1TE663/723 – Lab exercise 2 Microcontroller Programming

version November 2, 2016

1 Introduction

This lab exercise is meant to give you hands-on experience with the AVR microcontrollers. Previously the very first lab exercise using the AVR controllers was based around the STK500 development board by Atmel. These boards are still around for those who want to try out the controllers in a *more controlled* environment first. However, since the STK500 boards need a serial port on the host computer which is not available anymore on a standard laptop computer, I rewrote the lab instructions to be used with an AVR controller on a breadboard together with an USB flash programmer.

2 Before getting started

Even though I like the freedom and stability of Linux-based operating systems, I still prefer Windows (7, definitely not 8!) as my daily operating system.

☞ If you are using your own computer, you need to install some software first:

2.1 MacOS

Since I don't have any Mac myself to test any of these, I can only refer to other peoples' experiences. What you need is

- **CrossPack** – this package contains the compiler, the standard libraries and the uploader *avrdude* which you need to transfer the compiled code into your microcontroller. `http://www.obdev.at/products/crosspack/index.html`
- a plain text editor, preferably with syntax highlighting for C-source code.
- you **don't** need a separate hardware driver for the USB-programmer.

2.2 Linux

This is also not tested by myself… You should find all the following packages readily available in your Linux installation's package managers.

- **binutils**: normally already installed tools like the assembler, linker, etc.
- **gcc-avr**: the GNU C compiler (cross-compiler for avr).
- **avr-libc**: the basic standard libraries for the C compiler.
- **avrdude**: the software to transfer your compiled code into the microcontroller.

Some additional configuration steps might be necessary in order to get the USB programmer to work correctly. There is a description of a possible conflict here: `http://stackoverflow.com/questions/5412727`

The following lines describe our programmer:

```
# USBasp
ATTR{idVendor}=="16c0", ATTR{idProduct}=="05dc", MODE="660", GROUP="dialout"
```

Furthermore the user account needs to be member of the group `dialout`:

```
usermod -a -G dialout <user>
```

2.3 Windows

- **Atmel Studio** – I recommend to use version 7 (or newer) which you can download from `http://www.atmel.com/Microsite/atmel-studio/`.

  Atmel studio now includes the compiler binaries and libraries for compiling AVR source code under the Windows operating system. Transparently for the user it runs precompiler, compiler and linker, but it can also be used to create and run *Makefiles*.

  This is a Windows application, I am not sure about the integration in other operating systems like Linux or MacOS. If you don't want or cannot use *Atmel Studio* then you'll need a text editor for your source code and the compiler toolchain *avr-gcc*, which you can find under `http://gcc.gnu.org/`.

  If you prefer **Eclipse** then you might this link helpful: `http://www.protostack.com/blog/2010/12/avr-eclipse-environment-on-windows/`

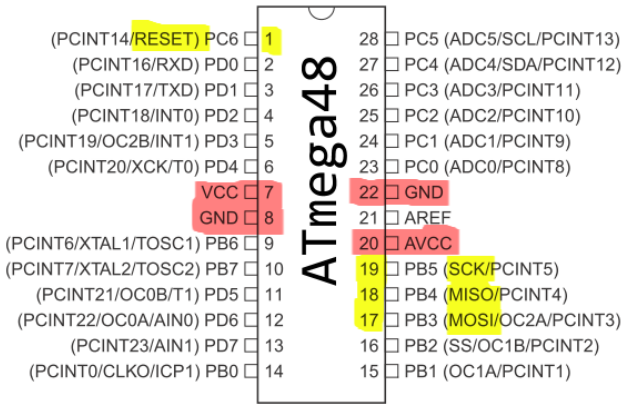
- the driver software for the USB programmer – you will have access to a USB programmer compatible with usbasp `http://www.fischl.de/usbasp/`. For Linux and MacOS no additional driver is needed since the USB functions can direcly be addressed from the application software. On Windows however you will need to install an additional driver which you will find on the above homepage and on Studentportalen.

- the programming tool **avrdude** - this is a command line application which will transfer the compiled code via the USB programmer into the flash-memory of the AVR controller. You can download precompiled binaries for Windows, documentation and source code (for compilation under Linux and MacOS) here: `http://download.savannah.gnu.org/releases/avrdude/`

- optionally you can also install one of many graphical frontends for the command line tool - personally I use a program called AVRDUDESS: `http://blog.zakkemble.co.uk/avrdudess-a-gui-for-avrdude/`.

3 Getting started

☞ Start by putting the ATmega328 controller onto your breadboard. For a new device you sometimes may need to bend the legs slightly inward in order to nicely fit into the holes in the breadboard.



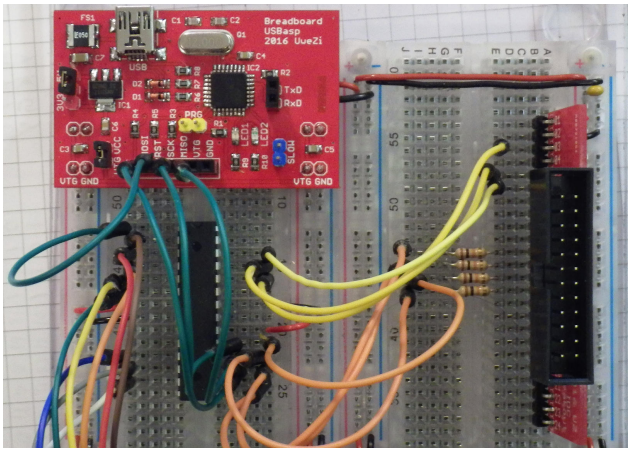
Pinout of the ATmega328 taken from the datasheet. In order to build our first circuit around this chip, we need to consider the highlighted pins.

A factory-new Atmega328 is configured to run with an internal clock frequency of 1 MHz. Though we might change this later during the course and for your projects, for now this setting is fine. We will need to slow down operation anyway in order to be able to observe what is happening. Therefore we will also ignore the connection pins for an external crystal oscillator, marked blue in the above schematic.

The long groups of holes on the breadboard are well suited to carry the supply voltage of the whole circuit, +5 V (VCC) and 0 V (ground or GND). Connect the corresponding pins of the ATmega328 to these supply *rails*. Preferably use **red** wires for +5 V and **black** wires for 0 V. Observe that there are two pins of either polarity which need to be connected (we will learn later why there are two…).

Then take the USB programmer and put it into the long rails of the breadboard

- this will connect *GND* to the 0 V (GND) rail of your breadboard
- and *5V* to the +5 V (VCC) rail of your breadboard
- connect the remaining pins to the corresponding pins *RESET*, *SCK*, *MOSI* and *MISO* of the microcontroller, marked in yellow in the pinout above.



An ATmega328 on a breadboard connected to the programming adapter.

4 First check...

☞ Under Windows you can skip this step and do the following test directly from within the graphical user interface of *AVRDUDESS*.

☞ If everything is connected and installed correctly we should now be able to *talk* with the ATmega328 through the USB programmer. As a very first test, we can run `avrdude` from the commandline – independent of the operating system we are using on our host computer:

```
prompt:> avrdude -B 5 -c usbasp -p m328

avrdude: set SCK frequency to 187500 Hz
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.04s

avrdude: Device signature = 0x1e9514

avrdude: safemode: Fuses OK (E:07, H:D9, L:62)

avrdude done.  Thank you.
```

What did happen here?

- the option `-B 5` reduces the serial transfer speed between the host computer through the USB programmer to the ATmega328 to a slow enough speed – a modern pc would otherwise be too fast for our microcontroller
- the option `-c usbasp` tells *avrdude* what kind of programmer we are using – there are literally dozens of possible choices available
- the option `-p m328` tells *avrdude* which type of AVR controller we are using – the abbreviated form of the ATmega328 is the same acronym that is also used (partly) by our compiler tool chain.

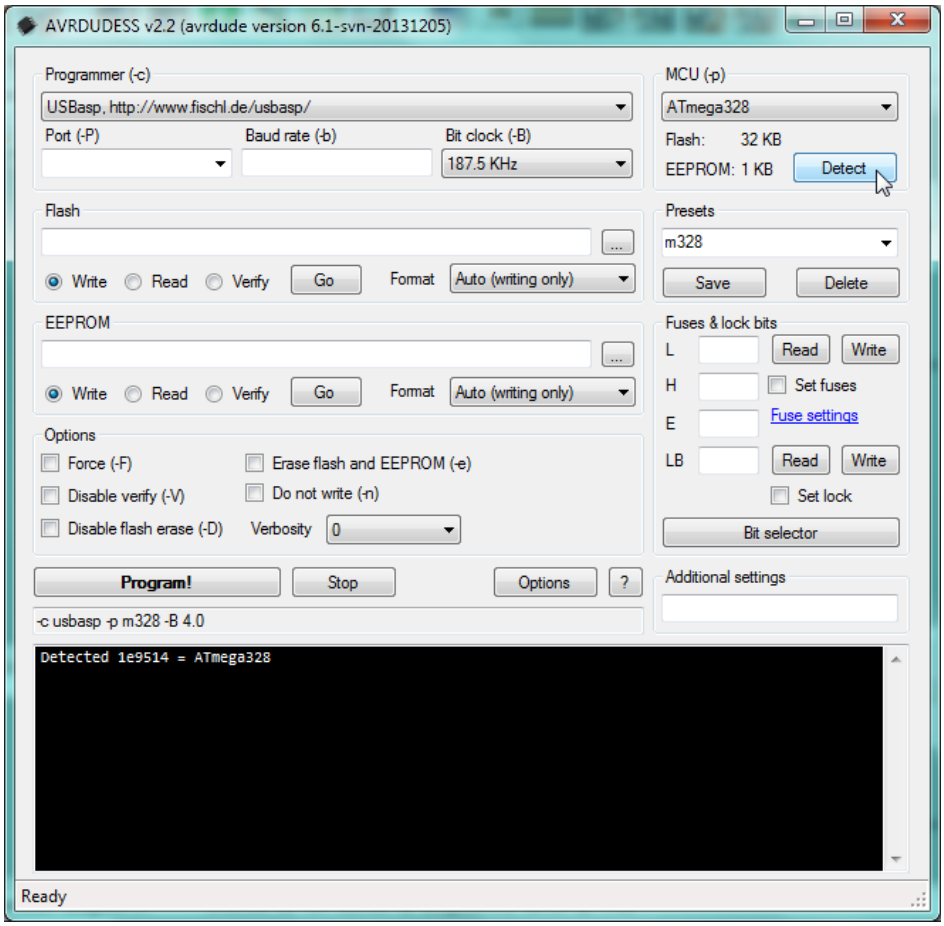
Then *avrdude* checks if it is able to establish a communication with the chip on the other side. If it succeeds it then reads back what is called the *signature* of the device: a three byte long identifier stored inside the microcontroller, identifying its exact type. In this case we get back the correct signature for the ATmega328, just a few more signatures:

ATmega168	`0x1e9406`	
ATmega168P	`0x1e940b`	low power version
ATmega328	`0x1e9514`	
ATmega328P	`0x1e950F`	low power version

(Some older versions of *avrdude* do not recognize the non-P version of the ATmega328, because it was introduced after the low-power version – there are ways to fix this, because apart from the signature both chips are treated exactly identical.)

5 AVRDUDESS

☞ Of course you can use all the function of *avrdude* from the commandline, you can even embed it into a *Makefile*, but if you want to use a more graphical utility there are some to choose from. *AVRDUDESS* is a .NET application, designed for Windows but it should also run under different operating systems – though with different settings.



Main window of AVRDUDESS under Windows.

With a program like this you do not need to remember all command line options, but you can choose from lists and menus instead. In order to configure the basic settings of your systems and your USB programmer you need to to choose *USBasp* from the list of programmers. For this programmer you do not need to select a *Port* nor a *Baud rate*. However, you should choose a *Bit clock* of 187,5 kHz.

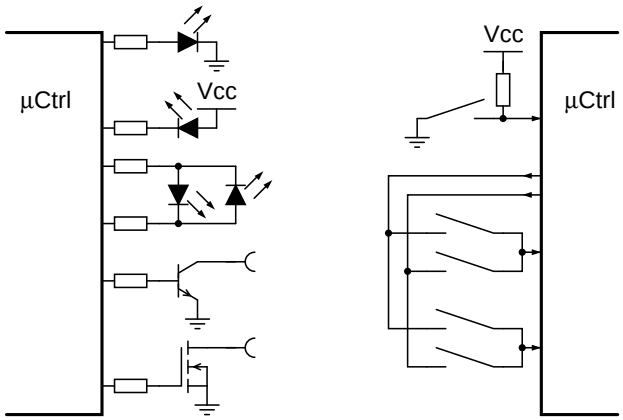
The easiest way to check the correct configuration of *AVRDUDESS* is to try to identify the attached microcontroller with the function Detect. In the message area of *AVRDUDESS* you will see the progress and whether or not the operation completes successfully.

☞ **Before doing major reconstruction in an existing circuit you should always disconnect the power supply on the USB programmer from the circuit!**

6 Writing the first lines of source code

Now that the hardware is attached and configured we can write out first piece of source code for the AVR microcontroller – at least within this course. It does not really matter whether you write the source code in the IDE of *Atmel Studio* or in any other text editor. The main convenience of *Atmel Studio* is that it will automatically generate the correct command line switches for the *GCC* compiler and also start pre-compiler, compiler and linker in the correct order on the correct input files – just with the pressing of a single button on the toolbar. There are more functions under the hood of *Atmel Studio* which you are free to explore, but which are not strictly necessary to use.

☞ We want to see some output from our microcontroller – otherwise it will be difficult to determine if our program will be running at all. Therefore you should start by connecting one side of a 150 Ω to 1000 Ω resistor to the pin `PB0` of our ATmega328. To the other leg of the resistor you connect the anode (positive, longer leg) of an LED and the cathode (negative, shorter leg) of the LED you connect to GND. As you remember from *Lab 1* a positive voltage (i.e. a logical 1) on the longer leg should now light-up the LED.



Connecting ”stuff” to the microcontroller – we start with a single LED to GND (top left).

If you now connect the USB programmer back to the circuit you should see nothing – since no program is loaded into the microcontroller it will (hopefully) do nothing and all GPIO pins are configured as inputs.

Lets write the first lines of code (you will find it as `labb_02_code_01.c` on Studentportalen. Either start a new project in *Atmel Studio* or just look at the source code in any text editor:

```
1  #define F_CPU 1000000UL // tell the compiler about the clock frequency
2  #include <avr/io.h>     // defines all macros and symbols
3  #include <util/delay.h> // defines time delay functions
4
5  int main (void)
6  {
7    DDRB = 0x01;          // pin 0 of PORTB as output
8
9    while (1)             // infinite main loop
10   {
11     PORTB = 0x01;       // switch PB0 to 1
12     _delay_ms(500);     // wait 1/2 second
13     PORTB = 0x00;       // switch PB0 to 0
14     _delay_ms(500);     // wait 1/2 second
15   }
16 }
```

Compile the code using the *Build⇒Build solution* function in *Atmel Studio*, or using a commandline version of the compiler or a Makefile.

This was some ugly spaghetti code which could be done much nicer, but it was our first program and it was meant to show some basic principles. You can find plenty of help on the internet on how to get the compiler manually to do what you want – I personally rely on the built-in creation and execution of a Makefile in *AVR Studio*.

The compiler will at the end tell you where it put the compiled code in the form of a `.hex`-file. This is the file we have to transfer into the flash-memory of our microcontroller now, using *avrdude* and the USB programmer (you might end up with a different filename!):

```
prompt:> avrdude -B 5 -c usbasp -p m328 -U flash:w:labb_02_code_01.hex
...
avrdude: Device signature = 0x1e9514
...
avrdude: reading input file "....hex"
...
avrdude: writing flash (xyz bytes):

Writing | ################################################## | 100% 0.17s

avrdude: xyz bytes of flash written
...
avrdude done.  Thank you.
```

In *AVRDUDESS* you would select the .hex file from a browser window and then write the contents to the flash memory of the microcontroller. You would see the same type of output from *avrdude* in the message area.

6.1 Summing up to this point

- `DDRA`, `DDRB`, `DDRC` and `DDRD` control the *direction* of GPIO pins, i.e. whether a pin is an input (corresponding bit in `DDRx` set to 0) or an output (corresponding bit in `DDRx` set to 1)
- `PORTA`, `PORTB`, `PORTC` and `PORTD` control the state of GPIO output pins, i.e. whether a pin is at a low voltage (0 V, corresponding bit in `PORTx` set to 0) or at a high voltage (e.g. 5 V, corresponding bit in `DDRx` set to 1)

7 Second micro-project

One of the tools you will find very important during the project work is the possibility to deliver debug-information from the microcontroller back to the surrounding world, especially to you. Our USB programmer does not support direct debugging in hardware, which in principle is possible for the modern AVR controllers. In the AVR studio you have the ability to interactively simulate the compiled code and directly see the influence it has on the microcontroller's hardware registers – in my personal experience this was a nice feature in the very beginning, but it is not useful anymore when you start to use software delays, interrupts or wait for external signals…

It is much more important to communicate from the code by sending out messages, values of variables and alike. This can of course be done using LEDs on the breadboard, but this can be very restricting. You have access to a module with a two-line 16-character LCD, which can be controlled in a 4-bit mode sacrificing 7 digital port pins. The module can be connected to the breadboard by using the (green) 24-pin adapter. The following example will also show you how to use third-party libraries in your projects.

☞ Download the LCD library from Studentportalen and copy the files `lcd.c` and `lcd.h` into your current AVR-GCC project directory.

☞ Connect the LCD to the ATmega328 according to the following table:

24-pin adapter			**ATmega328**	
Terminal	Symbol	Function	Pin	Function
1	U_{DD}	Supply voltage	*connect to the VCC supply rail*	
2	U_{SS}	GND	*connect to the GND supply rail*	
3	RS	LCD Register select	24	PC.1
5	E	LCD Enable	25	PC.2
6	R/W	LCD Read/Write	26	PC.3
11	DB5	LCD Data bit 5	17	PB.3 through 10 kΩ
12	DB4	LCD Data bit 4	16	PB.2 through 10 kΩ
13	DB7	LCD Data bit 7	19	PB.5 through 10 kΩ
14	DB6	LCD Data bit 6	18	PB.4 through 10 kΩ
17	S1	Switch 1	15	PB.1

☞ Open the header file `lcd.h` in the source code editor and check if the settings correspond to your connection scheme. As you can see you are quite free on how to connect the LCD to your controller later on.

```
85 #define LCD_DATA0_PORT   PORTB        /**< port for 4bit data bit 0 */
86 #define LCD_DATA1_PORT   PORTB        /**< port for 4bit data bit 1 */
87 #define LCD_DATA2_PORT   PORTB        /**< port for 4bit data bit 2 */
88 #define LCD_DATA3_PORT   PORTB        /**< port for 4bit data bit 3 */
89 #define LCD_DATA0_PIN    2            /**< pin for 4bit data bit 0  */
90 #define LCD_DATA1_PIN    3            /**< pin for 4bit data bit 1  */
91 #define LCD_DATA2_PIN    4            /**< pin for 4bit data bit 2  */
92 #define LCD_DATA3_PIN    5            /**< pin for 4bit data bit 3  */
93 #define LCD_RS_PORT      PORTC        /**< port for RS line         */
94 #define LCD_RS_PIN       1            /**< pin  for RS line         */
95 #define LCD_RW_PORT      PORTC        /**< port for RW line         */
96 #define LCD_RW_PIN       3            /**< pin  for RW line         */
97 #define LCD_E_PORT       PORTC        /**< port for Enable line     */
98 #define LCD_E_PIN        2            /**< pin  for Enable line     */
```

☞ Include `lcd.c` to the list of *Source files* in AVR studio by right-clicking on the project entry in the Solution explorer and selecting `Add existing item`. Then enter and compile the following short program `labb_02_code_02.c`:

```
#define F_CPU 1000000UL  // 1MHz internal clock

#include <avr/io.h>
#include <util/delay.h>
#include "lcd.h"

void init (void)         // collect hardware initializations here
{
  DDRB  = 0b00000001;    // LED still connected
  PORTB = 0b00000010;    // pull-up resistor active PB1

  lcd_init(LCD_DISP_ON); // initialize LCD
  lcd_puts("Hello world");
}

int main (void)
{
  init();

  while (1)                          // infinite main loop
  {
    PORTB ^= 0b00000001;             // invert LED
    lcd_puts("waiting for key\r\n");
    while (PINB & 0b00000010);       // wait for key
    lcd_puts("key pressed\r\n");
    _delay_ms(1000);                 // wait 1000ms between cycles
    lcd_clrscr();
  }
}
```

The `lcd.c` library uses bit-banging to send data and commands to the LCD controller. No special hardware feature of the AVR is needed, except for seven digital output lines.

8 Ideas...

☞ Program a timer with display on the LED, with start, stop, set and reset function. Try to determine the runtime of your code and adjust the delays accordingly. Or use a timer interrupt for time control...

9 To be continued...

Feel free now to experiment with the different features which were presented during the lectures. In the project boxes you will find different pieces of hardware which you can connect to the AVR controller – datasheets for almost everything should be found on Studentportalen.

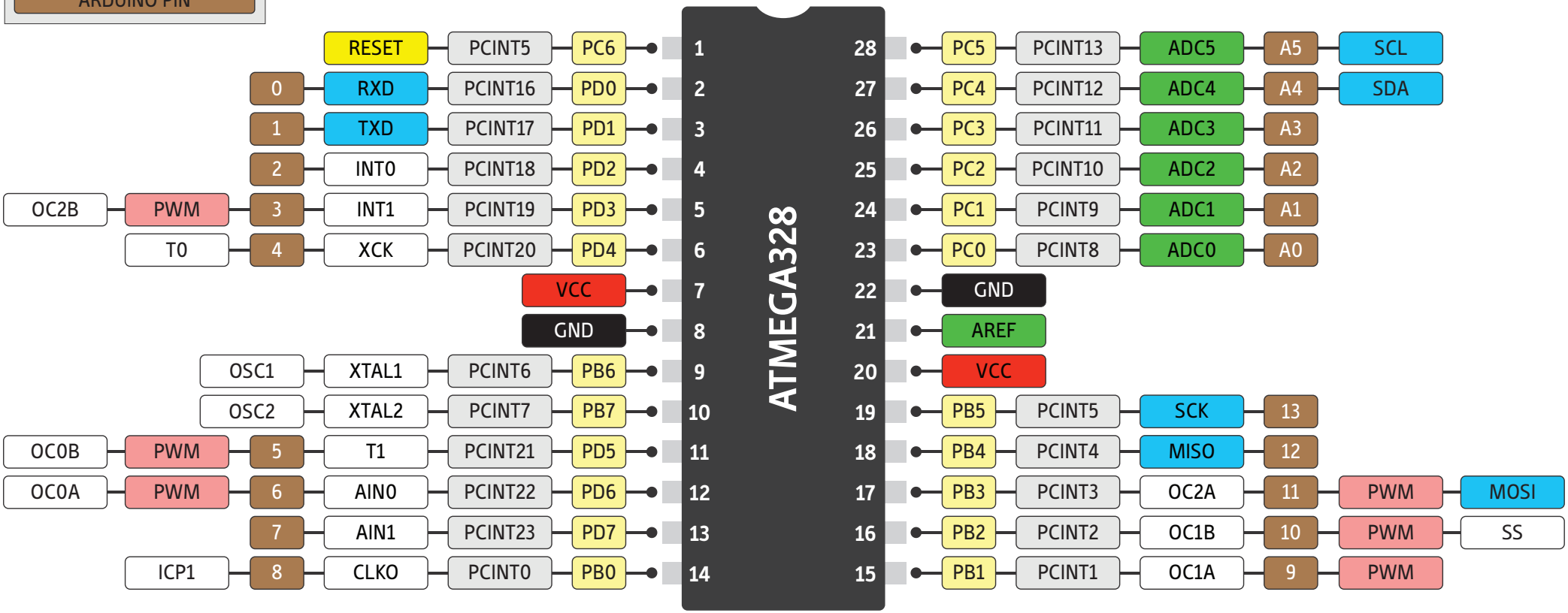
THE
UNOFFICIAL

ARDUINO & ATMEGA328

PINOUT DIAGRAM

LEGEND

- GND
- POWER
- CONTROL
- PORT PIN
- ATMEGA328 PIN FUNC
- DIGITAL PIN
- ANALOG-RELATED PIN
- PWM PIN
- SERIAL PIN
- ARDUINO PIN



www.pighixxx.com

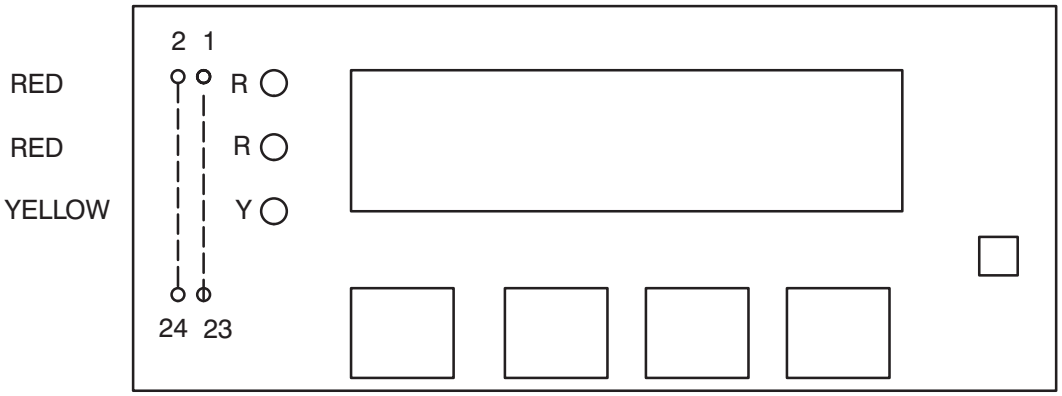
29 JAN 2013

2.2 CONNECTIONS (DESCRIPTION)

Terminal	Symbol	Function
1	U_{DD}	Supply voltage
2	U_{SS}	GND
3	RS	Register select
4	NC	No connection
5	E	Read/ write enable
6	R/W	Read/ write select
7	DB1	Data bit 1
8	DB0	Data bit 0
9	DB3	Data bit 3
10	DB2	Data bit 2
11	DB5	Data bit 5
12	DB4	Data bit 4
13	DB7	Data bit 7
14	DB6	Data bit 6
15	CLED(+)	LED C
16	ALED(+)	LED A
17	S1	Switch 1
18	BLED(+)	LED B
19	S3	Switch 3
20	S2	Switch 2
21	S4	Switch 4
22	NC	No connection
23	U_{SS}	GND
24	U_{DD}	Supply voltage

2.3 LED POSITION

Top view



2.4 BLOCK DIAGRAM