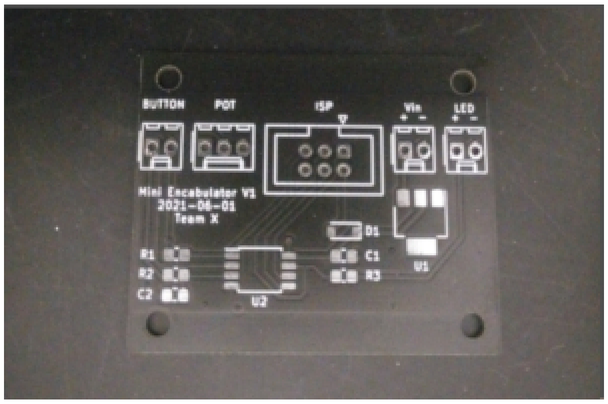
Soldering Exercise

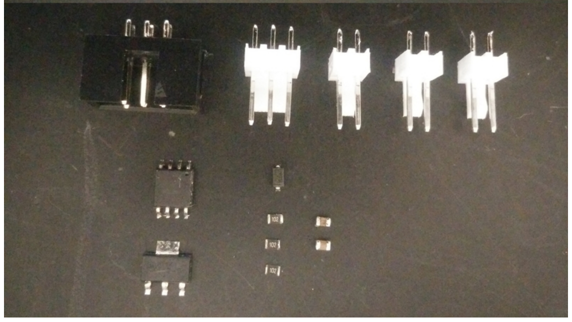
This is a continuation of the PCB Design Exercise. This assignment will show you how to populate PCBs with surface-mount and through-hole components, how to assemble an enclosure with panel-mount parts, and will cover the basics of programming a bare AVR chip. At the end of the assignment, you will have a completed encabulator and will demonstrate its function to a TA or Lab staff.

First, you will need to obtain the disposable parts from a TA or Lab staff member and make sure you have all of the following:

- 3x 1k resistors
- 2x 1uF capacitors
- 1x LM1117 5V regulator
- 1x ATTiny85
- 1x ISP header
- 3x 2 pin headers
- 1x 3 pin headers
- 1x panel mount LED
- 1x 2 pin MTA100 connector

A few images of all the components are below.





Skip to the programming section after populating your board if you are already familiar with surface-mount soldering.

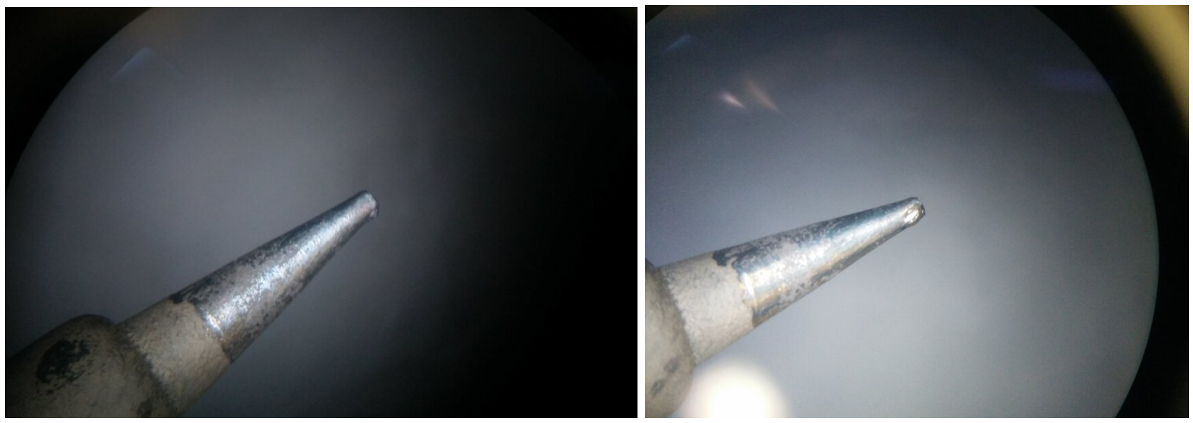
Soldering

Poor soldering is the cause of 95% of the issues with this assignment. So please pay attention to this section if you have not soldered before or if you just want a refresher.

We'll begin by populating the surface-mount components first. Many people use a method known as reflow soldering, where solder paste in a syringe is applied to each pad, components are placed on the paste, and the whole board is heated in an oven or hotplate. **We will not be doing this in this assignment.**

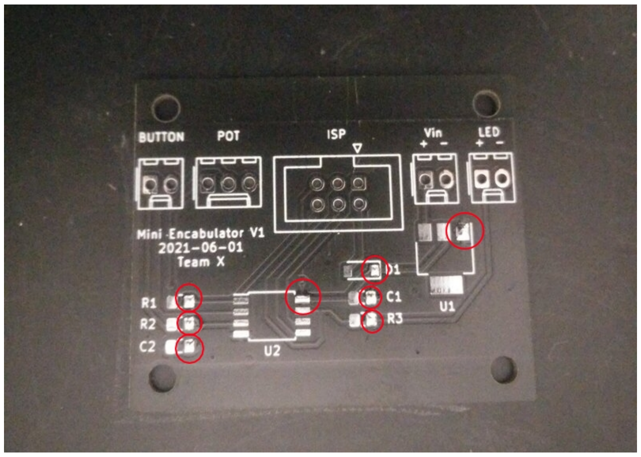
Instead, we'll be using a regular soldering iron for populating our boards. This is only possible if the components are large enough and have exposed pads, so keep this in mind before choosing components in QFN or BGA packages.

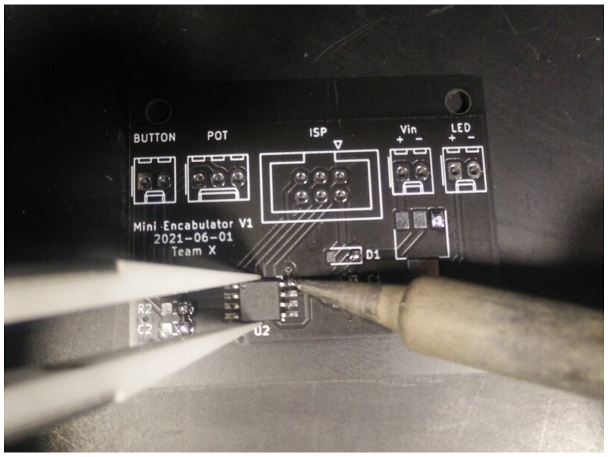
Start by wetting your sponge and cleaning the hot tip of your iron off. Apply a small bead of solder to your tip to help transfer heat from the iron to the part being soldered.



The right image above is an iron tip without a bead of solder, while the right has a bead of solder added.

To start off, we're going to "tack" each of our components to the board with one of their pins to hold them in place. Add a little bit of solder to just one pad of each part as in the picture below. Then use the tweezers and iron to tack down the microcontroller.

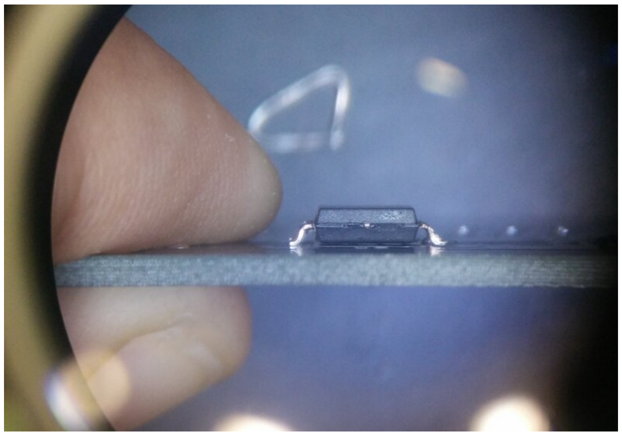


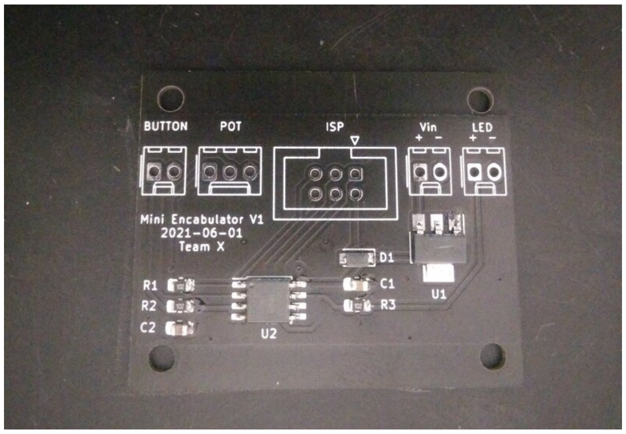


Keep these three steps in your head when soldering pins:

1. Put your iron in contact with both the leg of the component and the pad.
2. Apply some solder to the point of contact between the iron and the pad.
3. Leave the iron in contact with the pad for 1-2s to allow the solder to "soak" in.

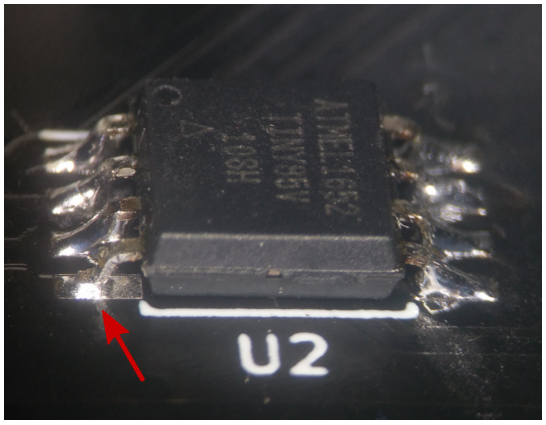
It's important that the IC is flat when you've tacked it down or it may be difficult to get all pads connected. Repeat this for the other surface-mount parts.

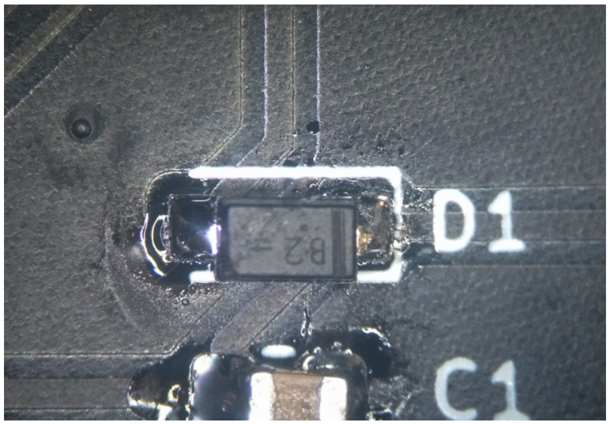




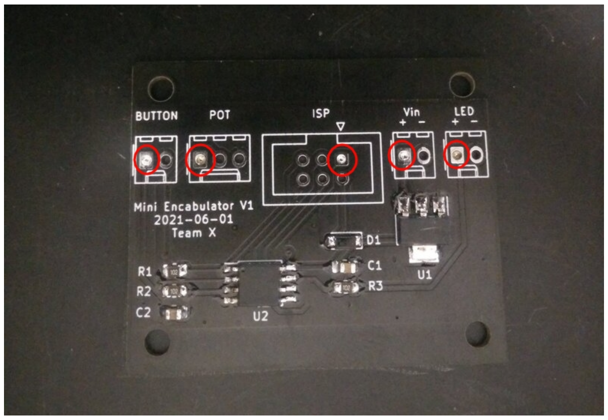
In the above images, the left shows a flat IC, and the right shows all the surface mount components soldered on the board.

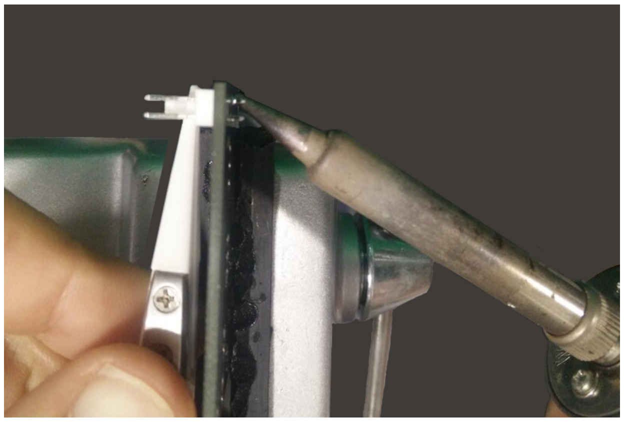
Solder the components fully after they are tacked down. The solder joint should be shiny and cover the pad completely. Pay attention to the orientation of the diode. See the below images, respectively.



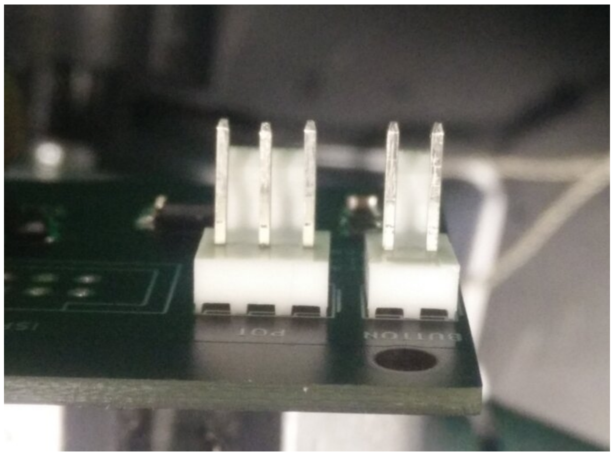
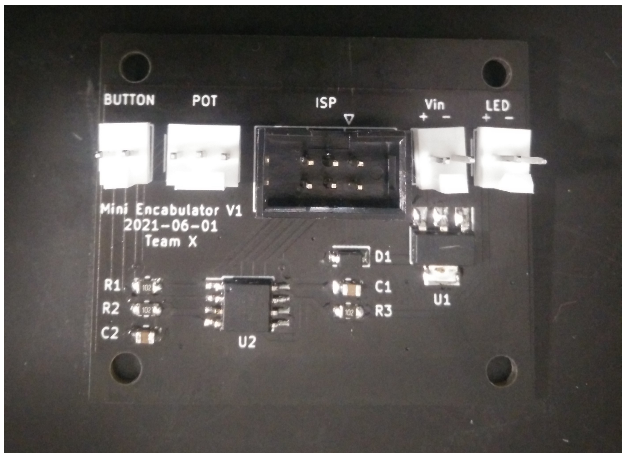


Do the same for the through-hole headers by adding solder to just one of the pads for each part. Clamp the PCB in the vise, and while heating the soldered pad, push the header through. Be sure to get the connector orientation right before soldering.





Check that the component is flat and repeat this for the other headers.

Like before, these solder joints should be shiny and cover the whole pad.

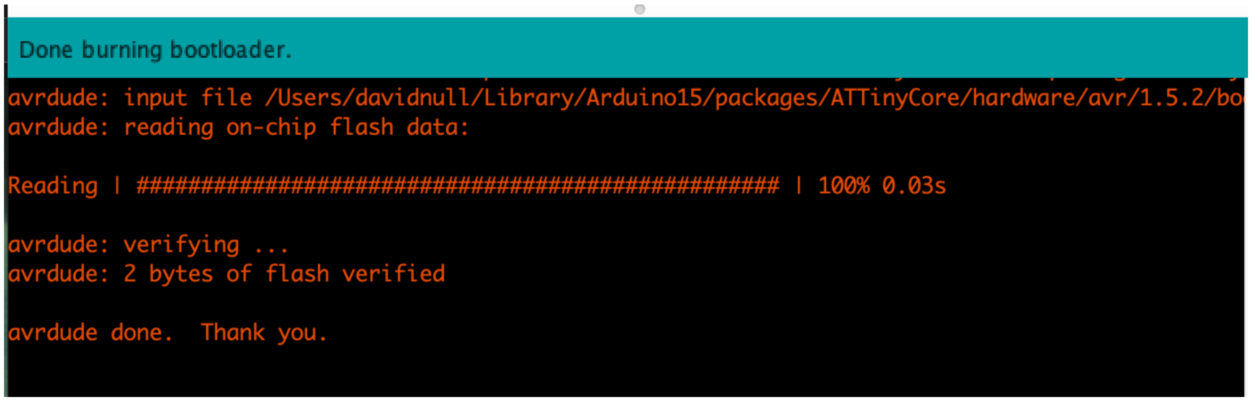
Programming

Plug the USBasp programmer into your computer and your board and follow the Configuration instructions in the Appendix.

Program the board using the Arduino IDE and the following test sketch:

```
// Test sketch

#define BUT_PIN PIN_PB4
#define LED_PIN PIN_PB2
#define POT_PIN A3

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUT_PIN, INPUT);
  // enable internal pullup
  digitalWrite(BUT_PIN, HIGH);
}

void loop() {
  // read potentiometer and rescale to 0-1s delay
  int delay_time = 1000 * int32_t(analogRead(POT_PIN)) / 1024;

  // blink while button depressed
  if (digitalRead(BUT_PIN) == LOW) {
    digitalWrite(LED_PIN, HIGH);
    delay(delay_time);
    digitalWrite(LED_PIN, LOW);
    delay(delay_time);
  } else {
    digitalWrite(LED_PIN, LOW);
  }

}
```

If the board is successfully programmed, you will see a message like below.

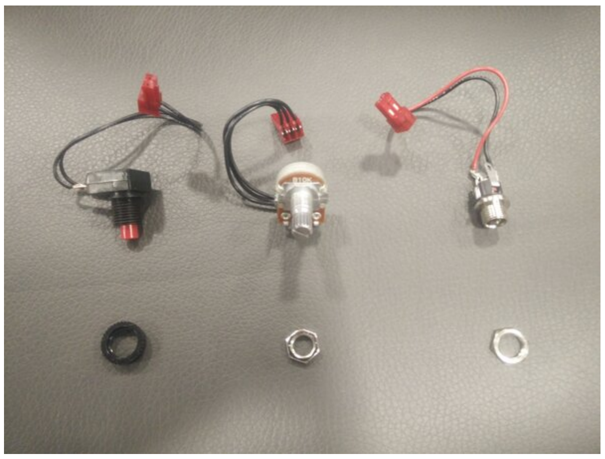
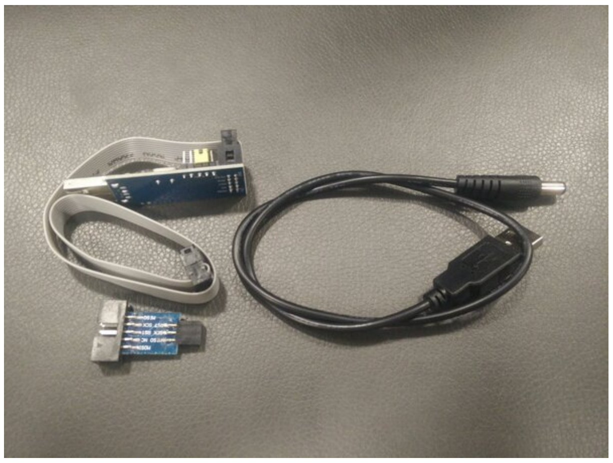


```
Done burning bootloader.
avrdude: input file /Users/davidnull/Library/Arduino15/packages/ATTinyCore/hardware/avr/1.5.2/bo
avrdude: reading on-chip flash data:

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

avrdude: verifying ...
avrdude: 2 bytes of flash verified

avrdude done.  Thank you.
```

Assembly

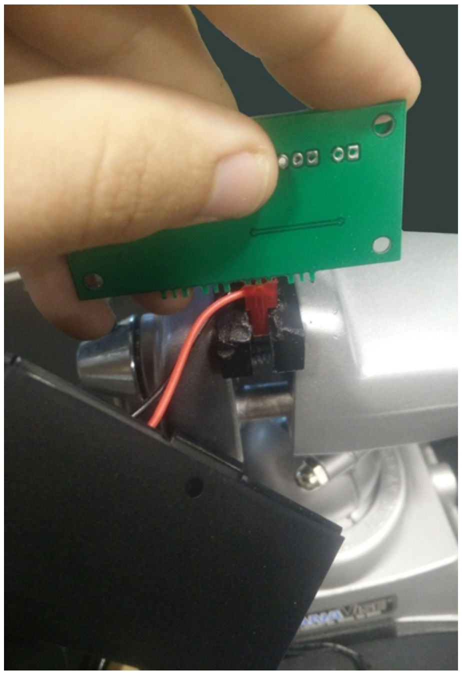
Obtain these parts from your TA or the lab staff. These parts will be returned after the assignment is complete, so do not damage them.

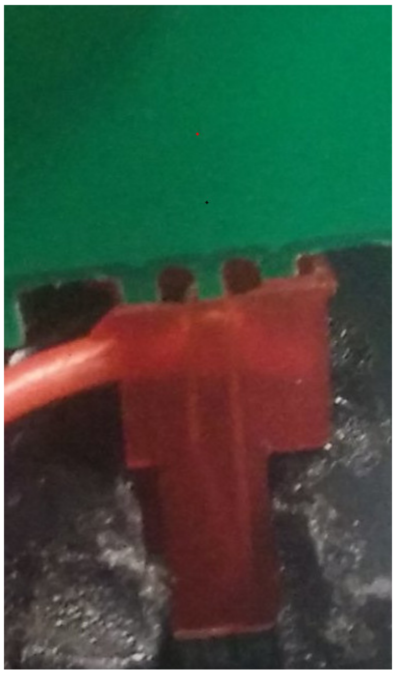
- 1x button w/ nut
- 1x potentiometer w/ nut
- 1x barrel jack w/ nut
- 1x USBASP programmer with adapter
- 1x barrel plug to USB cable
- 1x enclosure w/ screws

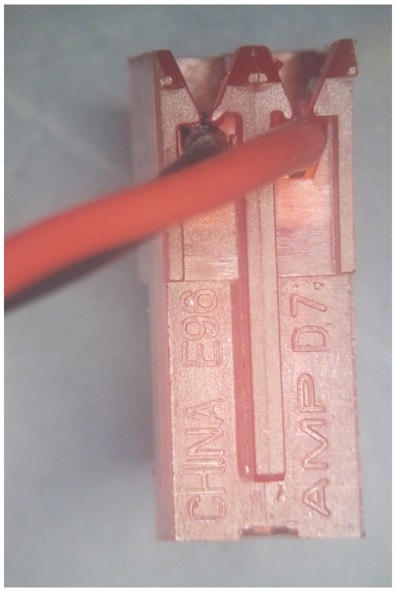



Cut the wire leads of the LED to about 3 inches in length and insert it into the enclosure (**do this before crimping the connector**). The wire leads should not be stripped.

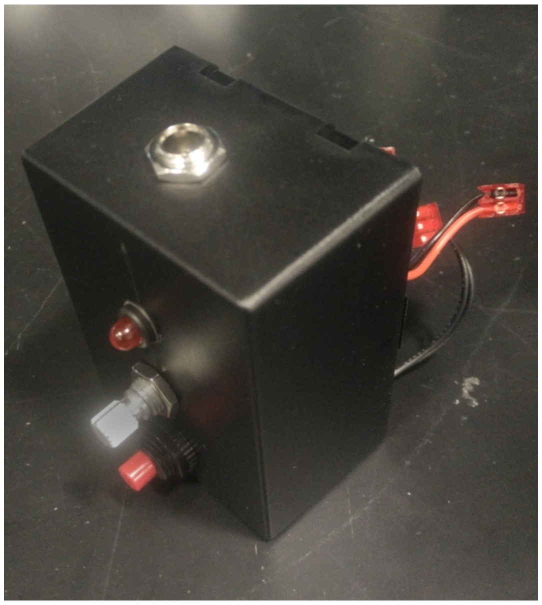
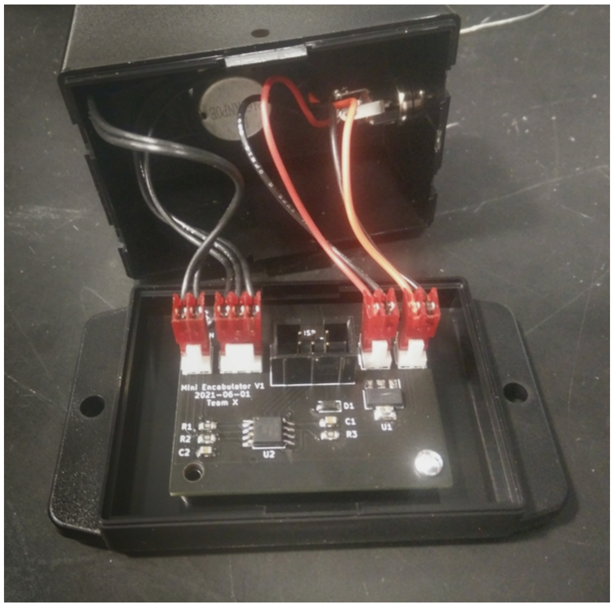
Using the vise, clamp the MTA-100 connector into place and use the crimping tool on the edge of your PCB to push the positive lead of the LED into position 1 of the connector. This may require some force.







Screw in the rest of the panel mount components, connect them to the PCB, and mount the PCB to the enclosure baseplate. See the below image:

Demonstrate your encabulator to the TA or Lab staff and be able to answer the following questions (ECE Students only):

- In the test sketch, why is it necessary to cast to `int32_t` when reading the potentiometer? (hint: check the Arduino docs for the size of the return type of `analogRead`)
- Why do we enable the internal pullup on `BUT_PIN`? What happens if we don't do this? (refer to the schematic)

Cleanup Checklist

Complete this checklist to get a full grade for the assignment:

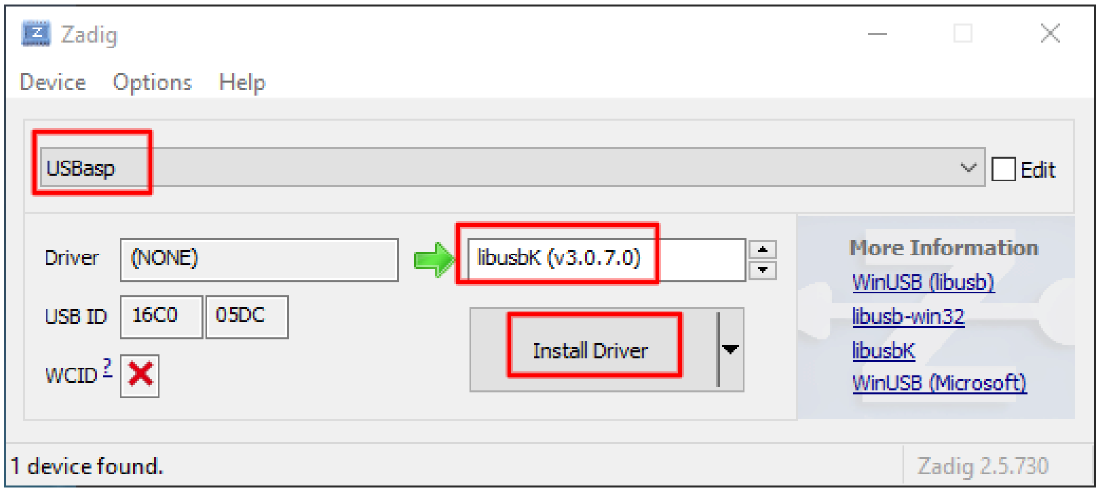
- disconnect and unscrew all panel mounted components and put them in the large bag (except the LED)
- remove board screws and place in large bag
- USBasp programmer placed in small bag
- remove LED from enclosure by cutting off MTA-100 connector with wire cutters

Return the two bags and enclosure to the TA or Lab staff for full points. Make sure that they have recorded your name (we can forget sometimes when the lab is busy).

Appendix A: Configuring the Arduino IDE

Note: You do not need to do step 1 and 2 on the lab computers.

1. Install the Arduino IDE (version >1.8.5 recommended).
2. (If on a Windows machine) With the USBasp programmer plugged in, install the `libusbK` driver using Zadig.



USBasp

Driver (NONE) → libusbK (v3.0.7.0)

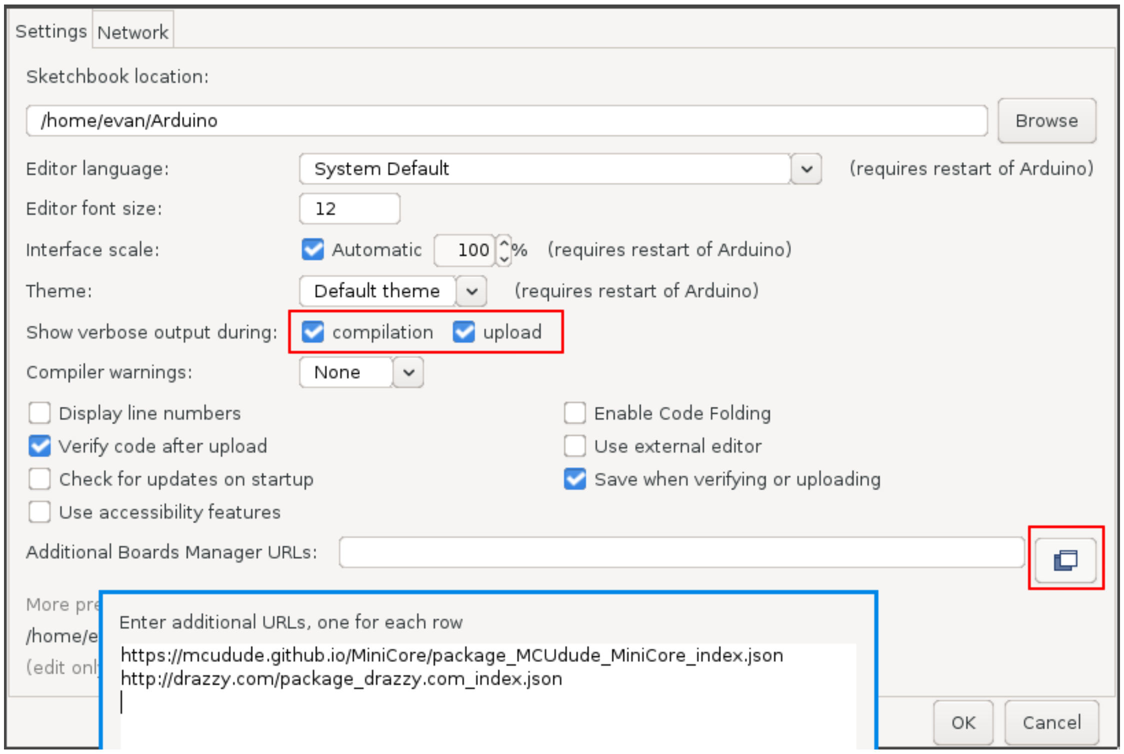
USB ID 16C0 05DC

WCID ?

1 device found.

3. Open Preferences in the Arduino IDE, enable verbose output and add these two board manager URLs:

```
https://mcudude.github.io/MiniCore/package_MCUdude_MiniCore_index.json
http://drazzy.com/package_drazzy.com_index.json
```



Sketchbook location: /home/evan/Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☑ compilation ☑ upload

Compiler warnings: None

☐ Display line numbers ☐ Enable Code Folding

☑ Verify code after upload ☐ Use external editor

☐ Check for updates on startup ☑ Save when verifying or uploading

☐ Use accessibility features

Additional Boards Manager URLs:

More pre

/home/e

(edit onl

Enter additional URLs, one for each row

```
https://mcudude.github.io/MiniCore/package_MCUdude_MiniCore_index.json
http://drazzy.com/package_drazzy.com_index.json
```

4. Under `Tools > Board > Boards Manager`, search for and install `ATTinyCore`.
5. Set the following board settings in the Tools menu:
    a. Board: `ATTinyCore > ATTiny25/45/85 (No bootloader)`
    b. Chip: `ATtiny85`
    c. Clock Source: `8 MHz Internal`
    d. Programmer: `USBasp (ATTinyCore)`

    Note the port option has no effect when using an external programmer.
6. Use `Tools > Burn bootloader` to apply these settings to the microcontroller (aka "burning the fuses”). This only needs to be done once.
7. Use `Sketch > Upload using programmer` and try uploading a blank sketch. The normal upload button will not work, as that assumes the microcontroller has a bootloader running.

You should now be able to program the microcontroller using the Arduino IDE as normal.