Dylan Falzone
12/11/2023
4539-8897

EE Design 1 Technical Report
Final Project
"Dog Operational Gear for Monitoring and Analysis (DOGMA)"

Introduction

In August, I got a dog. In just 4 months he has changed my life for the better, so I wanted to use this project as an opportunity to give back to him.

Figure 1: Pictures of Cornelius for reference and scale.

Despite his small stature, Cornelius' large personality has caused some trouble with the neighborhood dogs, so the first design requirement is to make him look as tough as he feels. He also has several health sensitivities, so the second design requirement is to keep track of his biometrics. These requirements combined with those assigned for this project led me to design DOGMA. This suit, controllable from an LCD with tactile inputs and audio feedback, will employ configurable LED lights, a body temperature sensor, and a heart rate monitor.

NOTE: DOGMA is a project whose scope is beyond that of this course, involving 3D printing and integration. For the sake of this report, I will just focus on the electronic systems.

Methods

System Overview:

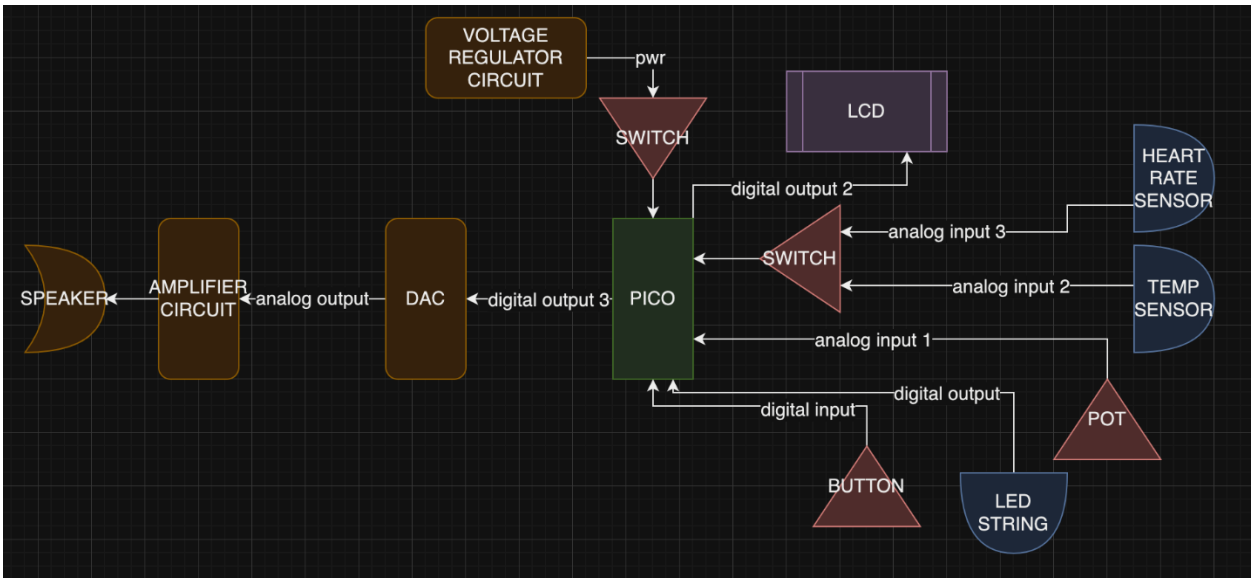


Figure 2: Hardware block diagram for DOGMA.

The major subsystems and components shown in figure 2 are detailed in the next section, so for now we will discuss the tactile inputs, pictured in red.

The potentiometer is used to select an option on the LCD, and the button is used to enter the selection. Another potentiometer is used but not pictured just to adjust the contrast on the LCD.

The power supply has a single switch to toggle the system on and off. Notably, there is also a switch used to alternate an ADC channel between the temperature and heart rate sensors. These sensors needed to share a channel since Raspberry Pi Pico only has 3 ADC channels the other two needed to be connected to the main control potentiometer and a ground reference.

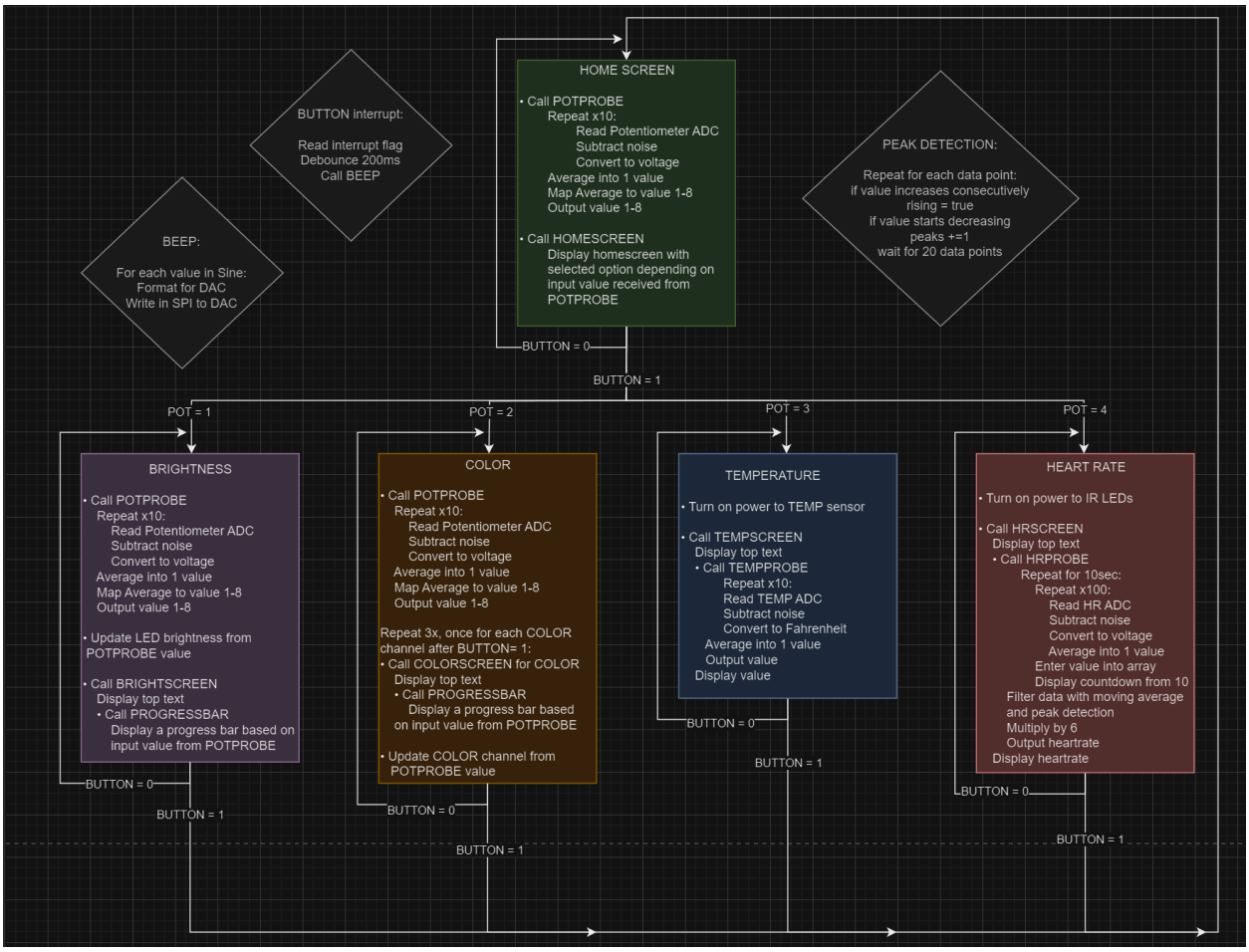


Figure 3: Software Diagram for DOGMA.

DOGMA's software can be thought of as a finite state machine with one initial state that can lead to any of 4 other states. The main state corresponds to the home screen, with an option for each of DOGMA's 4 functions: LED Brightness, LED Color, Temperature, and Heart Rate. The basics operations of each state are pictured, most of which include probing one or more ADC channels and displaying text and variables to the LCD. I utilized functions as much as possible to keep things simple and allow for easy re-use of code.

Also pictured are a few functions that didn't quite fit into the 5 states including the button interrupt, the audio output, and the peak detection algorithm for the heart rate state. All states are entered and exited by simply pushing the previously mentioned button.

Schematic Overview:

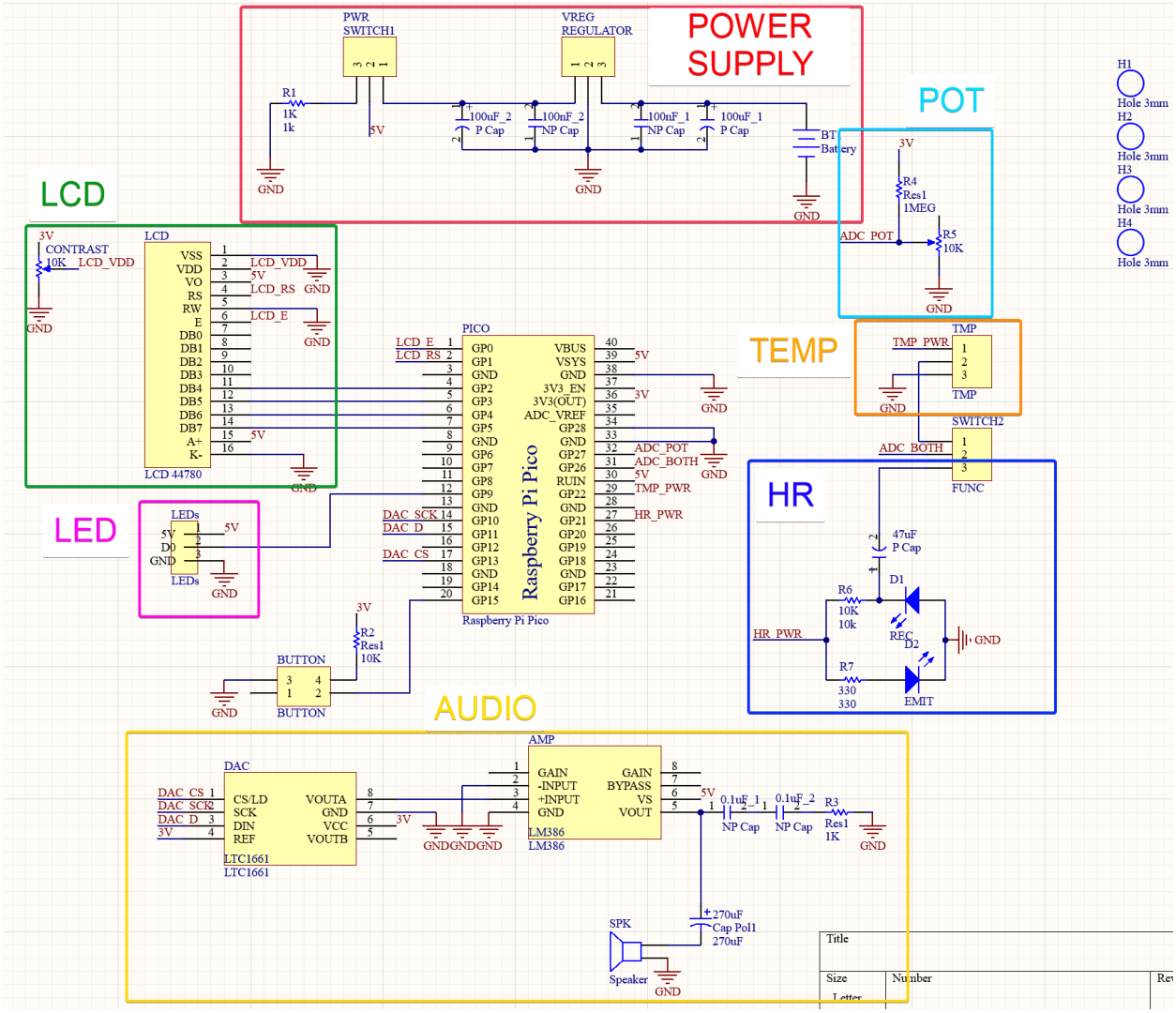


Figure 4: Altium Schematic with color coordinated selections.

Before diving into each of the more complicated subsystems, let's discuss the simpler ones. The LCD, pictured in green, mostly consists of connections directly to the microcontroller, spare the previously mentioned potentiometer to control the contrast. For the LCD, I used the exact same configuration and code that was shown to us in the lab, so it was a simple addition to the system.

A similarly easy addition is the LED string, pictured in pink. Needing only power, ground, and a data line, the LED string is also connected directly to the microcontroller. The LEDs are programmed by transmitting a basic 24bit color code via bit-banging on the data line [1]. I used a library from github designed for my exact microcontroller and LED string, which allowed me to program the color channels and overall brightness of the string individually [2].

Power Supply:

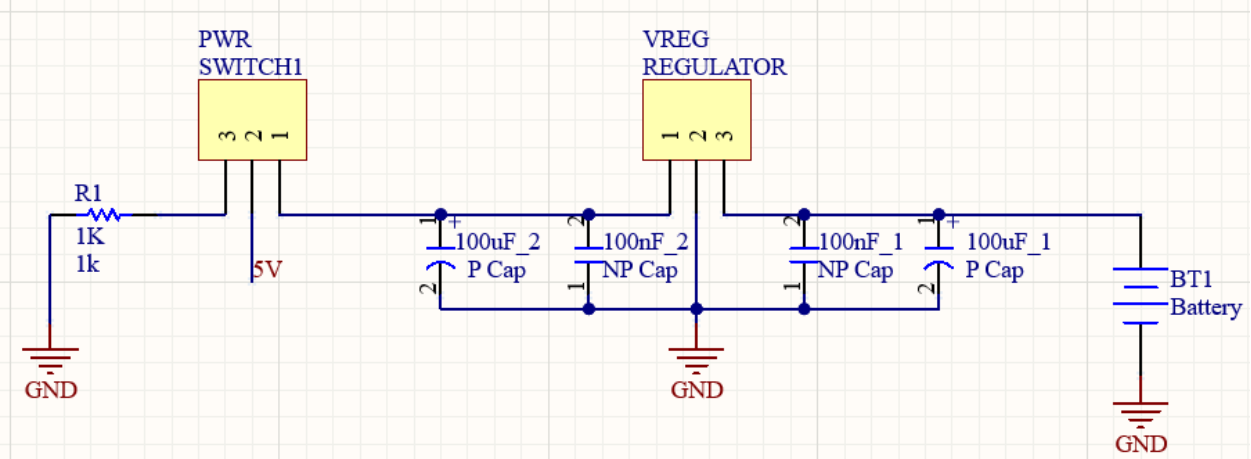


Figure 5: Power supply subsystem close-up.

The star of the power supply subsystem is the voltage regulator. It ensures that even when the supply voltage is higher, like the 9V that is currently in use, the system only receives a safe 5V. Decoupling capacitors on the input and output of the regulator also serve to protect the system from any high-frequency noise that may be coming from the power supply [3].

The previously mentioned power switch, when in the rightmost position, connects the system voltage (VSYS) to the output of the regulator, turning the system on. When in the other position, it connects VSYS to a grounded resistor, allowing the microcontroller to be plugged in without issue or just turning off the battery power.

It is also worth noting that I used modified header footprints in the Altium schematic for the regulator, switch, and any other parts I couldn't find.

Analog Output:

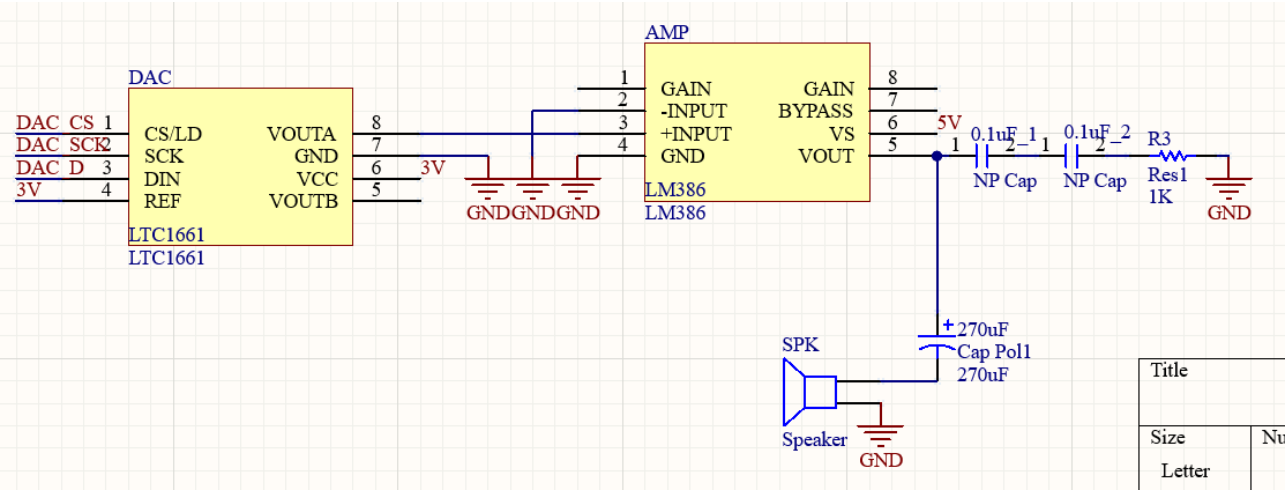
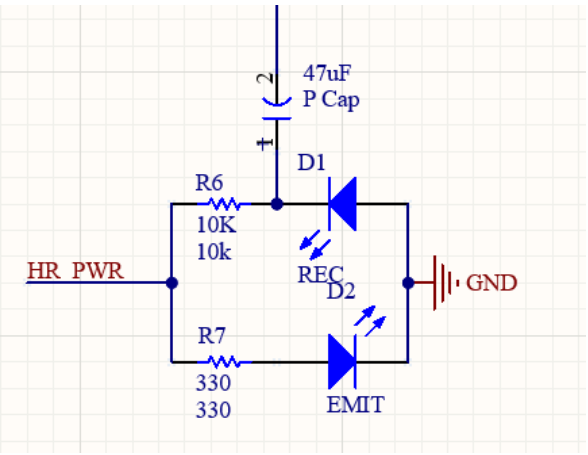


Figure 6: Analog output subsystem close-up.

The analog output subsystem uses the DAC to send a signal to the amplifier where it is boosted and then sent to the speaker. I attempted to replicate the configuration that I used successfully in the lab, which came from the LM386 datasheet, but I forgot a critical component [4]. This had some adverse effects, which I will discuss in the results section.

The most interesting aspect of this subsystem is the use of capacitors on the output. Two 0.1uF capacitors are used in series to form a 0.05uF capacitor, which acts as a bypass for high-frequency noise. Additionally, a 270uF coupling capacitor is used on the input to the speaker to remove any dc offset and allow the speaker to represent the signal accurately.

Heart Rate:



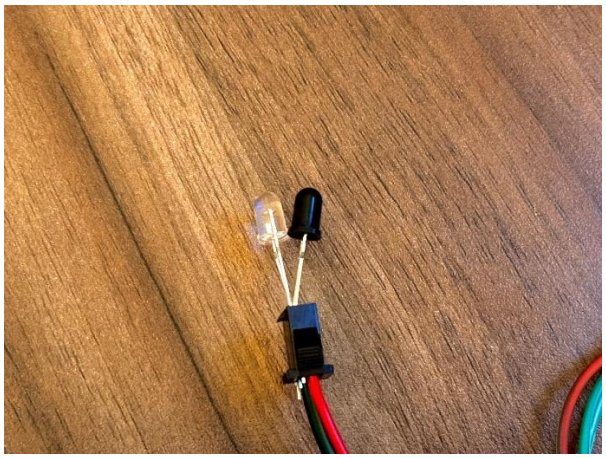


Figure 7: Heart Rate subsystem close-up and diode pair image.

Like even the most advanced heart rate sensors, at its core, this circuit is just two infrared diodes. The clear emitter diode takes in current and emits infrared light. The black receiver diode takes in infrared light and emits a current. From a physics perspective, the receiver is the same as any photodiode, just tuned for the longer wavelengths in the infrared range and coated in a black color to block visible light [5].

This circuit can detect heartbeats since its longer wavelengths pass through skin and are absorbed by oxygenated blood [6]. This means that when the diode pair is placed against skin, the amount of IR light received and consequently the current generated changes every time the user's heart beats. This change can be measured and used to determine heart rate.

The circuit configuration I used was an amalgamation of others that I saw online. A smaller resistor is used for the emitter so that a large current can't burn it out, and a coupling capacitor is used to remove the dc offset from the receiver's output.

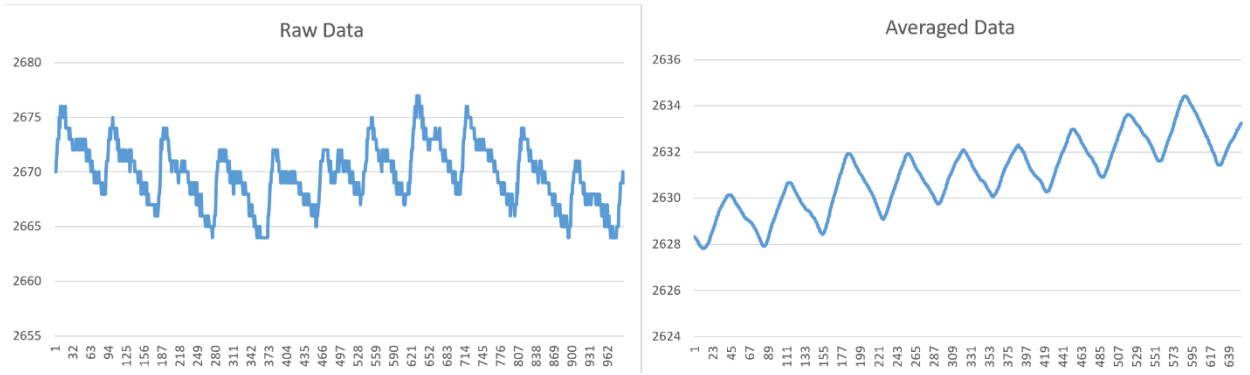


Figure 8: Raw heart rate data (left) and a different but similar set of heart rate data after filtering (right).

Data from this sensor is incredibly noisy and very sensitive to movements and lighting conditions. As such, a lot of processing had to be done to get the number of peaks within each ~1000 sample data set. First, a standard moving average filter had to be run on the data, smoothing it significantly as can be seen on the right side of figure 8. Next, I used a custom peak detection algorithm to measure relative increases and decreases to determine the number of peaks. A more detailed description of this algorithm can be found in figure 3. Coincidentally, taking ~1000 samples takes almost exactly 10 seconds, so the number of peaks from each cycle gets multiplied by 6 to get the number of predicted beats per minute.

Temperature:

Temp	Voltage
-40	0.1
25	0.75
50	1

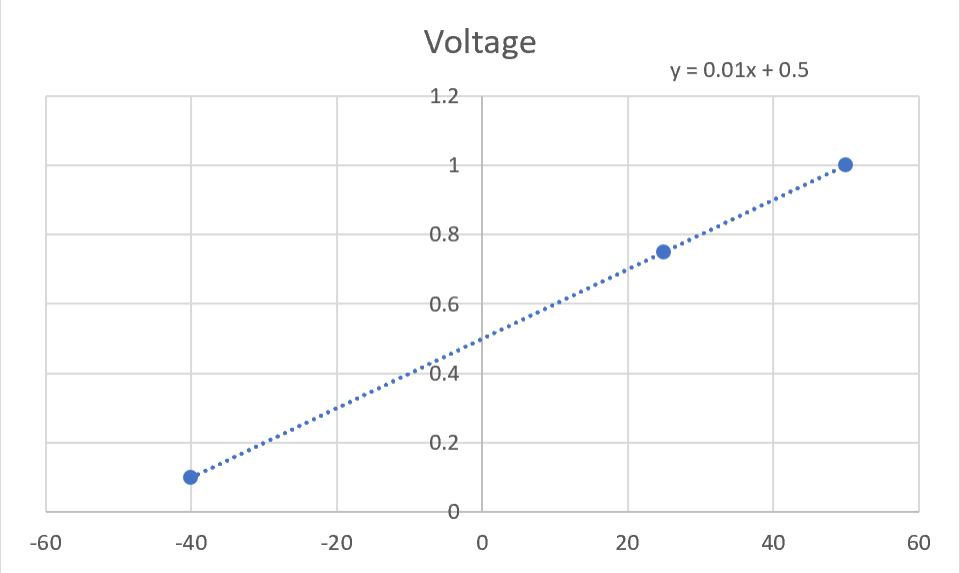


Figure 9: Characterization for temperature sensor.

The temperature sensor was much simpler to characterize. Though the datasheet does provide a conversion equation for voltage to temperature, I opted to try it on my own to confirm it. The temperature sensor itself works by measuring the voltage deviation of transistors as the temperature changes. This specific sensor uses BJT's since they are especially sensitive to temperature changes, and have a linear relationship between temperature and output voltage [7].

Materials and PCB:

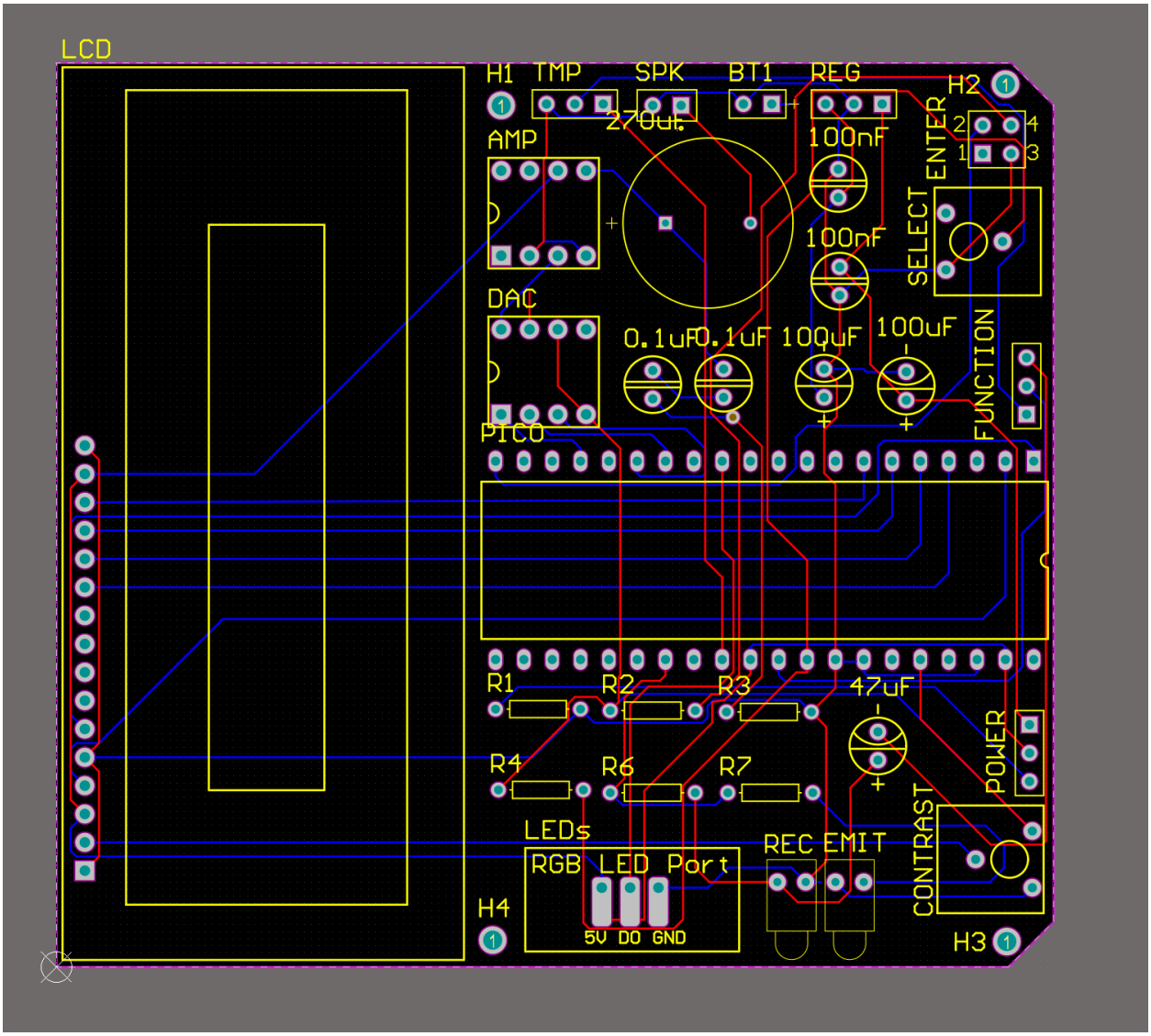


Figure 10: Altium PCB Layout.

My primary goal with the PCB was to keep everything as tight as possible, so that it will be able to fit on my dog's back. I made sure to put capacitors close to their corresponding components, and I put the tactile inputs on the bottom of the board so the user wouldn't have to reach over other components for access to the controls. All of the ancillary components were placed near the edges of the board so they will be able to be distributed throughout the suit. I also added copper pours for ground and 3.3V to help reduce noise.

Component	Price	Quantity	Link
LM7805 Regulator	$1.87	1.00	https://www.digikey.com/en/products/detail/onsemi/LM7805CT/458698
44780U LCD	$3.66	1.00	https://www.digikey.com/en/products/detail/universal-solder-electronics-ltd/LCD-1602-2x16-Blue-White/16821383
Raspberry Pi Pico H	$5.00	1.00	https://www.sparkfun.com/products/20172
ws2812b LED strip	$29.50	1.00	https://www.ledlightinghut.com/ws2812-digital-intelligent-rgb-led-strip-light.html
LTC1661 DAC	$5.65	1.00	https://www.digikey.com/en/products/detail/analog-devices-inc/LTC1661CMS8-PBF/890711
LM386 AMP	$1.28	1.00	https://www.digikey.com/en/products/detail/texas-instruments/LM386N-3-NOPB/148191
Speaker	$1.75	1.00	https://www.adafruit.com/product/1891?gad_source=1&gclid=CjwKCAiAg9urBhB_EiwAgw88mcSqs7t1qA-
TMP36GT9Z Temp	$2.30	1.00	https://www.digikey.com/en/products/detail/analog-devices-inc/TMP36GT9Z/820404
IR Diode pair x50	$10.99	1.00	https://www.amazon.com/XLX-Infrared-Emission-Receiver-Arduino/dp/B01MFCFLA7
Button	$0.37	1.00	https://www.digikey.com/en/products/detail/schurter-inc/1301-9314-24/8536716
Switch	$0.64	2.00	https://www.digikey.com/en/products/detail/c-k/JS202011SCQN/2094299
Potentiometer x12	$7	2.00	https://www.amazon.com/Potentiometer-Breadboard-Resistors-Assortment-

10k Resistor	$0.10	2.00	https://www.digikey.com/en/products/detail/stackpole-electronics-inc/CF14JT10K0/1741265
1k Resistor	$0.10	2.00	https://www.digikey.com/en/products/detail/yageo/MFR-25FBF52-1K1/13019
330ohm Resistor	$0.10	1.00	https://www.digikey.com/en/products/detail/yageo/CFR-25JB-52-330R/1636
270uF cap	$1.67	1.00	https://www.digikey.com/en/products/detail/rubycon/100ZLJ270M12-5X30/3133968
0.1uF cap	$0.23	2.00	https://www.digikey.com/en/products/detail/kemet/C410C104M5U5TA7200/674715
100uF cap	$0.37	2.00	https://www.digikey.com/en/products/detail/nichicon/UPW1C101MED/589520
100nF cap	$0.67	2.00	https://www.digikey.com/en/products/detail/kyocera-avx/TAP104K035SRW/1470992
47uF cap	$0.44	1.00	https://www.digikey.com/en/products/detail/nichicon/UKA0J470MDD/4992578
PCBx5	$25	1.00	JLCPCB.com
Total:	$107.78		https://www.digikey.com/en/products/detail/onsemi/LM7805CT/458698

Figure 11: Bill of Materials.

Results

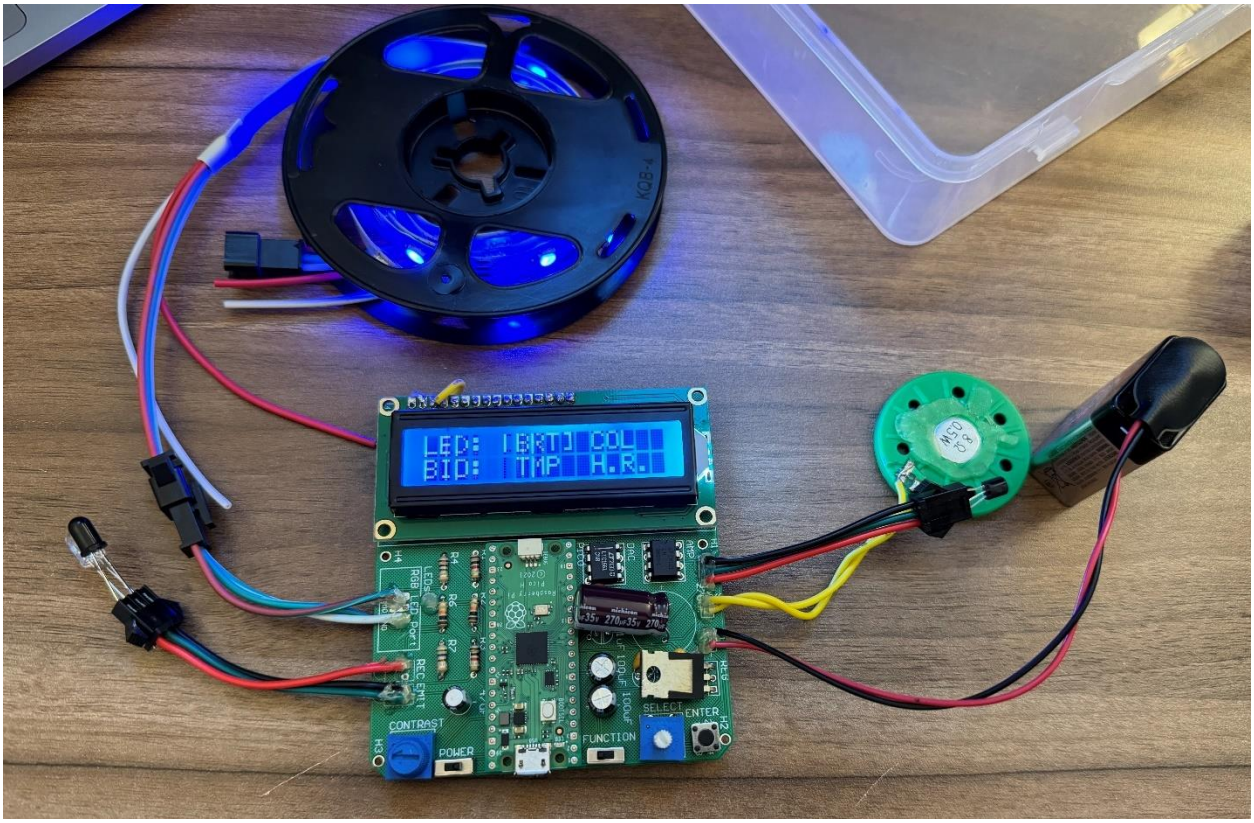


Figure 12: DOGMA Electronics fully functional, displaying home screen with brightness selected.

Overall, the system works! Every component is functional, with just a select few having limited functionality due to minor issues. The heart rate sensor, LEDs, temperature sensor, power system, and code all work exactly as intended.

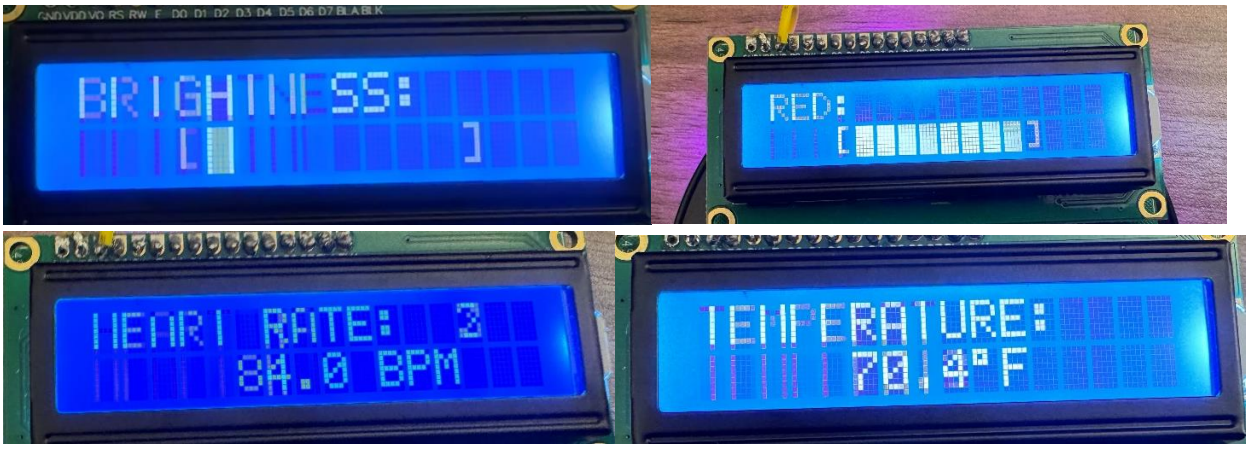


Figure 13: LCD displaying each of the 4 function screens.

As can be seen in the figures, the LCD does have some readability issues on the left side. The exact cause of this is unknown, but is likely due to some improvised soldering, desoldering, cutting, melting, and re-wiring that I had to do between the ground, VO and VDD pins. This was because I mixed up the VO and VDD pins on the PCB, but I did not realize it until after I had already soldered the LCD. It is still functional though, and just has some very faint white streaks on the left side of the screen.

There was also an auto-route error that caused the LED data line to short with the 5V line. This would have made the LEDs non-functional but I fixed it by changing the data pin in the code and soldering the data line directly to that pin.

Figure 14: Oscilloscope measurements of DAC output (Left) and Speaker input (Right).

As previously alluded to, I made a mistake when constructing the analog output subsystem. I did not account for the dc offset of the DAC's output. As a result, only 2 of the 25 data points in the sine wave were actually realized at the output of the amplifier. If I had simply added a coupling capacitor, the full sine wave would likely have been passed successfully.

The speaker had seemingly been working correctly during my testing. I even tried differen types of waves, all of which produced different tones, before settling on the sine wave. It took me until the demo to realize that what I had been hearing was not a sine wave at all, but two points of a sine wave and one point of unidentified origin. Despite my error, the system does have a 3 level analog output which surprisingly makes for excellent audio feedback

Discussion/Conclusion

Despite the shortcomings, DOGMA is fully operational. And while Cornelius will soon get to reap the rewards, I will not be left empty handed. I will take with me the many lessons learned from this project and this class as a whole, like when and how to use bypass, coupling, and decoupling capacitors, how to design a PCB and get it printed, how to characterize an

incredibly noisy and inconsistent sensor, and ultimately, how NOT to connect a DAC to an amplifier.

As far as the next phases of DOGMA, after finals week I plan to begin working on the 3D printing and integration. Given its scope, I'm not sure that I would have undertaken DOGMA at all had I not been allowed to design the electronics for this project, so both Cornelius and I are very grateful for the opportunity.

References

[1] Ws2812B Datasheet, https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf (accessed Dec. 12, 2023).

[2] Blaz-r, "Pi_pico_neopixel," GitHub, https://github.com/blaz-r/pi_pico_neopixel/blob/main/README.md (accessed Dec. 11, 2023).

[3] T. Zedníček, "Capacitor selection for coupling and decoupling applications," Passive Components Blog, https://passive-components.eu/capacitor-selection-for-coupling-and-decoupling-applications/ (accessed Dec. 11, 2023).

[4] LM386 low voltage audio power amplifier datasheet (rev. D), https://www.ti.com/lit/ds/symlink/lm386.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1701848499960 (accessed Dec. 12, 2023).

[5] Nawielectron and Instructables, "Intro to Ir Circuits," Instructables, https://www.instructables.com/Intro-to-IR-Circuits/ (accessed Dec. 11, 2023).

[6] D. Mercer, "Activity: Heart Rate Monitor Circuit," Activity: Heart Rate Monitor Circuit [Analog Devices Wiki], https://wiki.analog.com/university/courses/alm1k/alm-lab-heart-rate-mon (accessed Dec. 11, 2023).

[7] Swagatam, "How to use diodes, transistors, ICS as temperature sensors," Homemade Circuit Projects, https://www.homemade-circuits.com/how-to-use-diodes-transistors-ics-as-temperature-sensors/ (accessed Dec. 11, 2023).

Appendix

```
import machine

import math
```

```
import utime
import LCD
from neopixel import Neopixel

"""PINS=============================================================================
================================================="""
#Function Pins
pot_pin = machine.ADC(27) #ADC1 = POT
ground_pin = machine.ADC(28)#ADC2 = Ground ref
bio_pin = machine.ADC(26)#ADC0 = temp or HR
button_pin = machine.Pin(15, machine.Pin.IN) #GP15, pin20 = button
temp_toggle_pin = machine.Pin(22, machine.Pin.OUT) #GP22, pin 29 = power for TMP
hr_toggle_pin = machine.Pin(21, machine.Pin.OUT) #GP21, pin 27 = power for HR

#DAC pins
spi = machine.SPI(1, baudrate=1000000, polarity=0, phase=0, sck=machine.Pin(10), mosi=machine.Pin(11))
cs = machine.Pin(13, mode=machine.Pin.OUT, value=1)
ref = machine.Pin(12, mode=machine.Pin.OUT, value=1)

#LCD pins
EN = machine.Pin(0, machine.Pin.OUT)
RS = machine.Pin(1, machine.Pin.OUT)
D4 = machine.Pin(2, machine.Pin.OUT)
D5 = machine.Pin(3, machine.Pin.OUT)
D6 = machine.Pin(4, machine.Pin.OUT)
D7 = machine.Pin(5, machine.Pin.OUT)
PORT = [D4, D5, D6, D7]
```

```
"""FUNCTIONS================================================================================================"""


#Button ISR changes value of var: switch
def interrupt(pin):
    global flag
    global button
    if flag == 1:
        button = 1
        beep(1)
        utime.sleep_ms(200) #somewhat essential debouncing
    flag = 0


#Beep speaker for given length
def beep(length):
    for k in range(20*length):
        for i in sine:
            buf = (0xF000 | (i << 2)).to_bytes(2,1)
            cs(0)
            spi.write(buf)
            cs(1)


#Probe ADC0 for pot value 1-7
def potprobe():
    pot_values = [None] * 10
    for i in range(10):
        #Read ADC pins
        pot_v = pot_pin.read_u16()
        ground_v = ground_pin.read_u16()
        #Turn them into voltages
        pot_v = (pot_v / 65535.0) * 3.3
```

```
    ground_v = (ground_v / 65535.0) * 3.3
    #Get a voltage without noise
    pot_v = (pot_v-ground_v)
    #Edge case for 0v
    if pot_v <= 0:
        pot_v = 0.000805676
    #Fill array
    pot_values[i] = pot_v
#Average the array
pot_values_total = sum(pot_values)
pot_v = pot_values_total/10
pot_v = pot_v*10000

if 1<=pot_v<65:
    pot =1
elif 65<=pot_v<130:
    pot =2
elif 130<=pot_v<195:
    pot = 3
elif 195<=pot_v<260:
    pot = 4
elif 260<=pot_v<325:
    pot = 5
elif 325<=pot_v<390:
    pot = 6
elif 390<=pot_v<455:
    pot = 7
else:
    pot = 8
return(pot)
```

```
def tempprobe():
    temp_values = [None] * 10
    for i in range(10):
        #Read ADC pins
        temp_v = bio_pin.read_u16()
        ground_v = ground_pin.read_u16()
        #Turn it into voltage
        temp_v = (temp_v / 65535.0) * 3.3
        ground_v = (ground_v / 65535.0) * 3.3
        #Get a voltage without noise
        temp_v = (temp_v-ground_v)
        #Edge case for 0v
        if temp_v <= 0:
            temp_v = 0.000805676
        #Fill array
        temp_values[i] = temp_v
    #Average the array
    temp_values_total = sum(temp_values)
    temp_v = temp_values_total/10
    temp_c = (temp_v-0.5)/.01
    temp = temp_c*(9/5) + 32
    return(temp)

def hrprobe():
    global hr_v_last
    beats = 0
    start_t = utime.time()
    #while utime.time() - start_t < 10:
```

```
hr_values2 = [None] * 990
for j in range(990):
    hr_values = [None] * 100
    for i in range(100):
        #Read ADC pins
        hr_v = bio_pin.read_u16()
        ground_v = ground_pin.read_u16()
        #Turn it into voltage
        hr_v = (hr_v / 65535.0) * 3.3
        ground_v = (ground_v / 65535.0) * 3.3
        #Get a voltage without noise
        hr_v = (hr_v-ground_v)*1000
        #Edge case for 0v
        if hr_v <= 0:
            hr_v = 0.000805676
        #Fill array
        hr_values[i] = int(hr_v)
    #Average the array
    hr_values_total = sum(hr_values)
    hr_v = int(hr_values_total/100)

    hr_values2[j] = hr_v
    if j == 0:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("10")
    elif j == 100:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("9 ")
    elif j == 200:
        LCD.lcd_goto(13,0)
```

```
        LCD.lcd_puts("8")
    elif j == 300:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("7")
    elif j ==400:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("6")
    elif j ==500:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("5")
    elif j ==600:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("4")
    elif j ==700:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("3")
    elif j ==800:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("2")
    elif j ==900:
        LCD.lcd_goto(13,0)
        LCD.lcd_puts("1")



print(hr_values2)
smooth = moving_average(hr_values2,30)
print(smooth)
peaks = find_peaks(smooth, 0)
hr = peaks*6
print(hr)
```

```
    return(hr)


def moving_average(data, window_size):
    """Apply a simple moving average filter to the data."""
    filtered_data = []
    for i in range(len(data) - window_size + 1):
        window = data[i : i + window_size]
        average = sum(window) / window_size
        filtered_data.append(average)
    return filtered_data



def find_peaks(data, threshold):
    peaks_count = 0
    rising = 0

    for i in range(1, len(data) - 10):
        if rising ==0:
            if data[i] > data[i-2] and data[i-1] > data[i-3]:
                rising = 1
            else:
                rising = 0
        elif rising ==1:
            if data[i] >= data[i-2] and data[i-1] >= data[i-3]:
                rising = 1
            else:
                rising = 3
                timer = i
                peaks_count +=1
        else:
```

```
            if i == timer+20:
                rising = 0
            else:
                rising = 3




    return peaks_count

#Write variable to LCD
def lcdvar(a):
    LCD.lcd_clear()
    LCD.lcd_home()
    LCD.lcd_puts("{:.2f}".format(a))

#Write variable to LCD
def homescreen(a):
    LCD.lcd_clear()
    LCD.lcd_home()
    if a == 1:
        LCD.lcd_puts("LED: [BRT] COL  ")
        LCD.lcd_goto(0,1)
        LCD.lcd_puts("BIO:  TMP  H.R. ")
    elif a ==2:
        LCD.lcd_puts("LED:  BRT [COL] ")
        LCD.lcd_goto(0,1)
        LCD.lcd_puts("BIO:  TMP  H.R. ")
    elif a ==3:
        LCD.lcd_puts("LED:  BRT  COL  ")
        LCD.lcd_goto(0,1)
```

```
        LCD.lcd_puts("BIO: [TMP] H.R. ")
    else:
        LCD.lcd_puts("LED:  BRT  COL  ")
        LCD.lcd_goto(0,1)
        LCD.lcd_puts("BIO:  TMP [H.R.]")

def brightscreen(a):
    LCD.lcd_clear()
    LCD.lcd_home()
    LCD.lcd_puts("BRIGHTNESS:")
    progressbar(a)

def colorscreen(a, p):
    LCD.lcd_clear()
    LCD.lcd_home()
    if p == 0:
        LCD.lcd_puts("RED:")
        progressbar(i)
    elif p == 1:
        LCD.lcd_puts("GREEN:")
        progressbar(i)
    elif p== 2:
        LCD.lcd_puts("BLUE:")
        progressbar(i)

def tempscreen(a):
    if a == 0:
        LCD.lcd_clear()
        LCD.lcd_home()
        LCD.lcd_puts("TEMPERATURE:")
```

```
    t = tempprobe()
    LCD.lcd_goto(5,1)
    LCD.lcd_puts("{:.2f}".format(t))
    LCD.lcd_goto(9,1)
    LCD.lcd_putch(chr(0xdf))
    LCD.lcd_goto(10,1)
    LCD.lcd_puts("F")

def hrscreen(a):
    if a == 0:
        LCD.lcd_clear()
        LCD.lcd_home()
        LCD.lcd_puts("HEART RATE:")
    hr = hrprobe()
    LCD.lcd_goto(5,1)
    LCD.lcd_puts("{:.2f}".format(hr))
    LCD.lcd_goto(9,1)
    LCD.lcd_puts(" BPM")




def progressbar(a):
    LCD.lcd_goto(3,1)
    LCD.lcd_puts("[")
    for b in range(a):
        LCD.lcd_goto(4+b,1)
        LCD.lcd_putch(chr(0xff))
    """
```

```
    for b in range(8-a):
        LCD.lcd_goto(11-b,1)
        LCD.lcd_puts(" ")
    """
    LCD.lcd_goto(12, 1)
    LCD.lcd_puts("]")




"""VARIABLES/SETUP=============================================================
=================================================="""

sine = [0x200,0x27f,0x2f7,0x35e,0x3b0,0x3e7,0x3ff,0x3f7,
        0x3cf,0x38b,0x32d,0x2bc,0x240,0x1c0,0x144,0xd3,
        0x75,0x31,0x9,0x1,0x19,0x50,0xa2,0x109,
        0x181,0x200]

#FSM contorl variable: 0-4
state = 0

#Repeat counters
presses = 0 #0-3 for cycling colorscreen
repeats = 0 #0-1 for temp
hr_v_last = 0
hr_v_last2 = 0


#Button interrupt variables: both binary
flag = 1
```

```
button = 0


#LED variables
strip = Neopixel(60, 0, 8, "RGB")
brightness = 20 #0-255
green = 0      #0-255
red = 0        #0-255
blue = 255     #0-255
strip.fill((green,red,blue), brightness)
strip.show()



#LCD setup
screenhold = 0 #0-4 but also 10 when changing state
LCD.Configure()
LCD.lcd_init()

# Configure the button pin to trigger the interrupt on a rising edge
button_pin.irq(trigger=machine.Pin.IRQ_RISING, handler=interrupt)

#Beep for system on and to reset speaker
beep(1)

"""MAIN
LOOP================================================================================
=========================================="""
while True:

    #Home Screen - Default State
    if state == 0:
        i = potprobe()
```

```
    i = (i + 2 - 1) // 2 #map i range 1-8 to state/function range 1-4
    if i != screenhold:
        homescreen(i)
    if button == 1:
        state = i
        screenhold = 10
    else:
        state = state
        screenhold = i



#Brightness - State 1
elif state == 1:
    i = potprobe()

    brightness = (i-1)*30
    if i != screenhold:
        brightscreen(i)

    strip.fill((green,red,blue), brightness)
    strip.show()
    #Return to Home Screen
    if button == 1:
        state = 0
        screenhold = 10



#Color - State 2
elif state == 2:
    i = potprobe()
```

```
#Tune Red
if presses == 0:
    if i != screenhold:
        colorscreen(i,presses)
        red = (i-1)*30
        strip.fill((green,red,blue), brightness)
        strip.show()
    if button ==1:
        presses = 1
#Tune Green
elif presses == 1:
    if i != screenhold:
        colorscreen(i,presses)
        green = (i-1)*30
        strip.fill((green,red,blue), brightness)
        strip.show()
    if button ==1:
        presses = 2
#Tune Blue
elif presses == 2:
    if i != screenhold:
        colorscreen(i,presses)
        blue = (i-1)*30
        strip.fill((green,red,blue), brightness)
        strip.show()
    if button ==1:
        presses = 3
#Leave
else:
    presses = 0
```

```
        state = 0
        screenhold = 10


#Temperature - State 3
elif state == 3:
    temp_toggle_pin.value(1) #Turn on temp
    tempscreen(repeats)
    repeats = 1
    #Return to Home Screen
    if button == 1:
        state = 0
        screenhold = 10
        repeats = 0
        temp_toggle_pin.value(0) #Turn off temp


#Heart Rate - State 4
elif state == 4:
    hr_toggle_pin.value(1) #Turn on hr
    hrscreen(repeats)
    repeats = 1

    #Return to Home Screen
    if button == 1:
        state = 0
        screenhold = 10
        repeats = 0
        hr_toggle_pin.value(1) #Turn off hr
```

```
#Return home if state is somehow not between 0 and 4
else:
    state = 0



#Refresh button interrupt
button = 0
flag = 1
#Sleep
utime.sleep_ms(100)
```