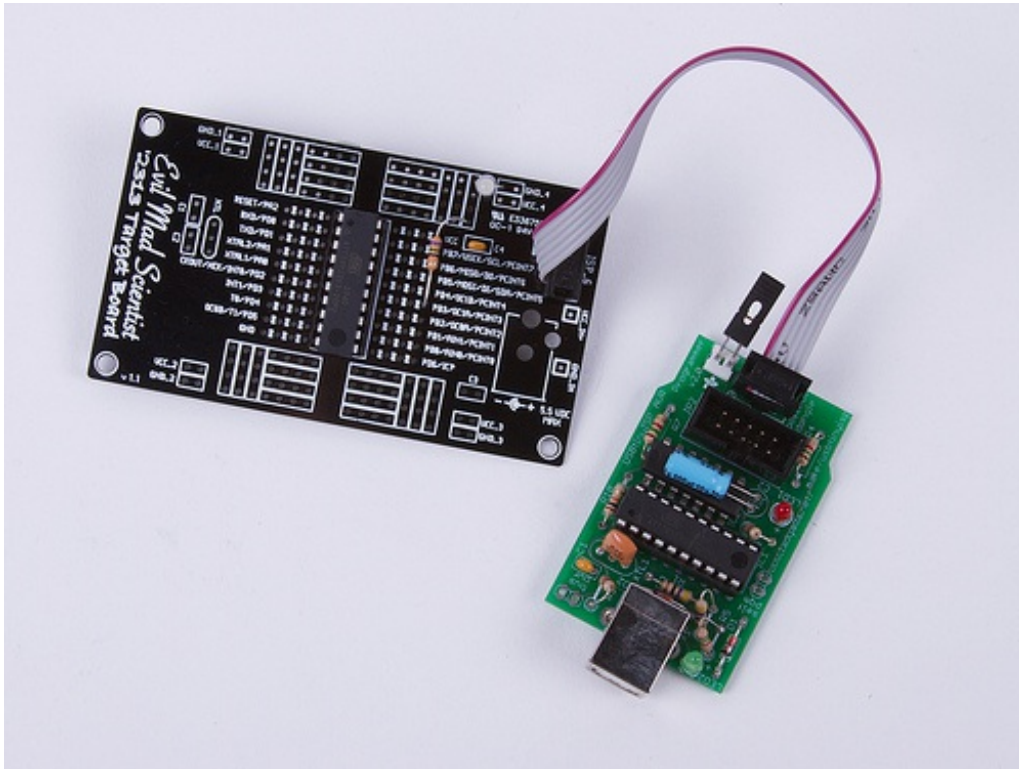


Blink an LED with an AVR

by RobotGrrl



Getting an AVR to blink might seem like an incredibly difficult task compared to the usual Arduino blink, but it really isn't! In this post we will be uploading a basic blink example to an ATtiny2313. This is perfect for projects where using an Arduino would be over the top. So let's get started!

You will need:

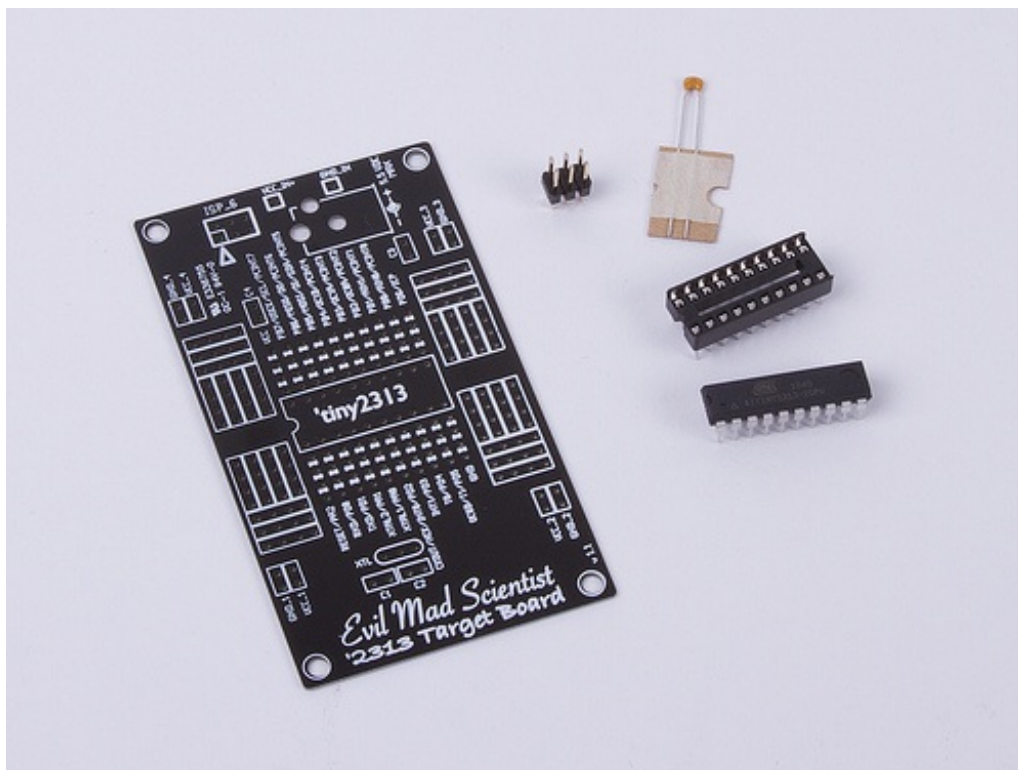
- [ATtiny2313 kit](#) or a [minimalist target board](#)!
- [USBTinyISP](#)
- Soldering iron & solder
- LED & resistor

We will be working on a Mac for this example. There are options available for Windows and Linux as well, the gist is the same as described in this post.

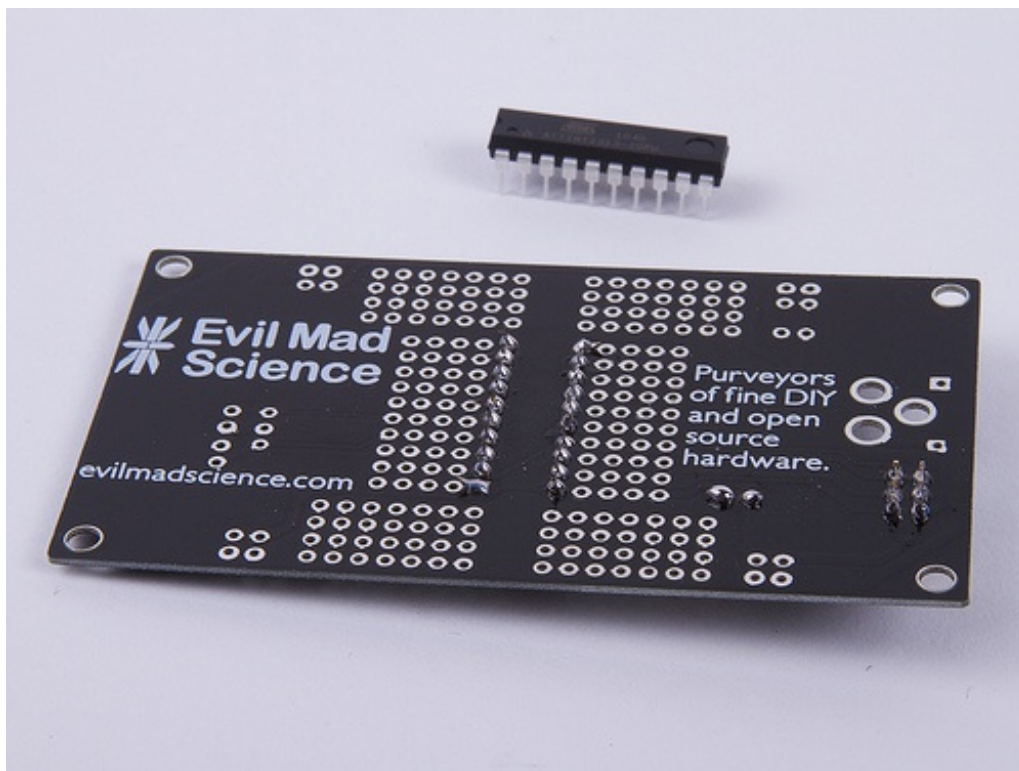
Get your [ATtiny2313 dev board kit](#) and let's open it up!



The ATtiny2313 is the main chip in this kit. It is worthwhile to flip through the datasheet and get an overview of how it works. [Look at the datasheet here.](#)

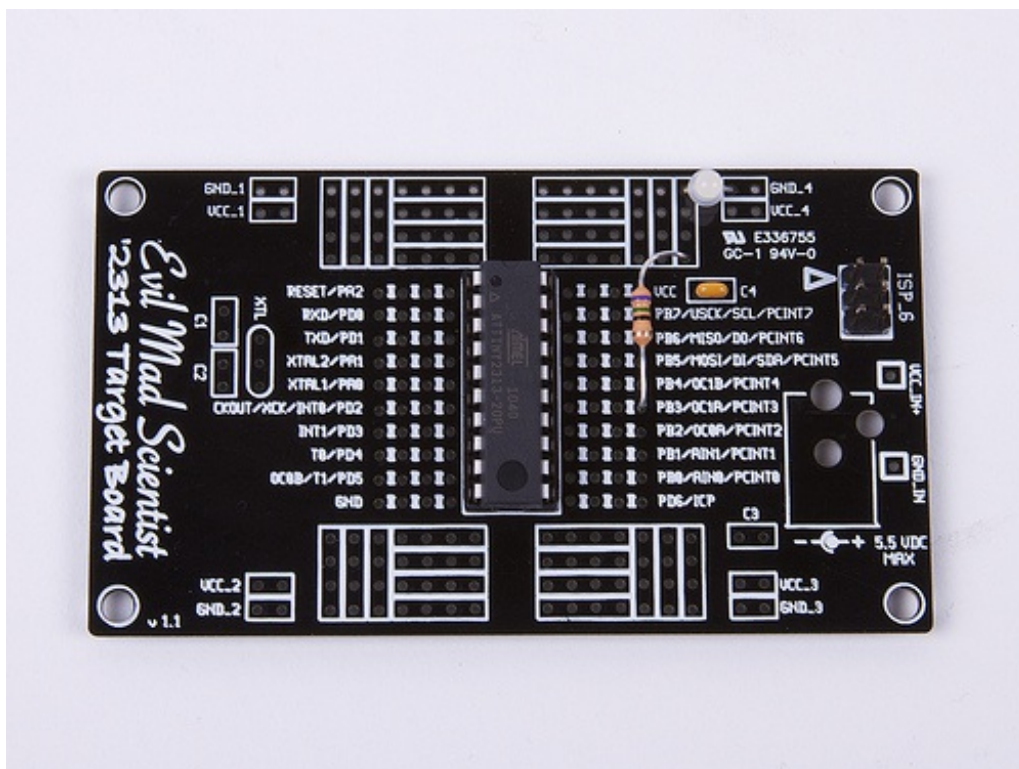


Solder in all of the pieces. Be sure to match up the DIP socket with the notch.

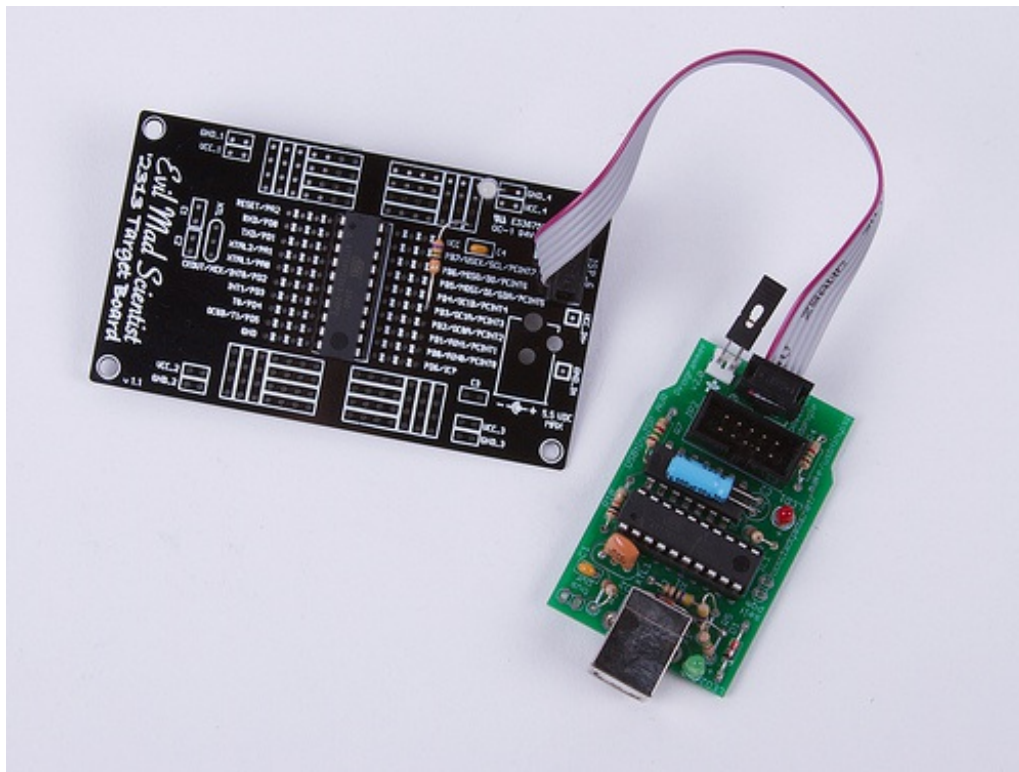


Time to add in an LED and a resistor. For this example, we are going to go with pin PB3. This pin is special because it has a 16 bit timer! This makes it quite delightful to use for fading.

Although the pin numbers are “weird” on this board, there are many advantages to this that you will find as you make more projects with AVR.



If you haven't assembled your [USBTinyISP](#), now is a good time to do so. You can [follow a tutorial here](#). The arrow on the wire attacher matches with the arrow on the board.



Time for the software part! There is a very handy package available that installs everything together called Crosspack. [Go and download the latest one here](#) and install it.

objective development
product

CrossPack for AVR® Development

CrossPack is a development environment for Atmel's AVR® microcontrollers running on Apple's Mac OS X, similar to AVR Studio on Windows. It consists of the GNU compiler suite, a C library for the AVR, the AVRDUDE uploader and several other useful tools.

Features

- Does not depend on Xcode for building AVR code.
- Runs on Mac OS X 10.6 and higher.
- Supports 8 bit AVR microcontrollers including XMEGA devives.
- Includes patches to gcc for new devices not yet supported by gcc's main distribution.
- Includes gdb for debugging with simulavr and avarice.

For a list of included software packages and versions see the [Release Notes](#).

Download

Cros

- On
- De
- Re
- Fe

Rela

V-US

micro

Spor

To get the code from the computer onto the chip, the procedure is to compile and then upload the code. We can handle this using a makefile. With a short terminal command, it will do everything that we detail it to do. [Here is the makefile we used for this project.](#)

The main parts of the makefile that have to change (if you were using a different chip or programmer) are the

device, clock, programmer, objects, and fuses.

code

MAKE SURE THE FUSES ARE CORRECT! The values in the makefile we are using are set for the ATtiny2313. You can actually brick your avr if you mess up the fuses. This is one thing that Arduino protects you from. However if you exercise diligence with regards to the fuses, it should not be an issue. [Check out the AVR fuse calculator here.](#)

If you are using a different programmer from the USBTinyISP, then you will have to change the PROGRAMMER variable.

```
# OBJECTS ..... The object files created from your source files.
#                  usually the same as the list of source files with
# PROGRAMMER ... Options to avrdude which define the hardware you
#                  uploading to the AVR and the interface where this
#                  is connected. We recommend that you leave it unchanged
#                  add settings like this to your ~/.avrduderc file
#                  default_programmer = "stk500v2"
#                  default_serial = "avrdoper"
# FUSES ..... Parameters for avrdude to flash the fuses appropriately

DEVICE    = attiny2313
CLOCK     = 8000000
PROGRAMMER = -c usbtiny -p attiny2313
OBJECTS    = blinklight.o
FUSES     = -U hfuse:w:0xdf:m -U lfuse:w:0x64:m

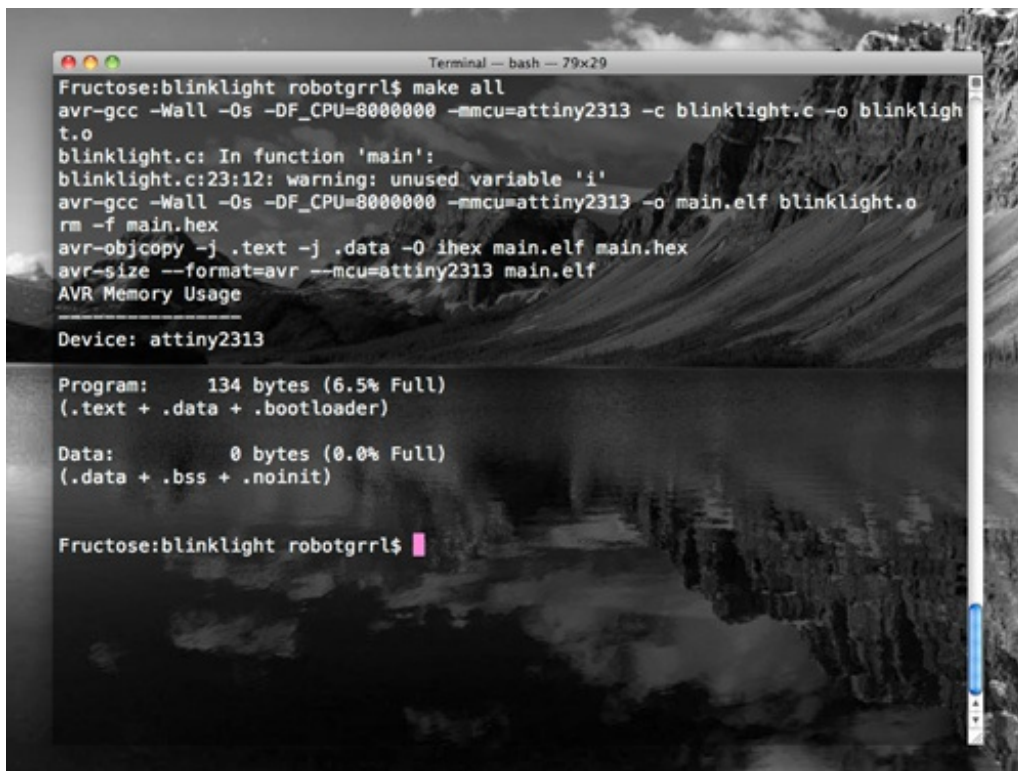
# For computing fuse byte values for other devices and options see
# the fuse bit calculator at http://www.engbedded.com/fusecalc/

# Tune the lines below only if you know what you are doing:

AVRDUDE = avrdude $(PROGRAMMER) -p $(DEVICE)
COMPILE = avr-gcc -Wall -Os -DF_CPU=$(CLOCK) -mmcu=$(DEVICE)

# symbolic targets:
```

Now let's test the makefile by opening Terminal (/Applications/Utilities/Terminal.app) and entering the command `make all`. You should see text as seen in the screenshot below. If you do not, be sure to fix this before continuing by checking the installation of Crosspack.



```
Fructose:blinklight robotgrrl$ make all
avr-gcc -Wall -Os -DF_CPU=8000000 -mmcu=attiny2313 -c blinklight.c -o blinklight.o
blinklight.c: In function 'main':
blinklight.c:23:12: warning: unused variable 'i'
avr-gcc -Wall -Os -DF_CPU=8000000 -mmcu=attiny2313 -o main.elf blinklight.o
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avr-size --format=avr --mcu=attiny2313 main.elf
AVR Memory Usage
Device: attiny2313

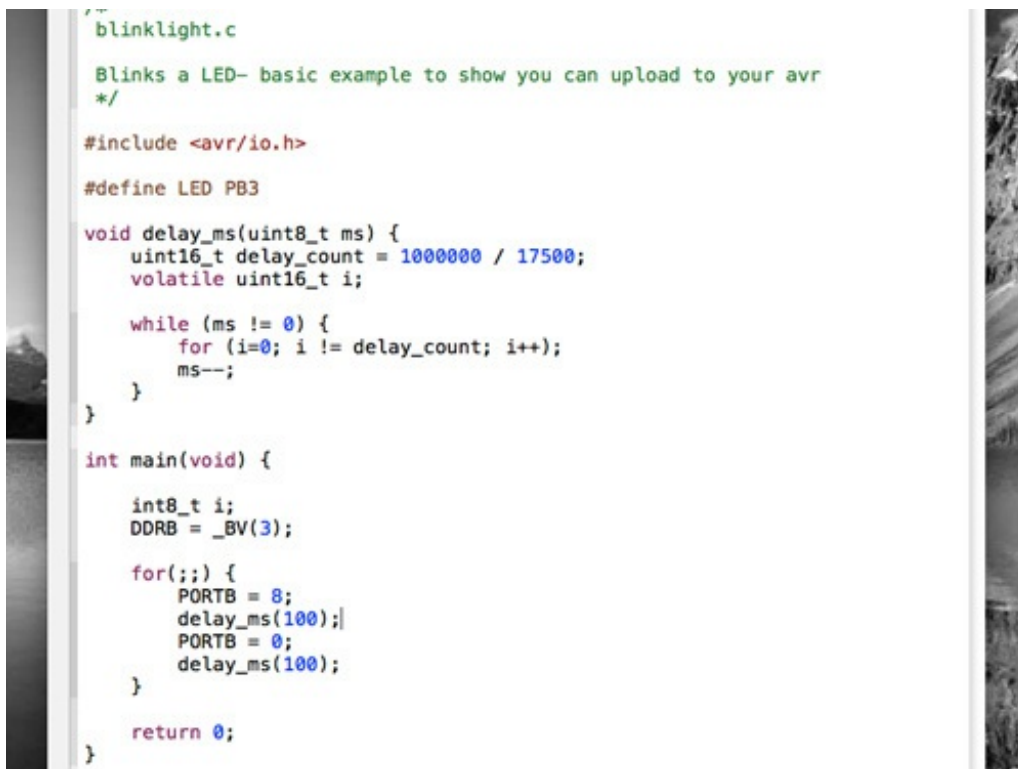
Program: 134 bytes (6.5% Full)
(.text + .data + .bootloader)

Data: 0 bytes (0.0% Full)
(.data + .bss + .noinit)

Fructose:blinklight robotgrrl$
```

Finally for the interesting part, writing the code to be uploaded to the chip! You can [look at our code here](#).

The `_BV()` is an avr macro to do a left bitshift. To turn on and off the LED, we set `PORTB` to 8 or 0 respectively. The `delay_ms` function is from [here](#), and the 17500 number comes from the computational time that it takes the loop to run. Cool eh!



```
/*
 * blinklight.c
 *
 * Blinks a LED- basic example to show you can upload to your avr
 */

#include <avr/io.h>

#define LED PB3

void delay_ms(uint8_t ms) {
    uint16_t delay_count = 1000000 / 17500;
    volatile uint16_t i;

    while (ms != 0) {
        for (i=0; i != delay_count; i++);
        ms--;
    }
}

int main(void) {
    int8_t i;
    DDRB = _BV(3);

    for(;;) {
        PORTB = 8;
        delay_ms(100);
        PORTB = 0;
        delay_ms(100);
    }

    return 0;
}
```

Enter the `make all` command into Terminal again since the code has changed since we last compiled. Be sure to fix any errors before proceeding. Plug in the USBtinyISP, and now let's upload the code! Enter the command `make install`. You should see some long text appear. The below screenshot shows the end part of the code.

If all was good, you now have a blinking LED using an avr! Totally cool, congrats! If something is not working, go back and check the makefile to see that it is okay.

Now challenge yourself to go further and play around more! There is a most excellent [list of resources available here](#). What will YOU make with your avr? Let us know on [Google+](#) and [Twitter](#), or come show it off on the [Robot Party](#)! Happy hacking!

