

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 AVR microcontrollers 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 AVR microcontrollers. 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**.

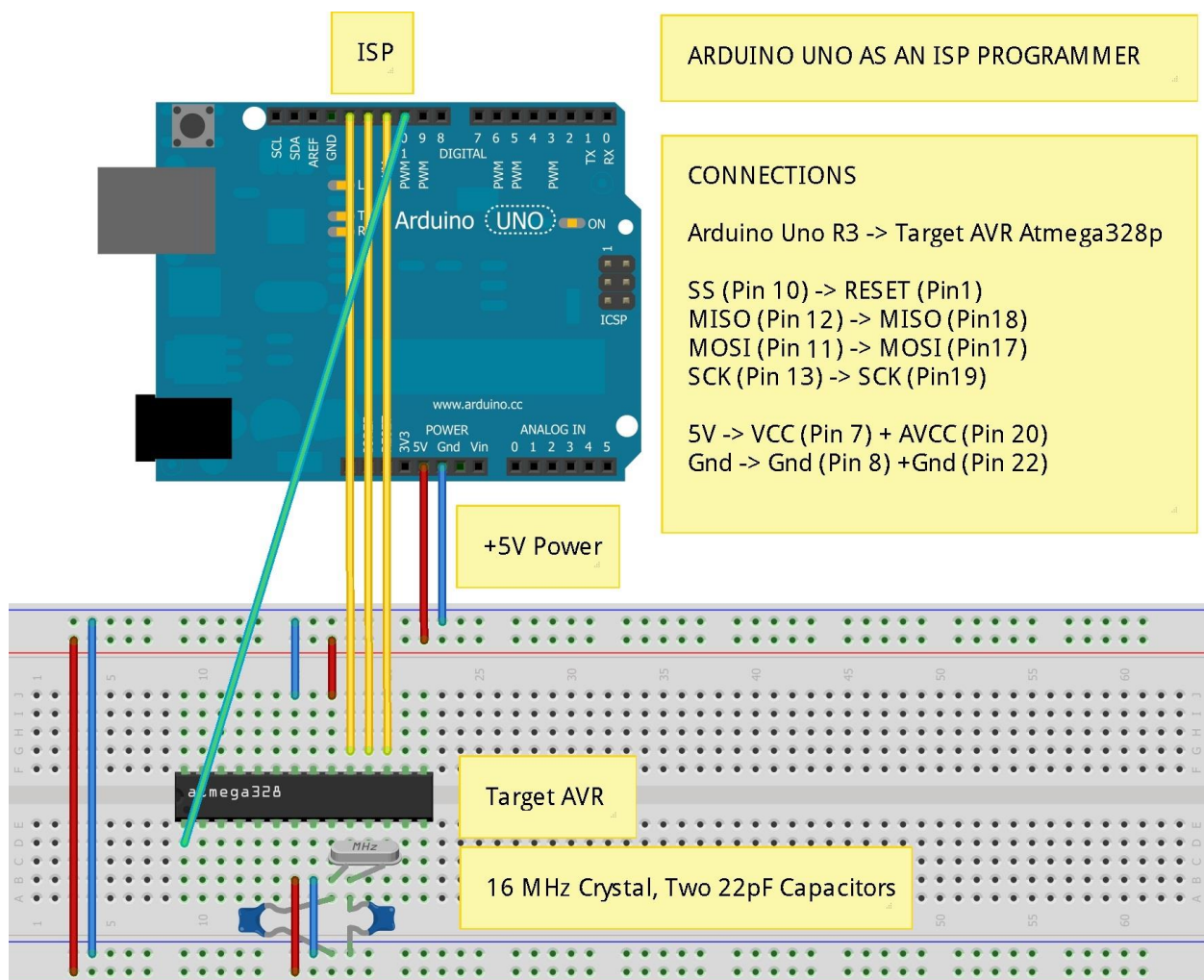


Fig. 1

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 AVR's, 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.

```
c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -u -U  
flash:w:"C:\Users\Victor\AppData\Local\Temp\build1325897986179133197.tmp\Blink.  
cpp.hex":i
```

```
avrdude: AVR device initialized and ready to accept instructions
```

```
Reading | ##### | 100% 0.16s
```

```
avrdude: Device signature = 0x1e950f  
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed  
        To disable this feature, specify the -D option.  
avrdude: erasing chip  
avrdude: reading input file  
"C:\Users\Victor\AppData\Local\Temp\build1325897986179133197.tmp\Blink.cpp.hex"  
avrdude: writing flash (1084 bytes):
```

```
Writing | ##### | 100% 1.76s
```

```
avrdude: 1084 bytes of flash written  
avrdude: verifying flash memory against  
C:\Users\Victor\AppData\Local\Temp\build1325897986179133197.tmp\Blink.cpp.hex:  
avrdude: load data flash data from input file  
C:\Users\Victor\AppData\Local\Temp\build1325897986179133197.tmp\Blink.cpp.hex:  
avrdude: input file  
C:\Users\Victor\AppData\Local\Temp\build1325897986179133197.tmp\Blink.cpp.hex contains  
1084 bytes  
avrdude: reading on-chip flash data:
```

```
Reading | ##### | 100% 1.31s
```

```
avrdude: verifying ...  
avrdude: 1084 bytes of flash verified
```

```
avrdude done.  Thank you.
```

Success! At this moment your **Target AVR** is programmed and ready to execute the code.

Most common errors you may get from avrdude:

```
1) c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -n  
avrdude: ser_open(): can't open device "\\.\COM3": The system cannot find the file  
specified.
```

Reason: The **USB** connection between the **PC** and **Arduino Uno** is not good. If the connection is good but the error still keeps coming press the **Reset** button of **Arduino Uno**.

```
2) c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -n  
avrdude: ser_open(): can't open device "\\.\COM3": Access is denied.
```

Reason: Another application that uses **COM3** port, likely a **Serial Monitor**, is running. Close it.

```
3) c:\>c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -n  
Access is denied.
```

Reason: c drive appears twice: "c:>c:>".

```

4) c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -n
avrdude: stk500_program_enable(): protocol error, expect=0x14, resp=0x50
avrdude: initialization failed, rc=-1
      Double check connections and try again, or use -F to override
      this check.
avrdude: stk500_disable(): protocol error, expect=0x14, resp=0x51
avrdude done. Thank you.

```

Reason: The **Target AVR** is not powered or does not exist.

This message is also useful for testing, before connecting any **Target AVR**, that **Arduino Uno** + **ArduinoISP** work as an **ISP programmer** and the **connection to the PC is proper**.

```

5) c:\>avrdude -P COM3 -b 19200 -c avrisp -p m328p -n
avrdude: stk500_getsync(): not in sync: resp=0xf0
avrdude done. Thank you.

```

Reason: **Arduino Uno** needs a **Reset**.

```

6) 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 = 0xffffffff (Device signature = 0x000000)
avrdude: Yikes! Invalid device signature.
      Double check connections and try again, or use -F to override
      this check.

```

Reasons:

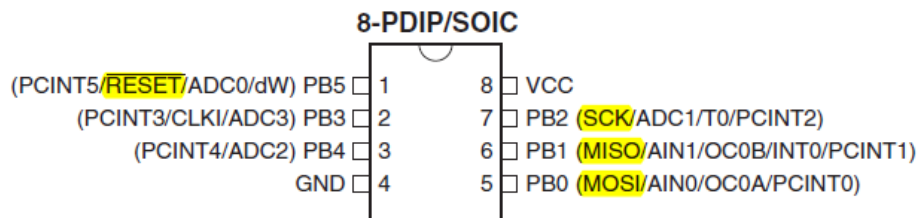
- ISP connections (**SS, MOSI, MISO, SCK**) -> (**RESET, MOSI, MISO, SCK**), between **Arduino Uno** and the **Target AVR**, are not good, wrong wiring.
- The AVR oscillator (external) does not work.

List of commonly used AVR's and how they should be connected to an Arduino Uno, for programming

If you do not use an Atmega328p target then you should adapt the wiring in Fig. 1 according to your specific target AVR (ex. Attiny13, 84, 85, Atmega8, etc.).

Attiny13

Uno -> pin Attiny13

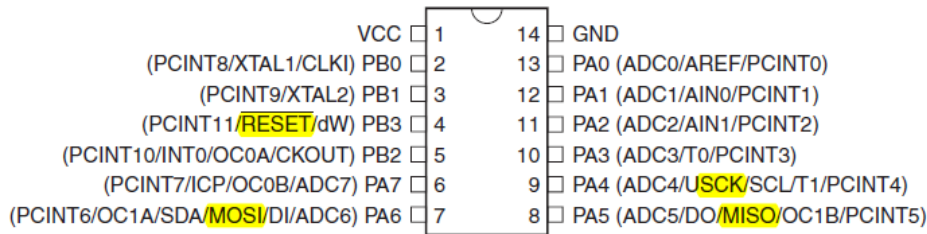


SS -> 1
MOSI -> 5
MISO -> 6
SCK -> 7

Uno -> pin Attiny84

Attiny84

PDIP/SOIC

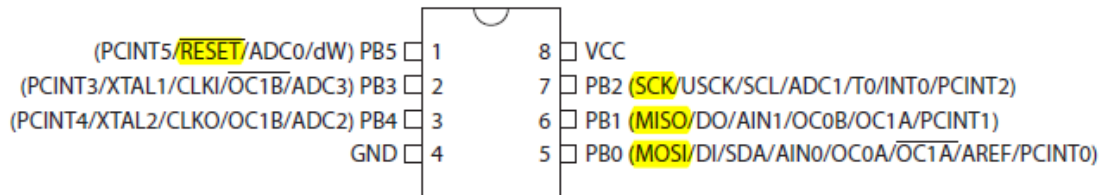


SS -> 4
MOSI -> 7
MISO -> 8
SCK -> 9

Attiny85

Uno -> pin Attiny85

PDIP/SOIC/TSSOP

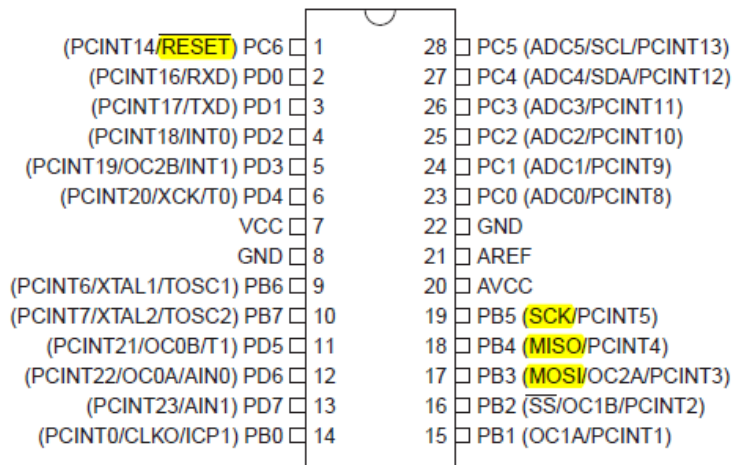
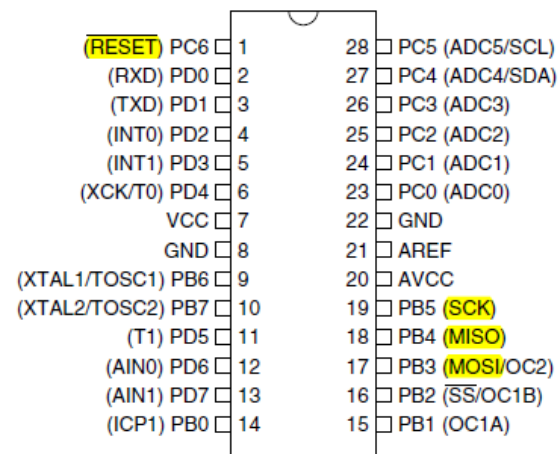


SS -> 1
MOSI -> 5
MISO -> 6
SCK -> 7

Atmega8

Atmega328p

PDIP

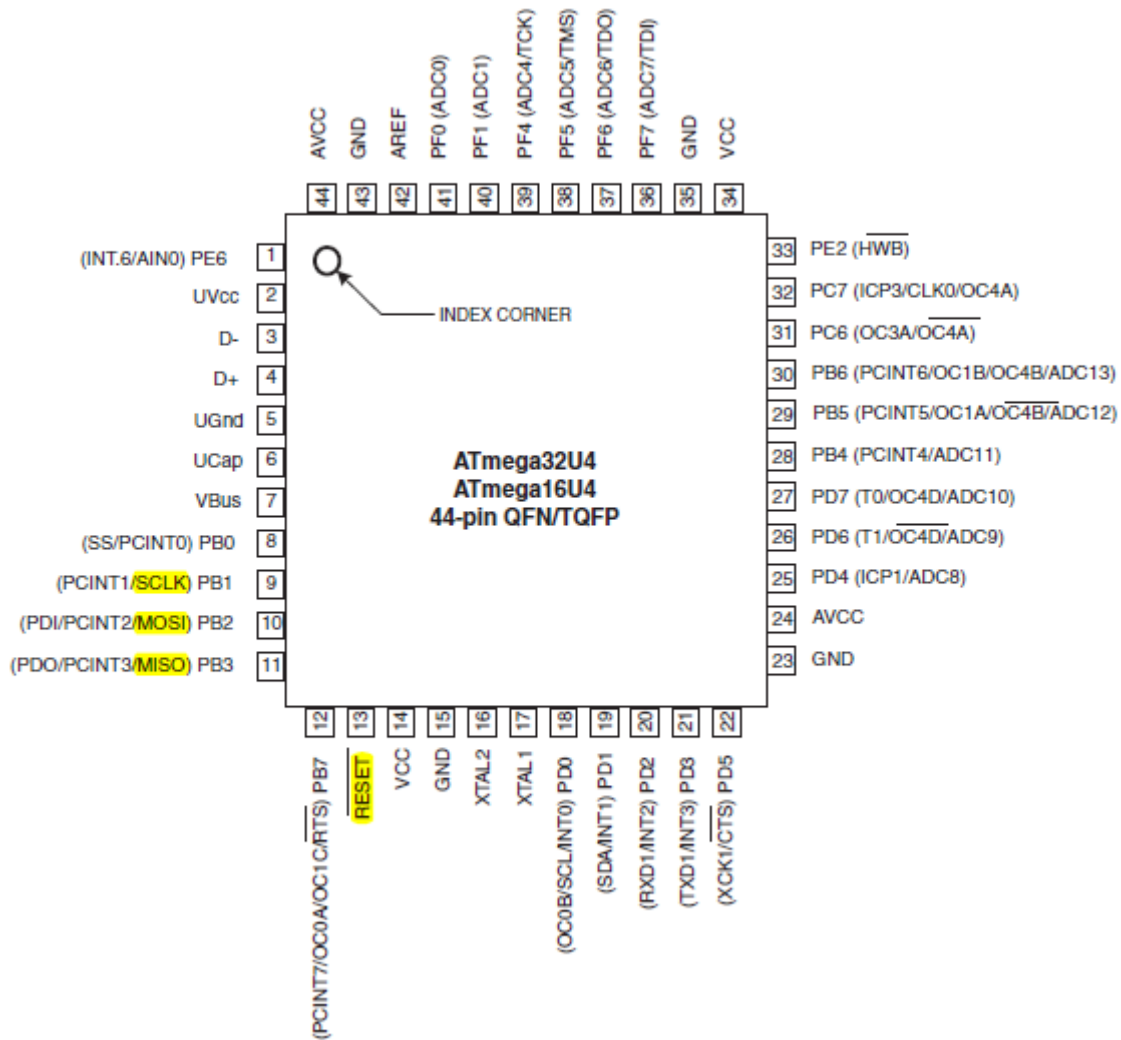


(Arduino Uno -> pin Atmega8, Atmega328p) – (SS -> 1, MOSI -> 17, MISO -> 18, SCK -> 19)

Atmega32U4

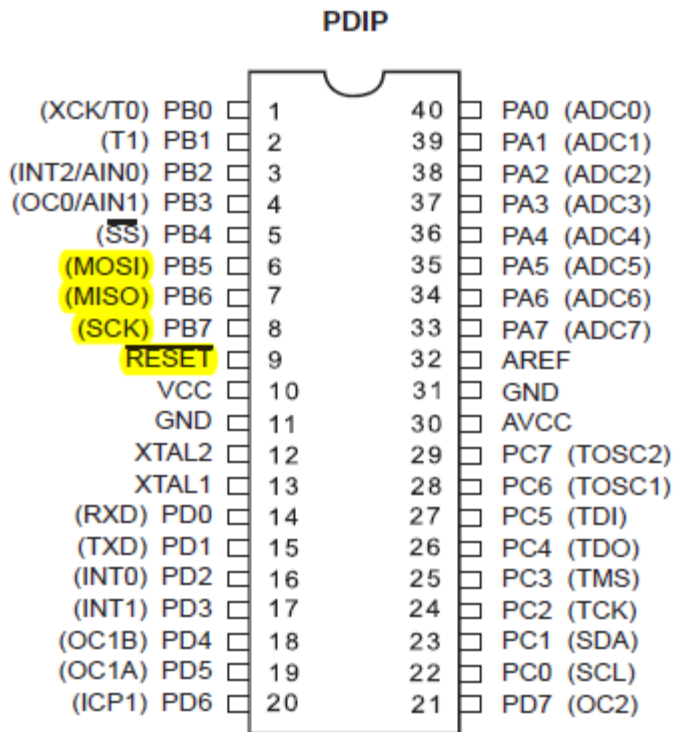
Uno>Atmega32U4

SS -> 13
MOSI -> 10
MISO -> 11
SCK -> 9



Atmega32

Uno -> pin Atmega32



SS -> 9

MOSI -> 6

MISO -> 7

SCK -> 8

Last update: July 3, 2016.