RescueAVR

This sketch can resurrect many classic AVR chips with wrong fuse settings using high-voltage (HV) programming. Sometimes, you may erroneously set a fuse bit, such as the clock source bit, and then the chip does not respond to ISP programming anymore. In this case, HV programming can help. One can easily reset all fuses to their factory setting, and then ISP programming is possible again. Note that the sketch does not implement a general HV programmer but can only perform some basic tasks, such as fuse setting and erasing the entire chip.

You need an Arduino Uno, Nano, or Pro Mini, a breadboard, two transistors, a few resistors, and an external regulated 12-volt supply. In addition, the sketch is also an alternative firmware for manekinen's Fusebit Doctor. The pin mapping is a bit different between these two versions. When the sketch is compiled for an Arduino Uno, Nano, or Pro Mini in the Arduino IDE, it will use the Arduino Uno pin mapping. Otherwise, it uses the pin mapping for the Fusebit Doctor. One can also force which version is produced by defining the compile-time constants `ARDUINO_MODE` or `FBD_MODE`, respectively.

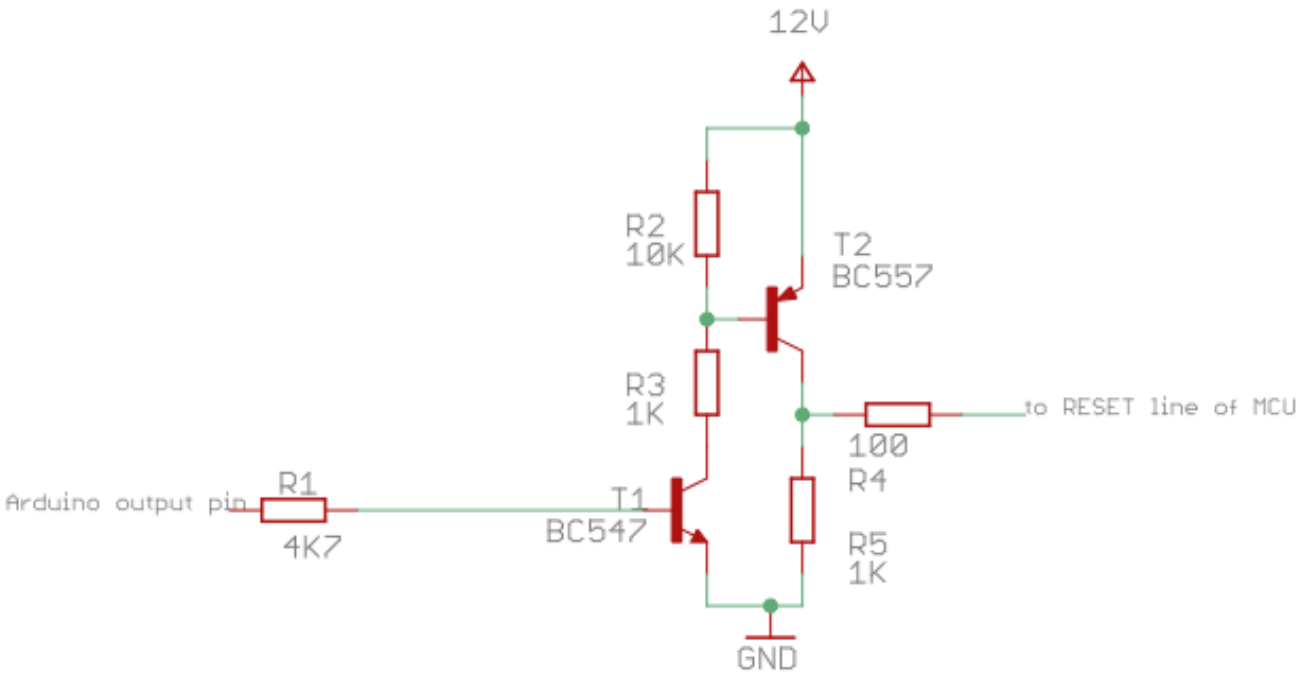
When you use the sketch, remember to set the monitor baud rate to 19200 baud (no parity, 1 stop-bit).

The sketch uses many of the ideas and code of MightyOhm's HV Rescue Shield 2 and is inspired by manekinen's Fusebit Doctor.

Using RescueAVR on an Arduino Uno

I tried the sketch with an ATtiny84, an ATtiny85, and an ATmega328 on the Arduino Uno. Fritzing wiring schemes for these three chips are included below. For other chips, you must consult the particular microcontroller's datasheet. The pin mapping is usually found in the section on ***Memory Programming***. For some more popular MCUs, I have included wiring instructions for the DIP/SOIC versions of the chips in Appendix B.

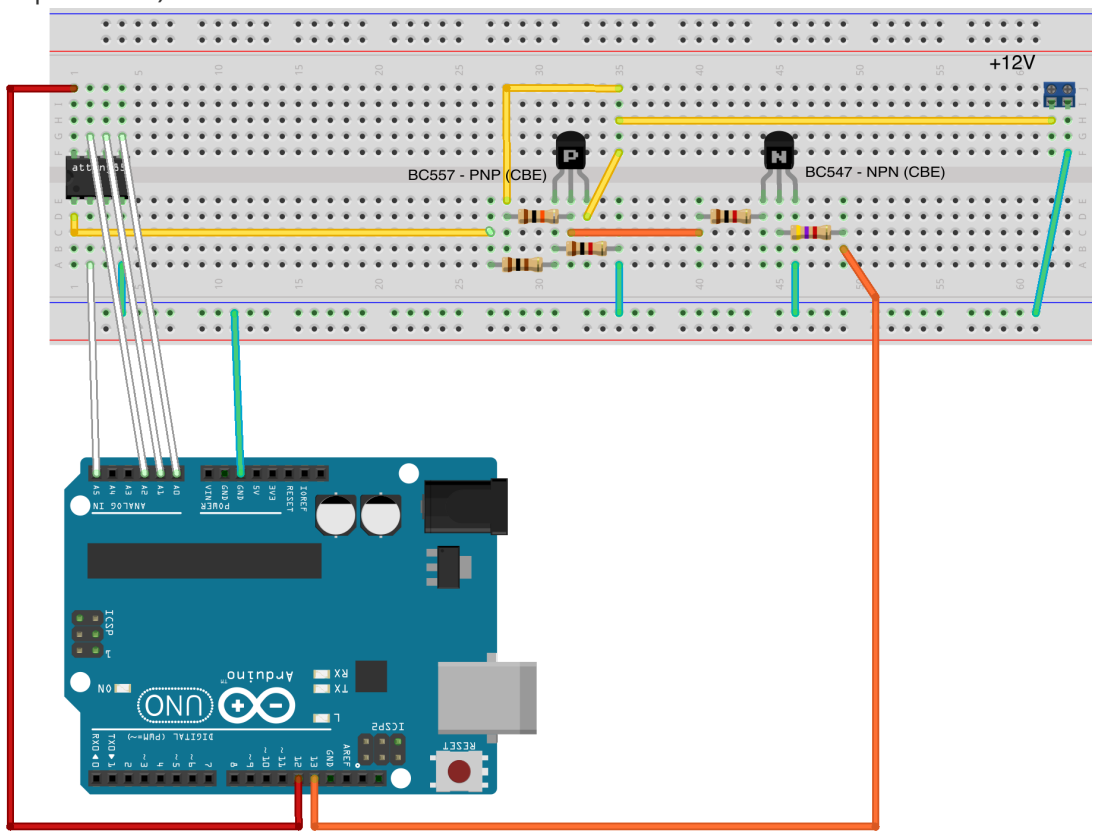
The most important part of high-voltage programming is the ability to put 12 volts into the RESET pin of the MCU. So, you need a regulated 12-volt supply and an electronic switch that applies this voltage to the RESET pin. Such a switch using two transistors is shown below. The transistors I have used are fairly standard ones. You can probably use any other reasonable type. But make sure that the pins are ordered as in the picture, i.e., CBE (otherwise, the Fritzing diagram is not correct).



The wiring is straightforward for small ATtinys because they use serial programming, and you need only a few wires. The Fritzing diagram for an ATtinyX5 looks as follows (and it also applies to ATtiny11, 12,13, 15, and 22 and a few other 8-pin MCUs).

HVSP-pin mapping

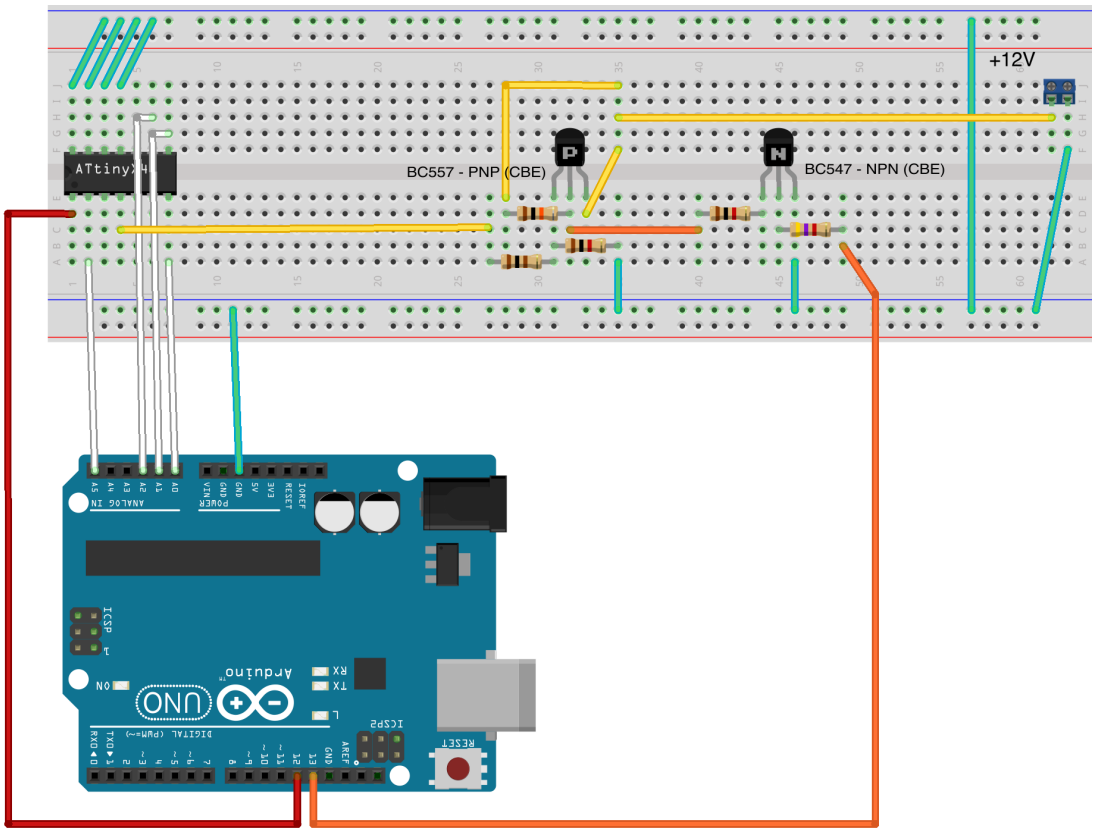
ATtinyX5 Pin	Function	Arduino
1	RESET 12V	
2	SCI	A5
3	N.C.	
4	N.C.	
5	SDI	A0
6	SII	A1
7	SDO	A2
8	Vcc	12



Similarly, the wiring for an ATtinyX4 is quite simple as well. As you can see, one needs just 2 data lines (SDI, SDO), one clock line (SCI), one control line (SII), and in addition one has to switch the RESET line and the Vcc line.

HVSP-pin mapping

ATtinyX4 Pin	Function	Arduino
1	Vcc	12
2	SCI	A5
3	N.C.	
4	RESET 12V	
5	N.C.	
6	N.C.	
7	SDI	A0
8	SII	A1
9	SDO	A2
10	N.C.	
11	GND	
12	GND	
13	GND	
14	GND	

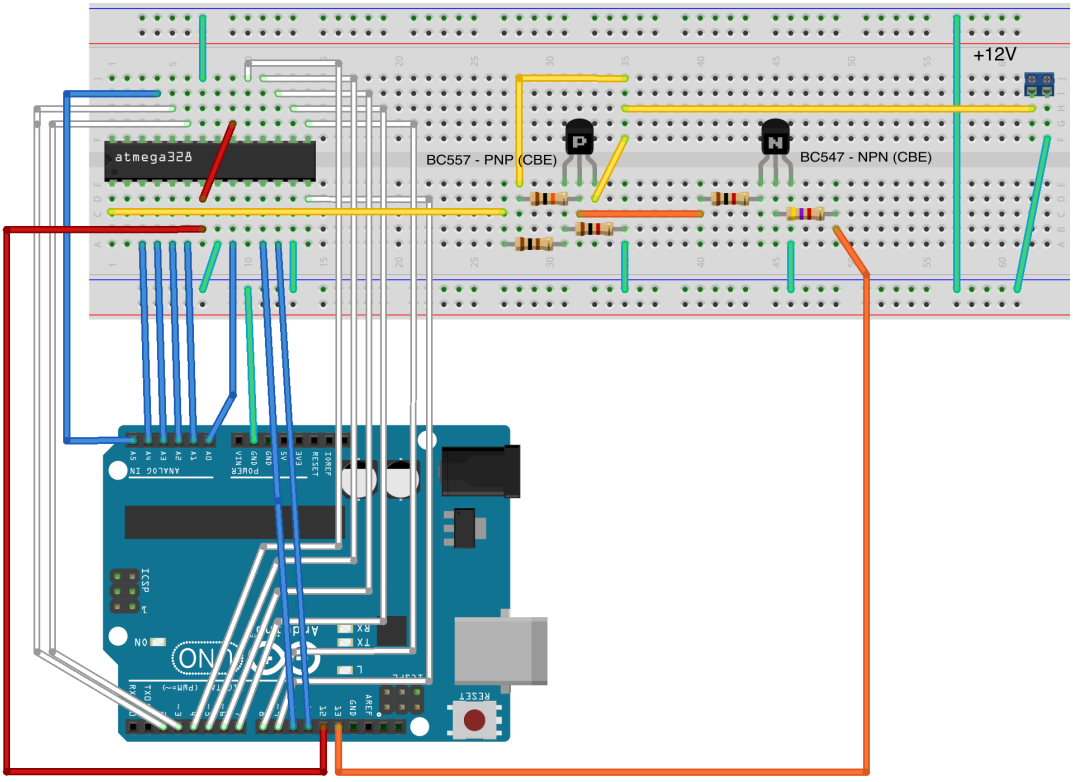


Made with Fritzing.org

For an ATmegaX8 MCU, the wiring is much more involved. Instead of 2 data lines, one clock line as well as three control lines (SII, RESET and Vcc), one has to deal with 8 data lines, one clock line, and 9 control lines! This may look like as in the following Fritzing diagram.

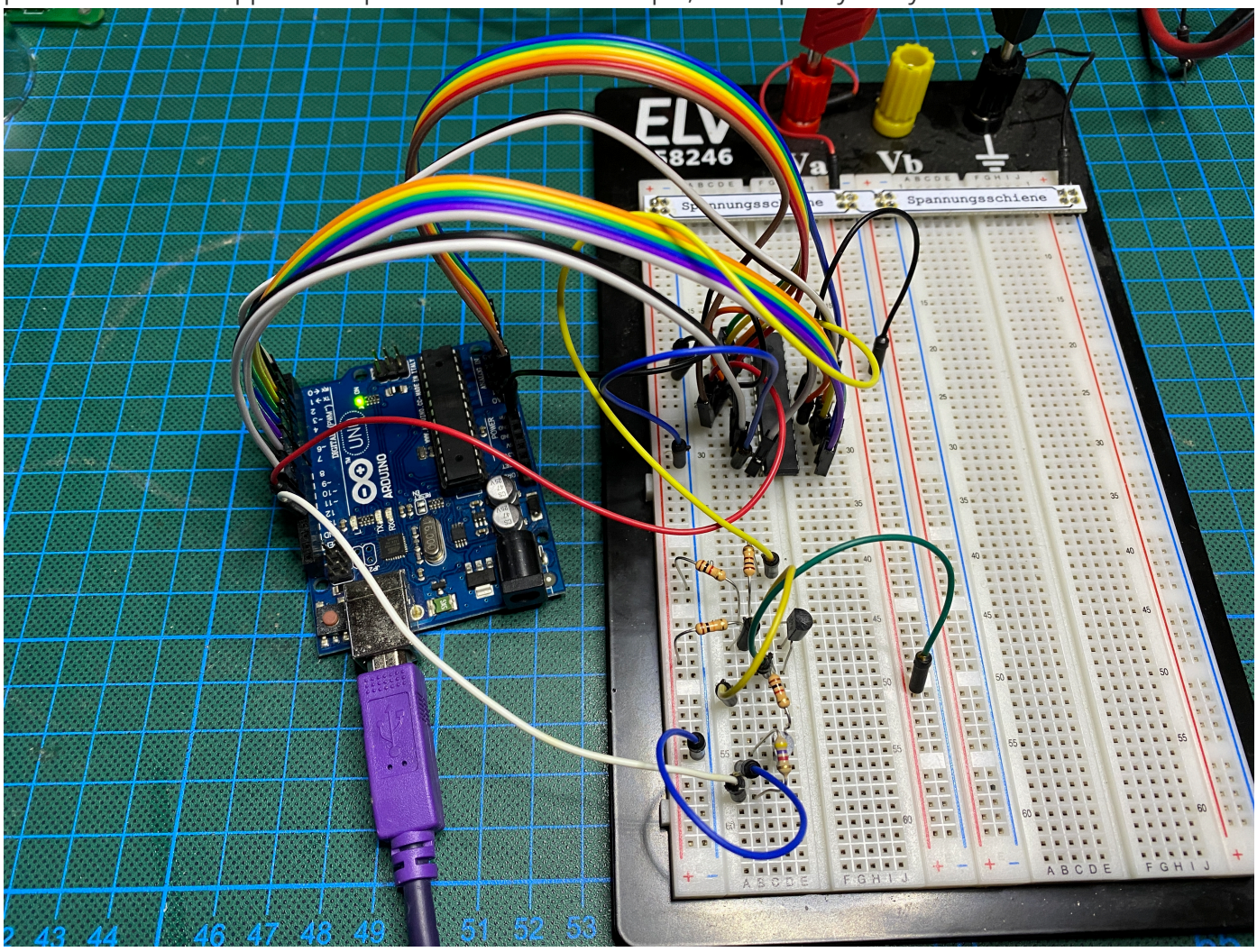
HVPP-pin mapping

ATmegaX8 Pin	Function	Arduino
1	RESET 12V	
2	N.C.	
3	RDY	A4
4	OEi	A3
5	WRi	A2
6	BS1	A1
7	Vcc	12
8	GND	
9	XTAL1	A0
10	N.C.	
11	XA0	10
12	XA1	11
13	PAGEL->GND	
14	DATA0	9
15	DATA1	8
16	DATA2	7
17	DATA3	6
18	DATA4	5
19	DATA5	4
20	Vcc	12
21	N.C.	
22	GND	
23	DATA6	3
24	DATA7	2
25	BS2	A5
26	N.C.	
27	N.C.	
28	N.C.	



Made with Fritzing.org

When this is put to work in reality, it can look as follows. Check the wiring twice before applying the external power. If 12 volt is applied to a pin that is not the RESET pin, the chip may easily die.



After you made all the connections (and double checked them!), open the Arduino monitor window, switch to 19200 baud, switch the external power supply on, and press reset on the Arduino. You are now in ***interactive rescue mode*** and can do a lot of things (see below).

Using RescueAVR on the Fusebit Doctor

The Fusebit Doctor can be run stand-alone or connected to a computer. In the stand-alone mode, after power-up, all LEDs are off and you can insert a chip. After pressing the button, the board will first try to recognize the chip:

- green LED on for three seconds: chip has been successfully recognized,
- green LED on for one second, then red LED on for three seconds: chip has been rcognized, but there is not enough information in the firmware to resurrect it,
- red LED is on for three seconds: no chip recognized.

After having recognized the MCU, the board tries to reset all lock bits and then tries to set the fuses to a safe default setting. If successful, the green LED flashes for 5 seconds, otherwise the red LED flashes for 5 seconds. If unsuccessful, you can try to set the erase jumper, which allows for erasing the entire chip in order to recover it.

If the serial line on the board is connected to a computer using 19200 baud (no parity, 1 stop-bit) then you can use the ***interactive rescue mode***, which gives you more control than the stand-alone mode.

Interactive Rescue Mode

When switched on or after a reset, the sketch will try to determine what kind of programming mode the MCU uses and which MCU is connected. If unsuccessful, the following is displayed:

```
No chip found!
Insert chip and reset or give details.

Choose from:
  P - HVPP
  T - HVPP for Tiny
  S - HVSP
  R - Start again
Choice:
```

When this message is shown, you either forgot to insert the MCU, the wiring is wrong, the external power supply is not switched on, or the chip is badly damaged. In the latter case, you might try then to select the programming mode, where *HVPP* is the high-voltage *parallel* programming mode for ATmegas, *HVPP for Tiny* is the same mode for ATtiny2313 and 4313 (but PAGEL and BS1 are both controlled by BS1, and BS2 and XA1 are both controlled by BS2, so PAGEL and XA1 should not be connected to the chip), *HVSP* is the high-voltage *serial* programming mode for ATtinyX4 and ATtinyX5. After having selected a programming mode, you can try to set fuses and lock bits. However, I have never been successful when the MCU could not be identified anymore. In any case, it is more likely that there is a wiring error or you forgot to plug the MCU into the socket (or breadboard).

Usually, the chip is detected and something along the following line is printed.

```
Signature: 1E910A
MCU name:  ATtiny2313
Current L/H/E-Fuses: 63 DF FF
Default L/H/E-Fuses: 62 DF FF
Current lock byte:   FF
Oscillator calibr.:  61
```

You can then choose from the following menu.

```
Choose:
  T - Try to resurrect chip by all means
  E - Erase chip
  D - Burn default fuse values
  L - Change low fuse
  H - Change high fuse
  X - Change extended fuse
  K - Change lock byte
  R - Restart
Action:
```

If you are only interested in unbricking your chip, press 'T'. This tries to unlock the chip. If this is not possible, it will try to erase the chip (if the 'chip erase' jumper on the Fuse-Doctor board is set). After it, it will try to reset the fuses to their default value. If 'T' does not help, you can probably say 'good bye' to the chip.

The other options are all self-explanatory. If you want to change individual fuses, you may want to consult the online fuse calculator AVR® Fuse Calculator by Mark Hämmerling.

Appendix A: Supported MCUs

In principle, the sketch should be able to work with almost all classic AVR chips (i.e., those that can be programmed using ISP). Currently, only a (very large) subset is supported, but it is very easy to extend the coverage of the sketch. If you have a particular MCU type you want to program which is not included, send me a note. Here are the MCUs that are already covered:

- AT90S2313,
- AT90S2333, AT90S4433
- AT90S2323, AT90S2343
- AT90S4434, AT90S8535
- AT90S8515,
- ATtiny11
- ATtiny12, ATtiny22
- ATtiny13
- ATtiny43U
- ATtiny2313, ATtiny4313
- ATtiny24, ATtiny44, ATtiny84
- ATtiny441, ATtiny841
- ATtiny15, ATtiny25, ATtiny45, ATtiny85
- ATtiny26
- ATtiny261, ATtiny461, ATtiny861
- ATtiny87, ATtiny167
- ATtiny28, ATtiny48, ATtiny88
- ATtiny828
- ATtiny1634
- ATmega8515
- ATmega8535
- ATmega8, ATmega8HVA, ATmega8HVB
- ATmega16, ATmega16HVA, ATmega16HVB, ATmega32, ATmega32C1, ATmega32HVB, ATmega64, ATmega64C1, ATmega64HEV2, ATmega64M1, ATmega128, ATmega128RFA1
- ATmega162

- ATmega48, ATmega48P, ATmega48PB, ATmega88, ATmega88P, ATmega88PB, ATmega168, ATmega168P, ATmega16PB, ATmega328, ATmega328P, ATmega328PB
- ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
- ATmega164, ATmega164P, ATmega324A, ATmega324P, ATmega324PA, ATmega644, ATmega644P, ATmega644RFR2, ATmega1284, ATmega1284Pm ATmega1284RFR2
- ATmega8U2, ATmega16U2, ATmega32U2
- ATmega16U4, ATmega32U4

Appendix B: Wiring tables for some MCU types

The wiring tables below apply only to the SOIC/DIP versions of the respective chips.

ATtiny11, 12, 13, 15, 22, 25, 45, 85, AT90S2323, 2343 (HVSP)

ATtiny pin	Function	Arduino UNO pin name
1	!RESET(12V switchable)	(triggered by 13)
2	SCI	A5
3		
4		
5	SDI	A0
6	SII	A1
7	SDO	A2
8	Vcc	12

ATtiny24, 44, 84, 441, 841 (HVSP)

ATtiny pin	Function	Arduino UNO pin name
1	Vcc	12
2	SCI	A5
3		
4	!RESET(12V)	(triggered by 13)
5		
6		
7	SDI	A0
8	SII	A1
9	SDO	A2
10		
11	GND	
12	GND	
13	GND	
14	GND	

ATtiny2313 and 4313, AT90S2313 (tiny HVPP)

ATtiny Pin	Function	Arduino UNO pin name
1	!RESET (12V switchable)	(triggered by 13)
2		
3	RDY	A4
4		
5	XTAL1	A0
6	!OE	A3
7	!WR	A2
8	BS1	A1
9	XA0	10
10	GND	GND
11	BS2(=XA1)	A5
12	DATA0	9
13	DATA1	8
14	DATA2	7
15	DATA3	6
16	DATA4	5
17	DATA5	4
18	DATA6	3
19	DATA7	2
20	Vcc (5V)	12

ATtiny26, ATtiny261, ATtiny461, ATtiny861 (HVPP tiny)

ATtiny pin	Function	Arduino pin name
1	!WR	A2
2	XA0	10
3	BS2(=XA1)	A5
4	BS1	A1
5	Vcc(5V)	12
6	GND	GND
7	XTAL1	A0
8	!OE	A3
9	RDY	A4
10	!RESET(12V)	(triggered by 13)
11	DATA7	2
12	DATA6	3
13	DATA5	4
14	DATA4	5
15	Vcc(5V)	12
16	GND	GND
17	DATA3	6
18	DATA2	7
19	DATA1	8
20	DATA0	9

ATtiny87, ATtiny167 (HVPP tiny)

ATtiny pin	Function	Arduino UNO pin name
1	DATA0	9
2	DATA1	8
3	DATA2	7
4	DATA3	6
5	Vcc(5V)	12
6	GND	GND
7	DATA4	5
8	DATA5	4
9	DATA6	3
10	DATA7	2
11		
12	RDY	A4
13	!OE	A3
14	XTAL1	A0
15	Vcc(5V)	12
16	GND	GND
17	BS1	A1
18	BS2 (=XA1)	A5
19	XA0	10
20	!WR	A2

ATtiny1634 (HVPP tiny)

ATtiny pin	Function	Arduino UNO pin name
1		
2	DATA7	2
3	DATA6	3
4	DATA5	4
5	DATA4	5
6	DATA3	6
7	DATA2	7
8	DATA1	8
9	DATA0	9
10	GND	GND
11	Vcc(5V)	12
12	XTAL	A0
13		
14		
15	!RESET(12V)	(triggered by 13)
16	!OE	A3
17	!WR	A2
18	BS1	A1
19	XA0	10
20	BS2(=XA0)	A5

ATmega8, ATmegaX8 (HVPP)

ATmega pin	Function	Arduino UNO pin name
1	!RESET(12V)	(triggered by 13)
2		
3	RDY	A4
4	!OE	A3

5	!WR	A2
6	BS1	A1
7	Vcc	12
8	GND	GND
9	XTAL1	A0
10		
11	XA0	10
12	XA1	11
13	PAGEL	GND
14	DATA0	9
15	DATA1	8
16	DATA2	7
17	DATA3	6
18	DATA4	5
19	DATA5	4
20	Vcc	12
21		
22	GND	GND
23	DATA6	3
24	DATA7	2
25	BS2	A5
26		
27		
28		

ATmegaX4 (HVPP)

ATmega pin	Function	Arduino UNO pin name
1	DATA0	9
2	DATA1	8
3	DATA2	7
4	DATA3	6
5	DATA4	5
6	DATA5	4
7	DATA6	3
8	DATA7	2
9	!RESET(12V)	(triggered by 13)
10	Vcc(5V)	12
11	GND	GND
12		
13	XTAL1	A0
14		
15	RDY	A4
16	!OE	A3
17	!WR	A2
18	BS1	A1
19	XA0	10
20	XA1	11
21	PAGEL	GND
22		
23		
24		
25		
26		

27		
28		
29		
30	Vcc(5V)	12
31	GND	GND
32		
33		
34		
35		
36		
37		
38		
39		
40	BS2	A5