A NeoPixel Pomodoro Timer

Created by Dave Astels

https://learn.adafruit.com/a-neopixel-pomodoro-timer

Last updated on 2024-06-03 02:24:23 PM EDT

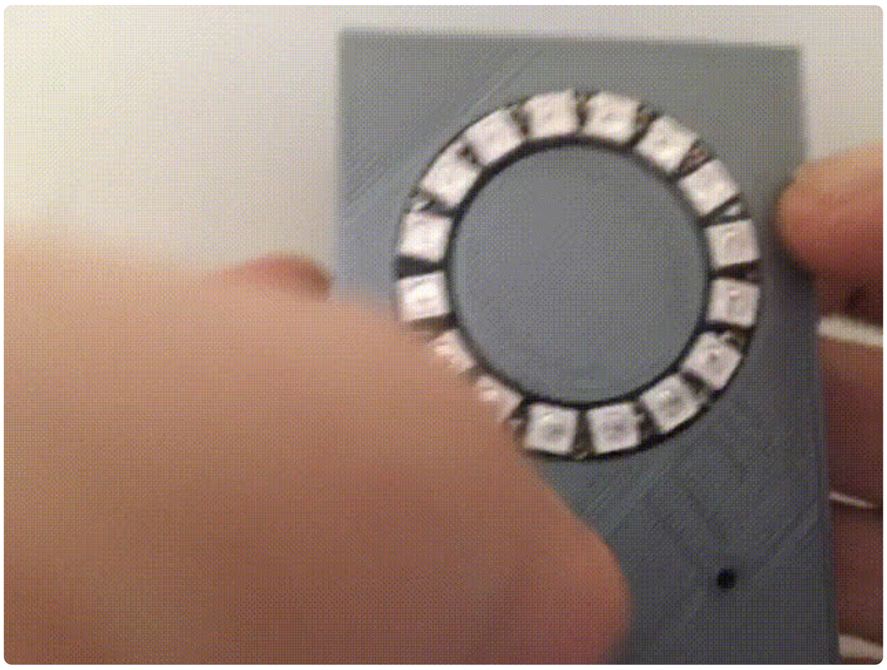
Table of Contents

Overview

I know many people who get value from a simple tool called the Pomodoro Technique.

It's a time management technique used to break work into periods of activity separated by short breaks. Each activity period is called a pomodoro. Why "pomodoro"? The technique was developed by Francesco Cirillo, who named the technique "pomodoro" as a nod to the tomato kitchen timer he initially used.

The technique has come a long way from those humble beginnings with phone apps, web versions, etc. I decided to make a simple hardware version in CircuitPython with a rotary encoder to set the time and mode!



Parts

Here's what you'll need from Adafruit to build the Pomodoro Timer:

1 x Adafruit ItsyBitsy M0 Express - for CircuitPython and Arduino IDE What's smaller than a Feather but larger than a Trinket? It's an Adafruit ItsyBitsy M0 Express!	https://www.adafruit.com/product/3727
1 x NeoPixel Ring - 16 x 5050 RGB LED with Integrated Drivers Round and round and round they go! 16 ultra bright smart LED NeoPixels are arranged in a circle with 1.75" (44.5mm) outer diameter.	https://www.adafruit.com/product/1463
1 x Piezo Buzzer Piezo buzzers are used for making beeps, tones and alerts. This one is petite but loud!	https://www.adafruit.com/product/160
1 x Rotary Encoder + Extras This rotary encoder is the best of the best, its a high quality 24-pulse encoder, with detents and a nice feel.	https://www.adafruit.com/product/377
1 x LiIon/LiPoly Backpack Adafruit LiIon/LiPoly Backpack Add-On for Pro Trinket/ ItsyBitsy	https://www.adafruit.com/product/2124

These nice switches are perfect for use with breadboard and perfboard projects.

1 x Breadboard-friendly SPDT Slide Switch https://www.adafruit.com/product/805
These nice switches are perfect for use with breadboard and perfboard projects.

1 x Lithium Ion Polymer Battery - 3.7v 500mAh https://www.adafruit.com/product/1578
Lithium ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light and powerful. This battery has a capacity of 500mAh for a total of about 1.9 Wh.

1 x Silicone Cover Stranded-Core Wire - 50ft 30AWG Blue https://www.adafruit.com/product/3166
Silicone-sheathing wire is super-flexible and soft, and its also strong! Available in various colors.

Materials and Supplies

- 3D Printer and Filament
- Superglue
- Small diameter heatshrink tubing

Tools

- Wire Strippers
- Wire Cutters
- Pliers
- Pin-vise drill and 0.8mm drill bit
- Soldering Iron
- Solder

Hardware

My goal was to make a pomodoro timer using a few basic parts and write the code for it using CircuitPython.

My requirements were pretty simple:

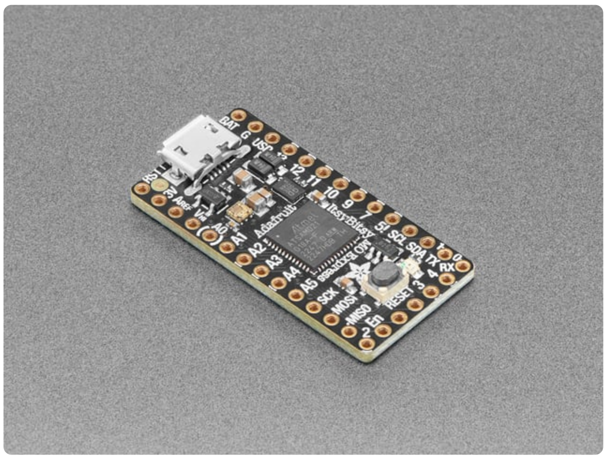
1. write the code in CircuitPython,
2. have an audible alert to indicate transition between the work/break phases,
3. have a display of some sort to show progress through each phase, and
4. have a way to set the length of each phase.

Which CircuitPython Board?

To satisfy requirement 1, I needed to use an Adafruit M0 or M4 microcontroller based board. I decided to see if I could make it work with an M0. These boards are great for simple CircuitPython projects, but if you have much code it's easy to run into memory limitations.

For very simple projects, a Trinket M0 does a great job, but I eventually decided to use a rotary encoder and wanted to try out the new `rotaryio` support in the latest 3.0 release of CircuitPython. That ruled the Trinket out.

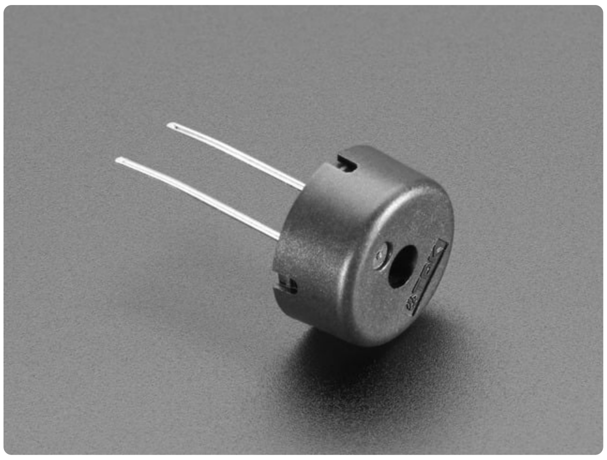
So that left the M0 Express versions of the Feather or ItsyBitsy. Either would work, but I decided to use the ItsyBitsy: it's smaller, so it would give me a bit more flexibility when it came time to designing a case. It would provide a bit more of a challenge, as well, due to the lack of mounting holes that the Feather has.



Adafruit ItsyBitsy M0 Express - for CircuitPython & Arduino IDE
What's smaller than a Feather but larger than a Trinket? It's an Adafruit ItsyBitsy M0 Express! Small, powerful, with a rockin' ATSAMD21 Cortex M0...
https://www.adafruit.com/product/3727

How To Make Noise?

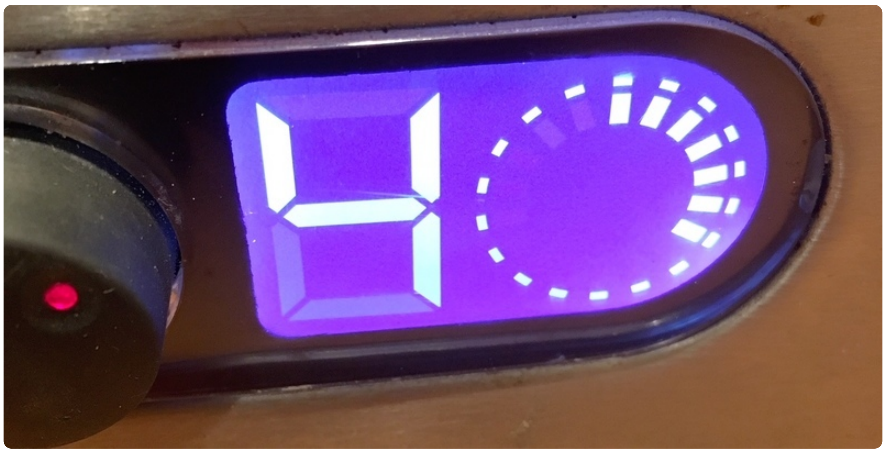
For requirement 2, I used the same piezo buzzer I used in my Humidity monitor guide (https://adafru.it/BIS). It's loud enough, small, and works nicely with the CircuitPython `pulseio` library.



Piezo Buzzer
Piezo buzzers are used for making beeps, tones and alerts. This one is petite but loud! Drive it with 3-30V peak-to-peak square wave. To use, connect one pin to ground (either one) and...
https://www.adafruit.com/product/160

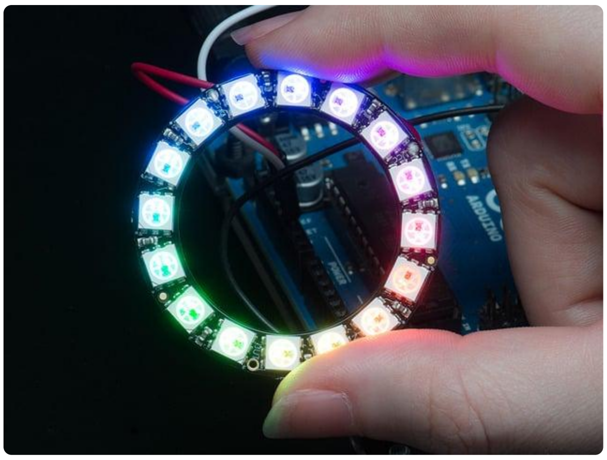
How To Show Time Progressing?

When I thought about requirement 3, I thought about my toaster's interface. It uses a continuous knob and a radial display for setting darkness.



What isn't obvious from the photo is that the highest valued segment of the dial blinks as it counts down. Also, as the count down proceeds, the number of segments lit decreases. When none remain lit, the toast is done.

This immediately reminded me of a Light Emitting Diode (LED) NeoPixel ring.



NeoPixel Ring - 16 x 5050 RGB LED with Integrated Drivers
Round and round and round they go! 16 ultra bright smart LED NeoPixels are arranged in a circle with 1.75" (44.5mm) outer diameter. The rings are 'chainable' - connect the...
https://www.adafruit.com/product/1463

How To Set Time?

For the final requirement I was still thinking about my toaster and a rotary encoder as input made sense, especially since I'd decided to use a ring as a display.

Rotary Encoder + Extras
This rotary encoder is the best of the best, it's a high-quality 24-pulse encoder, with detents and a nice feel. It is panel mountable for placement in a box, or you can plug it...
https://www.adafruit.com/product/377

Rotating the encoder could be used to set the times, while the push switch could be used to cycle through modes:

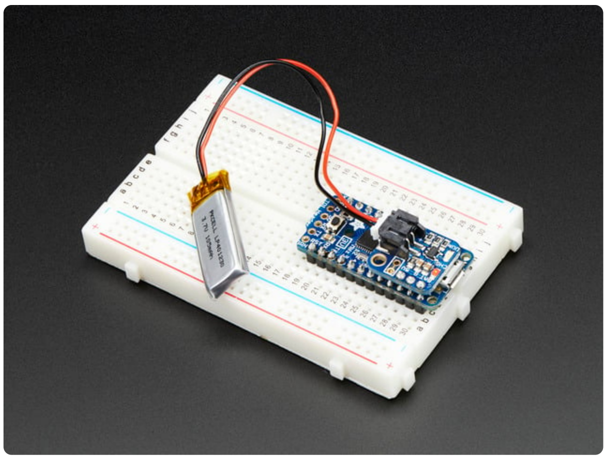
- timer -> set work time -> set break time -> timer

and so on.

I'd implemented rotary encoder handling in CircuitPython in a previous guide (https://adafru.it/BIT), but this was a good chance to try out the new bundled rotary encoder support.

Making It Portable

All that was left was adding a LiPo backpack so that it could be battery powered. The backpack is required since the ItsyBitsy doesn't have on-board battery support. That's one of the tradeoffs compared to the Feather. The nice thing about the backpack is that it doesn't increase the footprint, but it does add to the thickness of the ItsyBitsy. That wouldn't be a problem since the case will have plenty of thickness to accommodate it.



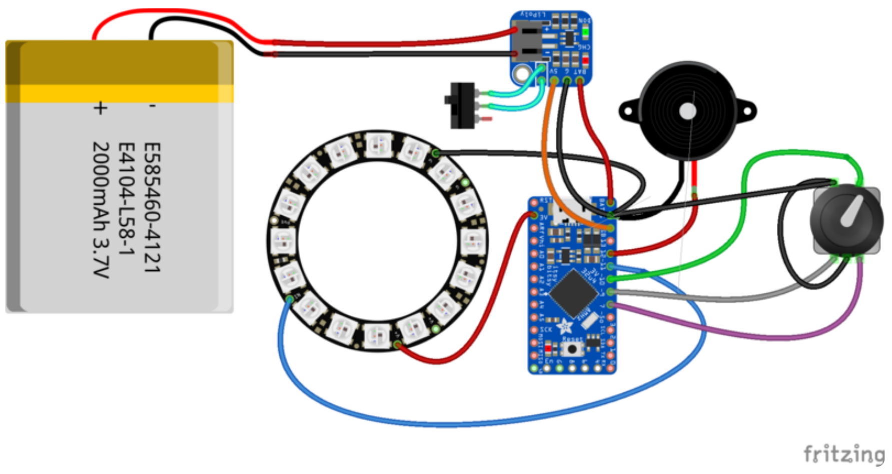
Adafruit LiIon/LiPoly Backpack Add-On for Pro Trinket/ItsyBitsy

If you have an ItsyBitsy or Pro Trinket you probably know it's the perfect little size for a portable project. This LiPoly backpack makes it really easy to do! Instead of wiring 2...

https://www.adafruit.com/product/2124

Final Wiring Diagram

That's it: an ItsyBitsy M0 Express with a NeoPixel ring, a rotary encoder, and a piezo buzzer. Add in a LiPo backpack, power switch, and battery for power, and that's it. Below is the wiring diagram. The only difference is that I'm using a 500mAh LiPo in my build. Making the case bigger will allow use of a bigger battery. That makes no difference to the wiring, though.



Wiring is pretty straight-forward and described in detail on the Assembly page.

The battery connects to the LiPo backpack. The slide switch does as well, allowing the device to be turned off when not in use. The backpack is connected to the power connections on the ItsyBitsy, typically (as I did) by using the included bit of long-pin header.

The buzzer connects to ground and D12.

The Neopixel ring has it's power connected to the ItsyBitsy's 3v output, and it's ground to the ItsyBitsy's. The ring has two ground connections, and the second can be used during construction to avoid having to connect everything directly to the ItsyBitsy's ground. The ring's Data In connects to D11.

The center of the encoder, and one side of its switch connects to ground. The other side of its switch connects to D10, while its encoder pins connect to D9 and D7. Done as shown, the code will work correctly with the rotation direction. If you reverse the encoder connections, rotation will be opposite what's expected. Reverse the connections or switch the encoder pins in the code to correct it.

Code

As with most of my projects these days, I used CircuitPython for this. Are you new to using CircuitPython? No worries, there is a full getting started guide here (https://adafru.it/cpy-welcome).

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and installation in this tutorial (https://adafru.it/ANO).

We'll go through the code, piece by piece, starting with the imports and setup.

```
import time
from math import ceil
import board
import rotaryio
import neopixel
from adafruit_debouncer import Debouncer
import digitalio
import pulseio

# Setup the hardware

encoder = rotaryio.IncrementalEncoder(board.D9, board.D7)
button_io = digitalio.DigitalInOut(board.D10)
button_io.direction = digitalio.Direction.INPUT
button_io.pull = digitalio.Pull.UP
button = Debouncer(button_io)
strip = neopixel.NeoPixel(board.D11, 16, brightness=1, auto_write=False)
```

Here the rotary encoder is set up using the new built-in support for the rotation of the switch and I use my debouncer library to clean up the encoder push switch. Finally the NeoPixel strip is configured. If you are unfamiliar with NeoPixels, there is a great guide on them (https://adafru.it/dhw).

Now let's jump to the main loop:

```
strip.fill(0x000000)
strip.show()
work_time = 6
break_time = 2
time_to_check = 0
state = False
mode, dial_color, time_remaining, increment = compute_mode_settings(True)

while True:
    # check whether the rotary encoder has been pushed. If so enter time-set mode.
    button.update()
    if button.fell:
        work_time = set_timer(0x400000, work_time)
        break_time = set_timer(0x004000, break_time)
        strip.fill(0x000000)
        strip.show()
        mode, dial_color, time_remaining, increment = compute_mode_settings(True)

    now = time.monotonic()
    if now &gt;= time_to_check:          #only check each second
        time_remaining -= 1
        if time_remaining &lt;= 0:       # time to switch modes?
            strip.fill(0x000000)      # clear the dial
            strip.show()              # make some noise
            beep(2, 0.5, 0.25, 4000)
            mode, dial_color, time_remaining, increment = compute_mode_settings(not
mode)
        state = not state             # have the top pixel toggle between the dial
color and white
        show_time(dial_color, ceil(time_remaining / increment), state)   #update
the dial
        time_to_check = now + 1.0
```

Each time through the loop, it checks for a push on the encoder switch. That's the job of the debouncer's update function. If one was detected (i.e. the button signal went from high to low... it fell) the code enters the mode to set the length of the work phase, followed by the break phase, then back to timing mode.

The next step checks to see if it's time to update the timer. It does this every second (as defined by the last line). So once a second, the time remaining is decreased by 1 (second) and the ring is updated. If it reaches 0, it's time to make some noise and change mode.

In addition to the setup and loop there are a handful of helper functions.

```
last_position = 0

def check_encoder():
    global last_position
    position = encoder.position
    if position &gt; last_position:
        direction = 1
    elif position &lt; last_position:
        direction = -1
    else:
        direction = 0
    last_position = position
    return direction
```

The `check_encoder` function tracks the position of the encoder, comparing it to the last known position. Based on that comparison it returns -1, 0, or 1 to indicate that the encoder rotated counter-clockwise, didn't move, or rotated clockwise.

```
def show_time(color, value, bright):
    strip.fill(0x000000)
    if value &gt; 0:
        for i in range(1, value + 1):
            strip[16 - i] = color
        if bright:
            strip[16 - value] = 0x404040
    strip.show()
```

The `show_time` function updates the time displayed on the NeoPixel ring. It sets pixels to the specified color starting at pixel zero (which is at the bottom of the ring in the final build) and moving clockwise. The final pixel is handled differently. Depending on the bright parameter, it's either set to the same color as the rest, or white. In the main loop we saw that this value gets toggled each second. The result is that the highest pixel blinks.

Note that for the work and break times, each pixel is worth a different number of seconds. This is because it's typical for the work phase to be a half hour, an hour, or longer, while the break phase is usually between five and fifteen minutes. To give a better, brighter display, I decided to use a different scale for each. This is all adjustable in the `compute_mode_settings` function:

```
def compute_mode_settings(new_mode):
    work_time_increment = 600   # each work phase pixel is worth 10 minutes
    break_time_increment = 300  # each break phase pixel is worth 5 minutes

    if new_mode:
        return True, 0x400000, work_time * work_time_increment, work_time_increment
    else:
        return False, 0x004000, break_time * break_time_increment,
break_time_increment
```

Depending on which mode is being entered, the appropriate set of values is returned. These are used to update the variables in the loop (see above).

When there's a switch from one phase to the next, we use the piezo buzzer and `pulseio` to make some noise. The `beep` function does that:

```
def beep(count, duration, interstitial, freq):
    pwm = pulseio.PWMOut(board.D12, duty_cycle = 0, frequency=freq)
    for _ in range(count):
        pwm.duty_cycle = 0x7FFF
        time.sleep(duration)
        pwm.duty_cycle = 0
        time.sleep(interstitial)
    pwm.deinit()
```

This is straightforward. It sets up the pulse-width modulation (PWM), loops for the number of beeps requested, and shuts down the PWM. For each beep it sets the duty cycle to 50%, waits for the beep duration, sets the duty cycle to 0% (effectively turning of the sound), then waits for the interstitial duration before playing the next beep (if any).

Next is maybe the most interesting function.

```
def set_timer(color, value):
    global last_position
    time_setting = value
    last_position = encoder.position
    for i in range(16):
        strip[i] = color
        strip.show()
    for i in range(16):
        strip[i] = 0x000000
        strip.show()
    while True:
        show_time(color, time_setting, False)
        button.update()
        if button.fell:
            return time_setting
        direction = check_encoder()
        time_setting += direction
        if time_setting &gt; 16:
            time_setting = 16
        if time_setting &lt; 0:
            time_setting = 0
```

To start, it initializes from the current encoder position and flashes the ring. Following that there's a loop that shows the current setting (using the `show_time` function we looked at above) and checks the encoder push switch. If it was pressed the current setting is returned. Otherwise the encoder's rotation is checked and the time setting changed based on the result. Finally the time setting is capped at 0 and 16, which reflects the size of the ring.

That's it. Each piece is fairly simple, but the overall functionality is interesting and useful. Below is all of it, with comments.

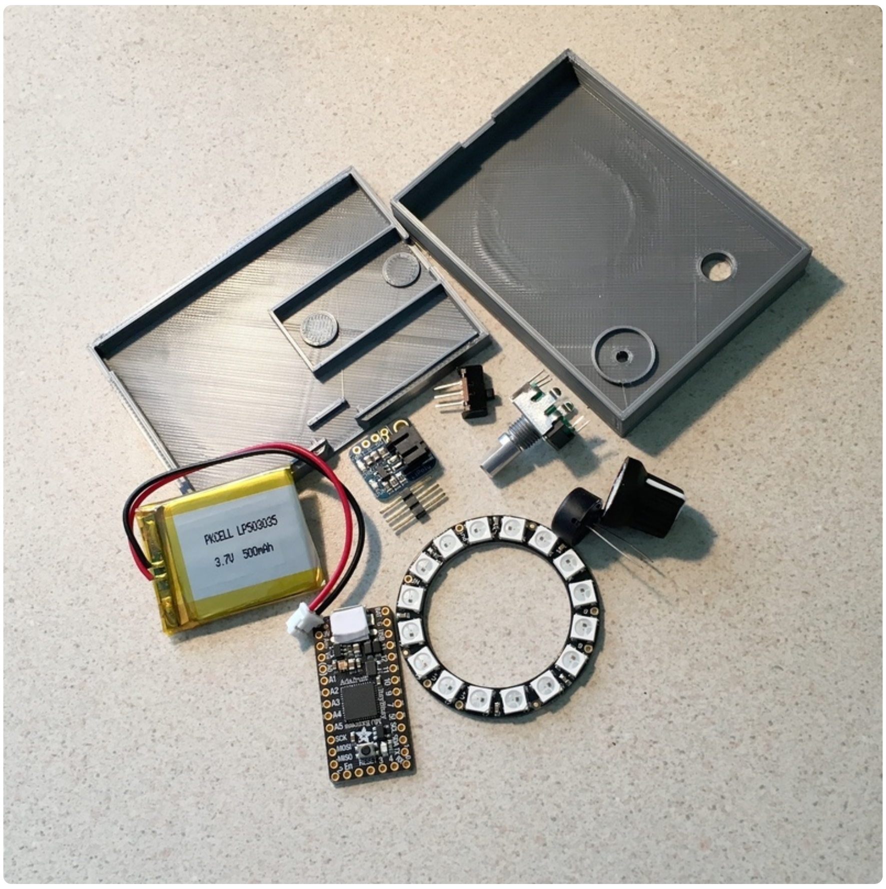
```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
The MIT License (MIT)
```

```
Copyright (c) 2018 Dave Astels

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
"""
# pylint: disable=global-statement

import time
from math import ceil
import board
import rotaryio
import neopixel
from adafruit_debouncer import Debouncer
import digitalio
import pwmio

# Setup the hardware

encoder = rotaryio.IncrementalEncoder(board.D9, board.D7)
button_io = digitalio.DigitalInOut(board.D10)
button_io.direction = digitalio.Direction.INPUT
button_io.pull = digitalio.Pull.UP
button = Debouncer(button_io)
strip = neopixel.NeoPixel(board.D11, 16, brightness=1, auto_write=False)

last_position = 0

def check_encoder():
    """Check if the encoder has been rotated.
    returns the direction (-1 or +1) if it has, 0 if not.
    """

    global last_position
    position = encoder.position
    if position > last_position:
        direction = 1
    elif position < last_position:
        direction = -1
    else:
        direction = 0
    last_position = position
    return direction


def show_time(color, value, bright):
    """Show remaining time on the ring.
    :param int color: the RGB value to use
    :param int value: how many pixels to light
    :param boolean bright: whether the highest pixel should be brighter (i.e white)
    """
    strip.fill(0x000000)
    if value > 0:
        for i in range(1, value + 1):
            strip[16 - i] = color
        if bright:
            strip[16 - value] = 0x404040
    strip.show()


def set_timer(color, value):
    """Set a time remaing value
    :param int color: the color to use on the ring
    :param int value: the initial value (number of pixels to light)
    Returns the new setting
    """
    global last_position
    time_setting = value
    last_position = encoder.position
    for i in range(16):
        strip[i] = color
        strip.show()
    for i in range(16):
        strip[i] = 0x000000
        strip.show()
    while True:
        show_time(color, time_setting, False)
        button.update()
        if button.fell:
            return time_setting
        direction = check_encoder()
        time_setting += direction
        if time_setting > 16:
            time_setting = 16
        if time_setting < 0:
            time_setting = 0


def beep(count, duration, interstitial, freq):
    """Make some noise
    :param int count: the number of beeps to make
    :param float duration: the length (in seconds) of each beep
    :param float interstitial: the length (in seconds) of the silence between beeps
    :param int freq: the frequency of the beeps
    """
    pwm = pwmio.PWMOut(board.D12, duty_cycle = 0, frequency=freq)
    for _ in range(count):
        pwm.duty_cycle = 0x7FFF
        time.sleep(duration)
        pwm.duty_cycle = 0
        time.sleep(interstitial)
    pwm.deinit()


def compute_mode_settings(new_mode):
    """Compute settings for a new mode
    :param boolean new_mode: the new mode
    Returns
      boolean mode       - the new mode
      int dial_color     - the dial color for the new mode
      int time_remaining - the initial time-remaining for the new mode
      int increment      - the pixel increment for the new mode
    """
    work_time_increment = 600
    break_time_increment = 300

    if new_mode:
        return True, 0x400000, work_time * work_time_increment, work_time_increment
    else:
        return False, 0x004000, break_time * break_time_increment,
break_time_increment
```

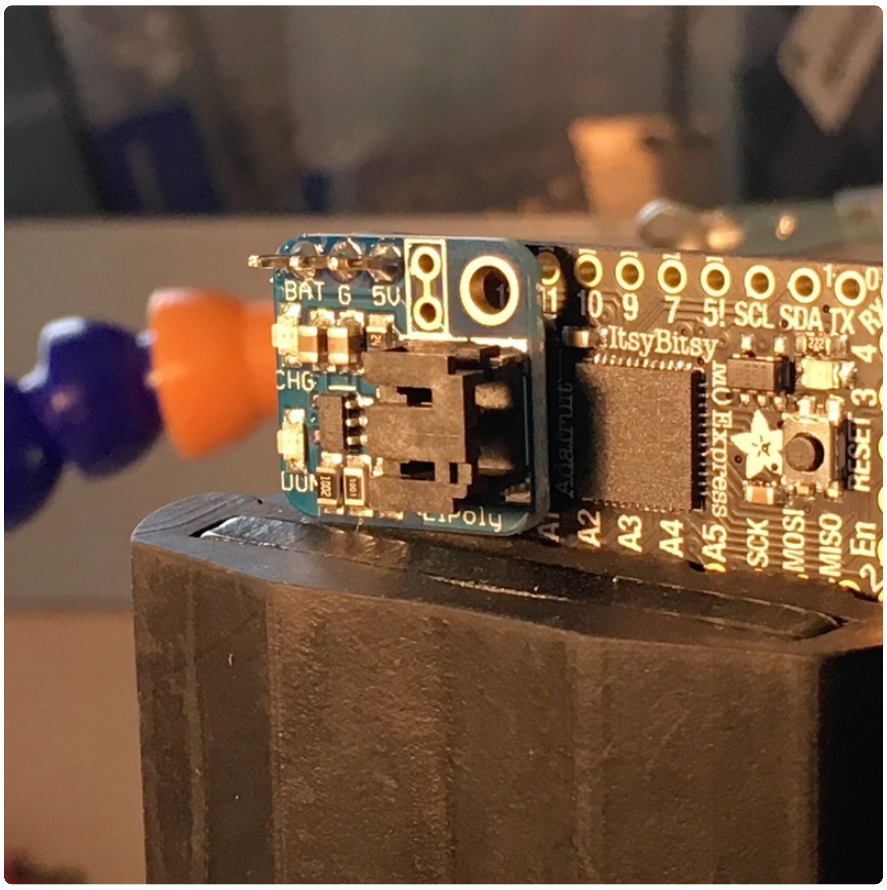
```
# Initialize things

strip.fill(0x000000)
strip.show()
work_time = 6
break_time = 2
time_to_check = 0
state = False
mode, dial_color, time_remaining, increment = compute_mode_settings(True)

# The main loop

while True:
    # check whether the rotary encoder has been pushed. If so enter time-set mode.
    button.update()
    if button.fell:
        work_time = set_timer(0x400000, work_time)
        break_time = set_timer(0x004000, break_time)
        strip.fill(0x000000)
        strip.show()
        mode, dial_color, time_remaining, increment = compute_mode_settings(True)

    now = time.monotonic()
    if now >= time_to_check:          #only check each second
        time_remaining -= 1
        if time_remaining <= 0:       # time to switch modes?
            strip.fill(0x000000)      # clear the dial
            strip.show()              # make some noise
            beep(2, 0.5, 0.25, 4000)
            mode, dial_color, time_remaining, increment = compute_mode_settings(not
mode)
        state = not state             # have the top pixel toggle between the dial
color and white
        show_time(dial_color, ceil(time_remaining / increment), state)   #update
the dial
        time_to_check = now + 1.0
```

Assembly

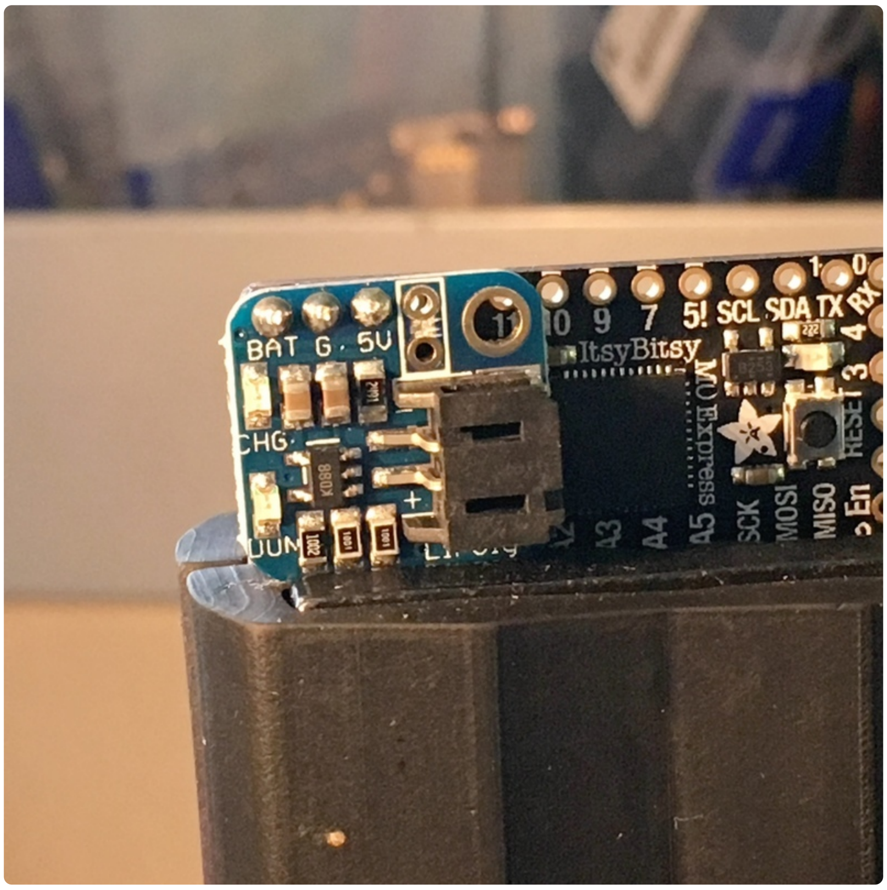
The picture below shows all the parts, ready for assembly. The STL files for printing the case are at the bottom of this page.



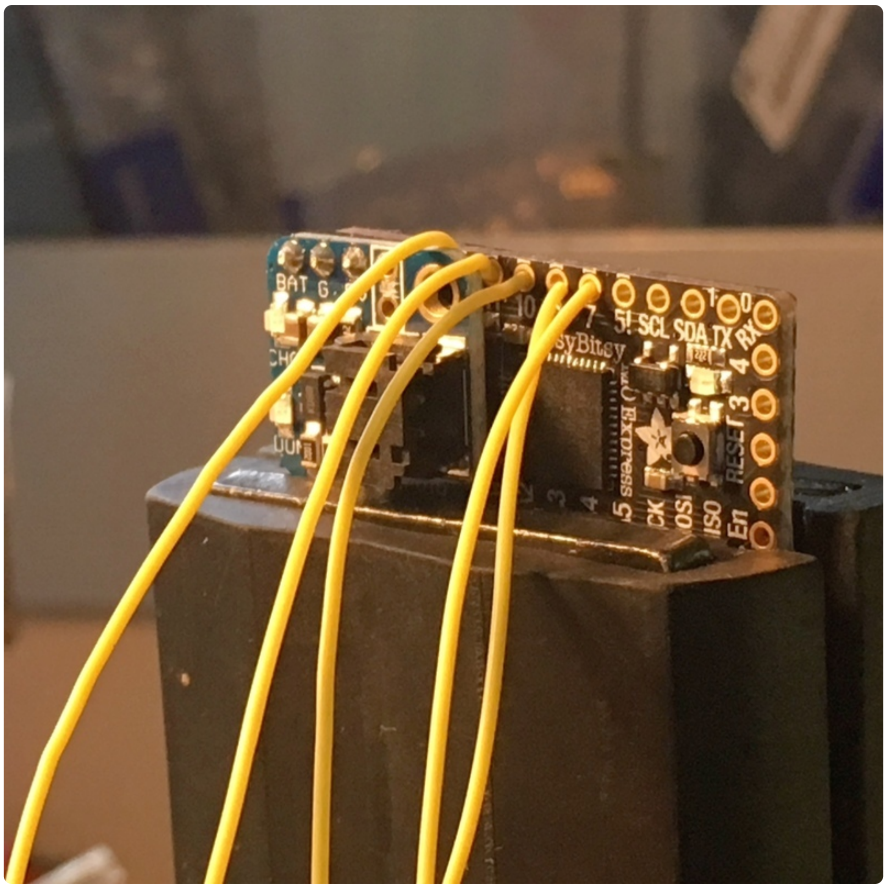
We'll start by mounting the LiPo backpack on the ItsyBitsy using the long header pins provided. Remember to trim the header pins afterward.



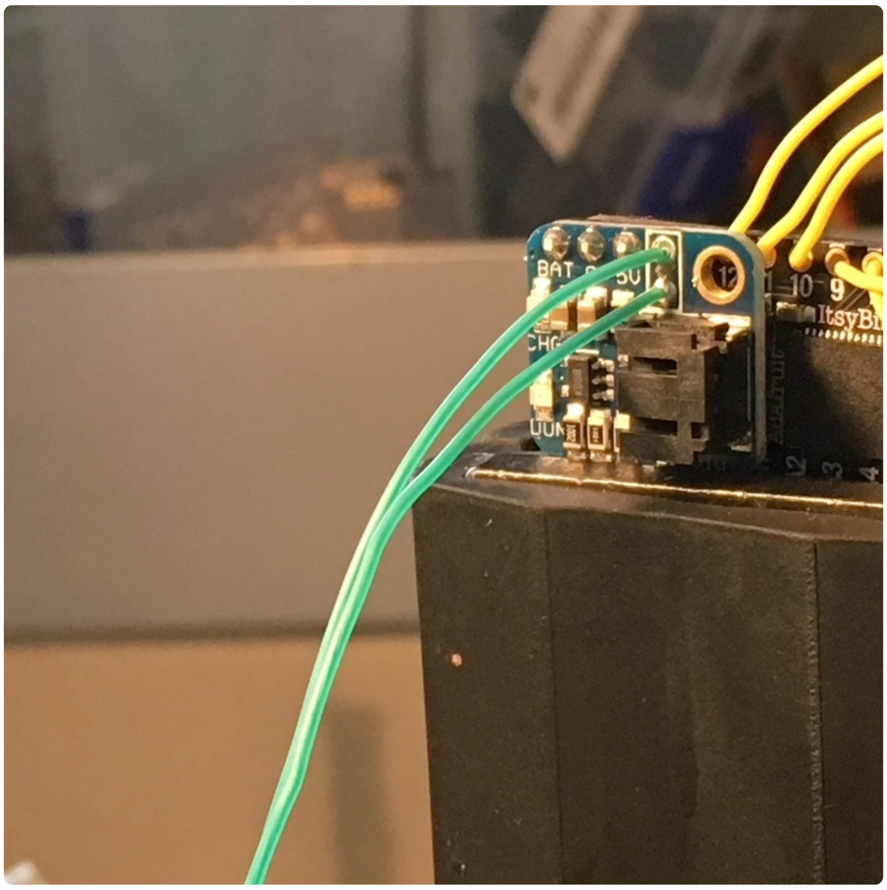
In order to use the switch for power, the trace between the switch pads must be cut. That's shown here.



The next step is to add wires for the NeoPixels, buzzer, and rotary encoder. I used yellow for these. Be sure to make them plenty long, you can always trim them later. 120cm (4-5 inches) is fine.

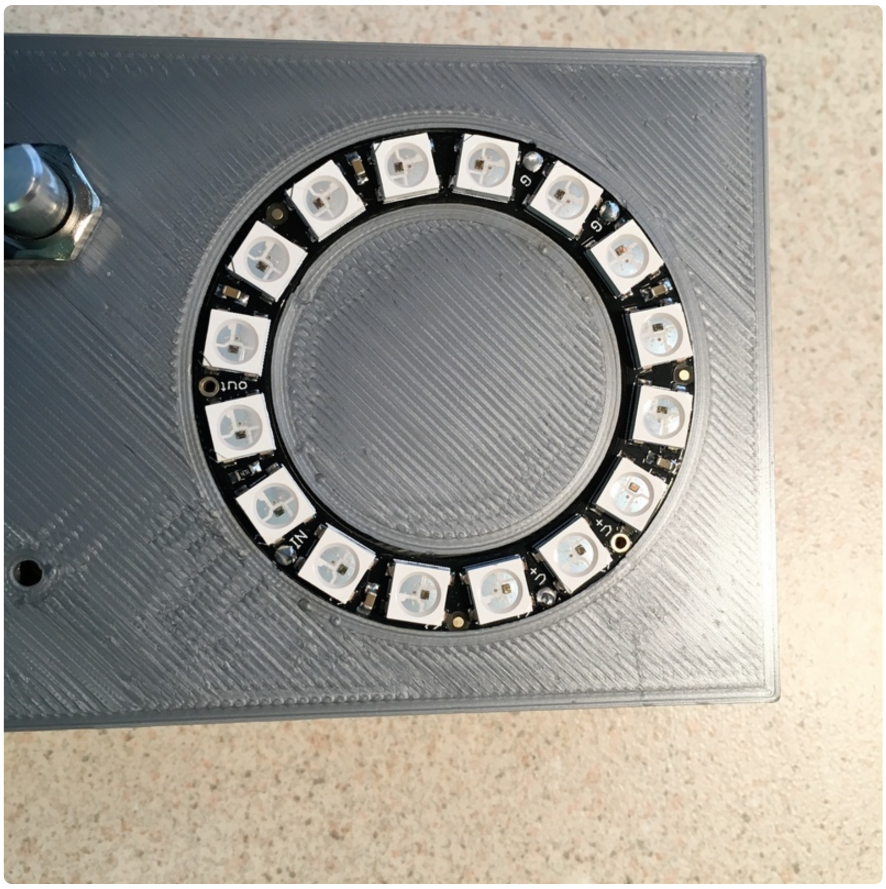


Next add two wires for the power switch. I used green for these. Again, 120cm or so (4-5 inches) is plenty.

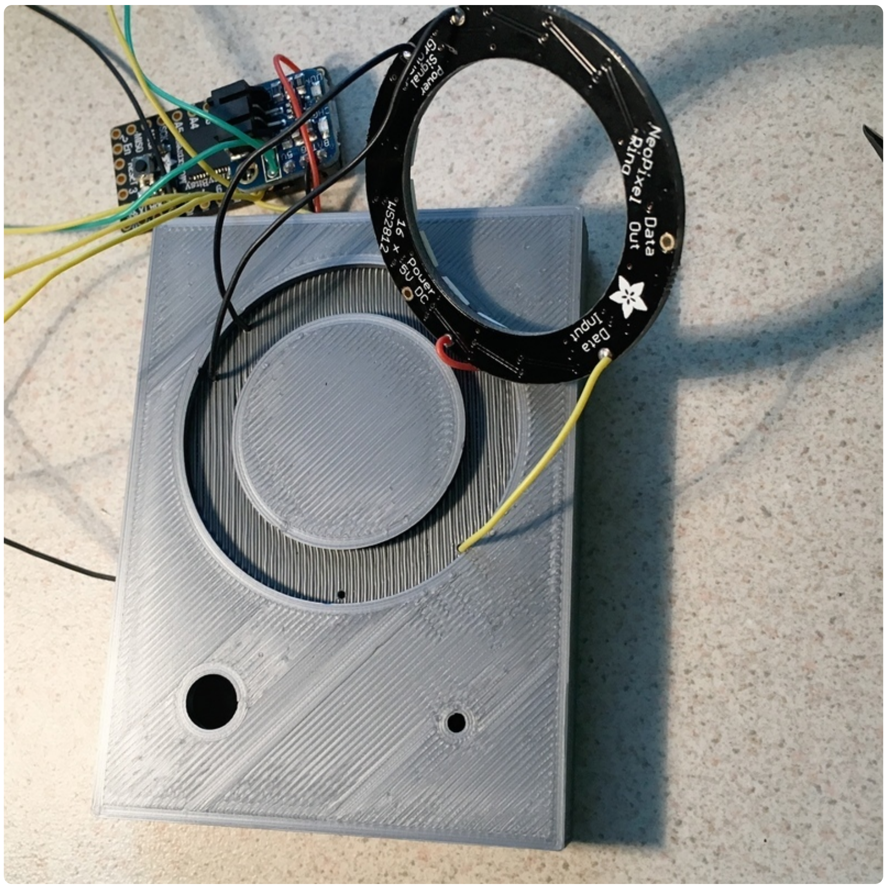


The last step for the ItsyBitsy is ground and power. The ground hole is being used by the LiPo backpack, so you will need to solder the ground wire onto the end on the pin on the backpack.

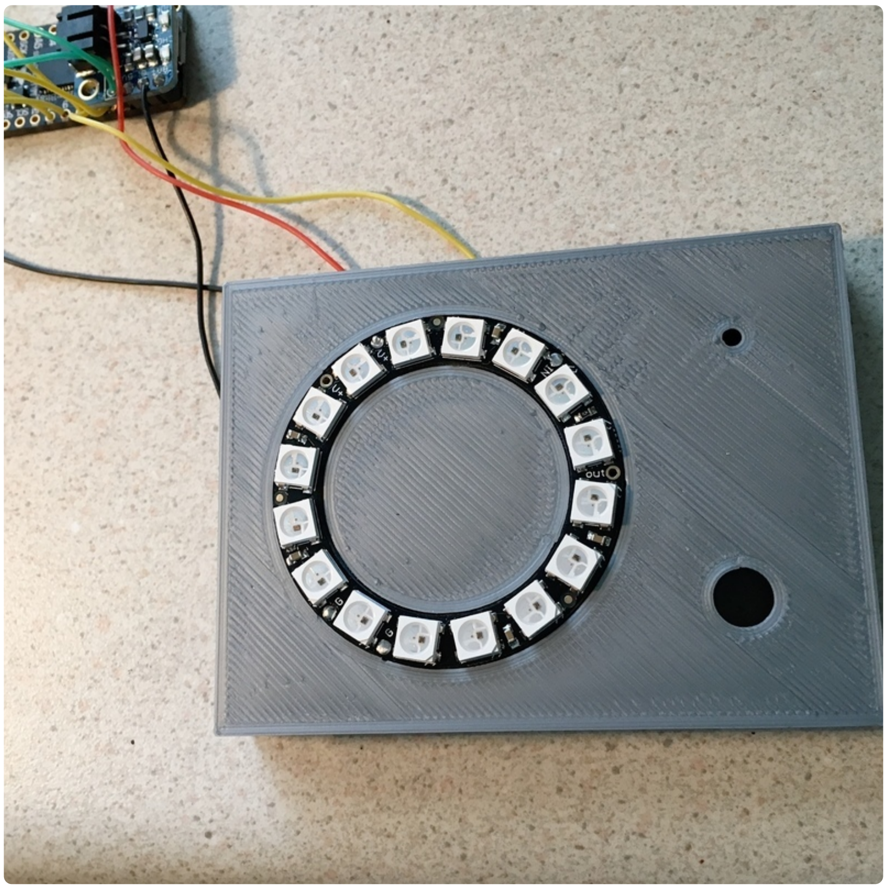
The next step requires a fine drill bit. I have a small jewelers drill that's idea for this, and a drill bit that fits a through-hole PCB hole. Position the NeoPixel ring in its place such that the Data-out connection is at the bottom. Drill a hole in the case at the Data-In, one power, and both ground holes.



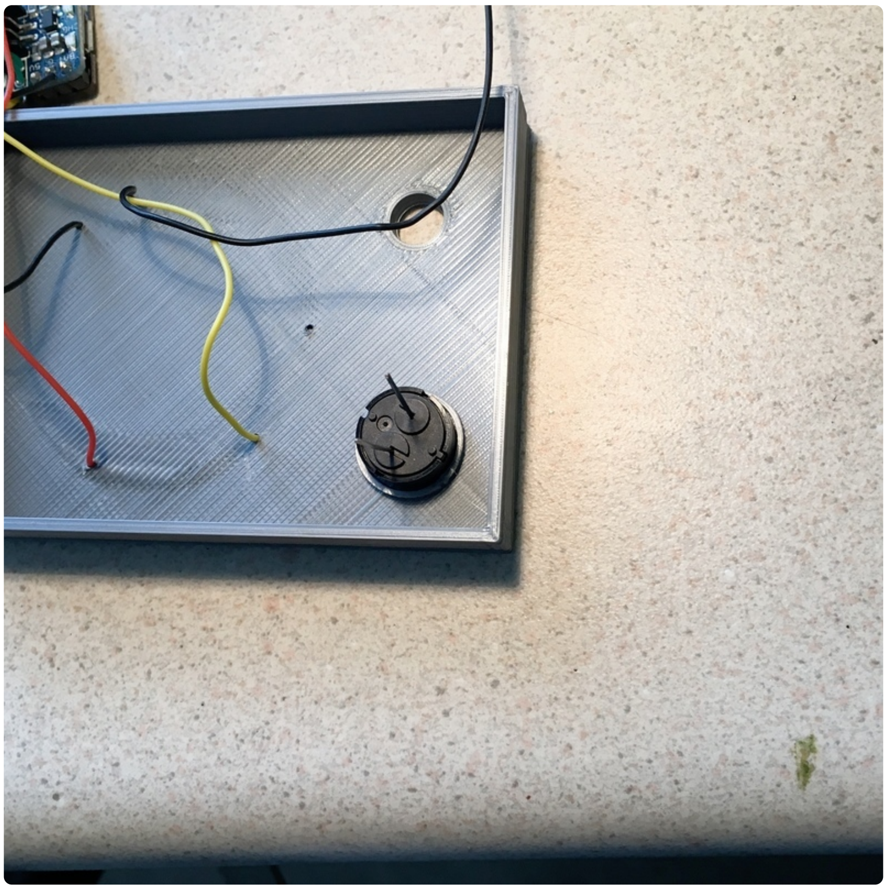
Feed the power, ground, and Neopixel connection (from D11 of the ItsyBitsy) through their respective holes you just drilled in the case and solder them to appropriate holes in the ring. Verify that it's soldered on both sides just in case the plated holes were compromised during the marking/drilling (it's unlikely but you can't be too safe). Cut another short (around 60cm or 2-3 inches) length of black wire and feed that through the second ground hole in the case. Solder that to the second ground connection on the ring.



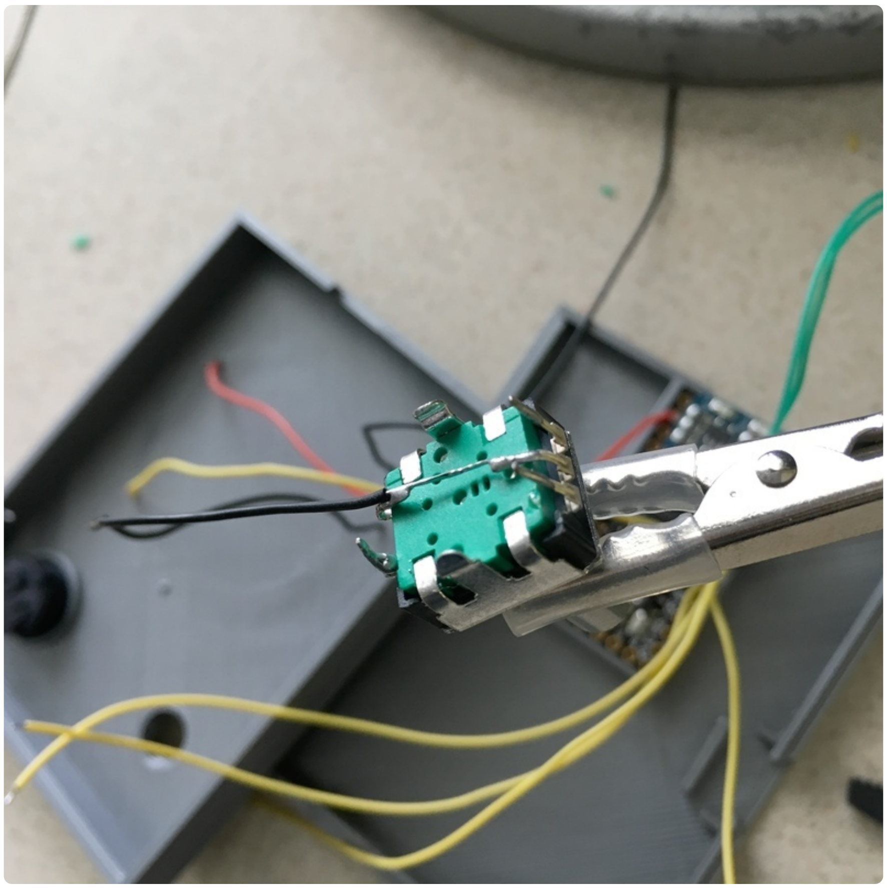
Now gently pull the wires through the front case and settle the ring into the circular groove. A few dabs of superglue will help it stay in place.



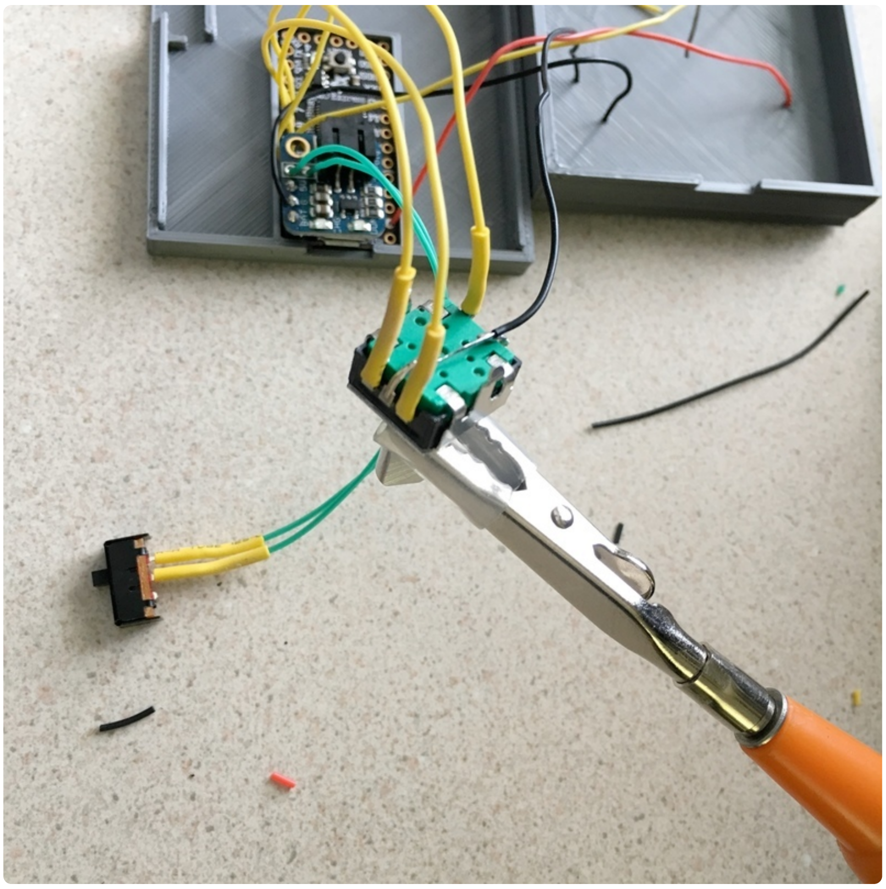
Next use a bit of superglue to hold the buzzer in place. There is a ring inside the top part of the case to help get it in the right spot.



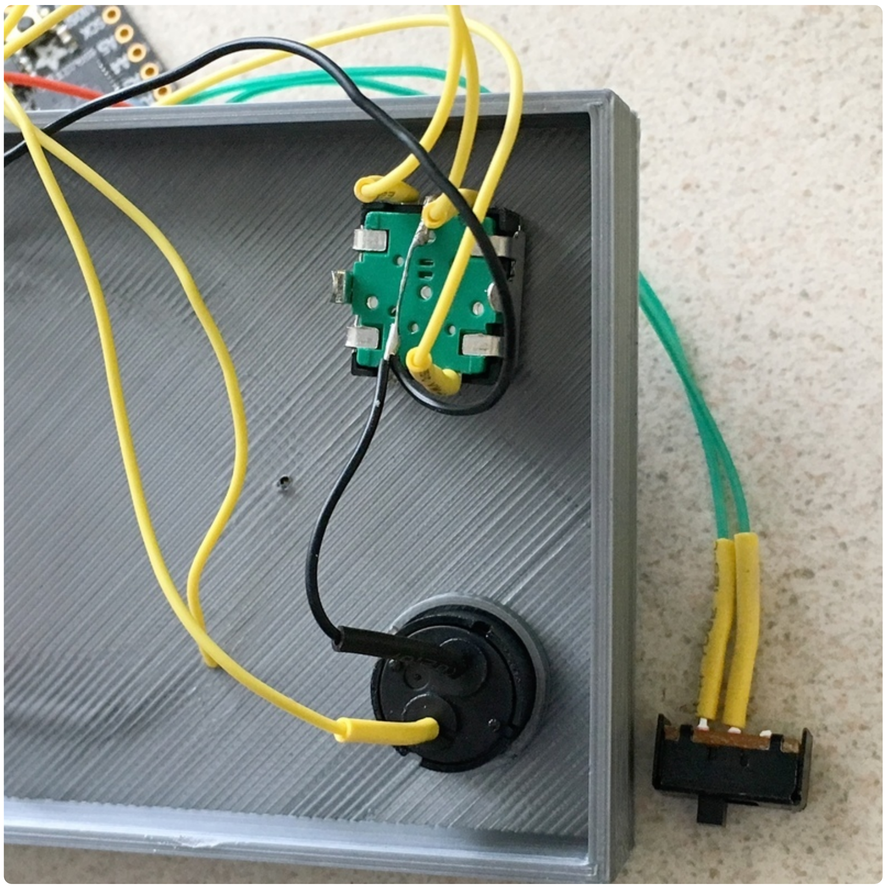
Next we can wire the rotary encoder. Start by stripping/tinning about 1 cm (0.5 inches) from the end of the ground wire coming from the ring. Connect this to one side of the encoders switch (the side with two connections) and the middle connection (of 3) on the other side.



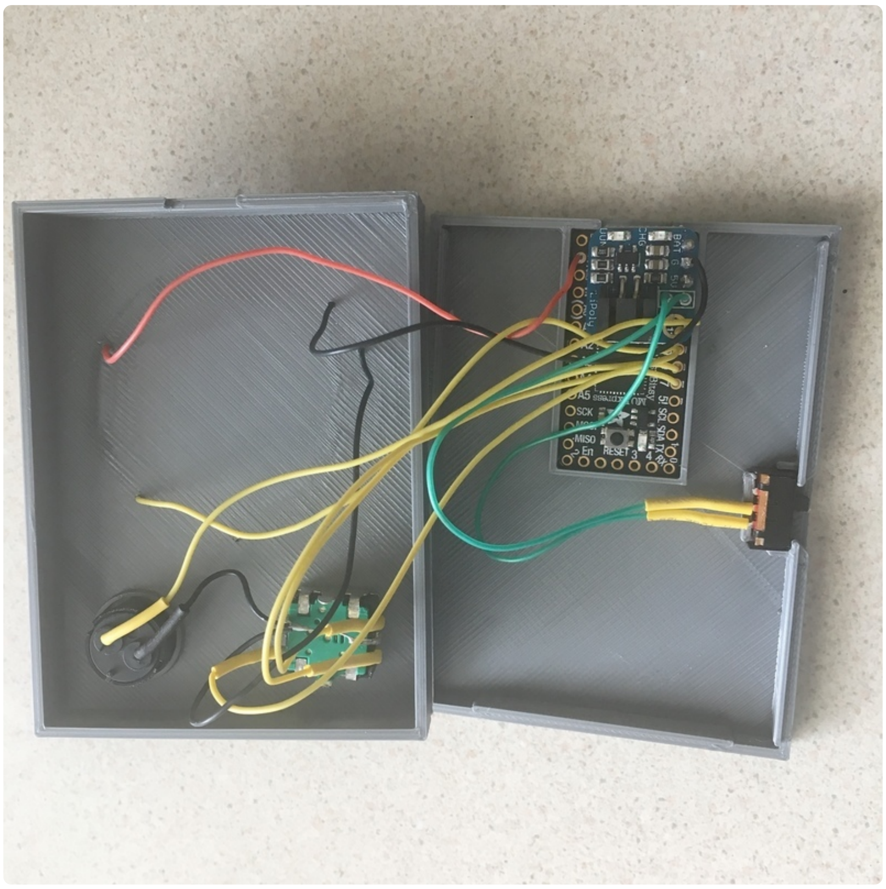
Connect the correct wire from the ItsyBitsy's digital pins to the encoders other pins (see the wiring diagram). I used short pieces of heatshrink on the connections to strengthen them as well as avoid any chance of shorting. Always a good idea. Wire the two green wires from the LiPo backpack to the center pin and one end pin of the slide switch. Which pins they connect to doesn't matter. I generally clip one of the end pins off. You can see this in the photos below.



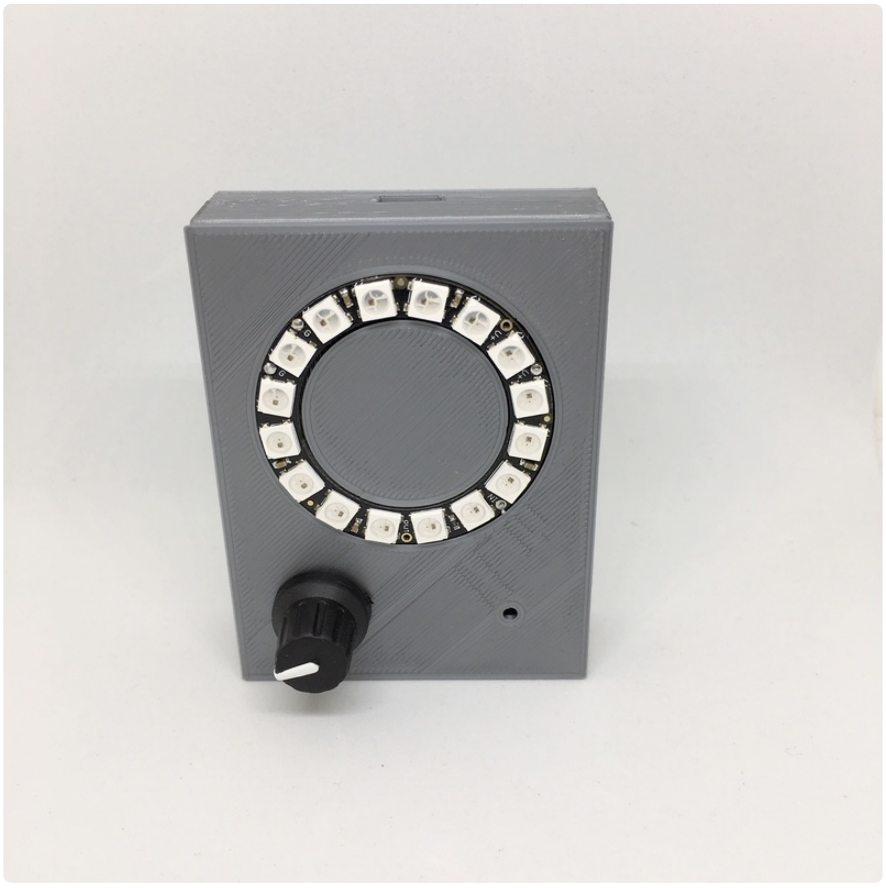
Connect the final wire from the ItsyBitsy (from D12) to the buzzer. It doesn't matter which connection. Cut a piece of black wire to reach from the ground connection on the encoder to the other connection of the buzzer. Again, I used some heatshrink on these connections.

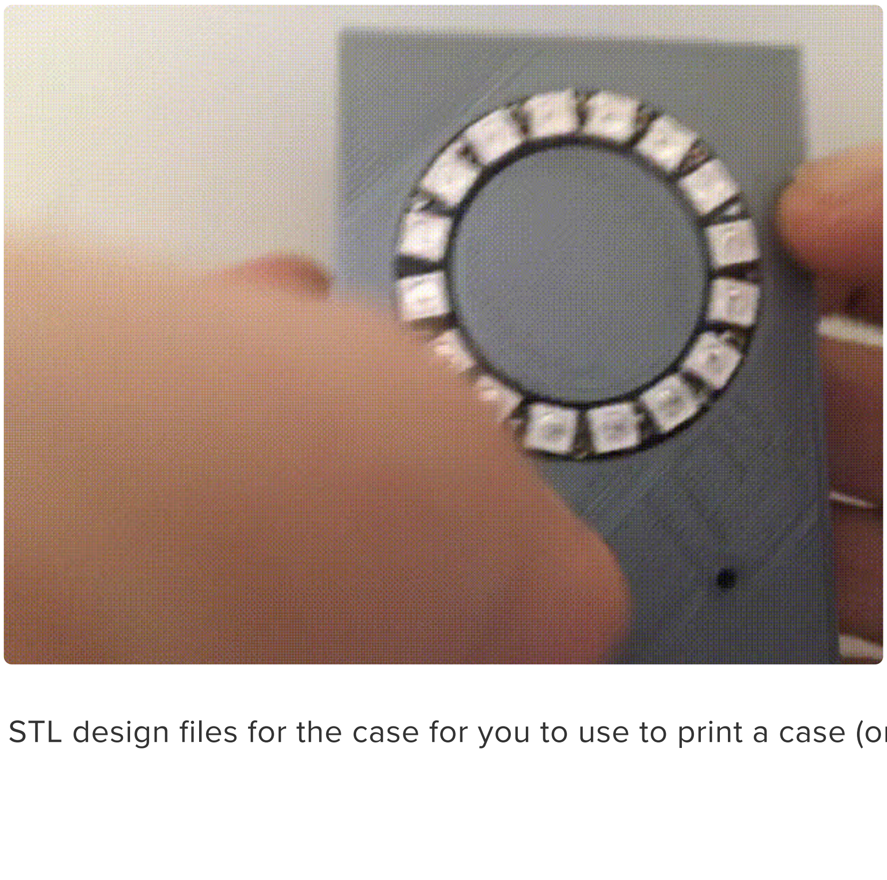


Now we can use superglue to mount the ItsyBitsy and the power switch.



All that's left is to connect a battery and seal up the case (careful not to catch any of the wiring between the parts of the case. My case design is snug so it might take a little finessing. I used a few dabs of superglue to hold the two pieces together. It could be redesigned with a fancier clip system if desired.





Here are the STL design files for the case for you to use to print a case (or sent out to be printed):

pomodoroBottom.stl

https://adafru.it/BJ9

pomodoroTop.stl

https://adafru.it/BJa