Programming a Target AVR using Arduino Uno board loaded with ArduinoISP

(April 21, 2015, author: B. Lazar)

Arduino Uno + **ArduinoISP** form an ordinary **ISP programmer** that is used by **avrdude** to program various 8 bit AVRs with any kind of hex file.
*No bootloader is needed, no hardware removal from **Arduino Uno** is necessary*

It may happen that you have an **Arduino Uno** that you bought, played a bit with it and now you want to move further, to build your own board, which can be just a simple AVR microcontroller + an oscillator + a LED, but you do not have an ISP programmer.

In reality, **Arduino Uno** is everything you need to program a large variety of AVRs. You just have to follow the four steps described below:

1) Open **Arduino 1.0.1** and upload **ArduinoISP** to **Arduino Uno**.
2) Connect the **ISP pins** of **Arduino Uno** to the **Target AVR** as indicated in the Fig.1.
3) Write your source code, compile it and generate the hex file.
4) Use **avrdude** command lines to upload the hex file into the **Target AVR**.

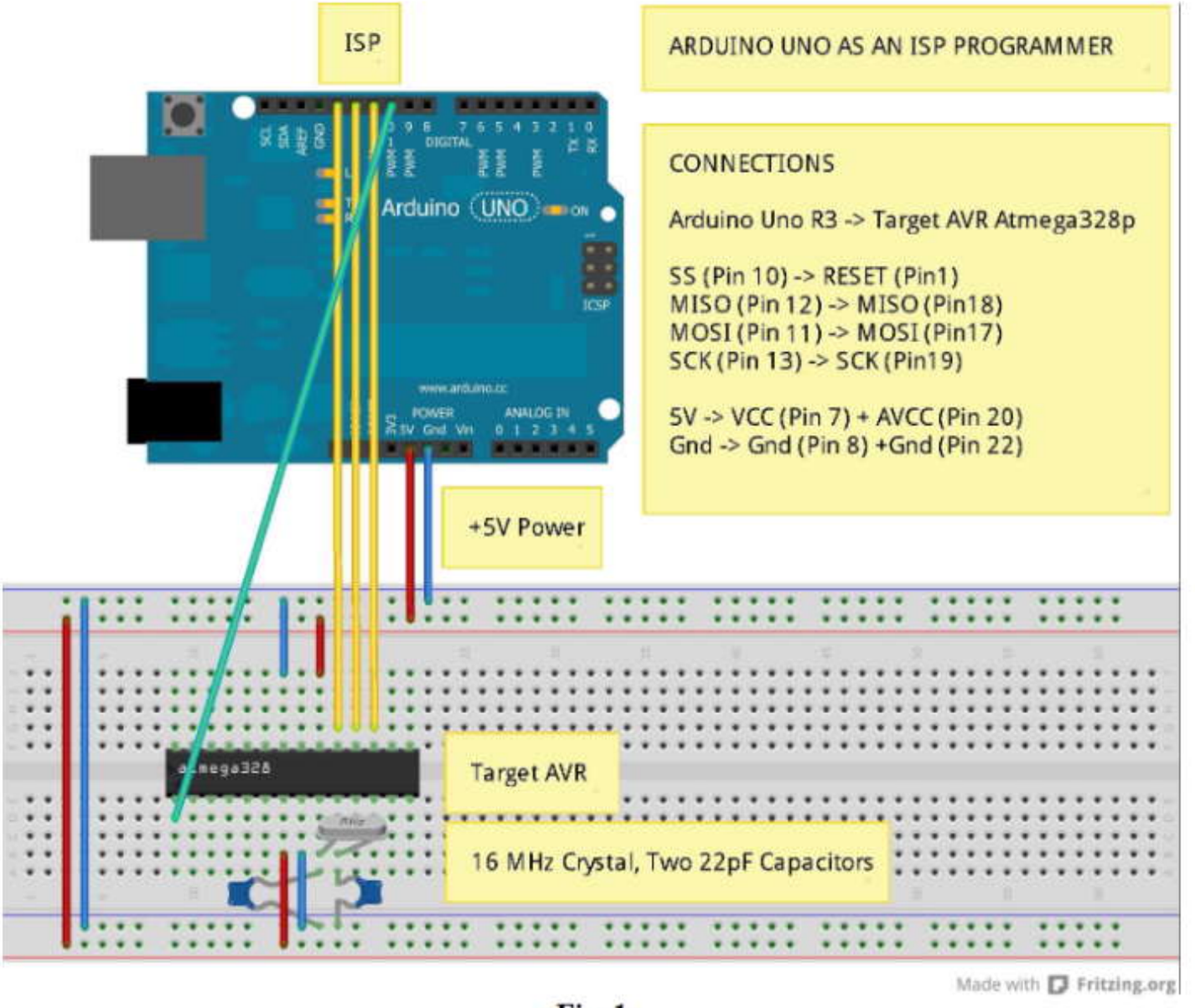

Fig. 1

Each of the four steps explained in detail:

(1) Open Arduino 1.0.1 and upload ArduinoISP to Arduino Uno

Start **Arduino 1.0.1**, go to **Files->Examples** and open **ArduinoISP**. Then from **Tools->Board** select **Arduino Uno**, from **Tools->Programmer** select **AVR ISP** or **Arduino as ISP**. Access **Tools->Serial Port** and write down the name of the port (ex. **COM3**) which is needed for **avrdude**.

Finally, press **Files->Upload** to load **ArduinoISP** to **Arduino Uno**. This operation transforms **Arduino Uno** into an **ISP programmer**.

(2) Connect the ISP pins of Arduino Uno to the Target AVR as indicated in Fig.1

It is extremely important to make sure that the ISP pins of **Arduino Uno** are connected to the **Target AVR** like this:

Arduino Uno	**Target AVR** (ex. Atmega328p)
SS	**RESET**
MOSI	**MOSI**
MISO	**MISO**
SCK	**SCK**

Frequent wiring mistakes are Arduino Uno(SS) -> Target AVR(SS) and Arduino Uno(RESET) -> Target AVR(RESET).

As a note: The crystal and its two capacitors have to be as close to XTAL1, XTAL2 pins of the **Target AVR** as possible (like in Fig. 1). If you position them in a different configuration, far from the processor, running wires between capacitors, crystal and the AVR, the newly formed oscillator can refuse to work and **avrdude** will throw the error "Invalid device signature" when you try to program the microcontroller.

(3) Write your source code, compile it and generate the hex file

In **Arduino 1.0.1->Files->Preferences** check **Show verbose output during compilation**. This will display a lot of information, when the code is compiled, including the location of the hex file.

Go to **File->Examples->01.Basics** and load **Blink** example. This will be considered as your source code.

Go to **Sketch** menu and press **Verify/Compile**.

When the compilation completes search in the lower side of the screen the path of the generated Blink hex file. It looks like this one:

```
C:\Users\Victor\AppData\Local\Temp\build1325897986179133197.tmp\Blink.cpp.hex
```

Select it with the mouse and press Ctrl-C followed by a Paste in Notepad, for instance. You will need the path for **avrdude**.

REMARK: The hex file can be generated with any IDE you like, not necessarily using **Arduino 1.0.1**. You can write your code in: **BASCOM**, **CodeVision**, **AVR Studio**, **IAR** which, in most cases, are far better choices than **Arduino 1.0.1**.

ATTENTION! If you use **Arduino 1.0.1** to generate a hex file, you have to select the right board/processor from **Tools->Board**. For example, you simply can not compile the **Blink** Led code for **Arduino Uno**

(**Atmega328p**) and expect it to work on a **target Atmega32** microcontroller. Also, even if, your target chip on the breadboard is an **Atmega328p DIL28**, you have to take into account that **pin 13** of **Arduino Uno** (the one that drives the led) corresponds to **pin 19 (SCK)** on your target **Atmega328p** microcontroller.
AS A CONCLUSION: Arduino 1.0.1 works well for compiling code written for the boards in its list.
For anything else, different boards or stand alone AVRs, it is highly indicated to use a general purpose IDE, not Arduino 1.0.1.

(4) Use avrdude command lines to upload the hex file into the Target AVR

Avrdude comes bundled with **WinAVR**, a software that is free and can be found on the internet.
Locate **avrdude** inside **WinAVR**, start the Windows standard application **Command Prompt** and launch **avrdude** from within **Command Prompt**.
A better choice would be to use a software, similar to **Command Prompt** but much easy to use, called **PowerCmd** because it allows you to save and copy-paste the long **avrdude** command lines, with many options and complicated paths to hex files, necessary to program an AVR (see the example).

At this moment, it is supposed that **Arduino Uno** is already loaded with **ArduinoISP**, connected to the **PC** with an **USB** cable and all wires, between the **Target AVR** and **Arduino Uno** acting as an **ISP** programmer, are in their place as in Fig. 1.

For the example that follows, the **Target AVR** is chosen to be an **Atmega328p** already set to run at **16 MHz**.

EXAMPLE:

Launch **PowerCmd** (or Command Prompt) and copy this line:

```
c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -n
```

in one of its command windows.

If your AVR is different then change the option **-p m328p** to **-p …** corresponding to your microcontroller. Most likely you do not have to modify the other options. (Note: The command ***c:\>avrdude -?*** will display a list with explanations regarding the meaning of the switches: ***–P, -b, -c, -p, -n*** and many others.)

Press **Enter**.

Avrdude should display the message:

```
c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -n
avrdude: AVR device initialized and ready to accept instructions
Reading | ################################################## | 100% 0.14s
avrdude: Device signature = 0x1e950f
avrdude: safemode: Fuses OK
avrdude done.  Thank you.
```

telling you that the **Target AVR** is accessible and can be programmed. (**-n** option instructs **avrdude** to make no modification inside the **Target AVR**).

You can program now the **Target AVR** with the **Blink.cpp.hex** file, generated at point (3), by copying the **avrdude** command, highlighted in red (see below), in **PowerCmd** and pressing **Enter**.
Do not forget to change the path to the hex file (the string of characters between "") according to the settings in your computer. The rest of the command line, including the final "**:i**" will likely remain unchanged.