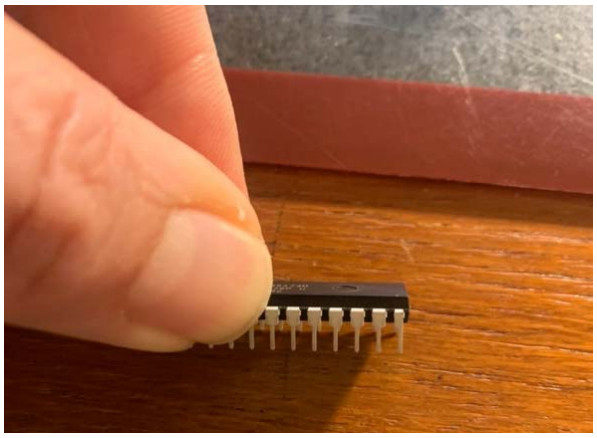
Tutorial: Programming the ATmega328P with the Sparkfun Pocket Programmer

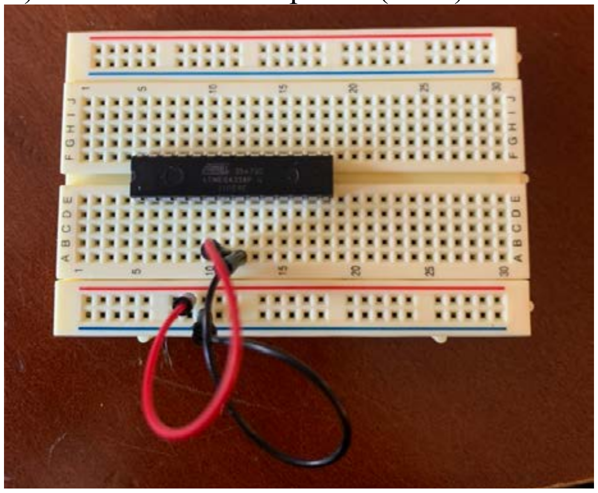
April 3, 2022

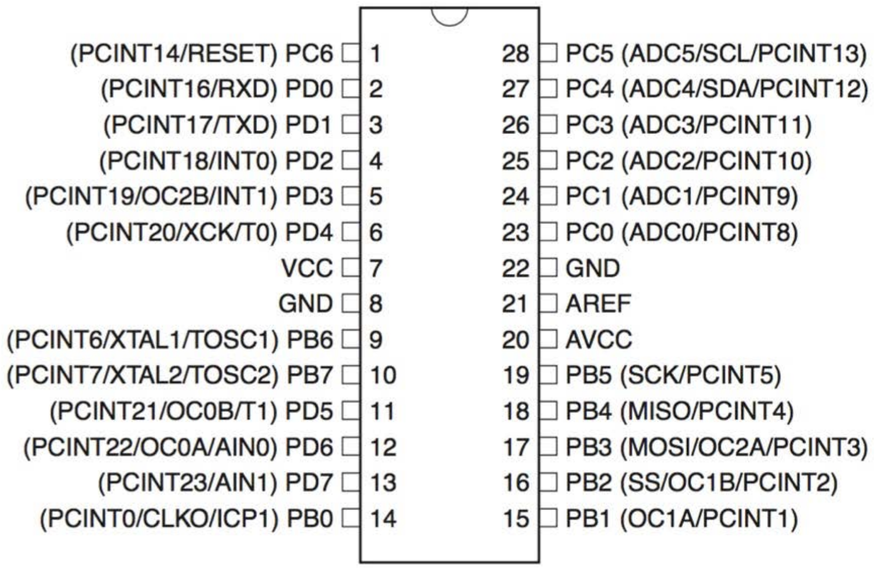
What you will need for this tutorial:
1) one Sparkfun AVR Pocket Programmer (aka programmer)
2) one programmer flat cable
3) one MiniUSB cable to connect the programmer to your computer's USB port
4) one ATmega328P, 28 pin plastic DIP package, DIP = Dual In-line Package (aka 328P)
5) six wires to connect programmer to ATmega328P
6) one solderless breadboard
7) LED, resistor, and a few more wires
8) 16 MHz crystal, & two 22-pf capacitors
9) USB to alligator clip leads
10) USB "wall wart" AC adapter
11) 7805 voltage regulator
12) 9V battery
13 9V battery clip and leads

Step 1: **Plug your 328P into your solderless breadboard.** You may need to gently press the pins on each side of the chip inward to make the chip fit into the breadboard. You can do this by pressing all the pins of a row against a hard surface.

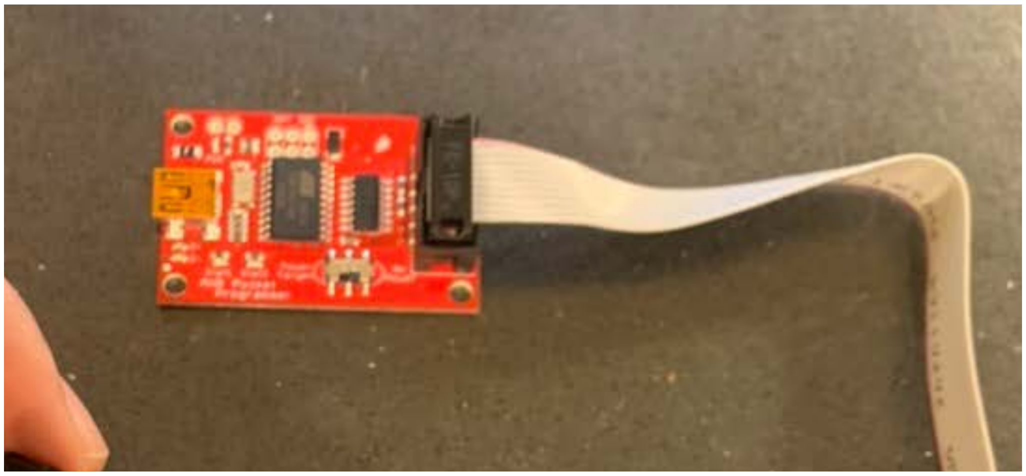


Step 2: **Connect pins 7 and 8 of your 328P to the VCC and GND** power rails on the breadboard. Use a black wire for ground (GND). Use a red wire for power (VCC). Refer to the pin diagram of the 328P.

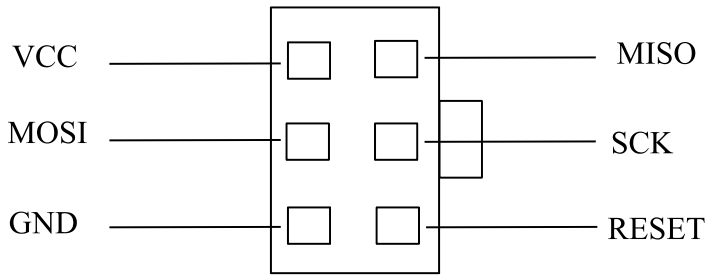




Step 3: **Plug the 10 pin connector** at one end of the flat cable into the Programmer port. The connector has a notch on one side allowing to only fit in one way.



Step 4: **The opposite end of the flat cable** is used to program the 328P using the Serial Programming Interface (SPI). The connecter is a socket with 6 holes. Locate the notch on the 6-pin connector:



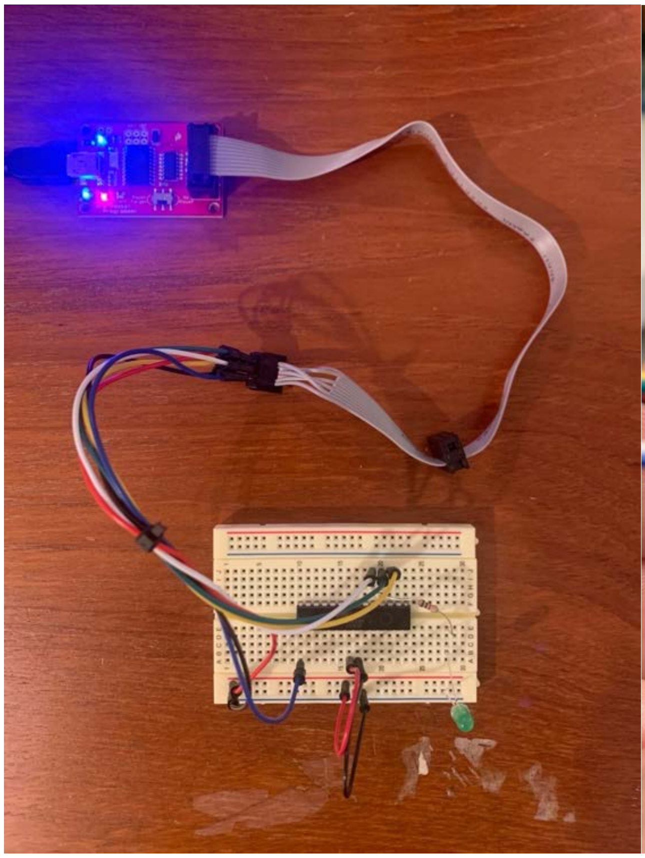
SPI 6-pin Programmer Socket

Step 5: **Make SPI connections from the 6-pin connector to your 328P** as follows:
1) red wire from Programmer VCC to breadboard VCC voltage rail
2 Programmer MOSI (Main Out Secondary In) to 328P PB3 (MOSI/OC2A/PCINT3) (Pin 17)
3) black wire from Programmer GND to breadboard GND rail
4) Programmer MISO (Main In Secondary Out) to PB4 (MISO/PCINT4) (Pin 18)
5) Programmer SCK to 328P PB5 (SCK/PCINT4) (Pin 19)
6) Progremmer RESET to PB6 (PCINT14/RESET) (Pin 1)

Use red and black wires where specified above. Use colors other than red and black for the other connections. Make a reference card noting all of your wire colors. Do not use red or black wires for anything other that power and ground. In my instructions, demos, photos, and videos, I will always use the following wire colors: VCC - red; MOSI - yellow; GND - black; MISO -green; SCK - white; RESET - blue. (see photos below)

Step 6: **We will program the 328P with code that blinks an LED connected to PB5**. Connect an LED with a current limiting resistor between PB5 (Pin 19) and the GND rail of the breadboard.

Step 7: **You may now connect the Pocket Programmer to your computer using a USB mini** cable provided in your course electronics kit. You will notice several LEDs glowing on the programmer. The programmer will provide 5V to power-up your ATmega328P. When the Pocket Programmer is disconnected from the 328P, we will use various other means to power the microcontroller and the circuit as described toward the end of this tutorial.



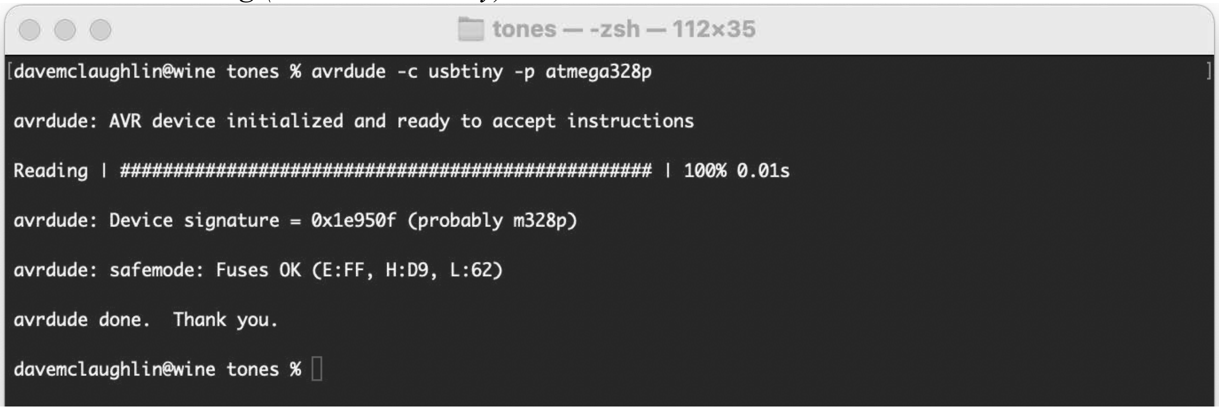
Step 8. **Install the windows driver** for the Programmer using the instructions provided by Sparkfun here: https://learn.sparkfun.com/tutorials/pocket-avr-programmer-hookup-guide/installing-drivers (If you are using macOS, you do not need to install a driver).

Step 9. **Verify that your programmer is able to communicate with the 328p** by typing the command "avrdude -c usbtiny -p atmega328p" from Terminal (macOS) or Command Prompt (Windows)

Here, usbtiny refers to the Pocket Programmer, while atmega328p refers to the 328p.

avrdude will respond by telling you that the AVR device is initialized and ready to accept instructions. It will also read the device signature, identifying the device as a 328p mcu.

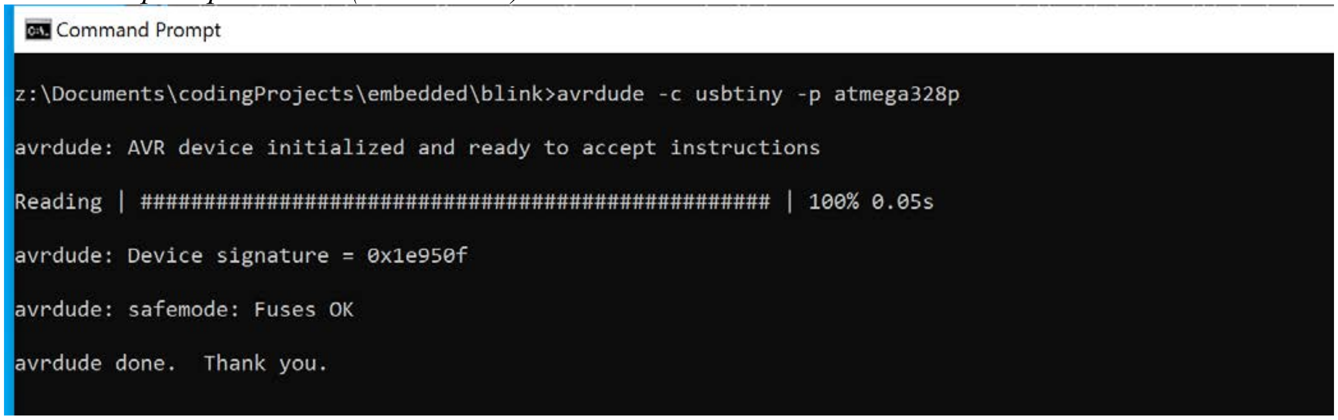
Terminal window dialog (macOS Monterey)



```
davemclaughlin@wine tones % avrdude -c usbtiny -p atmega328p

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e950f (probably m328p)

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

avrdude done.  Thank you.

davemclaughlin@wine tones %
```

FYI. on my macOs operating system, avrdude reads the E, H, and L fuse settings as shown above. On my windows operating system, avrdude does not read these fuses with the command given in this step (below). Fuses are discussed in more detail in the next step.

Command prompt window (windows 10)

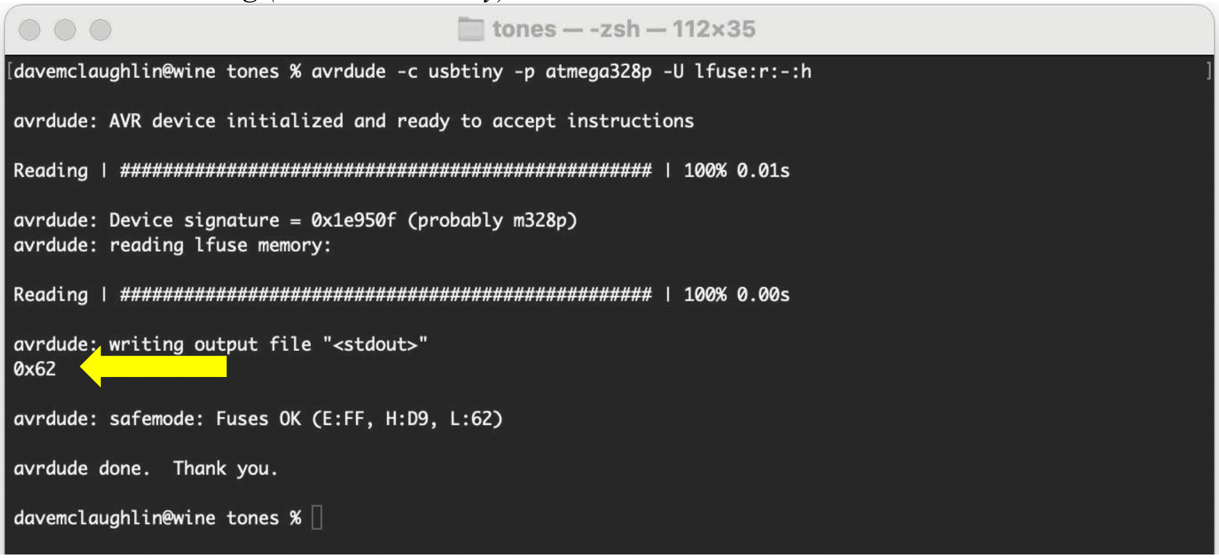


```
z:\Documents\codingProjects\embedded\blink>avrdude -c usbtiny -p atmega328p

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.05s

avrdude: Device signature = 0x1e950f

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Step 10. **Read Fuse settings**: The 328P has 3 non-volatile "fuses" that establish several key operating parameters for the device. (These are not like the fuses included in power supplies and other circuits, that "blow" or open-circuit when a certain current is reached; instead, they are simply important registers that contain critical settings about the microcontroller. Once values are written to the fuses, the values are retained even when the device is erased, reset or powered down. Fuses need to be programmed carefully, since it is possible to render a device unusable with certain fuse settings.) We are interested in the lower fuse setting which configures the clock of the 328p. Enter the following command to read the lower fuse setting:

```
avrdude -c usbtiny -p atmega328p -U lfuse:r:-:h
```

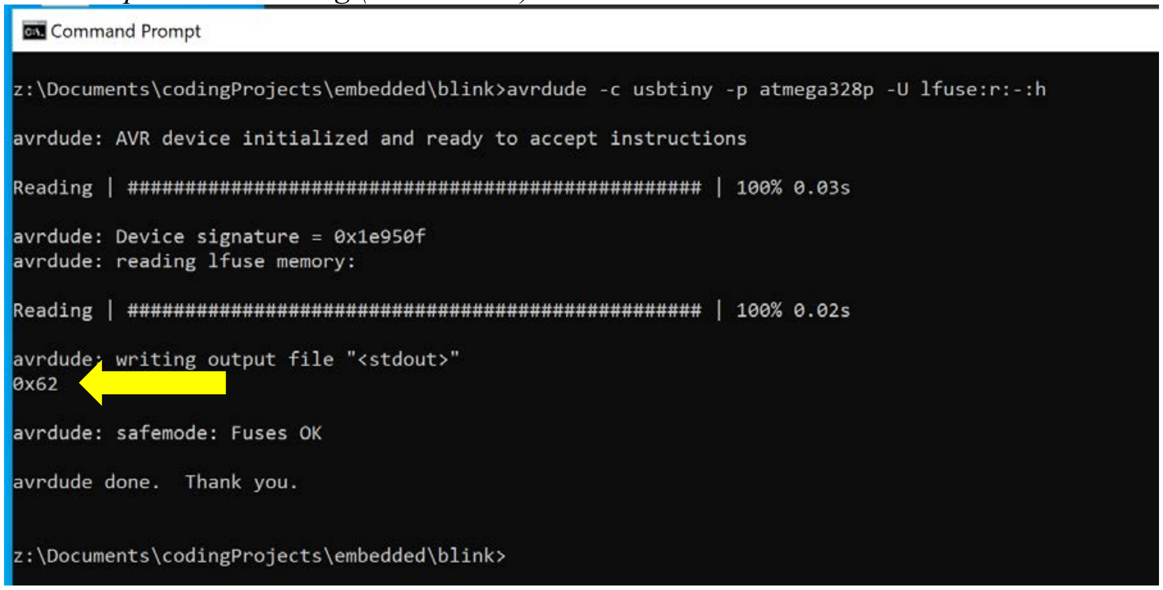
Terminal Window dialog (macOS Monterey)



```
davemclaughlin@wine tones % avrdude -c usbtiny -p atmega328p -U lfuse:r:-:h

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading lfuse memory:

Reading | ################################################## | 100% 0.00s

avrdude: writing output file "<stdout>"
0x62

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

avrdude done.  Thank you.

davemclaughlin@wine tones %
```

The lower fuse, indicated by the yellow arrow, is set to hex value 62.

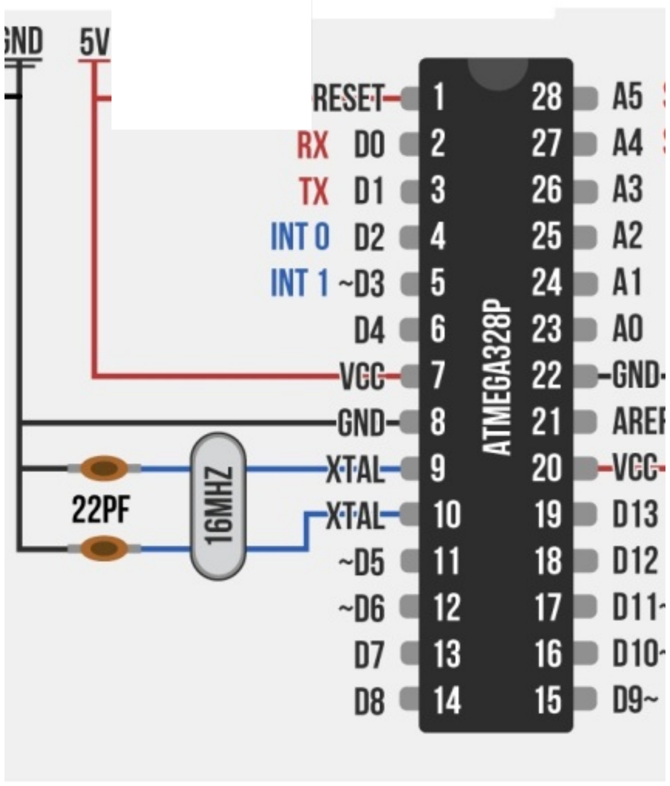
Command Prompt Window dialog (windows 10)



```
z:\Documents\codingProjects\embedded\blink>avrdude -c usbtiny -p atmega328p -U lfuse:r:-:h

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.03s

avrdude: Device signature = 0x1e950f
avrdude: reading lfuse memory:

Reading | ################################################## | 100% 0.02s

avrdude: writing output file "<stdout>"
0x62

avrdude: safemode: Fuses OK

avrdude done.  Thank you.


z:\Documents\codingProjects\embedded\blink>
```

The default lower fuse setting for a new 328P is factory programmed for a hex value of 62. *This indicates that the chip is configured to use an internal clock oscillator with frequency of 1 MHz.* You can keep this setting, but if you want to match the frequency of the ATmega328P chip in the Arduino Uno board, and make use of code you previously used with the Arduino Uno, you need to change the fuse to accept an external 16 MHz oscillator. Since much of our code involves timing, and we may want to run code on both the Arduino Uno and the 328P-on-breadboard, it makes sense to change the fuse setting on the 328P to use an external 16 MHz clock. Steps 11 and 12 walk you through this change.

Step 11. **Add a 16MHz crystal oscillator between 328P pins 9 and 10. Add two 22 pF capacitors between pins 9 and GND and 10 and GND.** (You received a 16 MHz crystal with your kit this semester. The 22 pF capacitors are in the Marston 221 lab and you should have picked them up, along with a 7805 voltage regulator, previously.)



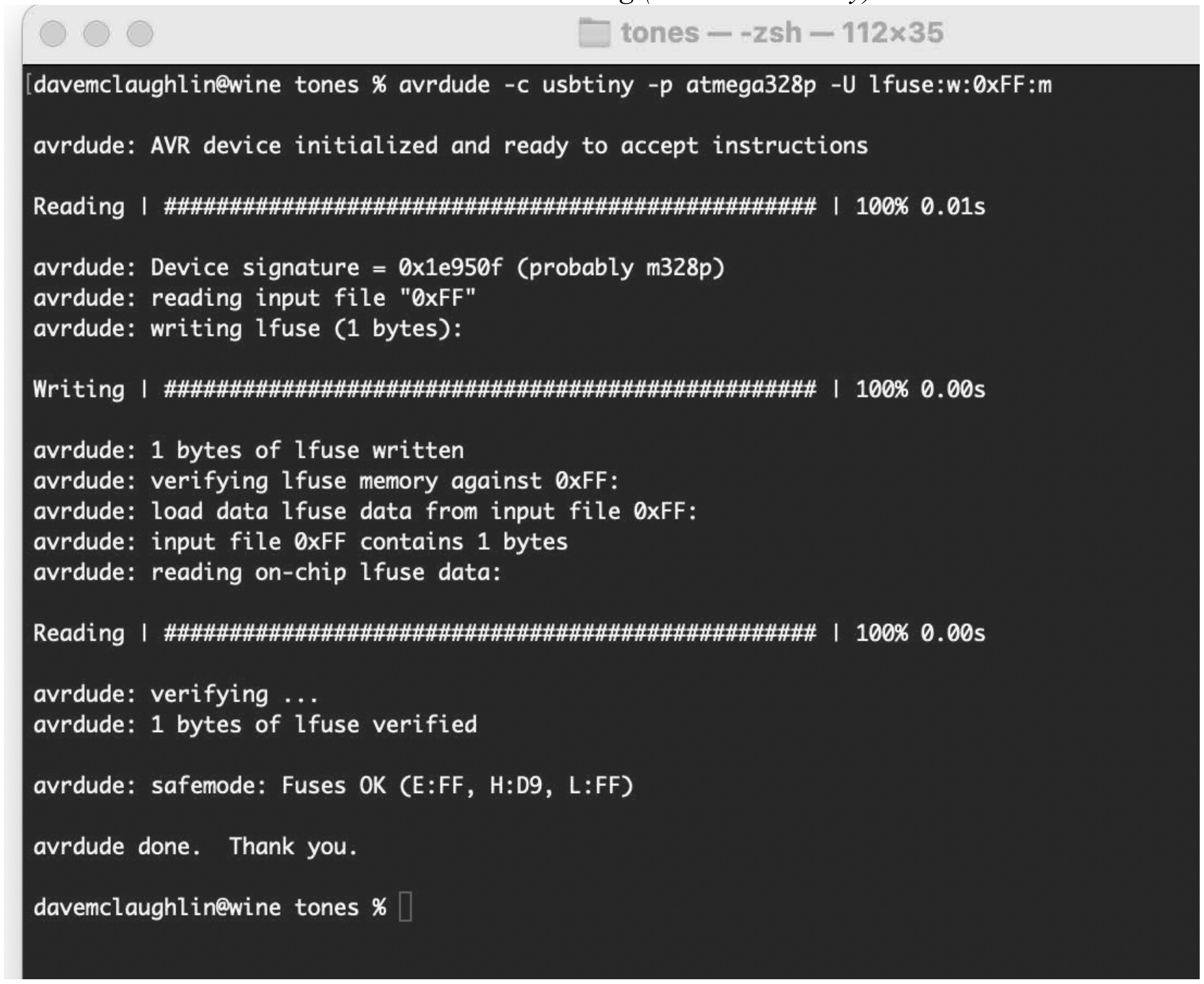
IMPORTANT: Do step 11 before you do step 12, otherwise you can make your 328P unusable.

Step 12. **Change the fuse settings of the 328P** to use an external clock. From a terminal (macOS) or command prompt (windows) window, type: avrdude -c usbtiny -p atmega328p -U lfuse:w:0xFF:m.

avrdude will respond by changing the lower fuse byte.

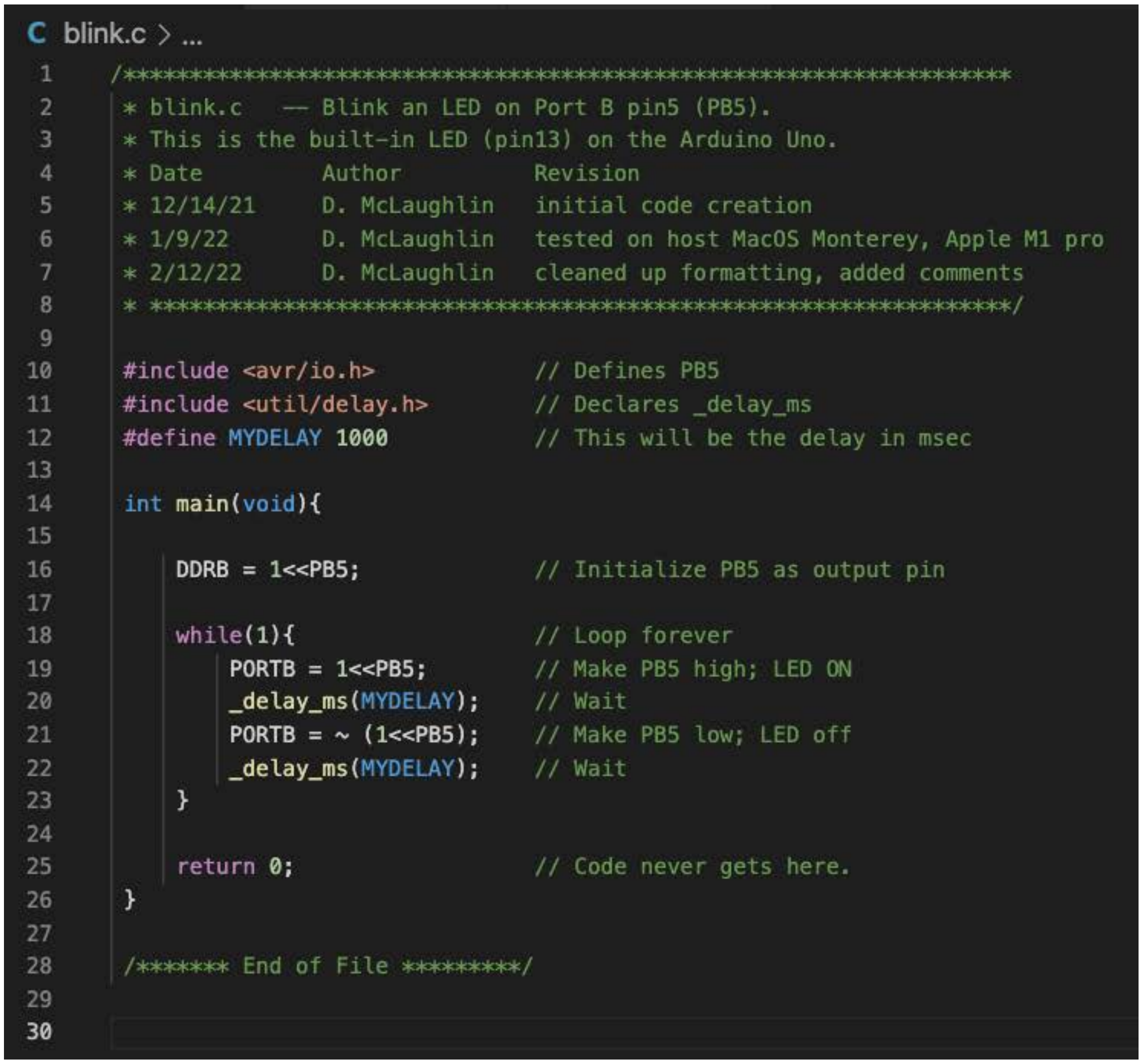
The Terminal window dialog for my macOS is shown below. The dialog for Command Prompt (windows) is similar and not shown.

Terminal Window Dialog (macOS Monterey)



```
davemclaughlin@wine tones % avrdude -c usbtiny -p atmega328p -U lfuse:w:0xFF:m

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file "0xFF"
avrdude: writing lfuse (1 bytes):

Writing | ################################################## | 100% 0.00s

avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0xFF:
avrdude: load data lfuse data from input file 0xFF:
avrdude: input file 0xFF contains 1 bytes
avrdude: reading on-chip lfuse data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of lfuse verified

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

avrdude done.  Thank you.

davemclaughlin@wine tones %
```

Step 13. **Write a program to blink an LED** on PB5 of the 328P. A suggested delay is 1000 ms.



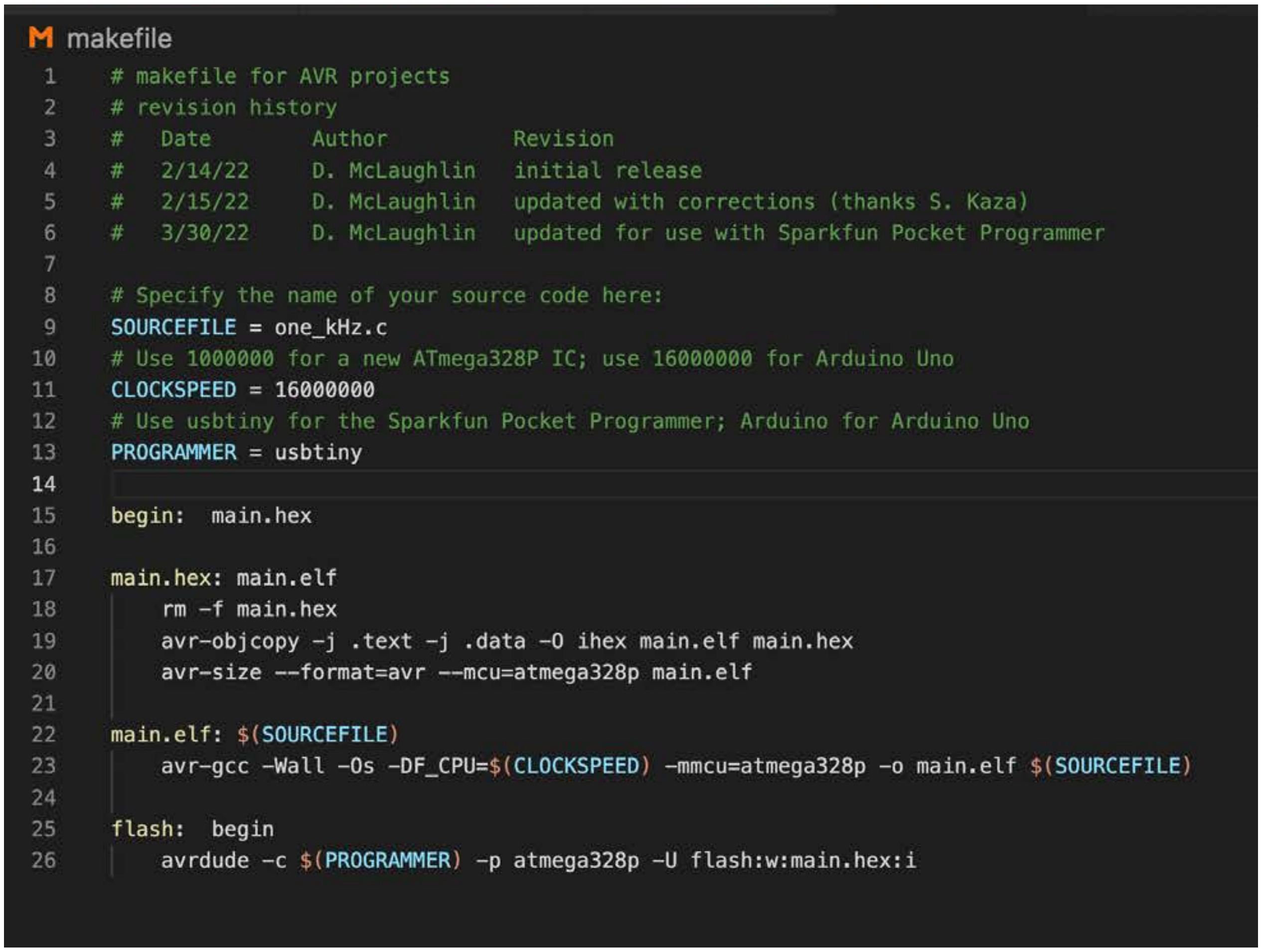
C blink.c > ...

```c
/*****************************************************************
 * blink.c   -- Blink an LED on Port B pin5 (PB5).
 * This is the built-in LED (pin13) on the Arduino Uno.
 * Date         Author          Revision
 * 12/14/21     D. McLaughlin   initial code creation
 * 1/9/22       D. McLaughlin   tested on host MacOS Monterey, Apple M1 pro
 * 2/12/22      D. McLaughlin   cleaned up formatting, added comments
 * ****************************************************************/

#include <avr/io.h>             // Defines PB5
#include <util/delay.h>         // Declares _delay_ms
#define MYDELAY 1000            // This will be the delay in msec

int main(void){

    DDRB = 1<<PB5;              // Initialize PB5 as output pin

    while(1){                   // Loop forever
        PORTB = 1<<PB5;         // Make PB5 high; LED ON
        _delay_ms(MYDELAY);     // Wait
        PORTB = ~ (1<<PB5);     // Make PB5 low; LED off
        _delay_ms(MYDELAY);     // Wait
    }

    return 0;                   // Code never gets here.
}

/******* End of File *********/

```

Step 14. **You need to make a few modifications to your makefile**

These instructions apply to the makefile previously used in ECE-231, Spring 2022.
(a) Change the programmer from *arduino* to *usbtiny* in the avrdude instruction
(b) If you have NOT changed the fuse settings as specified above, you need to change the clock speed from 16000000 to 1000000 in the avr-gcc instruction.
(c) as always, change the name of the SOURCEFILE variable to the name of your source code file.

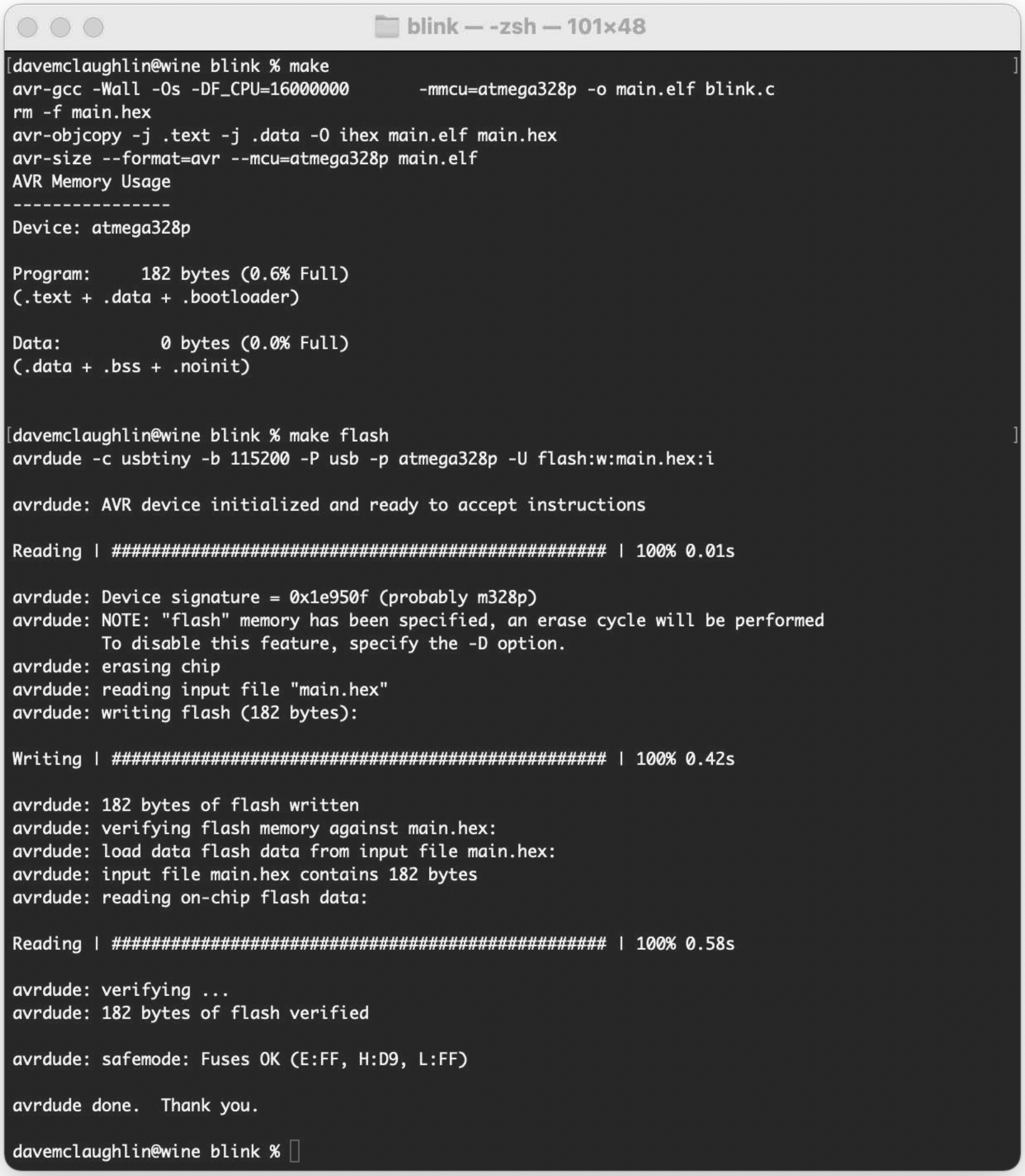
The following makefile has been rewritten with these changes You can use this makefile in place of the previous makefile you had been using. A copy of this makefile will be posted to moodle.



makefile

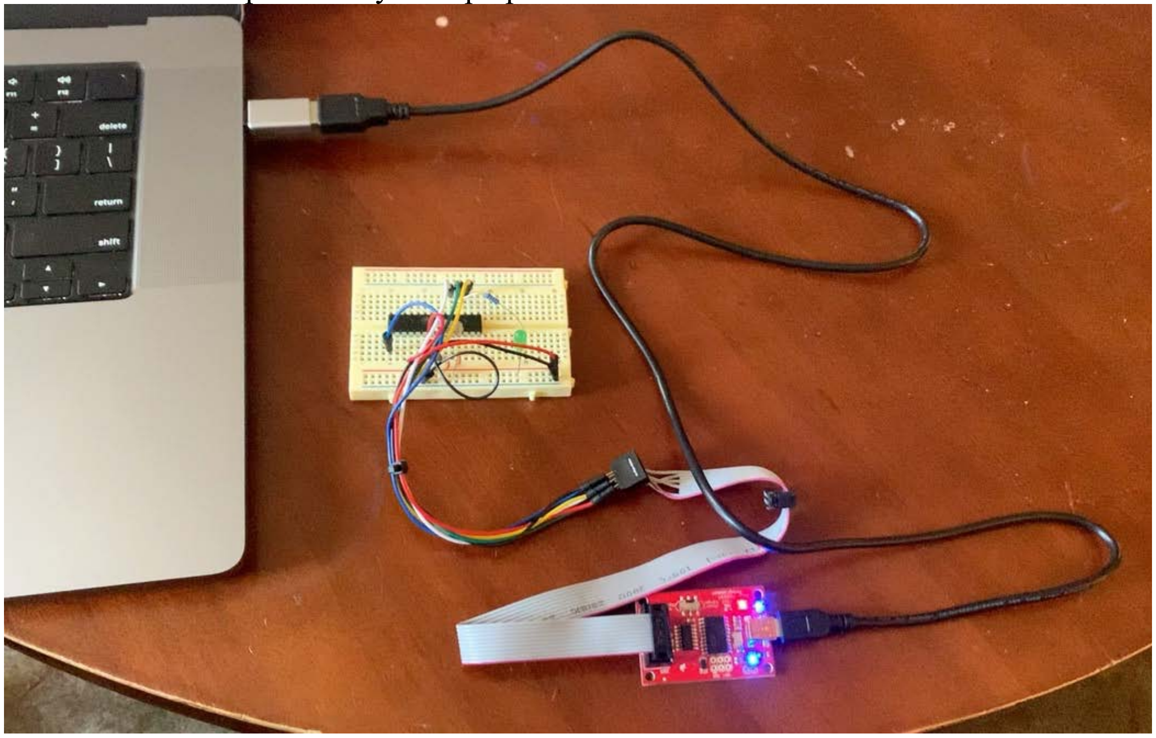
```
# makefile for AVR projects
# revision history
#   Date        Author          Revision
#   2/14/22     D. McLaughlin   initial release
#   2/15/22     D. McLaughlin   updated with corrections (thanks S. Kaza)
#   3/30/22     D. McLaughlin   updated for use with Sparkfun Pocket Programmer

# Specify the name of your source code here:
SOURCEFILE = one_kHz.c
# Use 1000000 for a new ATmega328P IC; use 16000000 for Arduino Uno
CLOCKSPEED = 16000000
# Use usbtiny for the Sparkfun Pocket Programmer; Arduino for Arduino Uno
PROGRAMMER = usbtiny

begin:  main.hex

main.hex: main.elf
	rm -f main.hex
	avr-objcopy -j .text -j .data -O ihex main.elf main.hex
	avr-size --format=avr --mcu=atmega328p main.elf

main.elf: $(SOURCEFILE)
	avr-gcc -Wall -Os -DF_CPU=$(CLOCKSPEED) -mmcu=atmega328p -o main.elf $(SOURCEFILE)

flash:  begin
	avrdude -c $(PROGRAMMER) -p atmega328p -U flash:w:main.hex:i
```

Step 15. **Compile and upload the blink code using the makefile**. You should see the LED blinking.

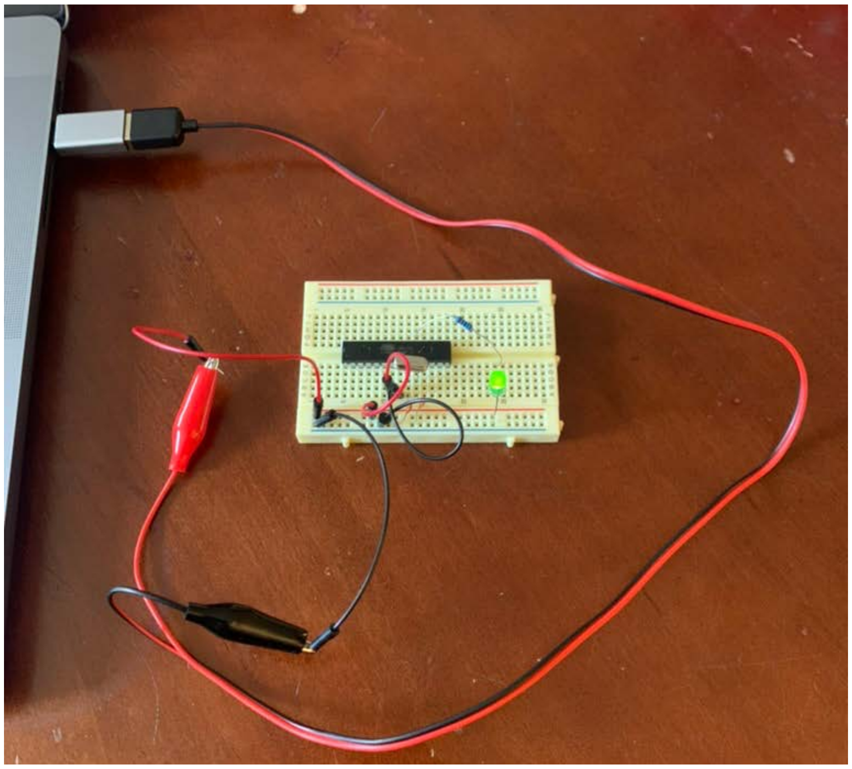

blink — -zsh — 101×48

```
[davemclaughlin@wine blink % make                                                                    ]
avr-gcc -Wall -Os -DF_CPU=16000000       -mmcu=atmega328p -o main.elf blink.c
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avr-size --format=avr --mcu=atmega328p main.elf
AVR Memory Usage
----------------
Device: atmega328p

Program:     182 bytes (0.6% Full)
(.text + .data + .bootloader)

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


[davemclaughlin@wine blink % make flash                                                              ]
avrdude -c usbtiny -b 115200 -P usb -p atmega328p -U flash:w:main.hex:i

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e950f (probably m328p)
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 "main.hex"
avrdude: writing flash (182 bytes):

Writing | ################################################## | 100% 0.42s

avrdude: 182 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 182 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.58s

avrdude: verifying ...
avrdude: 182 bytes of flash verified

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

avrdude done.  Thank you.

davemclaughlin@wine blink %
```

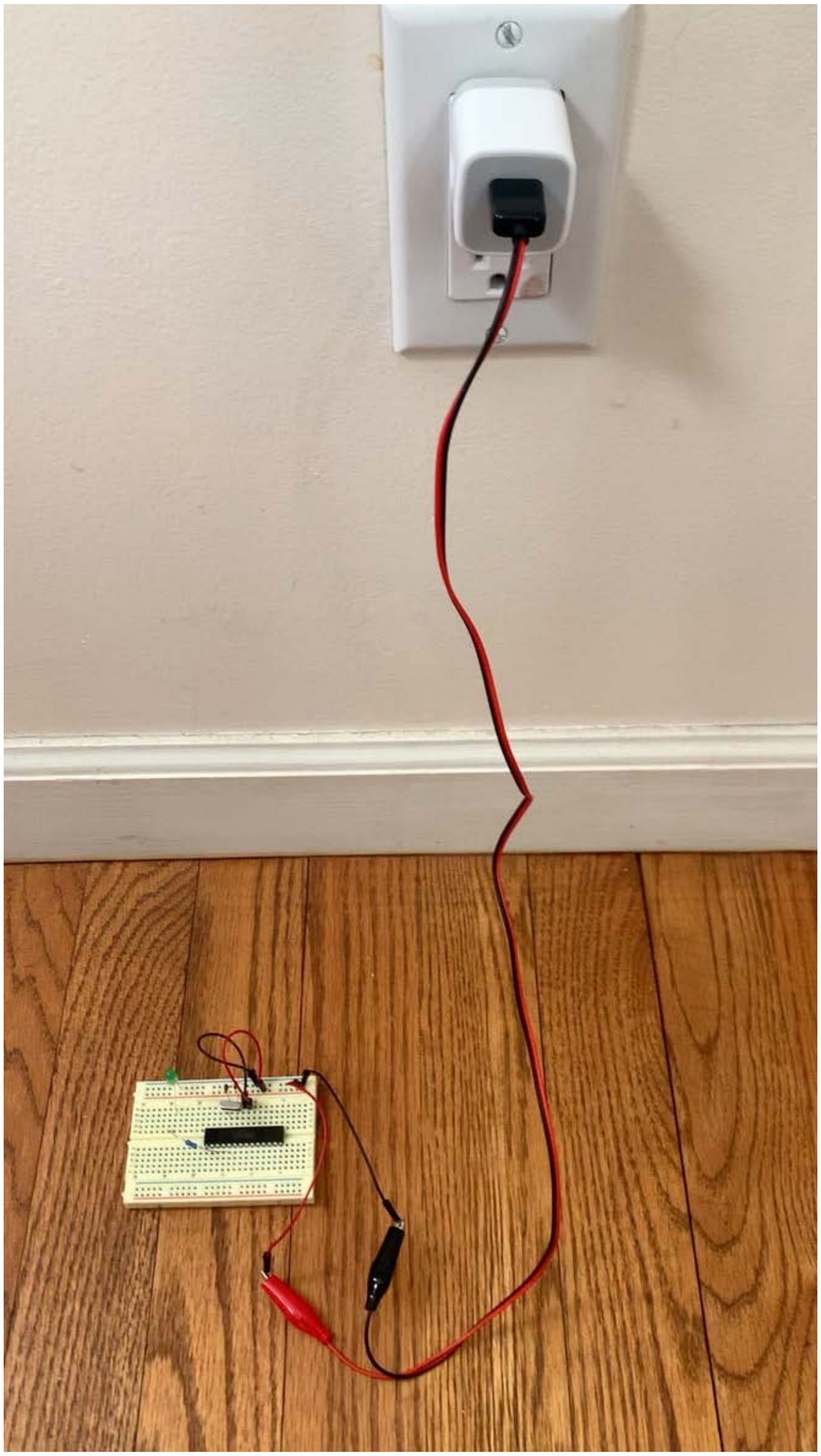
Step 16. **Power the circuit using USB.** At this point, your circuit is getting power through the Pocket Programmer via the USB port from your laptop.



After you have flashed the blink code to the 328P, disconnect the 6 SPI wires connecting the Pocket Programmer to your 328P. You can power your circuit using the 5V power directly from your USB port. Use the USB-to-alligator-clip leads that came with your kit to power the circuit.

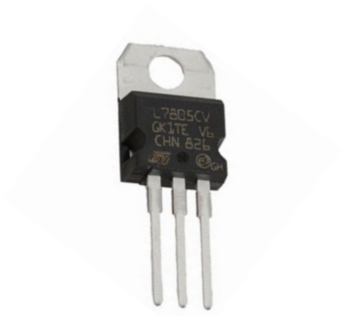


You can also use the USB AC adapter that came with you kit to power your circuit using a standard electrical outlet as shown:

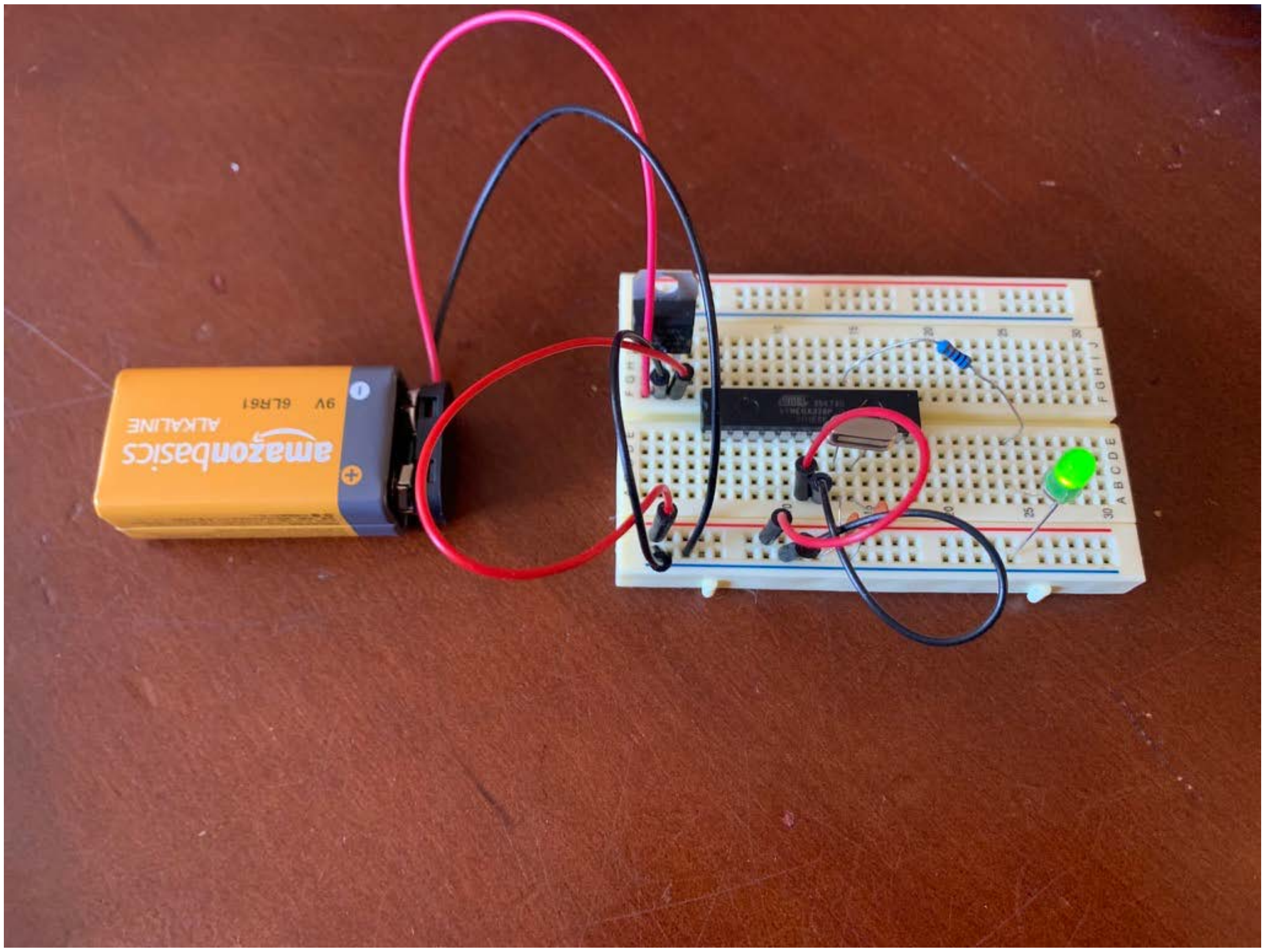


Step 17. **Add a 7805 voltage regulator** to your circuit. Connect a 9V battery between the Vin and GND pins of the regulator, and connect the Vout and GND pins to the VCC and GND rails of your breadboard.

IMPORTANT: When you re-connect the Pocket Programmer to the circuit using the 6 wire SPI interface, be sure to disconnect the 9V battery. (Also, of course, disconnect the 9V battery from the circut when you're not using it, since constant use will drain the battery.)

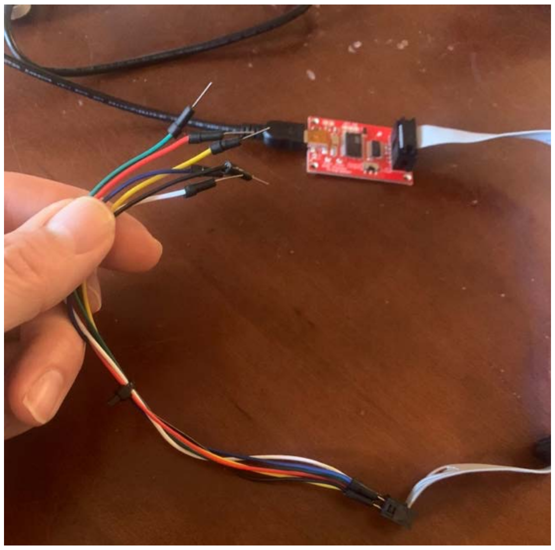


V_{in} GND V_{out}



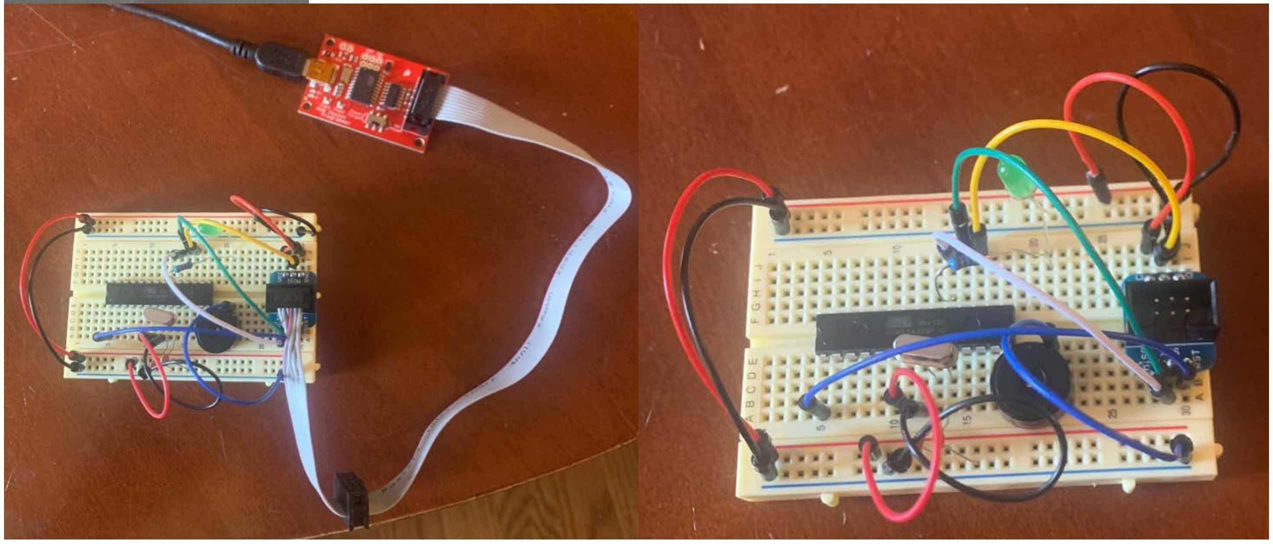
Step 18 (optional). You will likely be adding and removing the 6 wire SPI interface betweeen your programmer and 328p as you build and test circuits. Be careful to re-connecting wires to the correct pins on the 328p. Here are two ideas that can to make this easier on you:

(a) keep the 6 SPI wires together as a bundle with a piece of tape, keep one end inserted into the SPI connector, and make a note of which color wire corresponds to with 328p pin.



red: VCC rail
black: GND rail
blue: RESET (pin1)
yellow: MOSI (pin 17)
green: MISO (pin 18)
white: SCK (pin 19)

(b) You have a 6-pin adapter in your kit that can be assembled onto your breadboard and wired to your 328P for quick & easy attaching & detaching the SPI connector. This little adapter requires soldering 6 pins, which you can do in M. The adapter is described here: https://www.adafruit.com/product/1465



Document History

Revised on	**Version**	**Author**	**Description**
3/28/22	0.1	D. McLaughlin	Initial document creation
4/3/22	1.0	D. McLaughlin	Initial release
4/4/22	1.1	D. McLaughlin	released to ECE-231 Spring 2022