EECS 113 | Processor Software/Hardware Interfaces

Final Project

Noah Mathew
ID: 59622566
Date: 6/13/25

Objective

The objective of this project is to implement a BMS system that has an HVAC system, a fire alarm system, a security system, and ambient lighting. We are to use Raspberry Pi, breadboard, wiring, sensors, various components, etc.

Setup

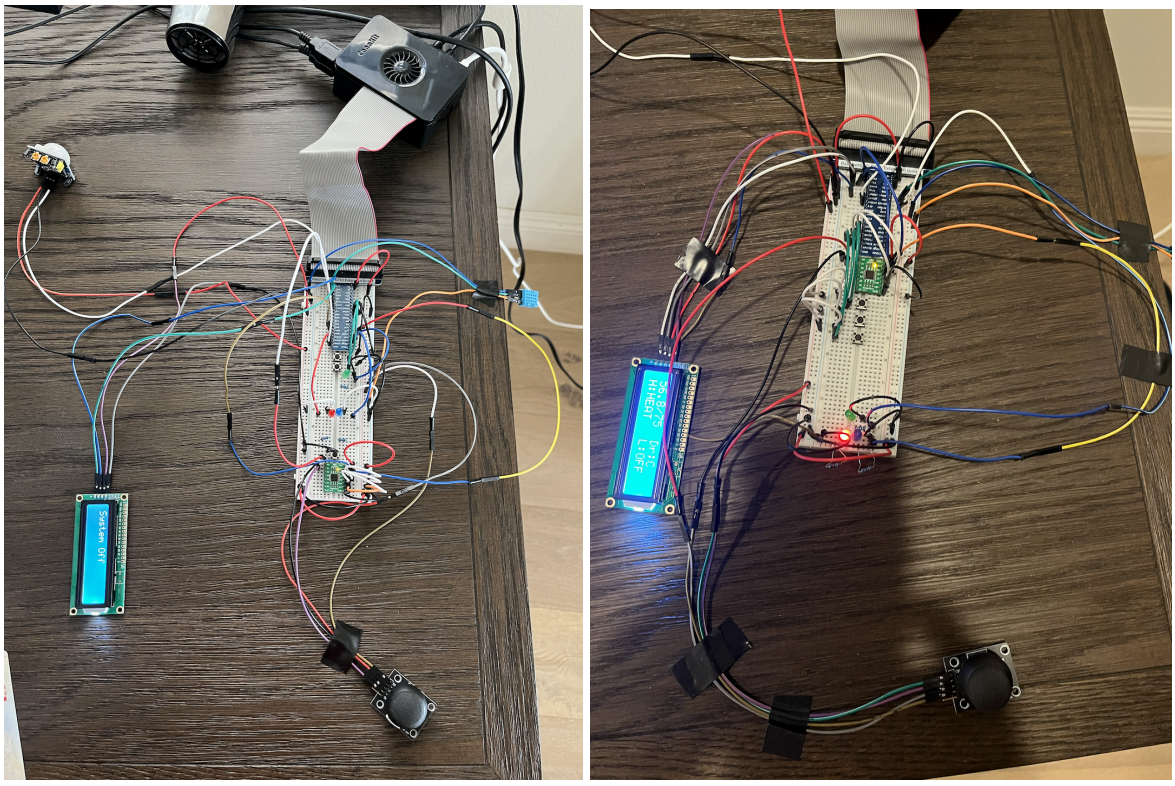
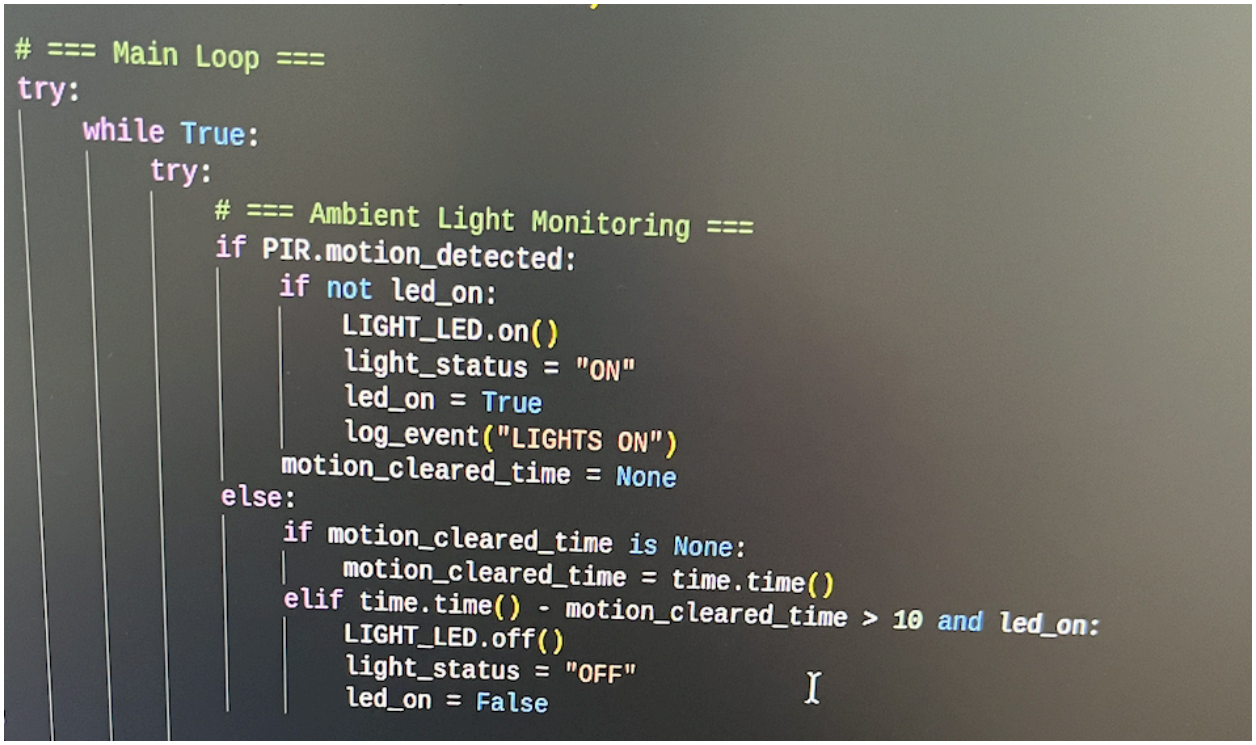


Figure 1: Initial Setup v.s Final Setup w/ Bonus (Wiring more clean as well)

Implementation/Results

Ambient Lighting

Please note that I did calibrate the potentiometer but the sensitivity was still somewhat high. However, the core functionality remains correct, and can be demonstrated in the video. The ambient lighting works where I monitor the motion from the PIR in a while loop and toggle the LED and make the status known that the LED is ON when motion is there, and when motion is not there for 10 seconds relative to the time that motion, we turn it off (i.e. time.time() - motion_cleared_time > 10).



```
# === Main Loop ===
try:
    while True:
        try:
            # === Ambient Light Monitoring ===
            if PIR.motion_detected:
                if not led_on:
                    LIGHT_LED.on()
                    light_status = "ON"
                    led_on = True
                    log_event("LIGHTS ON")
                motion_cleared_time = None
            else:
                if motion_cleared_time is None:
                    motion_cleared_time = time.time()
                elif time.time() - motion_cleared_time > 10 and led_on:
                    LIGHT_LED.off()
                    light_status = "OFF"
                    led_on = False
```

Figure 2: Ambient Light Logic

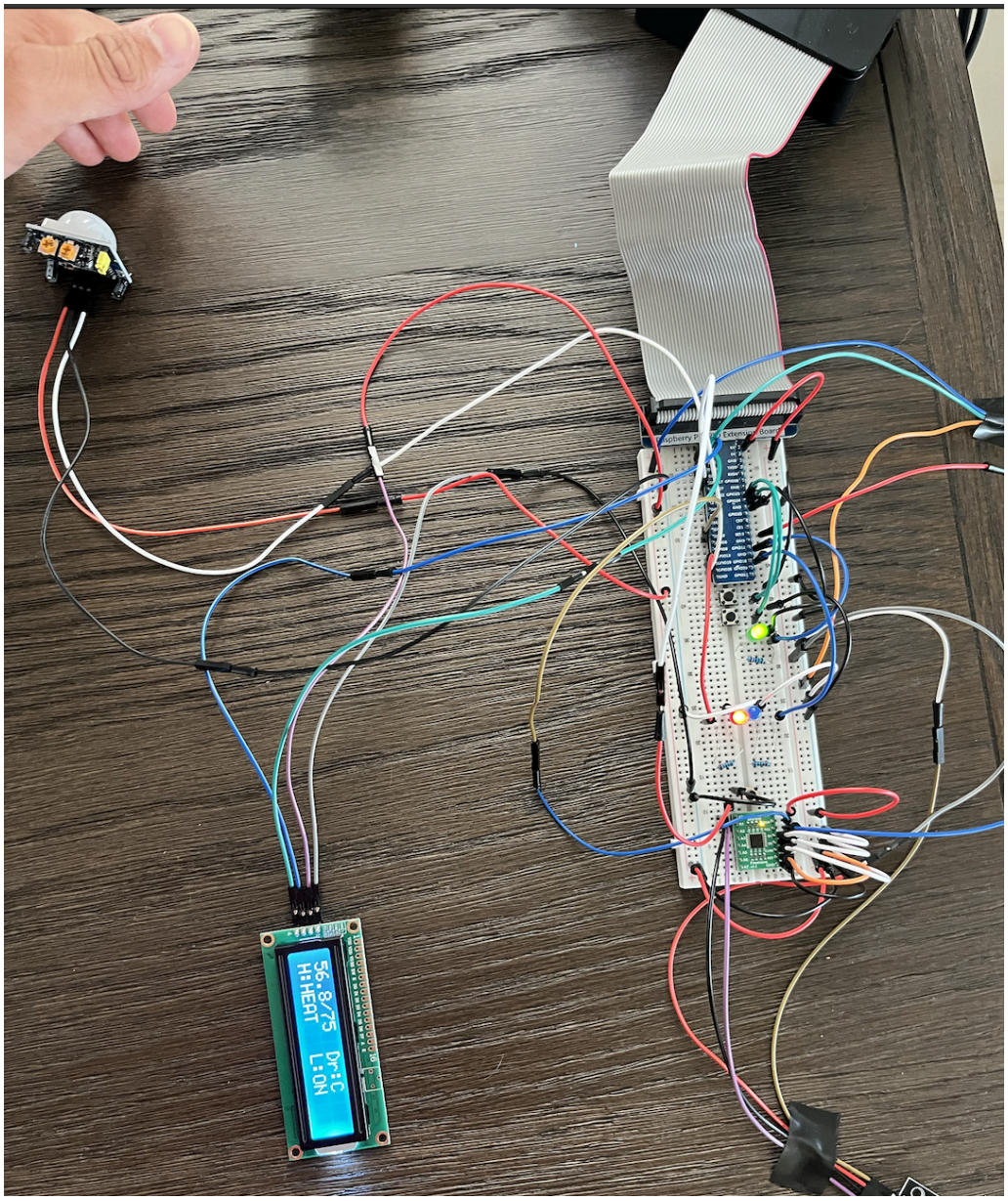


Figure 3: Light ON

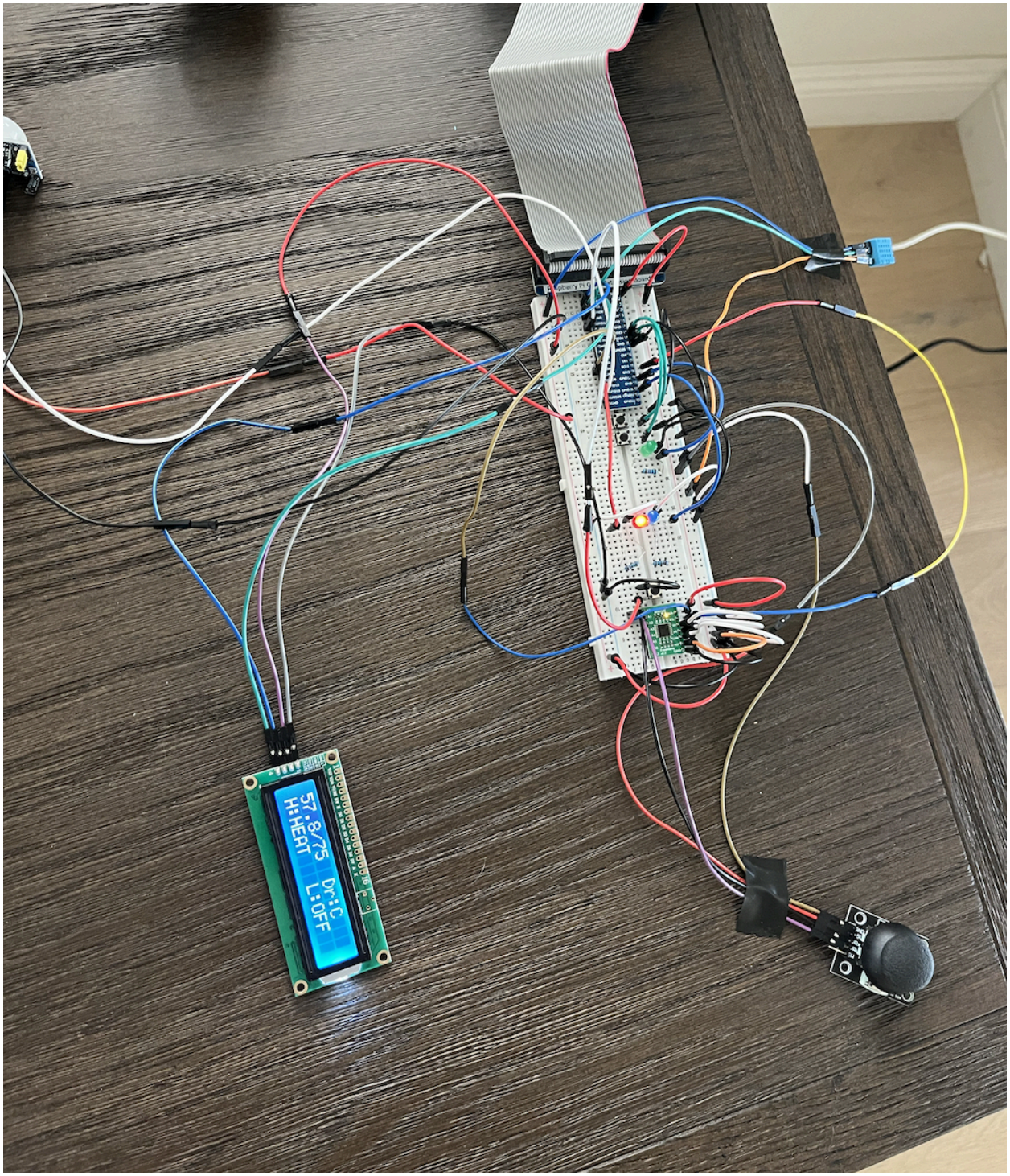


Figure 4: Light OFF

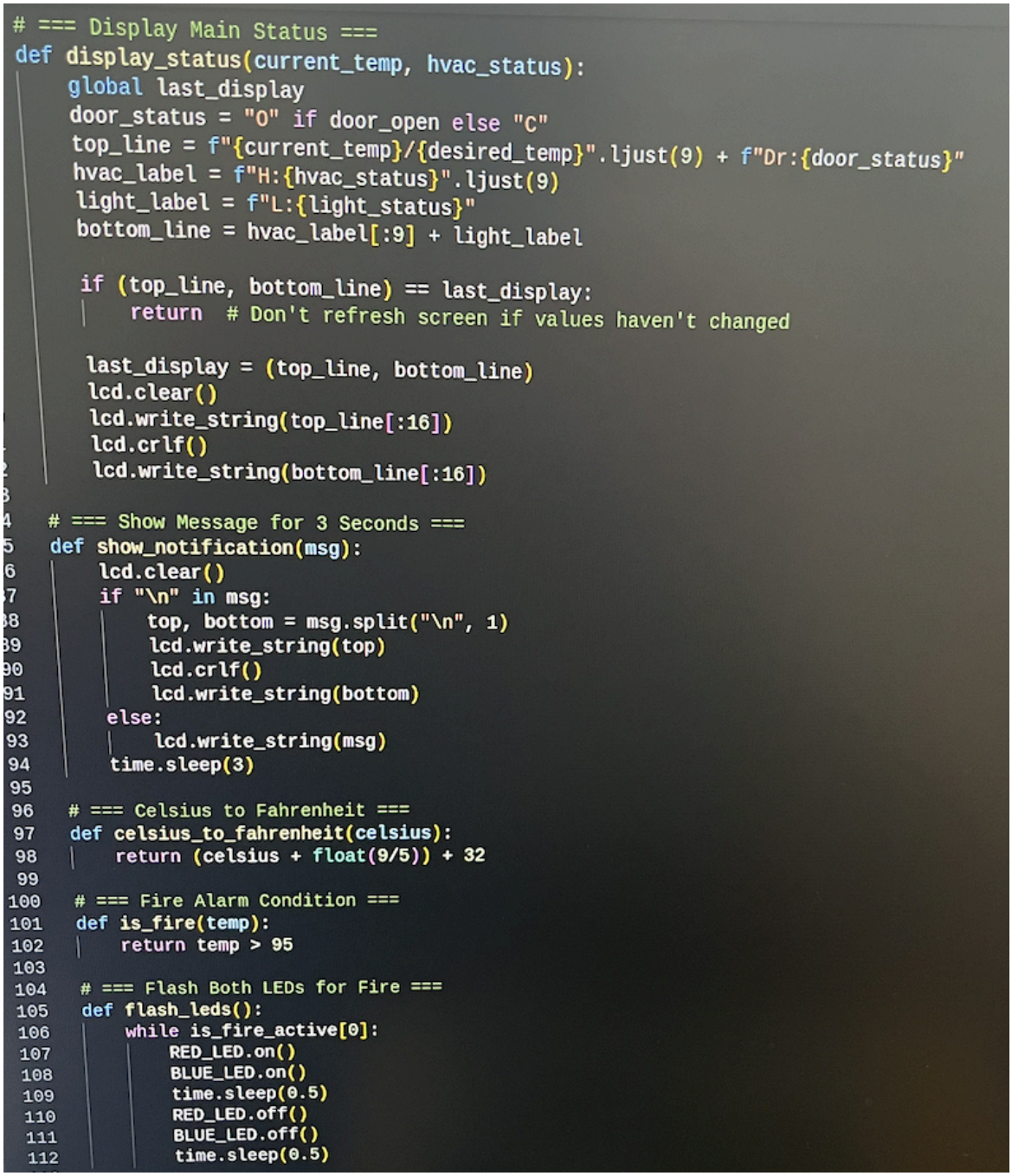
HVAC

The HVAC is implemented in a way where we get the hvac status, knowing alerts (i.e. fire) and determines what notification messages are to be output to the LCD screen as such. For incrementing and decrementing the desired temperature, I used threading to avoid parallel issues.

```
# === Temp Adjustment ===
def increment_temp():
    global desired_temp
    if desired_temp < 95:
        desired_temp += 1
        log_event(f"Desired temp increased to {desired_temp}")

def decrement_temp():
    global desired_temp
    if desired_temp > 65:
        desired_temp -= 1
        log_event(f"Desired temp decreased to {desired_temp}")

INC_BTN.when_pressed = lambda: threading.Timer(0.1, increment_temp).start()
DEC_BTN.when_pressed = lambda: threading.Timer(0.1, decrement_temp).start()

# === HVAC Update Function ===
def update_hvac(status, avg_temp):
    global last_hvac_status
    if last_hvac_status != status:
        log_event(f"HVAC {status}")
        last_hvac_status = status
        if status == "AC":
            show_notification("AC is on")
        elif status == "HEAT":
            show_notification("Heater is on")
        else:
            show_notification("HVAC is OFF")

    if status == "AC":
```

Figure 5: HVAC Threading for Incrementing/Decrementing & Updating HVAC Method

Furthermore, the HVAC logic is where as seen above, the update_hvac method sets the HVAC mode to AC, HEAT, or OFF based off of the temperature state, and then we can control the LEDs based off of the status variable toggling the blue LED for AC and red for LED for Heater.



```
# === Display Main Status ===
def display_status(current_temp, hvac_status):
    global last_display
    door_status = "O" if door_open else "C"
    top_line = f"{current_temp}/{desired_temp}".ljust(9) + f"Dr:{door_status}"
    hvac_label = f"H:{hvac_status}".ljust(9)
    light_label = f"L:{light_status}"
    bottom_line = hvac_label[:9] + light_label

    if (top_line, bottom_line) == last_display:
        return  # Don't refresh screen if values haven't changed

    last_display = (top_line, bottom_line)
    lcd.clear()
    lcd.write_string(top_line[:16])
    lcd.crlf()
    lcd.write_string(bottom_line[:16])

# === Show Message for 3 Seconds ===
def show_notification(msg):
    lcd.clear()
    if "\n" in msg:
        top, bottom = msg.split("\n", 1)
        lcd.write_string(top)
        lcd.crlf()
        lcd.write_string(bottom)
    else:
        lcd.write_string(msg)
    time.sleep(3)

# === Celsius to Fahrenheit ===
def celsius_to_fahrenheit(celsius):
    return (celsius + float(9/5)) + 32

# === Fire Alarm Condition ===
def is_fire(temp):
    return temp > 95

# === Flash Both LEDs for Fire ===
def flash_leds():
    while is_fire_active[0]:
        RED_LED.on()
        BLUE_LED.on()
        time.sleep(0.5)
        RED_LED.off()
        BLUE_LED.off()
        time.sleep(0.5)
```

Figure 6: FigureMore HVAC Logic

In the above, these are several essential functions for the HVAC and other functionalities. The display_status method will out put to the screen as the intended format as provided in the lab manual instructions. The show_notification method is for displaying the warning functions like when a door is opened/closed, for things like Heater/AC is on/off, and warning messages for the fire case scenario. For handling the celsius to fahrenheit conversion, we use the formula as seen above. Additionally, the HVAC logic maintains the last status to avoid redundant updates – that

way we can avoid unnecessary refreshes to the output of the LCD in the cases where no new values are made.

Note: when using the blow dryer, the PIR sensor detects the motion of the blow dryer and turns on the green ambient light.

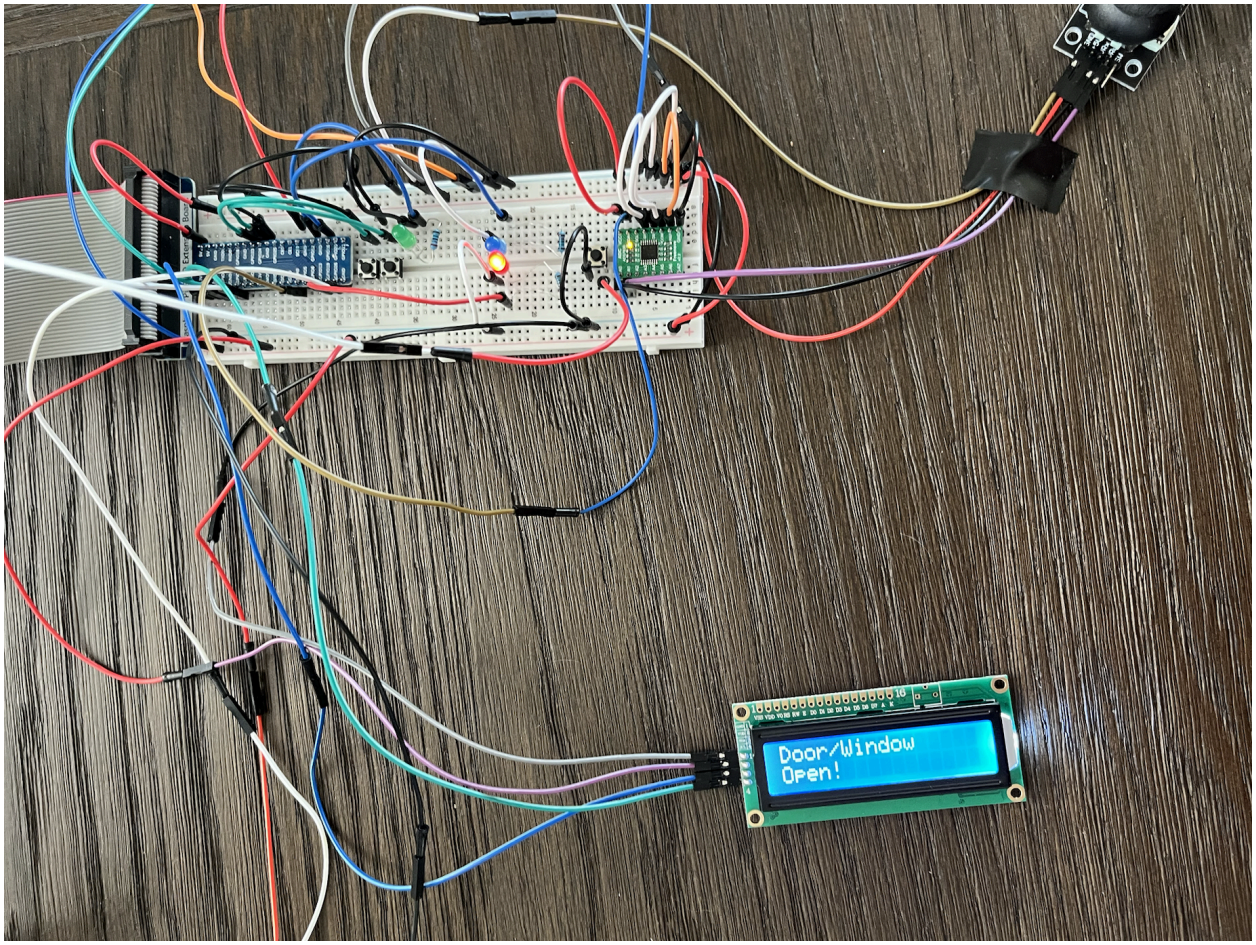


Figure 7: Button Pressed, Door Opened, Light OFF

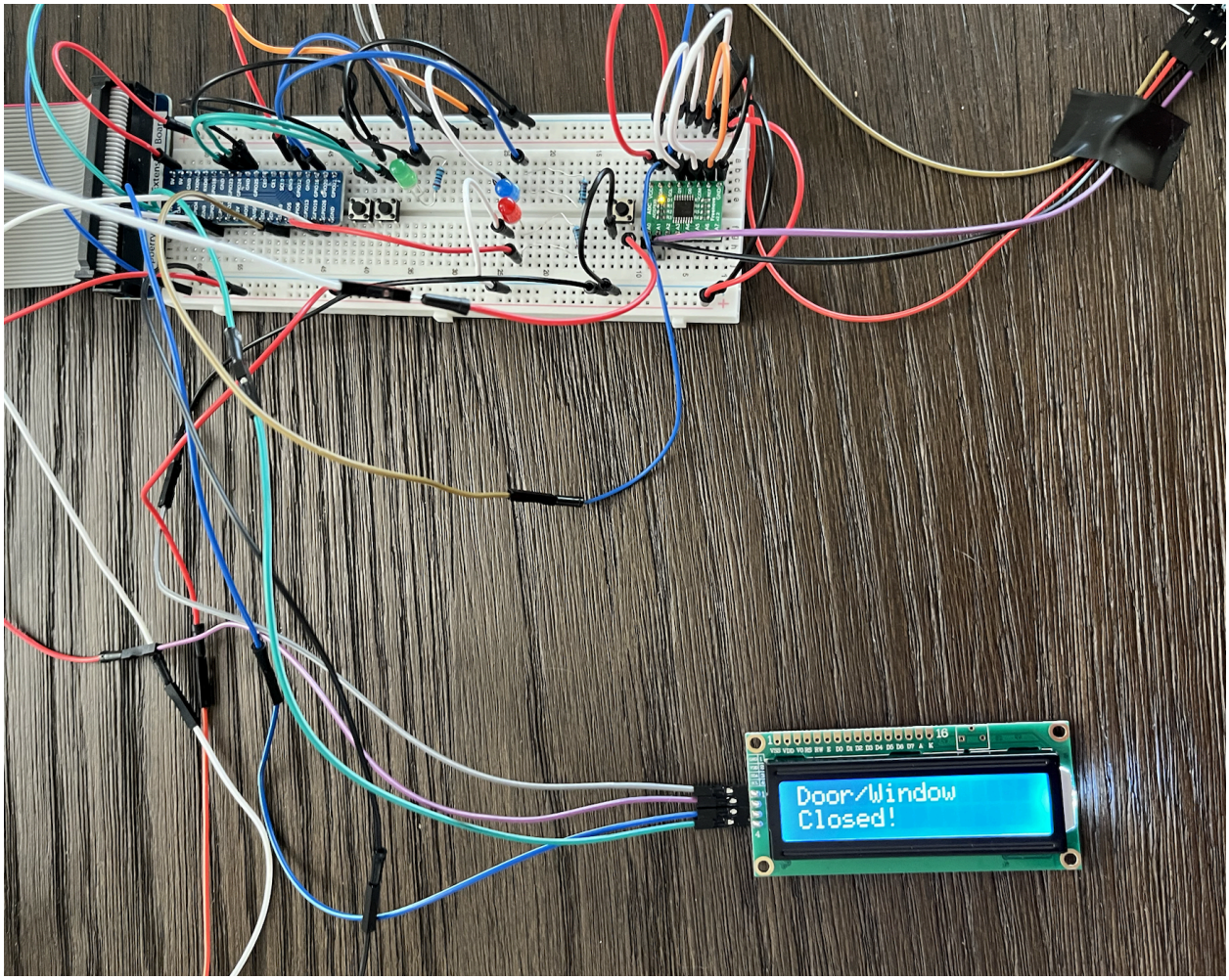


Figure 8: Button Pressed Again, Door Closed

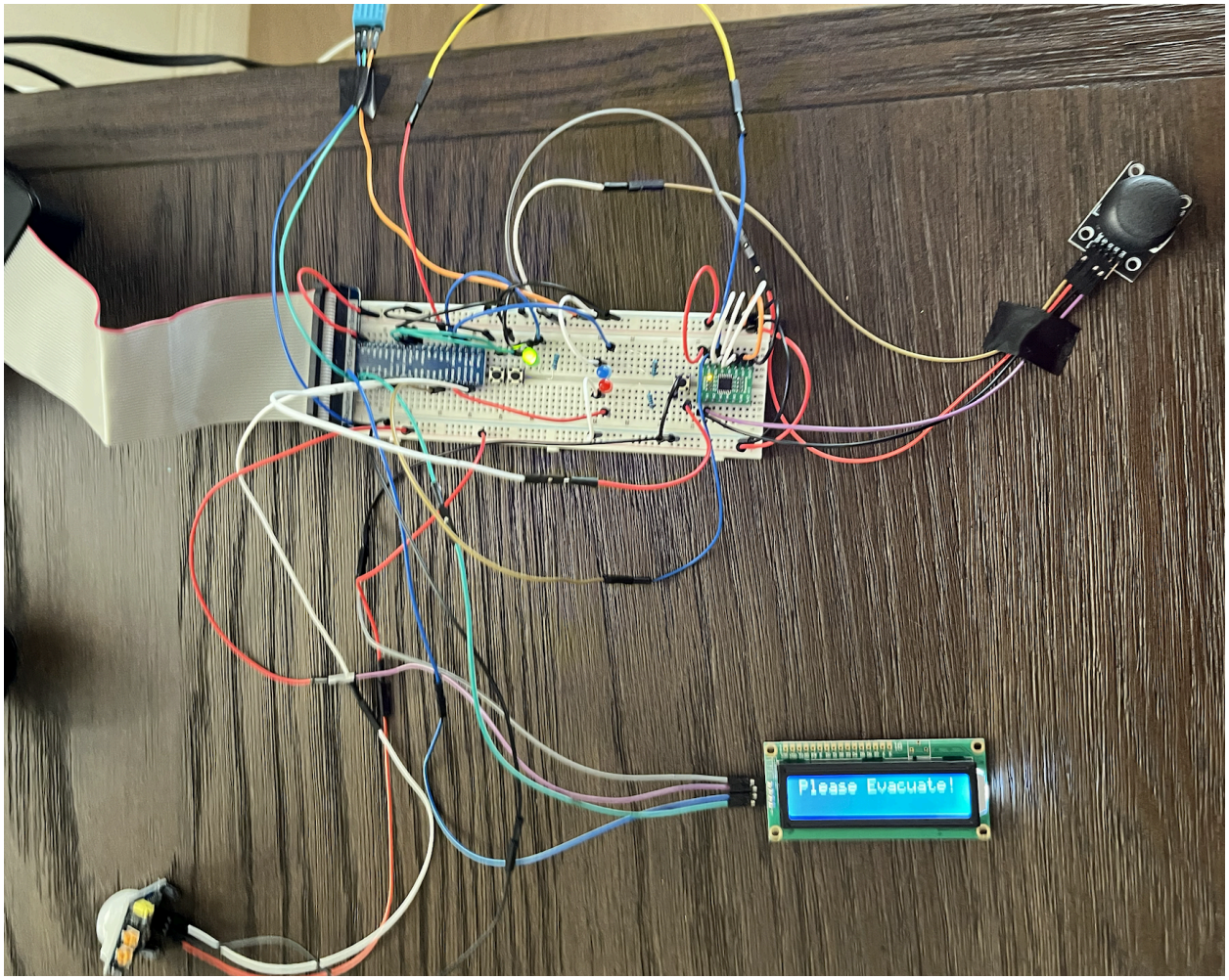


Figure 9: Fire Message (Evacuate, w/ showing 1 second Flash this one OFF)

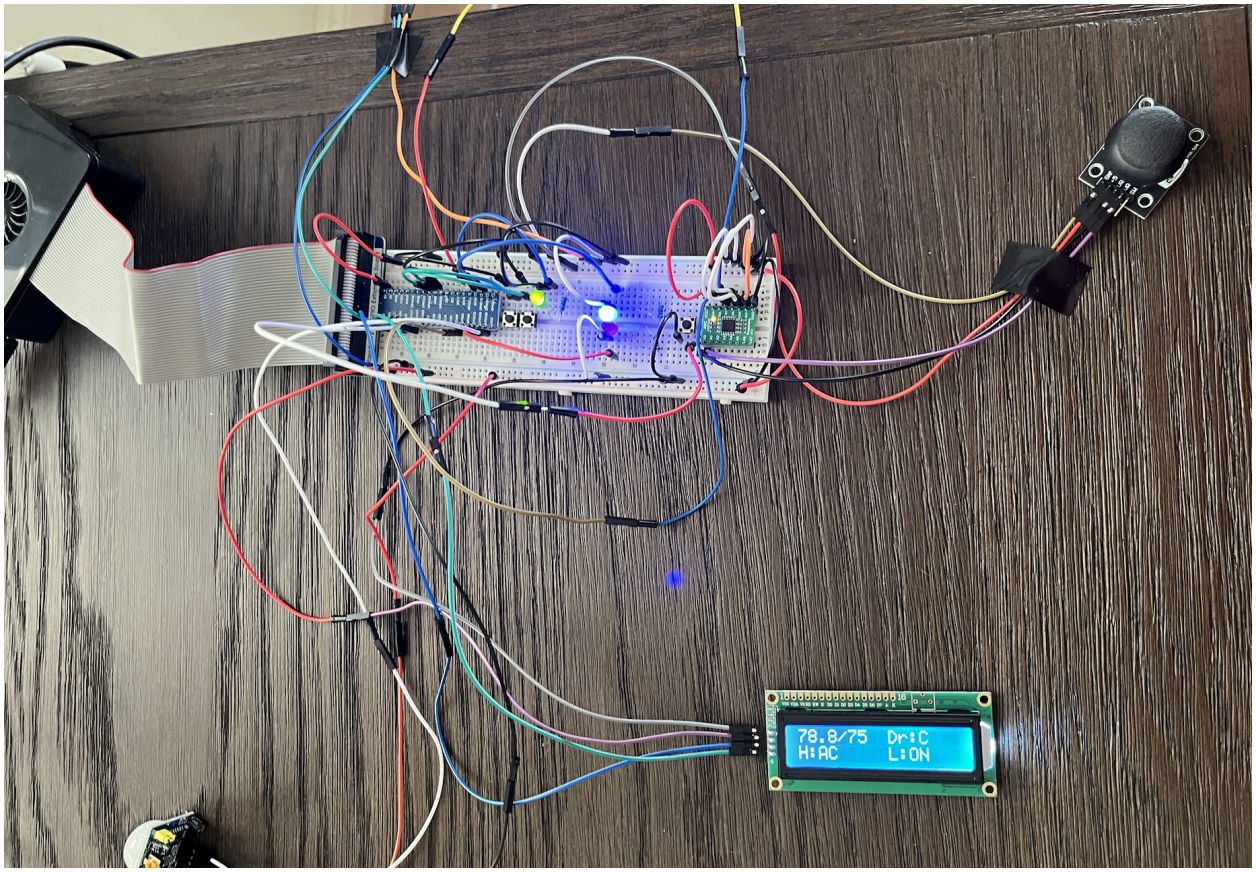


Figure 10: AC is ON

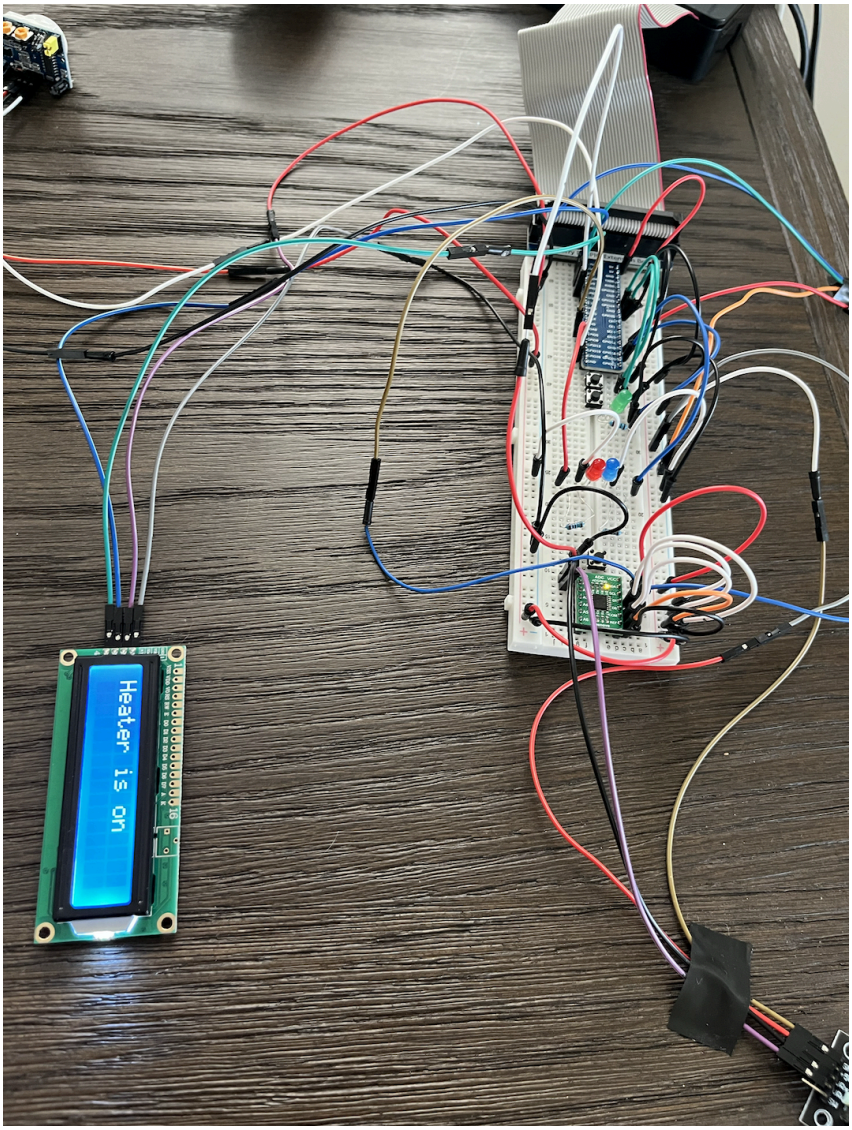


Figure 11: Heater is ON

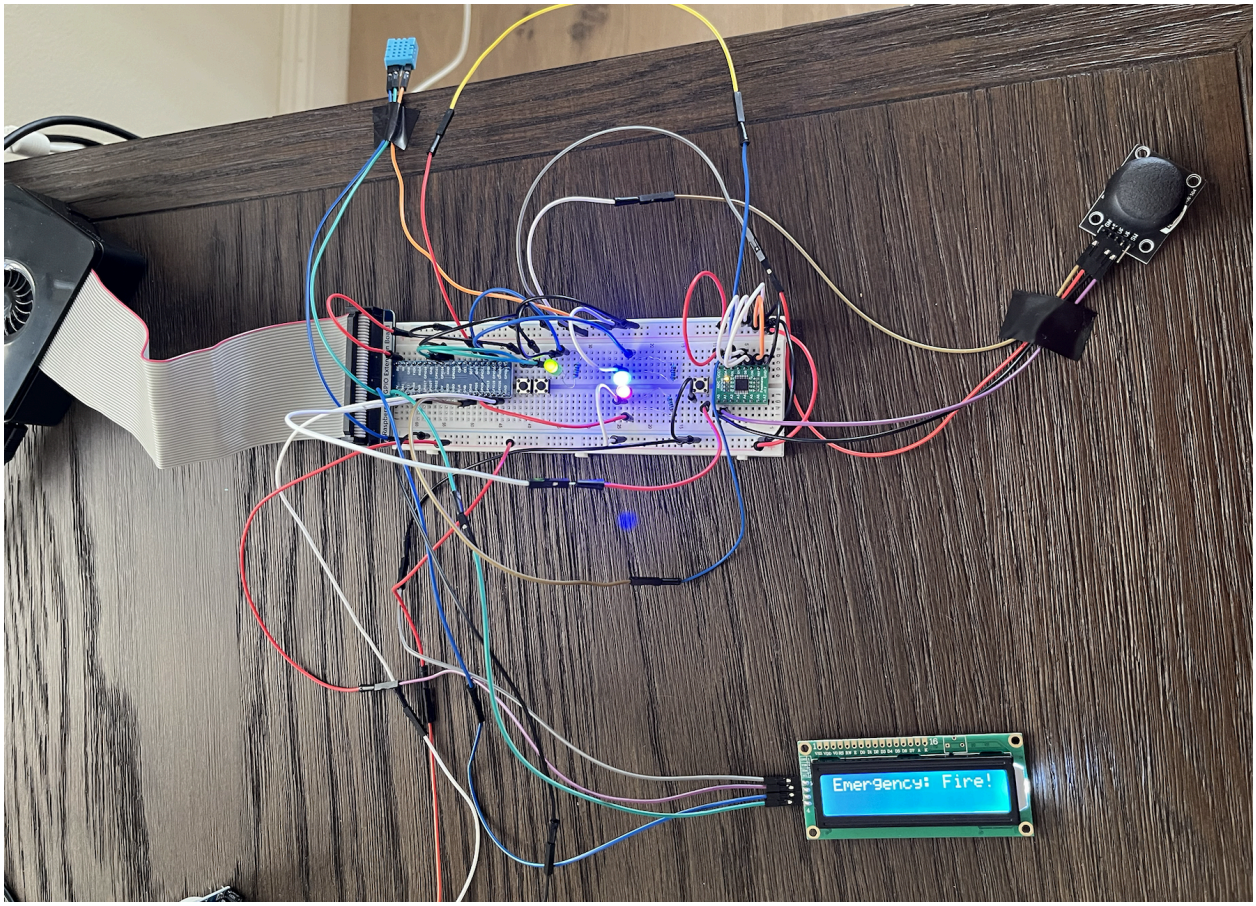


Figure 12: Emergency Flash (Fire)

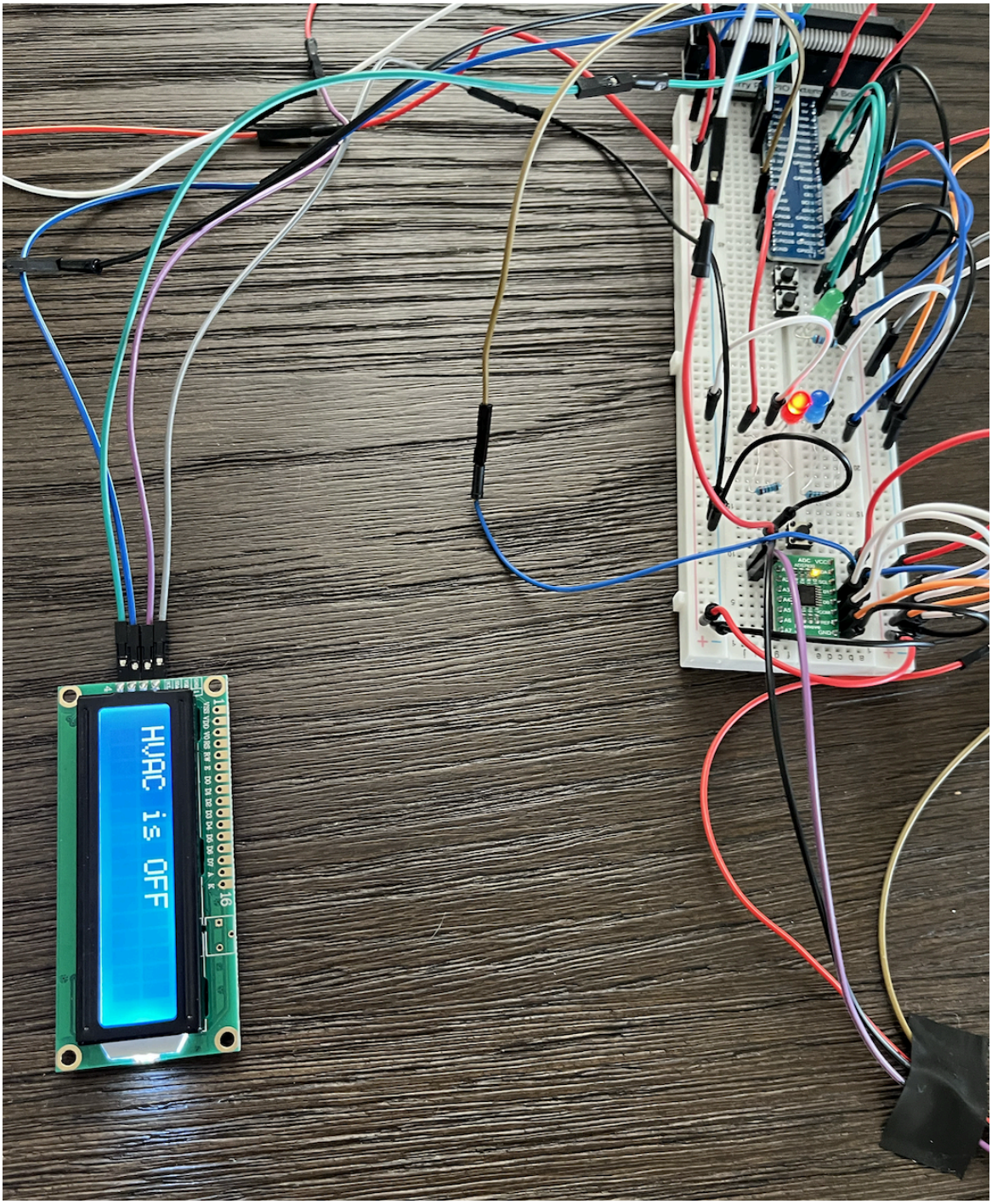
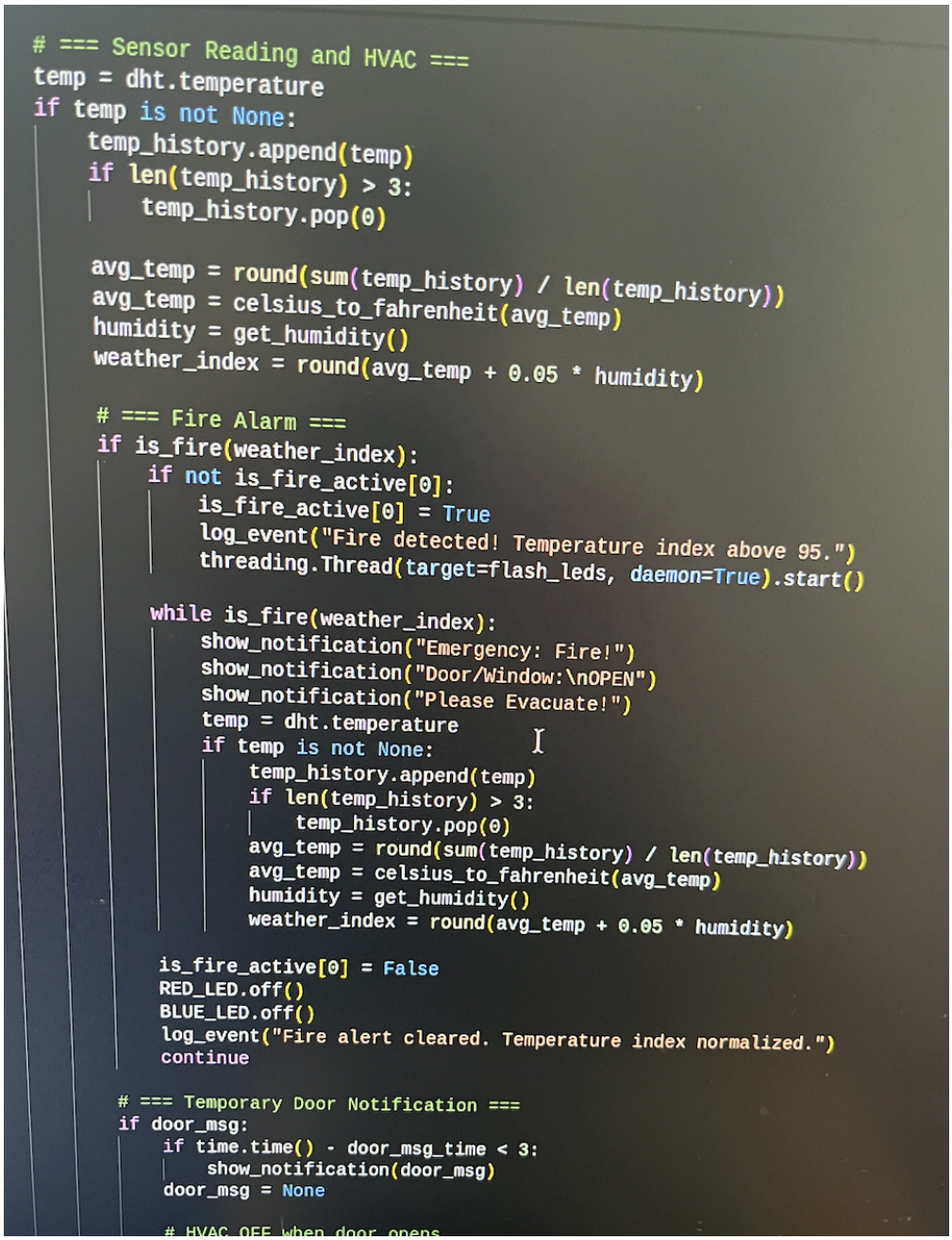


Figure 13: HVAC is OFF when open door

Fire Alarm System

As mentioned in the HVAC portion, we have an implementation of the fire alarm system. See figure 9 and figure 12 as cross reference for visual demonstration purposes, and the video. The implementation of the fire alarm system is simple, if the temperature reaches above 95 degrees, the HVAC is off, the door is open, and we give evacuation/emergency messages until the temperature is 95 degrees or below (fahrenheit).

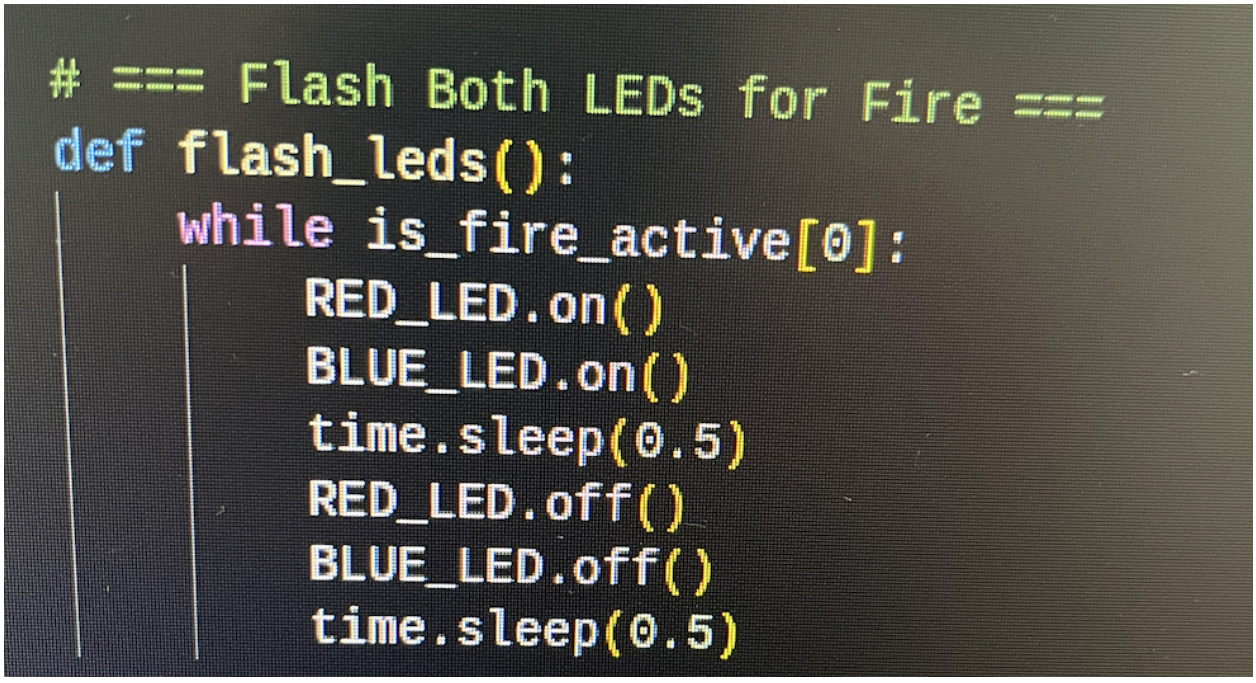


```
# === Sensor Reading and HVAC ===
temp = dht.temperature
if temp is not None:
    temp_history.append(temp)
    if len(temp_history) > 3:
        temp_history.pop(0)

    avg_temp = round(sum(temp_history) / len(temp_history))
    avg_temp = celsius_to_fahrenheit(avg_temp)
    humidity = get_humidity()
    weather_index = round(avg_temp + 0.05 * humidity)

    # === Fire Alarm ===
    if is_fire(weather_index):
        if not is_fire_active[0]:
            is_fire_active[0] = True
            log_event("Fire detected! Temperature index above 95.")
            threading.Thread(target=flash_leds, daemon=True).start()

        while is_fire(weather_index):
            show_notification("Emergency: Fire!")
            show_notification("Door/Window:\nOPEN")
            show_notification("Please Evacuate!")
            temp = dht.temperature
            if temp is not None:
                temp_history.append(temp)
                if len(temp_history) > 3:
                    temp_history.pop(0)
                avg_temp = round(sum(temp_history) / len(temp_history))
                avg_temp = celsius_to_fahrenheit(avg_temp)
                humidity = get_humidity()
                weather_index = round(avg_temp + 0.05 * humidity)

        is_fire_active[0] = False
        RED_LED.off()
        BLUE_LED.off()
        log_event("Fire alert cleared. Temperature index normalized.")
        continue

    # === Temporary Door Notification ===
    if door_msg:
        if time.time() - door_msg_time < 3:
            show_notification(door_msg)
        door_msg = None

        # HVAC OFF when door opens
```

Figure 14: Core Logic Fire Alarm

In Figure 14, we see the core logic for my fire alarm system as well as retireiving the values of the weather index which was given to us in the lab instructions. Once we retrieve the

temperature, we can go through a series of if statements to check if there is a fire, a need for AC, or heater. The fire alarm is the first check and once we detect the fire we use a while loop to keep showing that the fire is still occurring until the dropped temperature to 95 or below occurs. Figure 6 has the basic helper function to check if there is a fire “is_fire()”.



```
# === Flash Both LEDs for Fire ===
def flash_leds():
    while is_fire_active[0]:
        RED_LED.on()
        BLUE_LED.on()
        time.sleep(0.5)
        RED_LED.off()
        BLUE_LED.off()
        time.sleep(0.5)
```

Figure 15: Flashing LEDs when Fire

When the fire is active, we flash the LEDs in 1 seconds periods so we delay the period of on and off by 0.5 each (i.e. 0.5 + 0.5 = 1.0).

Security System

For the security system, a similar approach was taken to as how the pushbuttons for incrementing/decrementing the desired temperature was done where we use a thread to run the toggling to prevent gibberish or hidden overlays when the door state is open or closed. See

Figure 16.

```
# === Door Button Toggle ===
def security_callback():
    global door_open, door_msg, door_msg_time
    door_open = not door_open
    if door_open:
        door_msg = "Door/Window \nOpen!"
        log_event("DOOR OPEN")
    else:
        door_msg = "Door/Window \nClosed!"
        log_event("DOOR CLOSED")
    door_msg_time = time.time()

SECURITY_BTN.when_pressed = lambda: threading.Timer(0.1, security_callback).start()
```

Figure 16: Security Callback for Toggling

```
# === Temporary Door Notification ===
if door_msg:
    if time.time() - door_msg_time < 3:
        show_notification(door_msg)
    door_msg = None

    # HVAC OFF when door opens
    update_hvac("OFF", avg_temp)
    continue

# === Door Open: Suspend HVAC ===
if door_open:
    update_hvac("OFF", avg_temp)
    continue

# === HVAC Control ===
new_status = last_hvac_status # default to current

if weather_index >= desired_temp + 3:
    update_hvac("AC", avg_temp)
elif weather_index <= desired_temp - 3:
    update_hvac("HEAT", avg_temp)
elif door_open:
    new_status = "OFF"
else:
    if last_hvac_status not in ["OFF", "AC", "HEAT"]:
        update_hvac("OFF", avg_temp)
```

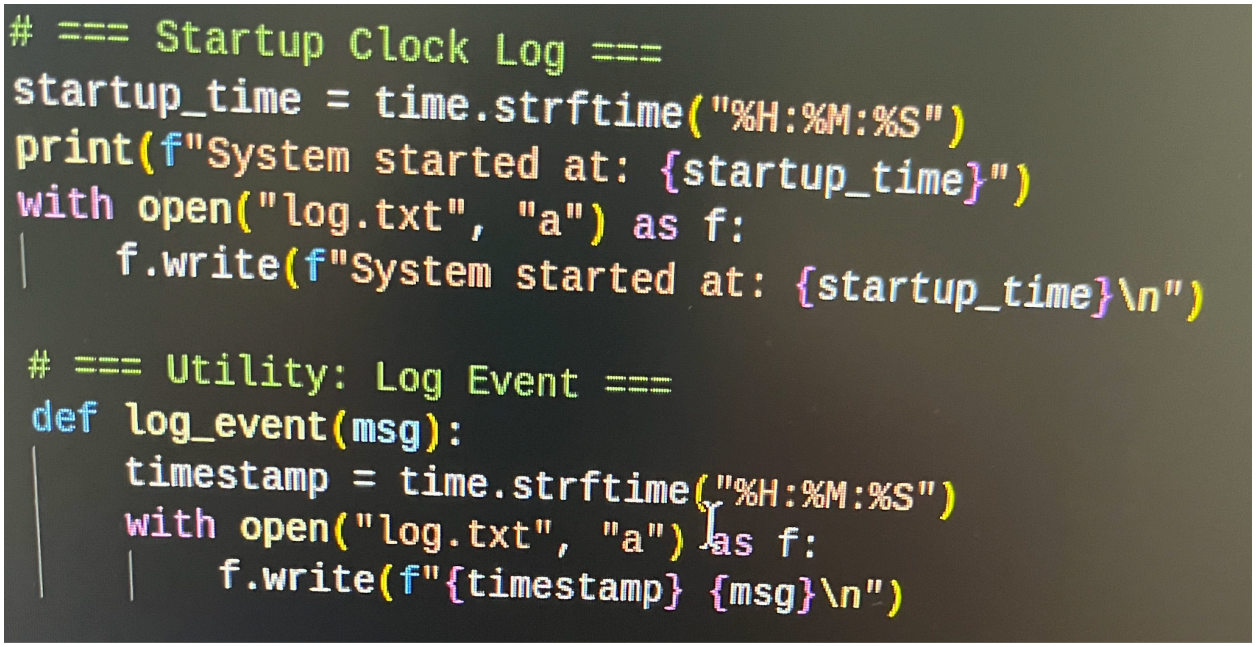
Figure 17: Door Logic within Main Loop

In Figure 17, we see that the HVAC and door message has no overlap and the order holds as intended. In other words, the notification when the door is opened it is shown for 3 seconds, then, the HVAC is off message will be shown shortly after.

Log File/Clock System/Misc.

This section is for the log file, the internal clock system, and small things to note.

For the log file we simply implement a method that passes the message we want to use. I used the examples given in the lab manual to tailor it as intended (e.g. “HVAC AC”).



```
# === Startup Clock Log ===
startup_time = time.strftime("%H:%M:%S")
print(f"System started at: {startup_time}")
with open("log.txt", "a") as f:
    f.write(f"System started at: {startup_time}\n")

# === Utility: Log Event ===
def log_event(msg):
    timestamp = time.strftime("%H:%M:%S")
    with open("log.txt", "a") as f:
        f.write(f"{timestamp} {msg}\n")
```

Figure 18: Internal Clock and Log File Appender

Each time the program is run we clear the log file using “open(“filename.txt”, “w”).close()”. Now, for the log_event I followed the format as such seen in Figure 18. For the clock log, we simply have a startup time and we can see the updates accordingly see Figure 19.

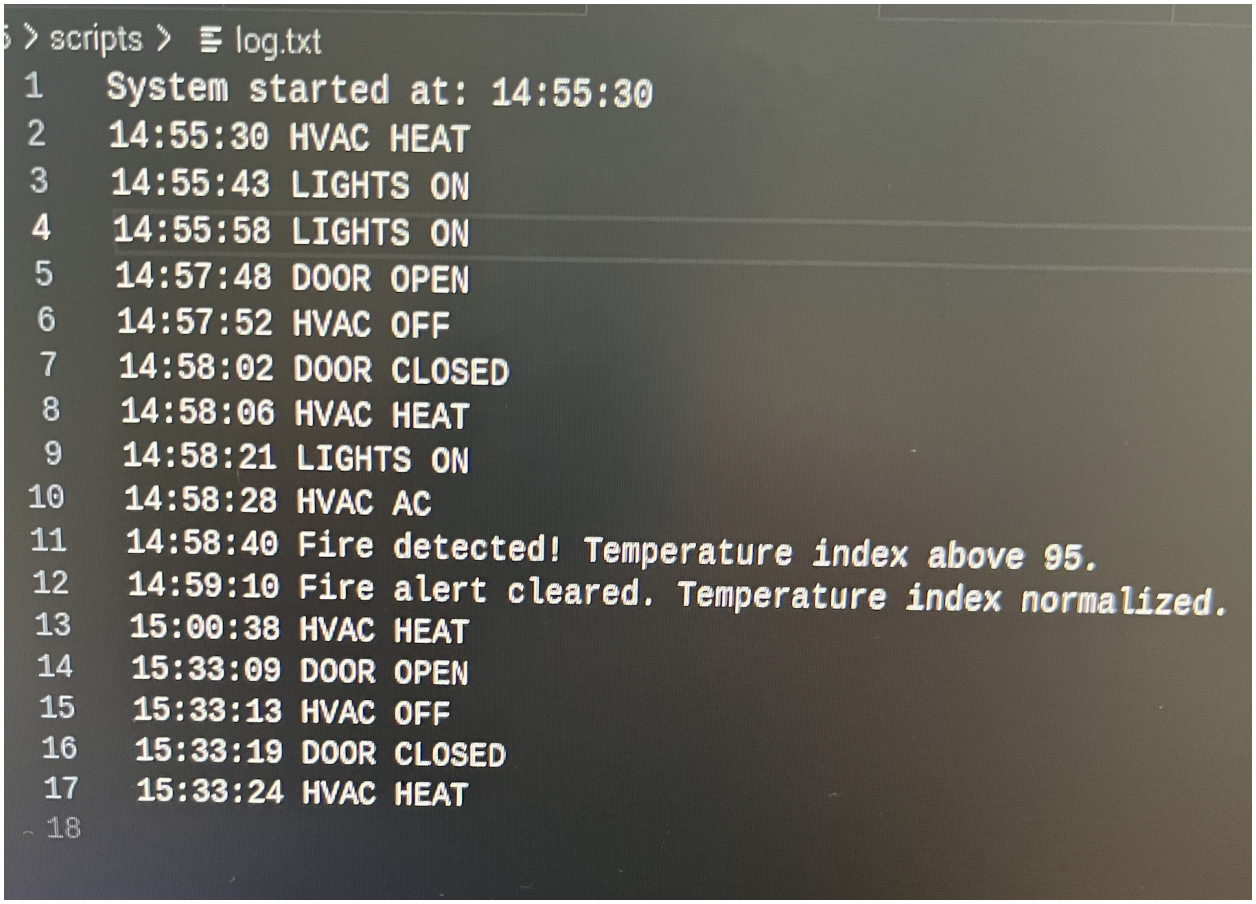
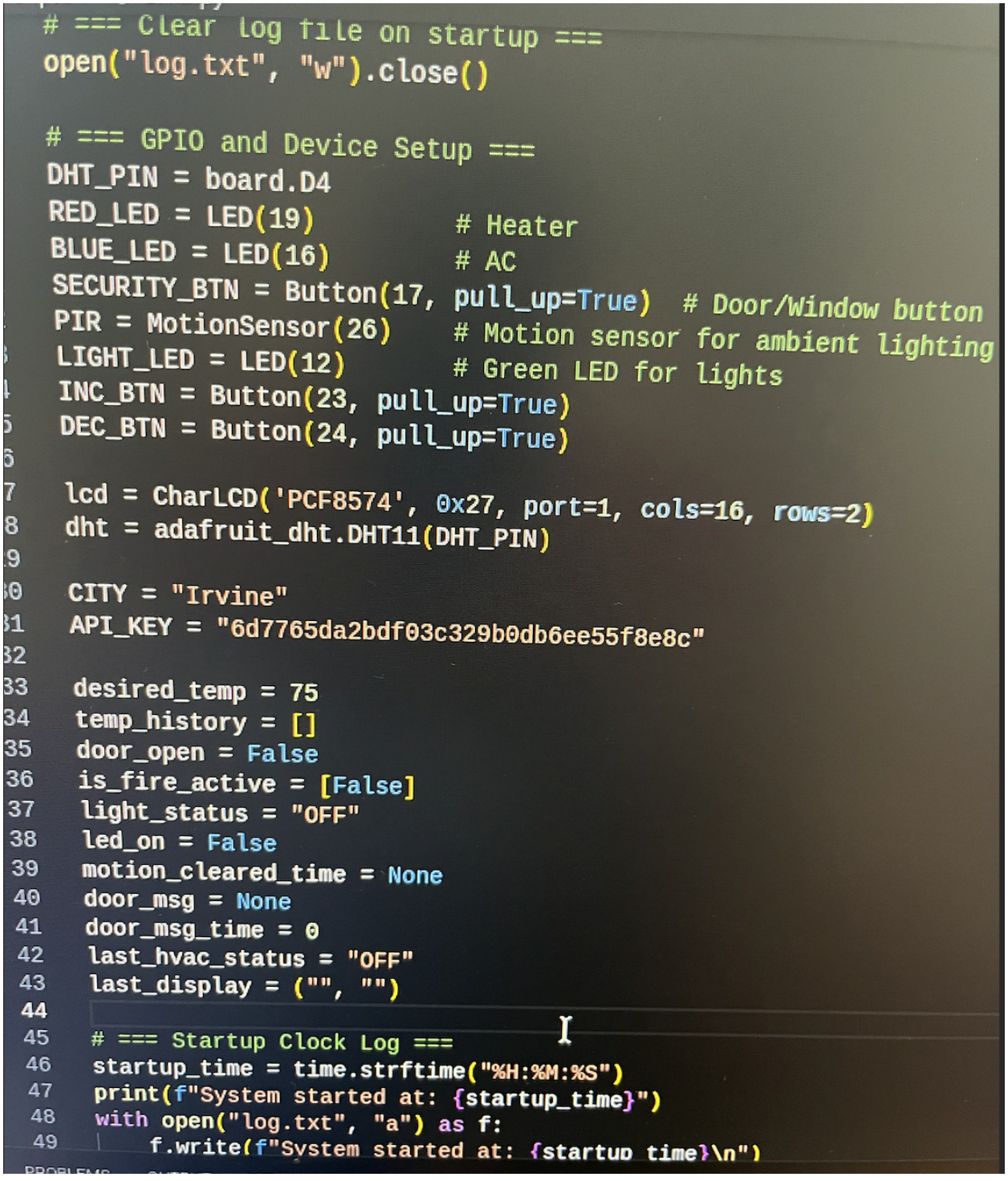


Figure 19: Log File

For the miscellaneous, I wanted to show the data types used in this program. The GPIO pins are displayed in Figure 20 as well as the states of the light status, hvac status, fire, door, etc. These booleans are important because they are used to toggle between true and false during event handling and become vital when multiple events are occurring simultaneously.



```
# === Clear log file on startup ===
open("log.txt", "w").close()

# === GPIO and Device Setup ===
DHT_PIN = board.D4
RED_LED = LED(19)          # Heater
BLUE_LED = LED(16)         # AC
SECURITY_BTN = Button(17, pull_up=True)  # Door/Window button
PIR = MotionSensor(26)     # Motion sensor for ambient lighting
LIGHT_LED = LED(12)        # Green LED for lights
INC_BTN = Button(23, pull_up=True)
DEC_BTN = Button(24, pull_up=True)

lcd = CharLCD('PCF8574', 0x27, port=1, cols=16, rows=2)
dht = adafruit_dht.DHT11(DHT_PIN)

CITY = "Irvine"
API_KEY = "6d7765da2bdf03c329b0db6ee55f8e8c"

desired_temp = 75
temp_history = []
door_open = False
is_fire_active = [False]
light_status = "OFF"
led_on = False
motion_cleared_time = None
door_msg = None
door_msg_time = 0
last_hvac_status = "OFF"
last_display = ("", "")

# === Startup Clock Log ===
startup_time = time.strftime("%H:%M:%S")
print(f"System started at: {startup_time}")
with open("log.txt", "a") as f:
    f.write(f"System started at: {startup_time}\n")
```

Figure 20: Device Setup

Bonus

For my bonus feature i implemented the joystick from the freenove starter kit. Additionally the ADC7830, an analog digital converter, was needed for reading analog values.It converter analofg signals into digital data that can be processed to a computer. Using the Vertical motion of the joystick I added the feature in where the joystick can change the temperature of the desired temperature.

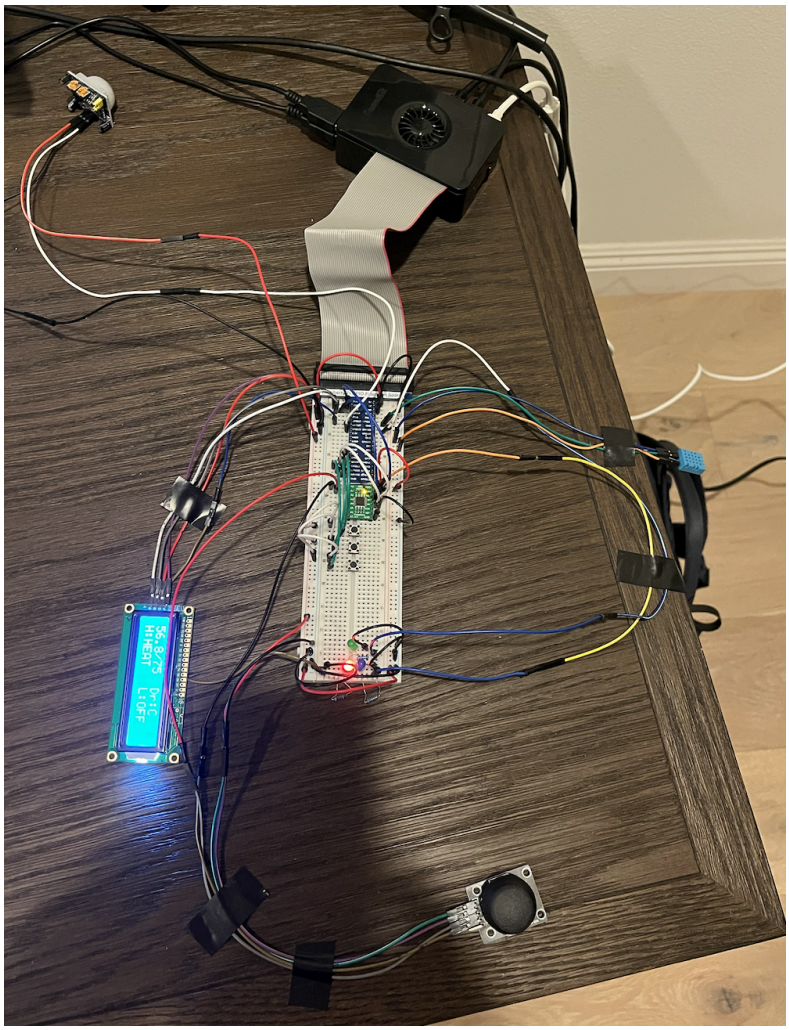
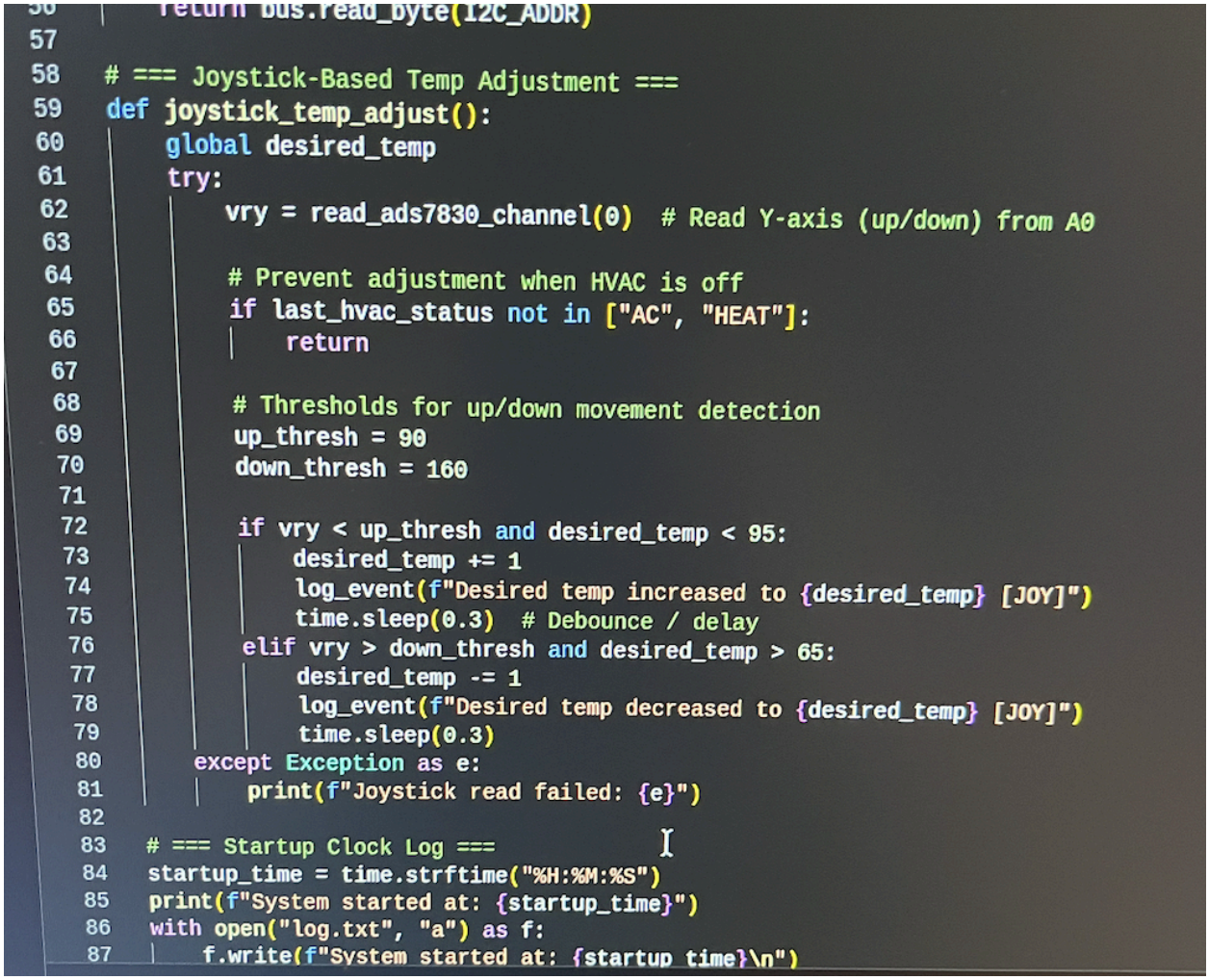


Figure 21: Bonus Feature and Revised Wiring

The green component is the ADC7830, and the joystick component is the bonus feature i have implemented.



```
    return bus.read_byte(I2C_ADDR)

# === Joystick-Based Temp Adjustment ===
def joystick_temp_adjust():
    global desired_temp
    try:
        vry = read_ads7830_channel(0)  # Read Y-axis (up/down) from A0

        # Prevent adjustment when HVAC is off
        if last_hvac_status not in ["AC", "HEAT"]:
            return

        # Thresholds for up/down movement detection
        up_thresh = 90
        down_thresh = 160

        if vry < up_thresh and desired_temp < 95:
            desired_temp += 1
            log_event(f"Desired temp increased to {desired_temp} [JOY]")
            time.sleep(0.3)  # Debounce / delay
        elif vry > down_thresh and desired_temp > 65:
            desired_temp -= 1
            log_event(f"Desired temp decreased to {desired_temp} [JOY]")
            time.sleep(0.3)
    except Exception as e:
        print(f"Joystick read failed: {e}")

# === Startup Clock Log ===
startup_time = time.strftime("%H:%M:%S")
print(f"System started at: {startup_time}")
with open("log.txt", "a") as f:
    f.write(f"System started at: {startup_time}\n")
```

Figure 22: Bonus Code Logic