you can switch it on and off with a simple program such as you have seen in experiment 1.

- Use the PWM functionality to change the light produced by the light bulb.

- Use a potentiometer in combination with the A/D conversion seen in the previous experiment to regulate the light produced by the bulb.

If you succeeded, congratulations!! If not, you may want to take a look at the solution of this exercise . It can be found on the EDB-CD filename: **Exercise4.bas**.

3.7 Other controller features

In this chapter we will discuss some common features that are often used, but not seen in one of the other experiments. You will see how to use the memory, the watchdog and the controllers internal timers.

3.7.1 About the memory

Theory

Part list
- none-

Goals
- Learning to know the memory

The ATMega88 microcontroller has 3 kinds of memory.

- FLASH -> Bootloader FLASH
  Application FLASH
- RAM
- EEPROM

The FLASH memory (8 kilobytes) can ***only*** be programmed with an external programmer such as the STK200/300 dongle or the Wiazania USB STK programmer (chapter 4).

However there is one exception, the bootloader. When you use the bootloader option you divide the FLASH into 2 sections. The bootloader itself resides in the Bootloader FLASH and is programmed with the STK programmer. The Application FLASH can then be programmed with the serial cable (chapter 2.3).

In the Application FLASH we store the program/firmware we want to execute. If you don't use the bootloader the entire FLASH is available for the program code. The FLASH is ***non volatile*** that means that it will be preserved when the chips power is cut.

Where FLASH stores your program/instruction codes, ***RAM memory*** is used to store the inputs and results of arithmetic and logic operations. Also bytes that need to be send/received from the UART are first loaded to the RAM. Also the return address of JUMP instructions are stored by Bascom in RAM.

RAM (Random Access) memory is a form of temporary storage. After the operation ends you need to store or trash the data. RAM is ***volatile*** that means that it will be lost when the chips power is cut.

A special form of memory is the ***EEPROM memory*** (Electrically Erasable/ Programmable Read Only Memory) Don't let this term fool you, in the old days EEPROM could only be READ... it was then called ROM memory and it could NEVER be erased by the controller itself. You could only program the ROM by use of a ROM programmer and you could erase the chip with an UV light box. (Chips had a transparent window).

Nowadays the controller can ***electrically*** erase and program the EEPROM, giving you the possibility of storing data even when the power is cut. This is thus ***non volatile***.

EEPROM is especially handy when your application has user programmable settings. Imagine what would happen if you had to program your tv-channels every time you switched it on.... you don't have to, because there is an EEPROM in you TV.

Experiment 17

Connect the serial cable between your EDB and your computer, program the EDBexperiment17.bas file and start the Bascom Terminal.

This program will prompt you to enter an alphanumerical character, and it will store the data... follow the instructions on the screen and observe what happens after you unplug and re-apply the power source.

Did you see that the values of the EEPROM are preserved?

Online resources:
Atmel AVR family:
http://www.atmel.com/dyn/products/param_table.asp?family_id=607

3.7.2 Watchdog

Part list
- none-

Goals
- Learning to know the Watchdog

A watchdog timer is a safety feature that prevents runaway software. If your software runs correct you should reset the timer periodically.

Theory

Now if the controllers gets stuck inside a piece of meaningless code (after an electrical problem) or if there is external hardware that does no longer work, the watchdog timer will no longer be reset. When this happens the watchdog will overflow and reset your controller. The microcontroller will then restart and begin executing your code from the beginning.

A typical watchdog implementation looks like this:

```
Config Watchdog = 2048                    'reset after 2048 mSec
Start Watchdog                            'start the watchdog timer

  Do
  'Print "Hello"
  Reset Watchdog
  'Your program code goes here, repeat the Reset Watchdog statement
  Loop
End
```

If you want, you can experiment with the watchdog, you can find the Watchdog.bas file on the EDB-CD. Uncomment the "Print "Hello" line and comment the "reset watchdog" line. You will notice the reset on your terminal.

The maximum watchdog time for the ATMega88 = 8 seconds. But for most applications it is recommended not to use a time that long. Take 2048 or 1024 instead.

Experiment
18

3.7.3 Using a timer

Part list
-none-

Goals
- Learning how to use the timer

The ATMega88 has a build in timer. You can use it to measure time intervals or if you want to make your own wait routine. You will now see an example of a seconds timer.

Connect the serial cable between your EDB and your computer, program the EDBexperiment18.bas file and start the Bascom Terminal.

You will now see a seconds counter on the Terminal display. Read the comments in the EDBexperiment18.bas file and see if you understand what happens.

Note:
The EDBexperiment18.bas program is accurate for about 10 minutes, it uses the internal RC Oscillator of the ATMega88. If you want a more accurate time you better use a real time clock, such as the Maxim Dallas DS1307 Real Time Clock IC. The DS1307 has an I²C interface. You will learn how to use I²C in the next chapter.

OVERVIEW OF AVAILABLE TIMERS

The ATMega88 has these on board timers:

8-bit Timer/Counter0 with PWM
16-bit Timer/Counter1 with PWM
8-bit Timer/Counter2 with PWM and Asynchronous Operation

You can find how to initialize these timers and which interrupt sources they have in the Bascom help. Press F1 in the Bascom AVR IDE window and look for "CONFIG TIMER" in the index.

Experiment
19

3.8 Other interfaces

Beside the UART there are other interfaces that are commonly used. In this chapter you will see how to use the I²C protocol and we will see how we can use the USB module that is available for the EDB board.

3.8.1 The I²C bus

Theory

Part list

1x DS1624
2x Resistor 4k7
1x Computer with COM port

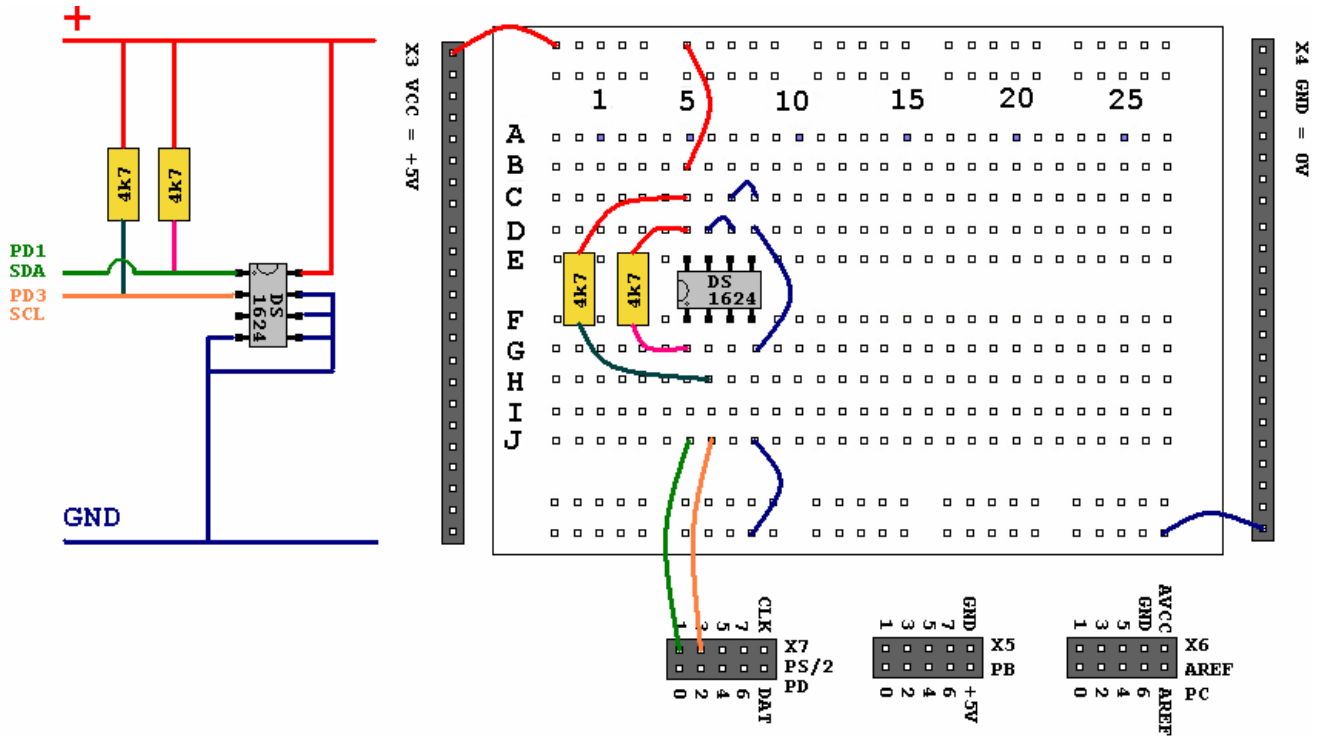
Goals

- Learning about the I²C bus

The DS1624 is a digital thermometer that measures temperatures from -55°C to +125°C. It is connected by use of an I²C bus. I²C is a serial two-wire protocol. The two wires are Serial DAta (SDA) and Serial CLock (SCL).

In I²C the microcontroller is called MASTER and the DS1624 is a slave. The master is the device that generates the SCL clock. You can connect lots of slaves but only one master.

Please build the drawing below:



If you wish to know more on the theory of the I²C bus, we like to suggest you read the DS1307 datasheet of Maxim/Dallas semiconductor. The DS1307 is not the same chip as we use in this experiment but it uses the same bus protocol. And the DS1307 datasheet is better if you wish to understand I²C.

The DS1307 datasheet can be found on the EDB-CD or can be downloaded from www.maxim-ic.com. Please begin reading from page 5 section "2–WIRE SERIAL DATA BUS" and do not read page 9 and further on. (Keep in mind that we use a different chip, so the principle is the same but the address- and command bytes are not the same.)

Program the ATMega88 with EDBexperiment19.bas and connect the serial cable between your EDB and your computer.

You should be able to read the temperature in your terminal.

Read the remarks in the EDBexperiment19.bas file and see if you understand what happens.

Online resources:
I2C Background:
http://www.semiconductors.philips.com/markets/mms/protocols/i2c/

I2C Interface Support in Windows:
http://www.microsoft.com/whdc/archive/i2c.mspx

3.8.2 USB Interface

Although most computers are equipped with USB ports these days, not often developers take the step to implement USB into their systems. This chapter will give you an introduction to USB development.

First of all you need to know that there are 3 options if you wish to design your own USB equipped hardware.

1. **USB device as virtual COM port**
   This is the easiest option if you want USB, you connect your *device* to the computers USB port and the operating system will treat your device as if it was a COM port.

   Advantages:
   - Easy
   - Compatible with terminal programs and existing software

   Disadvantages:
   - User sees the COM port which is not to professional.
   - COM port can be configured wrong.
   - User has to install 2 drivers.

2. **USB device with default VID&PID and driver**

   In general: Applications do not access USB ports, they access devices with a certain VID&PID (Vendor & part ID) on the bus.

   With this second option you can use the drivers provided by Linx Technologies (www.linxtechnologies.com) to access your device.

   Advantages:
   - Relatively easy in use

   Disadvantage:
   - You'll have to use the VID&PID that are registered by Linx.

3. USB device with your own VID&PID and driver

With this option you can actually create your own device. If you buy your VID from MCS Electronics you also will receive a software tool that writes the *. inf installation file for you. If you choose not to buy your VID thru MCS, take in account that you may have to write the *.inf file yourself.

Advantages:
- You can use your own VID&PID

Disadvantage:
- You will have to buy your VID from MCS or the USB consortium
- You need to write your own *.inf file.

USB devices are primarily used to interact with computers, therefore you almost always have to write computer applications in ***addition*** to your microcontroller code. That and the costs of the USB module (and optional VID) can form obstructions for the implementation of USB. It the future COM-ports are no longer available on computer systems and USB modules will become more affordable. That is the reason why we added this USB chapter to the EDB manual.

A comprehensive lecture of the USB bus lays beyond the scope of this manual. (www.usb.org/developers) We will describe how to connect the EDB as described in option one and two. We will also show some Visual Basic 6 code and Windows scripting examples.

The third option will not be described, we will only introduce a tool to write the USB-modules EEPROM to change the VID&PID.

Online resources:

USB module: http://www.linxtechnologies.com/
USB chip inside Linx module: http://www.ftdichip.com/Products/FT232BM.htm
USB general : www.usb.org
Get your own vendor ID: www.mcselec.com

3.8.3 Installation procedure

If you want to use option 1 (virtual com) please follow the installation as described in chapter 3.8.3.1. For the other options follow the instruction of chapter 3.8.3.2.

If you wish to switch from option 1 (virtual com) to option 2, 3. Or if you wish to switch from option 2, 3 to option 1 (virtual com) you first need to remove your old device driver. Driver uninstall instructions can be found in chapter 3.8.3.3

Make sure your computer has an operational USB port. USB ports will not work under Windows NT and some versions of Windows 95. We advise to use the same USB port every time you connect your EDB, this way you don't have to reinstall the drivers.

3.8.3.1 Installation procedure Virtual Com Port Windows

A type

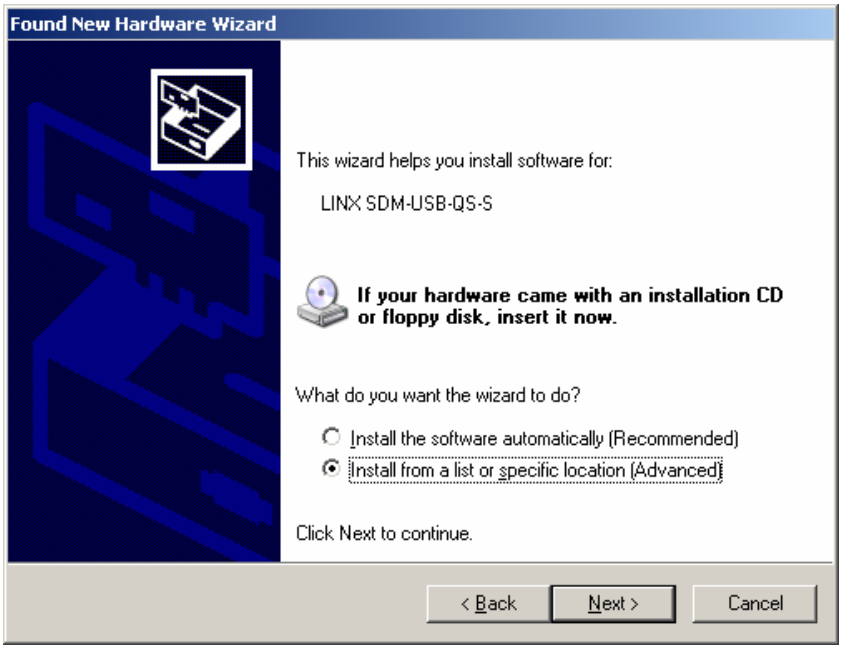
B type

Connect the power to your EDB and connect the USB-A to USB-B cable between the EDB and your computer.

You should then see this screen:

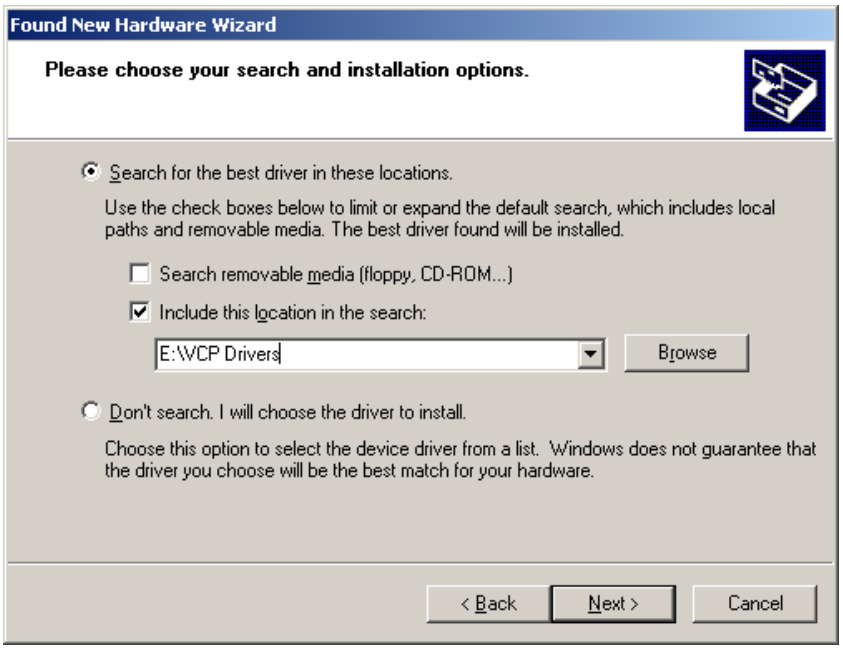
Select "No, not this time" and click "Next"

(After a moment) You should see this screen:



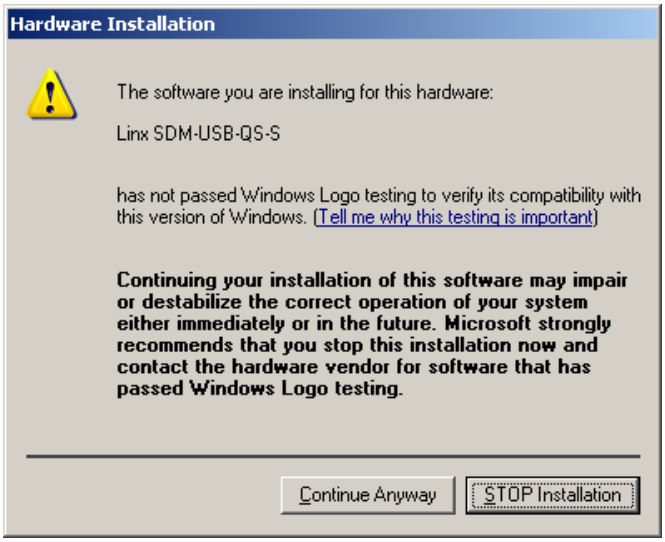
Select "Install from a list or specific location (Advanced)" and click "Next"

(After a moment) You should see this screen:



Insert the EDB-CD and use "Browse" to select folder
Your CDrom drive:\USB_Module\Virtual_COM_Drivers\ and click "Next"

(After a moment) You should see this screen:



This window is simply a warning that the driver has not gone through Microsoft's certification process and could potentially pose a problem for the system. The drivers provided for the USB module have been independently tested and should not pose any problems unless modified by the user. Thus click "Continue Anyway".

After a moment there will be a window with the message "The wizard has finished installing the software for Linx SDM-USB-QS-S" click finish.

Do not unplug the USB or power connector now, the installation is not finished yet.

Then the computer will again detect new hardware. Actually it now finds the Virtual COM port. To install the Virtual COM port you will have to redo all the steps mentioned on the previous pages. (begin reading from 3.8.3.1 from "Found new hardware wizard")

The Virtual COM port is now installed.

-> Continue reading in chapter 3.8.4

3.8.3.2 Installation procedure USB Device Windows

A type

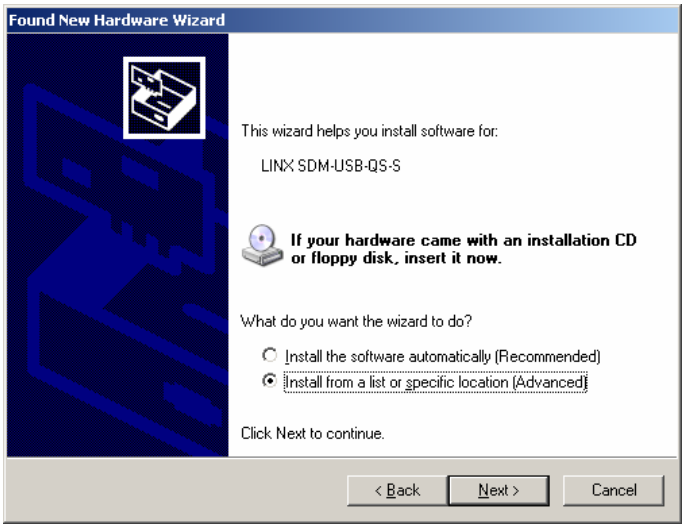
B type

Connect the power to your EDB and connect the USB-A to USB-B cable between the EDB and your computer.

You should then see this screen:

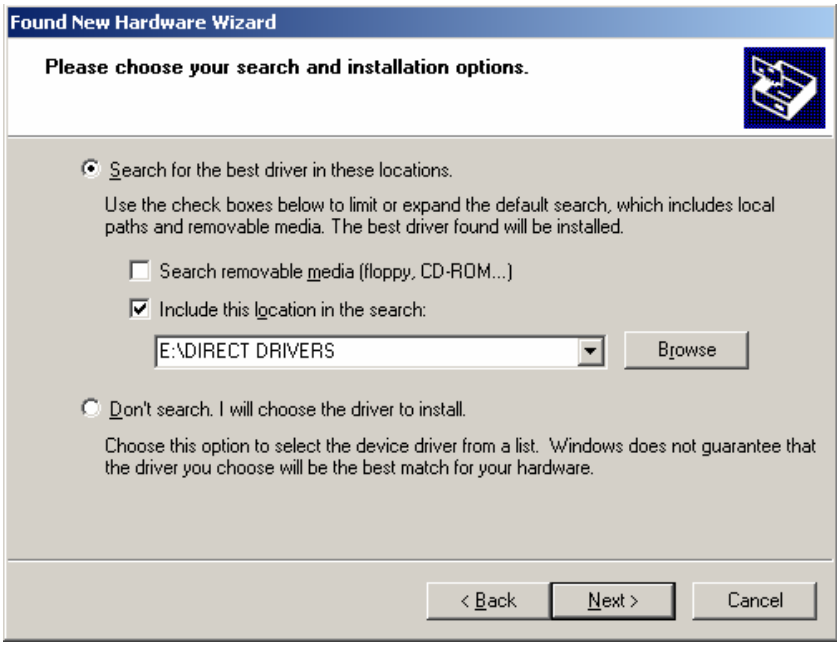
Select "No, not this time" and click "Next"

(After a moment) You should see this screen:



Select "Install from a list or specific location (Advanced)" and click "Next"

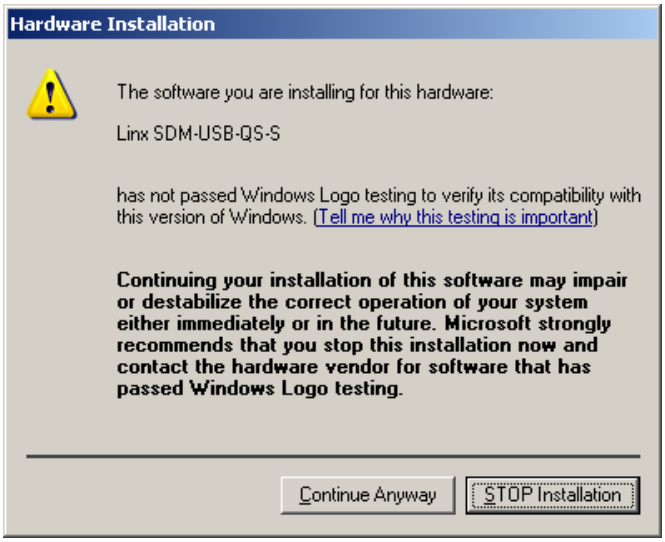
(After a moment) You should see this screen:



Insert the EDB-CD and use "Browse" to select folder
Your CDrom drive: \USB_Module\Direct_Drivers\ and click "Next"

If you have programmed your own VID&PID,
you have to specify your self written driver here.

(After a moment) You should see this screen:



This window is simply a warning that the driver has not gone through Microsoft's certification process and could potentially pose a problem for the system. The drivers provided for the USB module have been independently tested and should not pose any problems unless modified by the user. Thus click "Continue Anyway".

After a moment there will be a window with the message "The wizard has finished installing the software for Linx SDM-USB-QS-S" click finish.

The USB device is now installed.

-> Continue reading in chapter 3.8.5

3.8.3.3 Driver uninstall

If you wish to switch from option 1 (virtual com) to option 2, 3. Or if you wish to switch from option 2, 3 to option 1 (virtual com) you first need to remove your old device driver. This chapter tells you how to uninstall.

First insert your EDB-CD.

If you have installed the Virtual COM port driver.
Use your file explorer to navigate to your CD-rom drive.
Open the "VCP Drivers folder" and open "FTDIUNIN.EXE"

USB_Module\Virtual_COM_Drivers\FTDIUNIN.EXE

You should see this:

Click "Continue" and then after a moment "Finish". If you whish you can install the direct driver see chapter 3.8.3.2.

If you have installed the DIRECT driver.
Use your file explorer to navigate to your CD-rom drive.
Open the "Direct Drivers" and open "FTD2XXUN.EXE"

USB_Module\Direct_Drivers\FTD2XXUN.EXE You should see this:

Click "Continue" and then after a moment "Finish". If you whish you can install the direct driver see chapter 3.8.3.1.

Experiment
20

3.8.4 USB interface, device as Virtual COM port

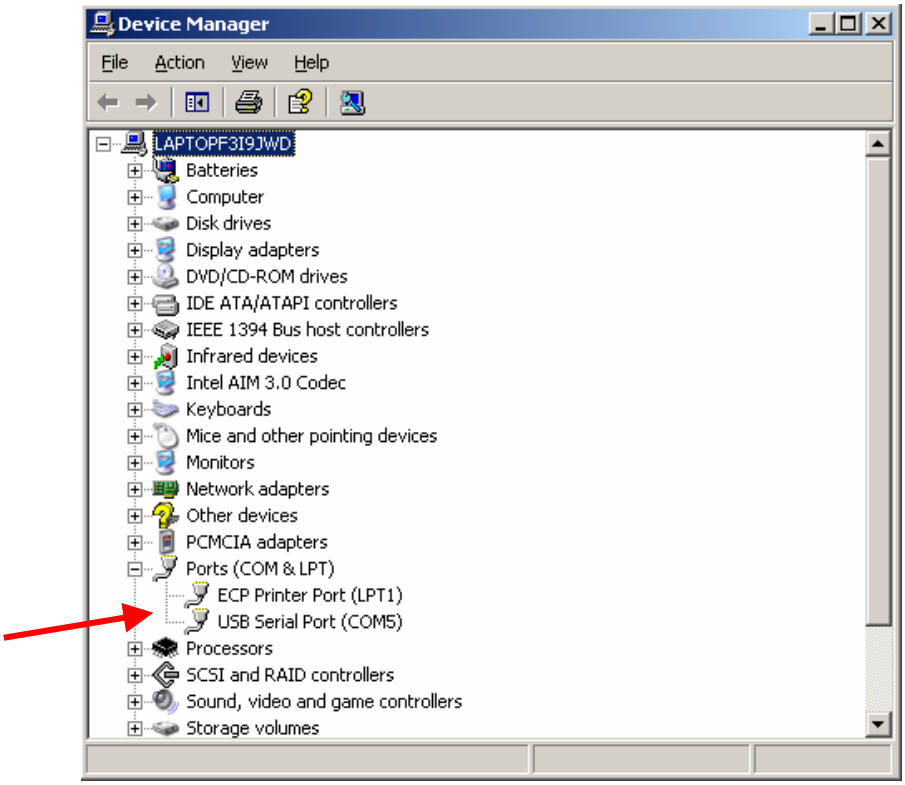
Part list
1x USB-A to USB-B cable
1x Computer with USB port

Goals
- Learning to interface a device as Virtual COM port

This experiment can only be carried out if you installed the drivers as described in chapter 3.8.3.1. **Make sure switch "S1 USB/RS232" is set to USB.**

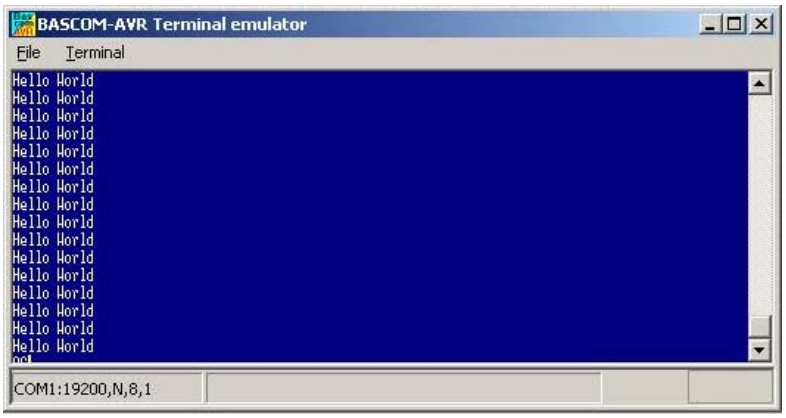
Open the EDBexperiment6a.bas file from the EDB-CD and program it into the ATMega88. We use the same program as in Experiment 6a to test the Virtual COM port. After you have programmed the controller and you connected the USB cable, open the terminal emulator by clicking on  in Bascom.

The settings should still be 19200 baud, no parity, 8 data bits, 1 stop bit
Make sure you use the appropriate (Virtual) COM port, you can find the appropriate COM port in the configuration panel. (System, Hardware Tab, and Device Manager)



In this case you should use COM5 but it is probably different in your system.

You should see the EDBexperiment6a.bas program printing "Hello World" to the screen of your PC like this:



It is possible to write your own application software for Windows/Linux etc. If you use Windows you can use Visual Basic (Basic as in Bascom AVR). You can also use a language that support other operating systems, for example C++ Borland Builder, Pascal or any other language that supports the use of ActiveX components.

We use an ActiveX (software-)component to send and receive data from the COM-port to an application in Windows/Linux etc. I suggest you use the royalty free OCX component (mcscomm.ocx) supplied by MCS Electronics. (Microsoft supplies mscomm32.ocx but the MS OCX does not work with events.)

You can find sample programs in the Visual Basic folder on the EDB-CD. The mcscomm.ocx file can also be found in that folder.

Experiment
21

3.8.5 USB interface, device with default PID&VID

Part list

1x USB-A to USB-B cable
1x Computer with USB port

Goals

- Learning to interface a device with default PID&VID

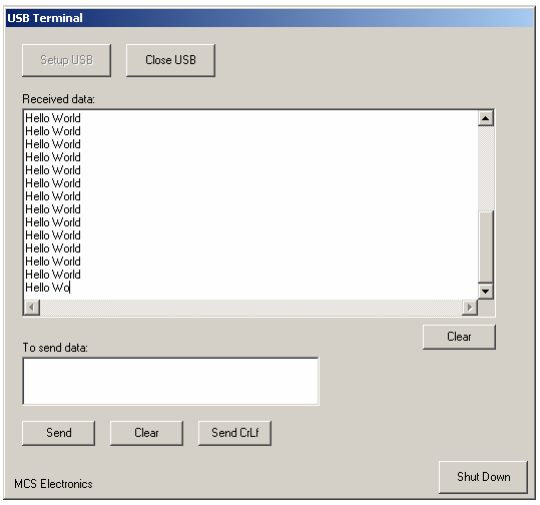
This experiment can only be carried out if you installed the drivers as described in chapter 3.8.3.2.

Make sure switch "S1 USB/RS232" on the EDB is set to USB.

Open the EDBexperiment6a.bas file from the EDB-CD and program it into the ATMega88. We use the same program as in Experiment 6a to test the USB interface default PID&VID.

Now open the "USB Terminal" program from the "Visual_Basic\USB_Terminal" folder on the EDB-CD.

Click "Setup USB", you should now see the program printing "Hello World" on the computer screen. Also try EDBexperiment6b.bas.



It is possible to write your own USB application software for Windows/Linux etc. If you use Windows you can use Visual Basic (Basic as in Bascom AVR). You can also use a language that support other operating systems, for example C++ Borland Builder, Pascal or any other language that supports the use of DLL components.

We use a DLL (software-)component to send and receive data from the USB-port to an application in Windows/Linux etc.

You can find sample programs in the Visual Basic folder on the EDB-CD. The DLL file can also be found in that folder. More samples in different languages can be found on the FTDI website http://www.ftdichip.com/Projects/CodeExamples.htm.

3.8.6 USB interface, device with your own PID&VID

This chapter is meant for advanced developers only, the chapter only gives a brief summary of the advanced possibilities. It is not a step-by-step guide which tells you all the steps you have to take to create your own PID device.

WARNING: Changing the EEPROM (PID&VID) of the USB module may cause the chip to be inaccessible. Changes made to EEPROM and drivers are at your own risk. MCS Electronics does not provide support for issues caused by changes in the EEPROM nor does it support for issues caused by modified device drivers.

EEPROM programming can only be carried out if you installed the drivers as described in chapter 3.8.3.2.

The USB interface is realized with the SDM-QS-S1-S USB module from Linx technologies which has an FT232xx chip from FTDI on board.

You can find the Linx website here: www.linxtechnologies.com
You can find the FTDI website here: www.ftdichip.com

You can read and write the EEPROM of the USB SDM-QS-S1-S USB module. This gives you the possibility of changing the modules PID&VID.

We strongly recommend you to visit the websites mentioned above to read the designers manuals before your do any EEPROM programming. You can find the EEPROM programming application in the "USB Module" folder on the EDB-CD but also on the manufacturers websites.

Before you can use your own PID device with Windows/Linux, etc. you first need to modify the *.inf device driver files. Read the *.inf files that can be found on the EDB-CD as example.

Online resources:
Get your own vendor ID: www.mcselec.com

Experiment
22

3.9 Motors

Besides human interfacing and data communications, microcontrollers also interface actuators. The most common actuators are DC and stepper motors. This chapter shows you how to use DC and stepper motor's.

3.9.1 Stepper motor

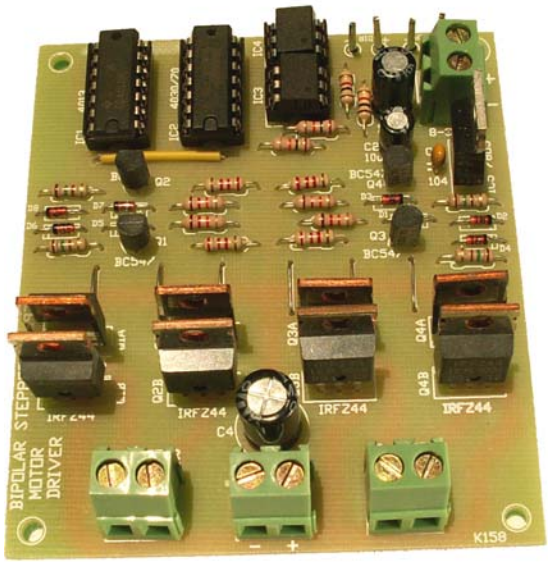
Part list

1x Bi-polar Stepper Motor
1x Quasar Stepper Motor Driver

Goals

- Learning to use a stepper motor

For this experiment we use the Quasar Bi-Polar Stepper Motor Driver 3158, more information can be found at the Quasar website www.quasarelectronics.com/kit-files/3000/3158.pdf.



If you haven't got a stepper motor you can use a stepper motor from a floppy disk drive. They are mostly not bi-polar so it is only good for experimenting (in combination with this driver), but not for use in a real application.

Connect the stepper motor to the "TO MOTOR" connectors on the driver.

Connect Portd.0 to the "STEP + " pin,
Connect Portd.1 to the "DIR +" pin,
Connect GND from the EDB to "STEP -" and "DIR-"

Connect 8...30 V to both power connectors on the driver. Start with a low voltage if you haven't got your motor's specifications. (galvanic isolation is possible, see the 3158 pdf file for details.)

If you used a floppy disk stepper motor, you might only have 3 wires to connect the motor. You then must leave one "TO MOTOR" connector pin free.

Program the ATMega88 with EDBexperiment22. You should see the motor running to one side and then back to the other.

Read the remarks in the EDBexperiment22.bas file and see if you understand what happens.

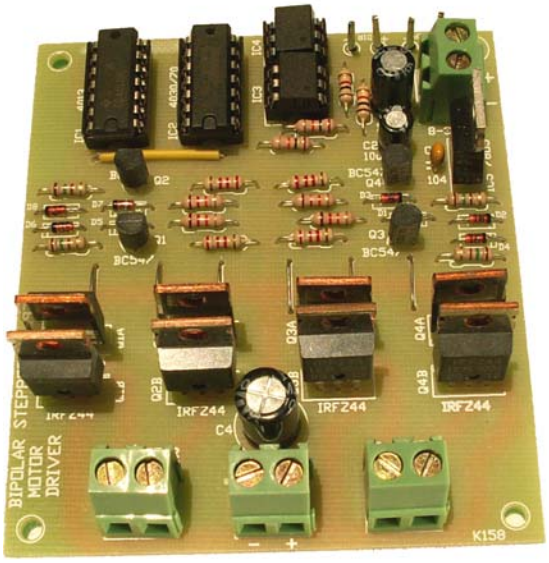
Experiment
23

3.9.2 PWM controlled DC motor

Part list
1x DC Motor
1x Quasar Stepper Motor Driver

Goals
- Learning to use a DC motor

For this experiment we use the Quasar Bi-Polar Stepper Motor Driver 3158, more information can be found at the Quasar website www.quasarelectronics.com/kit-files/3000/3158.pdf.



If you haven't got a DC motor you can use a DC motor from a tape deck..

Connect the DC motor to one of the to the "TO MOTOR" terminals on the driver.

Connect Portd.0 to the "STEP + " pin,
Connect Portd.1 to the "DIR +" pin,
Connect GND from the EDB to "STEP -" and "DIR-"

Connect 8...30 V to both power connectors on the driver. Start with a low voltage if you haven't got your motor's specifications. (galvanic isolation is possible, see the 3158 pdf file for details.)

Program the ATMega88 with EDBexperiment22. You should see the motor running to one side for a while and then back to the other side.

Read the remarks in the EDBexperiment22.bas file and see if you understand what happens.

Online resources:
Motor driver: http://www.quasarelectronics.com/motor_controllers_drivers.htm

Chapter

4

4. Other programming methods

This chapter explains how to program the ATMega88 using the parallel or the USB port.

You can also program the ATMega88 using the parallel or USB port. There are three reasons for doing this.

1. You don't have a serial port,
2. You want to program a blank/erased microcontroller,
3. You want to program the full memory space.
   This cannot be done with a boot loader since the boot loader itself uses memory space of the controller.

Chapter 4.3 describes how to reprogram the bootloader.

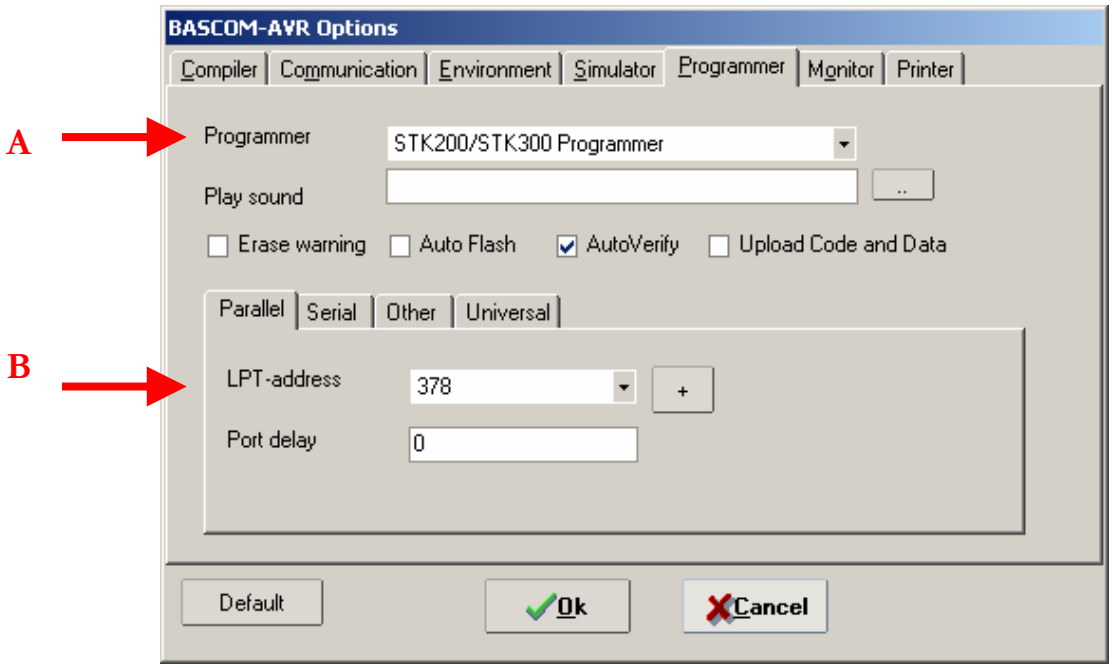
4.1 Programming with STK200/300 dongle

You can obtain an STK200/300 dongle from MCS Electronics, but you may also build your own. The schematics for the STK dongle can be found in annex 1.

- **Before you continue please verify that the parallel port BIOS setting is set to ECP mode.** The dongle does not work in some other modes for example the bi-directional mode.

Now connect the STK200/300 dongle to the parallel port of the computer and connect the flat cable of the STK200/300 dongle to connector X9 marked "ISP" on the Educational Development Board.

In Bascom click on "Options" and "Programmer" now you should see this screen:



- First select the "STK200/STK300 Programmer" from the drop down box (A).
- Then select your LPT-address. If you use LPT1 this is "378" by default and for LPT2 choose "278".
- Now press the OK-button.

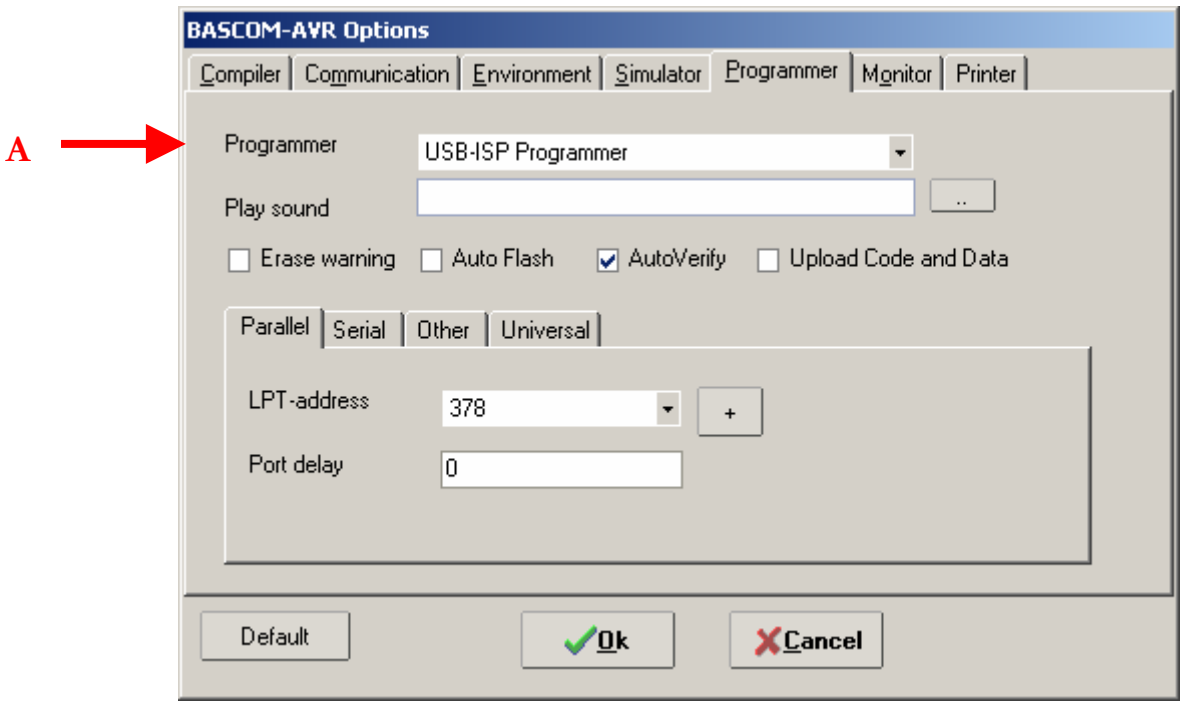
If you wish to reprogram the boot loader please continue to chapter 4.3 otherwise you can program your code as described in chapter 2.3 from the line **"Now let's program the flash"**.

4.2 Programming with Wiazania USP ISP

You can obtain a Wiazania USB ISP programmer dongle from MCS Electronics. The USB ISP programmer has the same possibilities as the STK200/300 dongle only uses it the USB port instead of a parallel port.

Connect the USP-ISP programmer to one of the USB ports of the PC, connect the flat cable of the USB-ISP programmer to connector X9 marked "ISP" on the Educational Development Board.

In Bascom click on "Options" and "Programmer" now you should see this screen:



- Select the "USB-ISP Programmer" from the drop down box (A).
- Now press the OK-button.

If you wish to reprogram the boot loader please continue to chapter 4.3 otherwise you can program your code as described in chapter 2.3 from the line **"Now let's program the flash"**.

4.3 Reprogramming the bootloader

Using the programming methods of chapter 4.1 or 4.2.

- Open the bootloader.bas file (EDB-CD) in Bascom AVR.
- Compile the boot loader (press F7)
- Make sure you use one of the STK dongles described in chapter 4.1 and 4.2 and program into chip (F4)
- Program the fuse bits of the ATMega88 as described hereunder.
- And finally select "MCS Bootloader" from programmers as described in chapter 2.3

Programming the fuse bits

The ATMega AVR family uses fuse bits to set advanced features of the chip. In normal condition they are only programmed once, although they can be programmed many times. You can program the fuse bits with the Bascom build in programmer.

You only need to follow the steps on this page if one of the conditions mentioned hereunder are true.

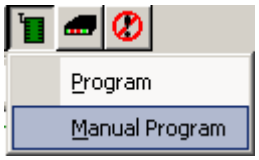
1. You are not using an ATMega88 provided by MCS electronics or you have changed the lock bits of the chip provided by MCS electronics.
2. Programming of the chip as described on the previous page did not work.

You need an STK200/300 dongle or Wiazania USB programmer to program the fuse bits.

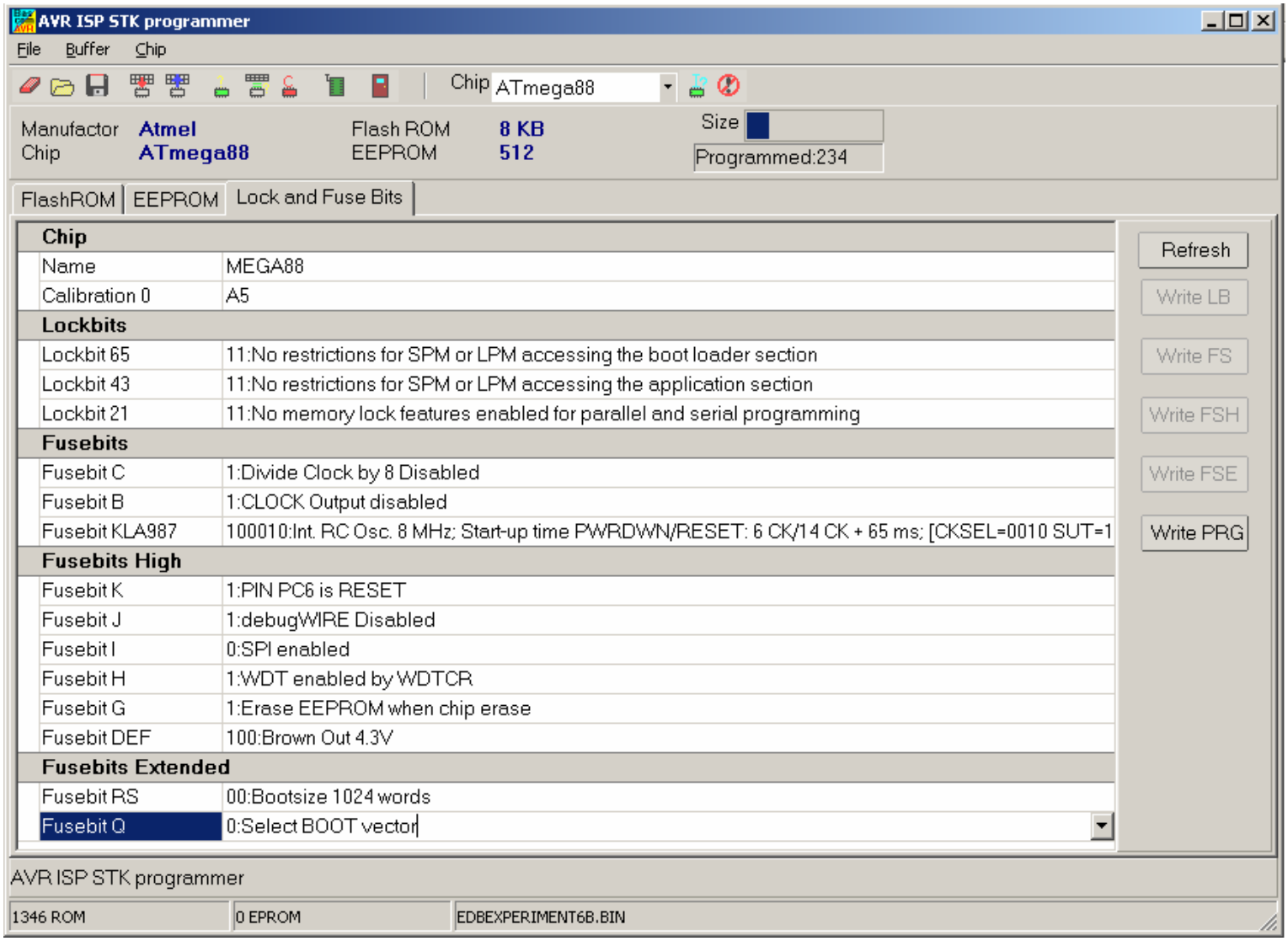
These steps will set the fuses back for use of the ATMega88 on the EDB.

Open the Bascom IDE, open a *.bas or an empty file.

Open the programmer like this:



Select the “Lock and Fuse Bits” tab and maximize the programmer window.



Make sure that the fuse and lock bits are set in the way you see above, press the “Refresh” button to see the actual settings. Fusebit Q should be set to BOOT.

Warning pay attention to Fusebit KLA987 which sets the oscillator options, it should be 100010: Int. RC Osc.... SUT = 1. If you change this option the device will no longer run and you cannot change it back without an external frequency oscillator. (frequency generator). Make sure you only set 100010.

You can write the bits with the “Write XXX” buttons that become available if you have changed an option (drop down box) and clicked an other fuse/lock bits section.

Atmel AVR In System Programming Dongle

Atmel offers a software package called the Atmel AVR ISP which allows the programming of AVR Microcontrollers in circuit with a simple dongle which is attached to the Parallel Port. This dongle is detailed below. It can be built cheaply, making it an ideal starting point for developing with ATMEL AVR micros.

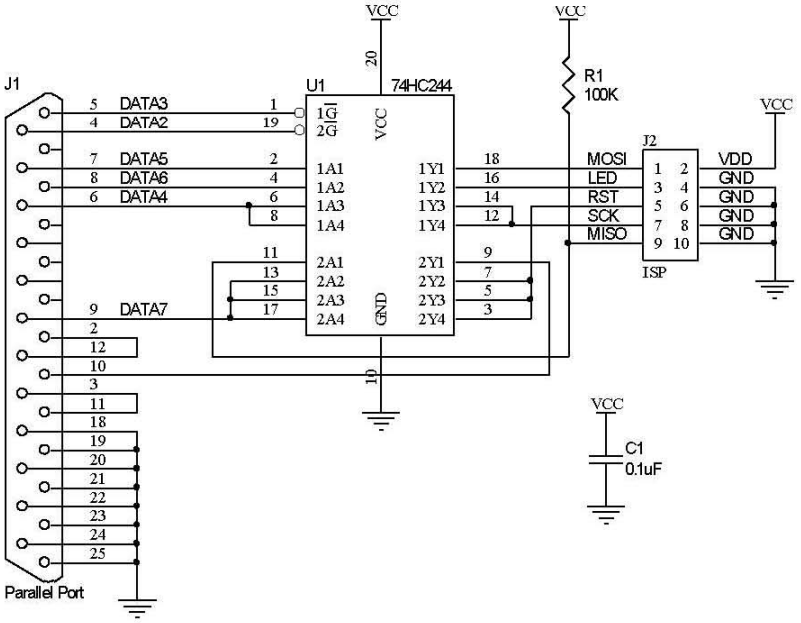


Figure 1 : Schematic for the ATMEL ISP Value Added Dongle

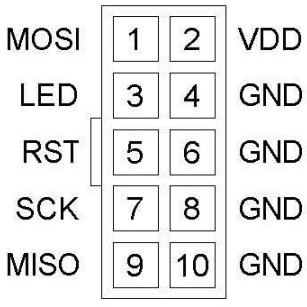
Using a 74HC244 Tri-State buffer as the main component, operation is extremely simple. The two loopback connections, pin 2 to 12 and 3 and 11 is used to identify the dongle. With both links in place the dongle is identified as a Value Added Pack Dongle. With only pins 2 and 12 links, it is reported as a STK300 or AVR ISP Dongle. With only 3 and 11 the dongle is reported as an STK200 or old Kanda ISP Dongle.

DATA2 and DATA3 of the Parallel Port Drive the TriState Outputs. A low will allow the passing of the serial clock and data during programming. MOSI, LED, SCK and Reset being outputs are buffered from the Parallel Port's DATA5, DATA6, DATA4 and DATA7 Respectively. The only input, MISO is fed into nACK, a status input of the Parallel Port.

There are two standard ISP Connectors for Atmel AVR Microcontroller ISP Programming. One standard is the 10 pin version using a DIL 5x5 header of 0.1" Pitch, shown in the above schematic. This is used on the ATMEL STK Kits. The other is a more compact 6 pin version, once again using a DIL 3x3 header of 0.1" pitch. This 6 pin version is the standard connector for ATMEL ISP Programmers.

The main advantage of the 10 pin header is the clean and easy use of 10 pin IDC crimp headers.

Name	Function	Description
MOSI	Master Out - Slave In	Data being transmitted to the part being programmed is sent on this pin
LED	Program LED	Optional Programming LED
RST	Target MCU Reset	Connects to Target AVR. Target AVR is programmed while in Reset State.
SCK	Shift Clock	Serial Clock Generated by the Programmer
MISO	Master In – Slave Out	Data received from the part being programmed is sent on this pin
VCC	ISP Power	Power Supply for the ISP. ISP Header must supply power to the dongle.
GND	Ground	Common Ground



10 Pin Header

Annex 2 ASCII Table

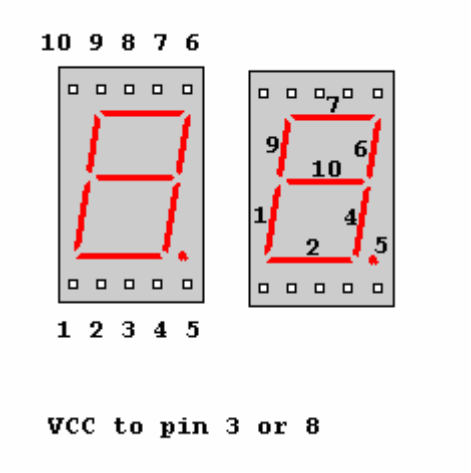
Decimal	Hex	Binary	Value	
000	000	00000000	NUL	(Null char.)
001	001	00000001	SOH	(Start of Header)
002	002	00000010	STX	(Start of Text)
003	003	00000011	ETX	(End of Text)
004	004	00000100	EOT	(End of Transmission)
005	005	00000101	ENQ	(Enquiry)
006	006	00000110	ACK	(Acknowledgment)
007	007	00000111	BEL	(Bell)
008	008	00001000	BS	(Backspace)
009	009	00001001	HT	(Horizontal Tab)
010	00A	00001010	LF	(Line Feed)
011	00B	00001011	VT	(Vertical Tab)
012	00C	00001100	FF	(Form Feed)
013	00D	00001101	CR	(Carriage Return)
014	00E	00001110	SO	(Shift Out)
015	00F	00001111	SI	(Shift In)
016	010	00010000	DLE	(Data Link Escape)
017	011	00010001	DC1 (XON)	(Device Control 1)
018	012	00010010	DC2	(Device Control 2)
019	013	00010011	DC3 (XOFF)	(Device Control 3)
020	014	00010100	DC4	(Device Control 4)
021	015	00010101	NAK	(Negative Acknowledgement)
022	016	00010110	SYN	(Synchronous Idle)
023	017	00010111	ETB	(End of Trans. Block)
024	018	00011000	CAN	(Cancel)
025	019	00011001	EM	(End of Medium)
026	01A	00011010	SUB	(Substitute)
027	01B	00011011	ESC	(Escape)
028	01C	00011100	FS	(File Separator)
029	01D	00011101	GS	(Group Separator)
030	01E	00011110	RS	(Request to Send)(Record Separator)
031	01F	00011111	US	(Unit Separator)
032	020	00100000	SP	(Space)
033	021	00100001	!	(exclamation mark)
034	022	00100010	"	(double quote)
035	023	00100011	#	(number sign)
036	024	00100100	$	(dollar sign)
037	025	00100101	%	(percent)
038	026	00100110	&	(ampersand)
039	027	00100111	'	(single quote)
040	028	00101000	(	(left/opening parenthesis)
041	029	00101001	)	(right/closing parenthesis)
042	02A	00101010	*	(asterisk)
043	02B	00101011	+	(plus)
044	02C	00101100	,	(comma)
045	02D	00101101	-	(minus or dash)
046	02E	00101110	.	(dot)
047	02F	00101111	/	(forward slash)
048	030	00110000	0	
049	031	00110001	1	
050	032	00110010	2	

Decimal	Hex	Binary	Value	
051	033	00110011	3	
052	034	00110100	4	
053	035	00110101	5	
054	036	00110110	6	
055	037	00110111	7	
056	038	00111000	8	
057	039	00111001	9	
058	03A	00111010	:	(colon)
059	03B	00111011	;	(semi-colon)
060	03C	00111100	<	(less than)
061	03D	00111101	=	(equal sign)
062	03E	00111110	>	(greater than)
063	03F	00111111	?	(question mark)
064	040	01000000	@	(AT symbol)
065	041	01000001	A	
066	042	01000010	B	
067	043	01000011	C	
068	044	01000100	D	
069	045	01000101	E	
070	046	01000110	F	
071	047	01000111	G	
072	048	01001000	H	
073	049	01001001	I	
074	04A	01001010	J	
075	04B	01001011	K	
076	04C	01001100	L	
077	04D	01001101	M	
078	04E	01001110	N	
079	04F	01001111	O	
080	050	01010000	P	
081	051	01010001	Q	
082	052	01010010	R	
083	053	01010011	S	
084	054	01010100	T	
085	055	01010101	U	
086	056	01010110	V	
087	057	01010111	W	
088	058	01011000	X	
089	059	01011001	Y	
090	05A	01011010	Z	
091	05B	01011011	[	(left/opening bracket)
092	05C	01011100	\	(back slash)
093	05D	01011101	]	(right/closing bracket)
094	05E	01011110	^	(caret/circumflex)
095	05F	01011111	_	(underscore)
096	060	01100000	`	
097	061	01100001	a	
098	062	01100010	b	
099	063	01100011	c	
100	064	01100100	d	
101	065	01100101	e	
102	066	01100110	f	
103	067	01100111	g	
104	068	01101000	h	

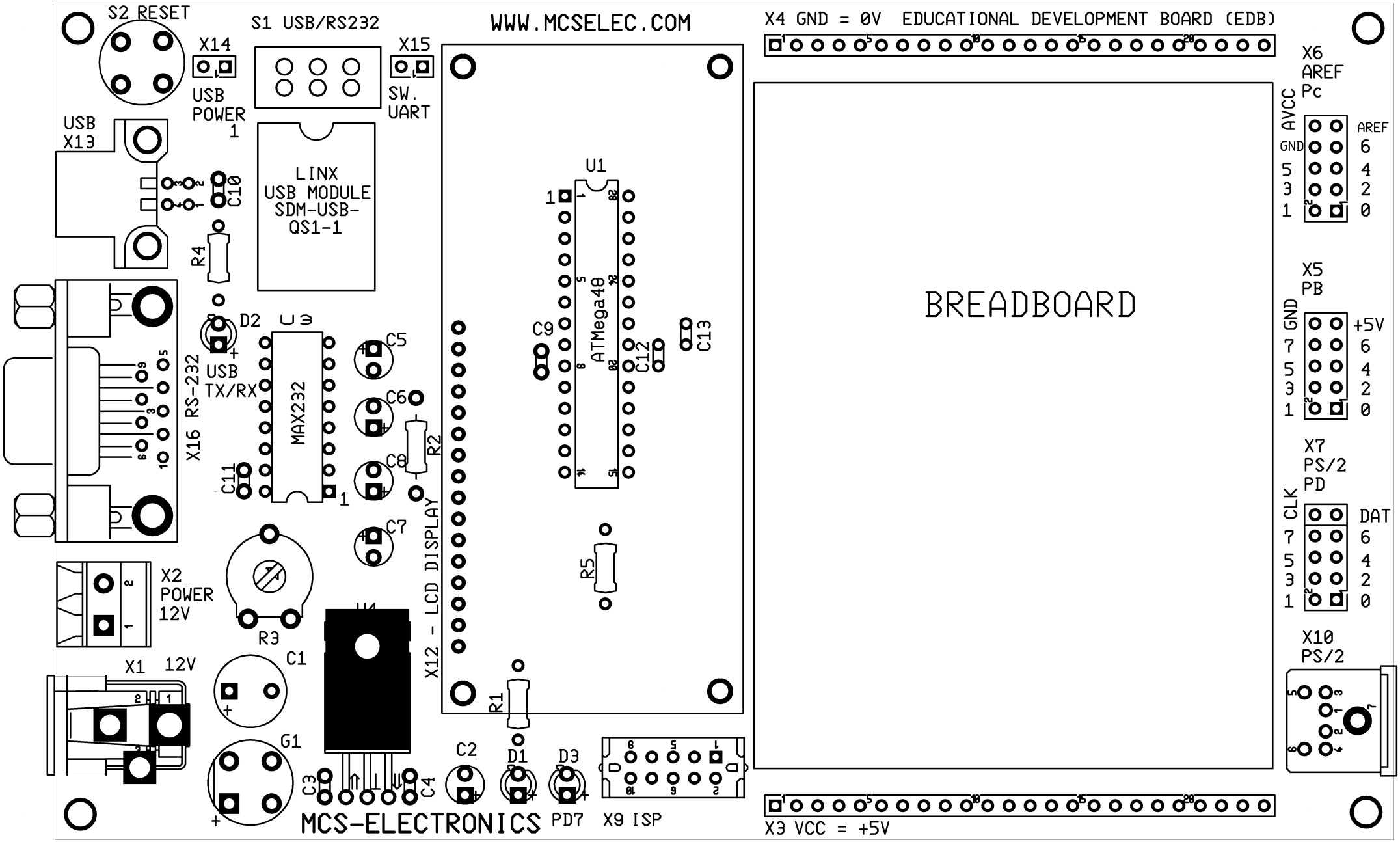
Decimal	Hex	Binary	Value	
105	069	01101001	i	
106	06A	01101010	j	
107	06B	01101011	k	
108	06C	01101100	l	
109	06D	01101101	m	
110	06E	01101110	n	
111	06F	01101111	o	
112	070	01110000	p	
113	071	01110001	q	
114	072	01110010	r	
115	073	01110011	s	
116	074	01110100	t	
117	075	01110101	u	
118	076	01110110	v	
119	077	01110111	w	
120	078	01111000	x	
121	079	01111001	y	
122	07A	01111010	z	
123	07B	01111011	{	(left/opening brace)
124	07C	01111100	|	(vertical bar)
125	07D	01111101	}	(right/closing brace)
126	07E	01111110	~	(tilde)
127	07F	01111111	DEL	(delete)

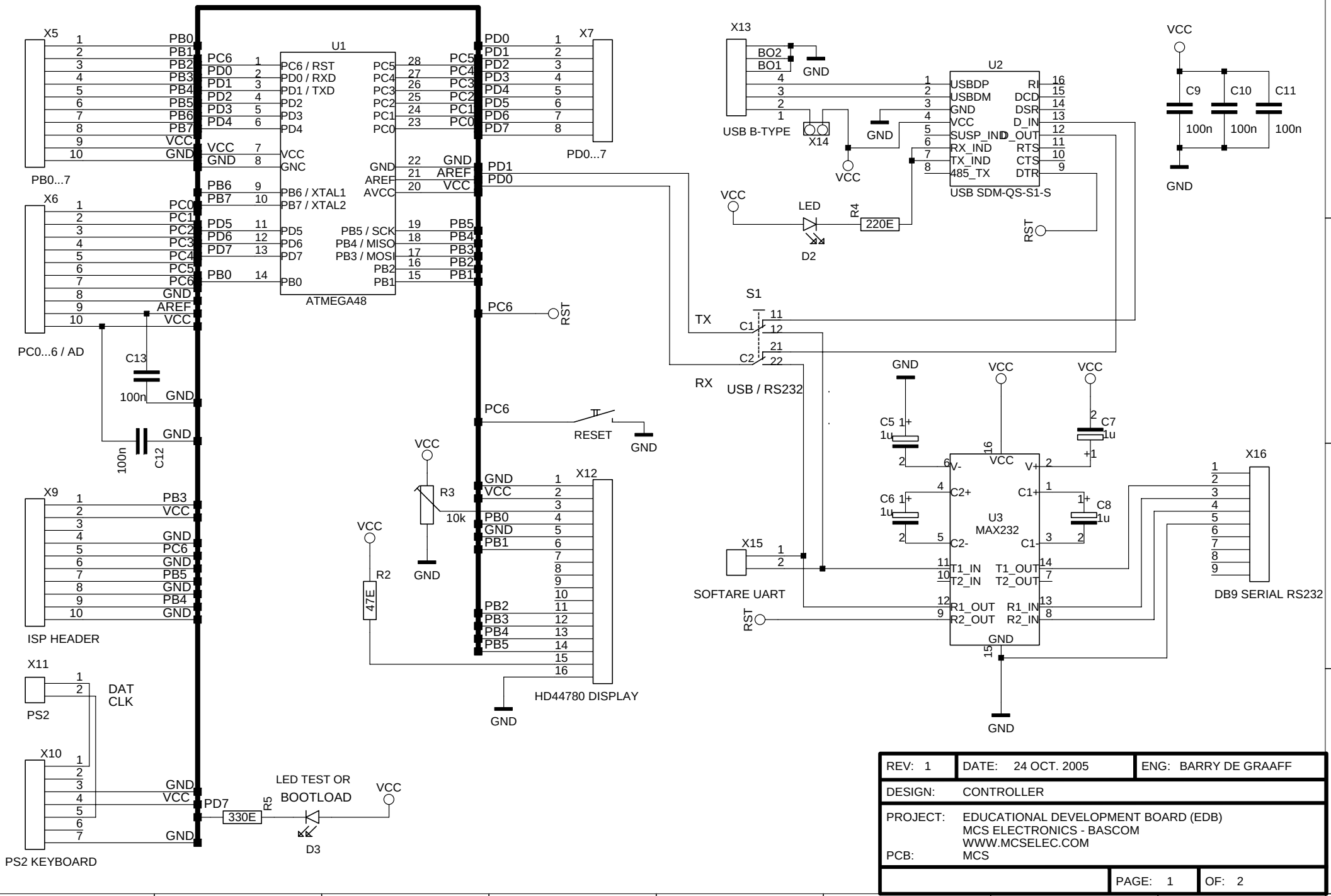
Annex 3 7 segment display pin outs

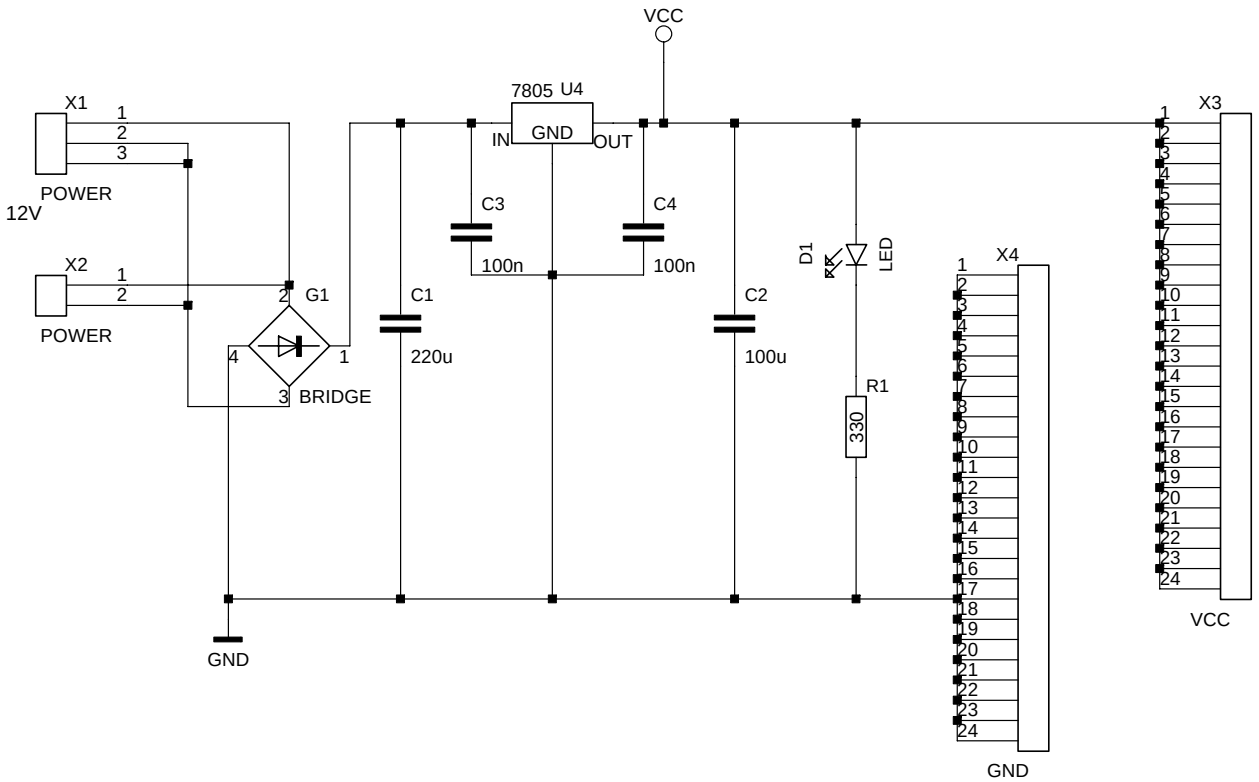
Pin out for the SL119 or OS516HWA 7 Segment display



Annex 4 EDB schematics


S2 RESET
S1 USB/RS232
WWW.MCSELEC.COM
X4 GND = 0V EDUCATIONAL DEVELOPMENT BOARD (EDB)
X14
X15
USB
POWER
1
SW.
UART
USB
X13
LINX
USB MODULE
SDM-USB-
QS1-1
U1
ATMega40
C10
R4
D2
USB
TX/RX
X16 RS-232
MAX232
C5
C6
C8
R2
C11
C9
C12
C13
X12 - LCD DISPLAY
R5
X2
POWER
12V
R3
C7
U4
X1 12V
C1
G1
C2
D1
D3
C3
C4
R1
MCS-ELECTRONICS
PD7
X9 ISP
X3 VCC = +5V
BREADBOARD
X6
AREF
Pc
AVCC
GND
AREF
X5
PB
GND
+5V
X7
PS/2
PD
CLK
DAT
X10
PS/2


X5
PB0...7
X6
PC0...6 / AD
X9
ISP HEADER
X11
PS2
DAT
CLK
X10
PS2 KEYBOARD
U1
ATMEGA48
PC6 / RST
PD0 / RXD
PD1 / TXD
PB6 / XTAL1
PB7 / XTAL2
PB5 / SCK
PB4 / MISO
PB3 / MOSI
X7
PD0...7
X12
HD44780 DISPLAY
X13
USB B-TYPE
X14
U2
USB SDM-QS-S1-S
USBDP
USBDM
SUSP_IND
RX_IND
TX_IND
485_TX
D2
LED
220E
S1
TX
RX
USB / RS232
X15
SOFTARE UART
U3
MAX232
X16
DB9 SERIAL RS232
RESET
R2
R3
10k
47E
R4
R5
330E
D3
LED TEST OR
BOOTLOAD
C1
C2
C5
C6
C7
C8
C9
C10
C11
C12
C13
100n
1u
VCC
GND
RST
REV: 1
DATE: 24 OCT. 2005
ENG: BARRY DE GRAAFF
DESIGN: CONTROLLER
PROJECT: EDUCATIONAL DEVELOPMENT BOARD (EDB)
MCS ELECTRONICS - BASCOM
WWW.MCSELEC.COM
PCB: MCS
PAGE: 1
OF: 2



REV: 1	DATE: 24 OCT. 2005	ENG: BARRY DE GRAAFF
DESIGN:	POWER	
PROJECT:	EDUCATIONAL DEVELOPMENT BOARD (EDB) MCS ELECTRONICS - BASCOM WWW.MCSELEC.COM	
PCB:	MCS	
	PAGE: 2	OF: 2

Copyright Notice

Disclaimer of warranties and liability