PROCESSOR HW/SW INTER

EECS 113

Final Project

Kyle Zyler Cayanan

80576955

06/11/2022

For this assignment we had created a Building Management System (BMS). This system will control a building's electrical equipment such as ventilation, lighting, power systems, and security systems. *I must note that I was not able to implement the energy bill portion.

We have split the code into sections

1. Libraries
2. Pin/LCD setup
3. GPIO setup/cleanup
4. HVAC
5. Ambient Lighting System
6. Door Security System
7. CIMIS data
8. Main

Let’s examine the code by section

1 Libraries

```
#!/usr/bin/env python3
########################################################################
# Filename    : FinalProject.py
# Description : Building Management System
# author      : Kyle Zyler Cayanan
# modification: 2022/06/06
########################################################################
import json
import threading
import time
from datetime import date

import RPi.GPIO as GPIO
import requests

# LCD and DHT Libraries
# files can be found in parent directory for project
import Freenove_DHT as DHT
from Adafruit_LCD1602 import Adafruit_CharLCD
from PCF8574 import PCF8574_GPIO
```

The above screenshot contains the libraries used in order to drive the code. The json library is used to parse and format the humidity data retrieved from CIMIS. The threading library is needed to handle each function as threads. The time/datetime library was used to get time data for humidity and delay functions. Rpi.GPIO was used to handle pins and GPIO on the raspberry pi. requests library was meant for gathering cimis data.

The last three libraries are provided in the submission files as they were pulled from the Freenove Ultimate Starter Kit.

2 Pin/LCD Setup

```
DHTPin = 27  # define DHT pin
IR_ledPin = 18  # define IR led pin
IR_sensorPin = 17  # define IR sensor pin

HVAC_buttonPin = 6  # define HVAC button pin
heater_buttonPin = 13  # define heater button pin
door_buttonPin = 5  # deine door button pin

HVAC_ledPin = 12  # define HVAC led pin
heater_ledPin = 24  # define heater led pin

# threading mutex variable
lcdLock = threading.Lock()

# initial GPIO
GPIO.setmode(GPIO.BCM)  # use GPIO numbering
GPIO.setwarnings(False)

PCF8574_address = 0x27  # I2C address of the PCF8574 chip
PCF8574A_address = 0x3F  # I2C address of the PCF8574A chip

# Create PCF8574 GPIO adapter
try:
   mcp = PCF8574_GPIO(PCF8574_address)
except:
   try:
       mcp = PCF8574_GPIO(PCF8574A_address)
   except:
       print(" I2C adress Error !")
       exit(1)

# Create LCD, passing in MCP GPIO adapter
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4, 5, 6, 7], GPIO=mcp)
```

Here we can see our pin declarations and we use these declarations to follow our schematic since we have used BCM mode for the program. Very straightforward code please refer to the images below to see if it matches pin declarations above

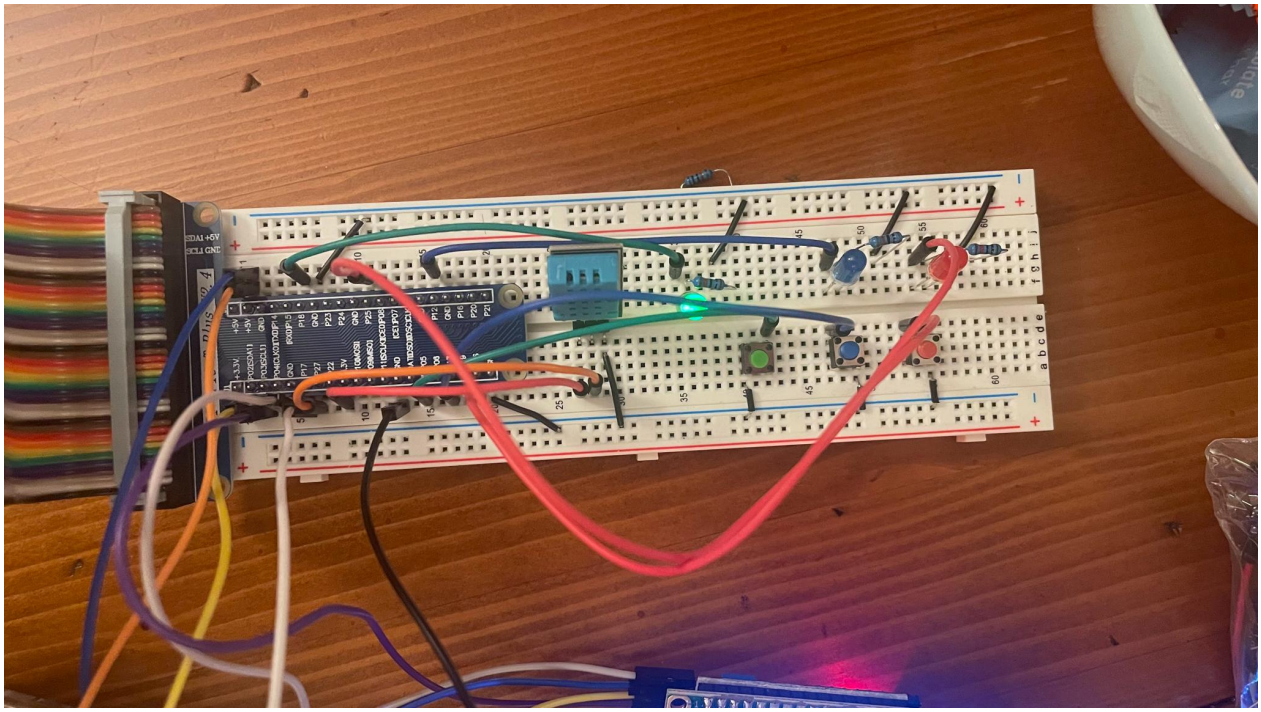

Figure 1: Breadboard Schematic

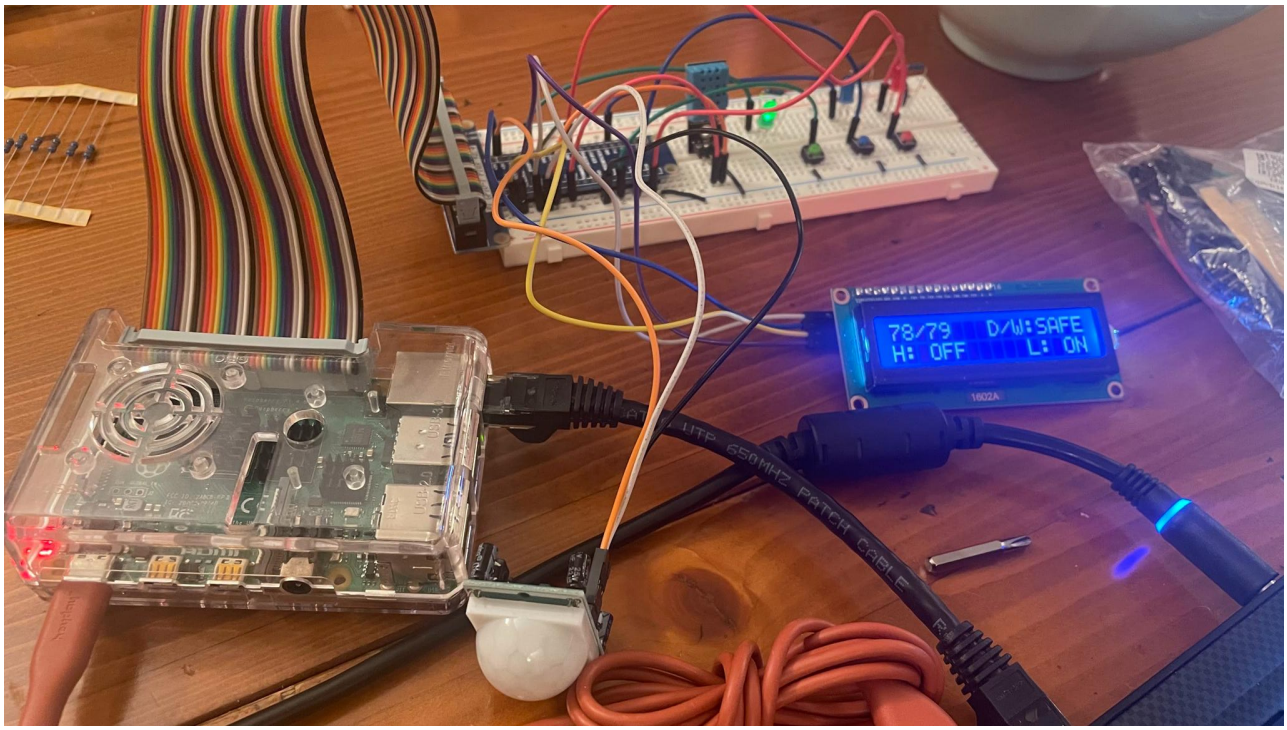

Figure 2: Breadboard Schematic with IR sensor and LCD with display

The schematic is setup with 3 buttons, 3 LEDs, a DHT-11 sensor, an IR sensor, and a 16x2 LCD

3 GPIO setup/cleanup

```
def setup():
   GPIO.setup(IR_ledPin, GPIO.OUT, initial=GPIO.LOW)  # set IR ledPin to OUTPUT
mode
   GPIO.setup(IR_sensorPin, GPIO.IN)  # set IR sensorPin to INPUT mode

   GPIO.setup(HVAC_buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # set HVAC
buttonPin to INPUT mode
   GPIO.setup(heater_buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # set heater
buttonPin to INPUT mode
   GPIO.setup(door_buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # set door
buttonPin to INPUT mode
   GPIO.setup(HVAC_ledPin, GPIO.OUT, initial=GPIO.LOW)  # set HVAC ledPin to OUTPUT
mode
   GPIO.setup(heater_ledPin, GPIO.OUT, initial=GPIO.LOW)  # set heater ledPin to
OUTPUT mode


def destroy():
   # lcd
   lcd.clear()

   # ambient lighting
   GPIO.output(IR_ledPin, False)

   # HVAC
   GPIO.output(HVAC_ledPin, False)
   GPIO.output(heater_ledPin, False)

   # for all
   GPIO.cleanup()
```

This code section is responsible for assigning our pins to their respective I/O as well as cleaning up the GPIO components such as clearing the LCD, turning off all LEDs and cleaning up the pin declarations.

In the setup we can see that the buttons for controlling temperature and the door/window are marked as inputs while the corresponding LEDs to those inputs are set as outputs.

4 HVAC

```
def hvacSystem():
   humidity = int(cimisData())
   dht = DHT.DHT(DHTPin)
   read = dht.readDHT11()
   temp = int(dht.temperature * (9 / 5) + 32)
   weather = int(temp + 0.05 * humidity)
   setTemp = weather
   lcdLock.acquire()
   lcd.setCursor(0, 0)
   lcd.message('' + str(weather) + '/' + str(setTemp))
   lcd.setCursor(0, 1)
   lcd.message('H: OFF')
   lcdLock.release()
   tempList = []
   tempList.append(temp)
   tempavg = int(temp)

   while True:
       # calculate temperature every second and avg last 3 measurements,
calculating weather index
       time.sleep(1)
       read = dht.readDHT11()
       temp = int(dht.temperature * (9 / 5) + 32)
       tempList.append(temp)
       if len(tempList) > 3:
           tempList.pop(0)
       if len(tempList) > 2:
           tempavg = int((tempList[0] + tempList[1] + tempList[2]) / 3)
       weather = int(tempavg + 0.05 * humidity)
       lcdLock.acquire()
       lcd.setCursor(0, 0)
       lcd.message('' + str(weather) + '/' + str(setTemp))
       lcdLock.release()

       if not GPIO.input(HVAC_buttonPin):
           print('Temp lowered\n')
           setTemp -= 1
           lcdLock.acquire()
           lcd.setCursor(0, 0)
           lcd.message('' + str(weather) + '/' + str(setTemp))
           lcdLock.release()
       if not GPIO.input(heater_buttonPin):
           print('Temp increased\n')
           setTemp += 1
           lcdLock.acquire()
```

```
            lcd.setCursor(0, 0)
            lcd.message('' + str(weather) + '/' + str(setTemp))
            lcdLock.release()

        if setTemp >= (weather + 3) and GPIO.input(door_buttonPin):
            GPIO.output(HVAC_ledPin, False)
            GPIO.output(heater_ledPin, True)
            lcdLock.acquire()
            lcd.clear()
            lcd.setCursor(0, 0)
            lcd.message('HEAT TURNED ON')
            time.sleep(3)
            lcd.clear()
            lcdLock.release()
            while setTemp >= (weather + 3) and GPIO.input(door_buttonPin):
                if (not GPIO.input(HVAC_buttonPin)) or (not
GPIO.input(heater_buttonPin)):
                    break
                lcdLock.acquire()
                lcd.setCursor(0, 1)
                lcd.message('H:HEAT')
                lcd.setCursor(0, 0)
                lcd.message('' + str(weather) + '/' + str(setTemp))
                lcdLock.release()
        if (setTemp < (weather + 3)) and (setTemp > (weather - 3)):
            GPIO.output(HVAC_ledPin, False)
            GPIO.output(heater_ledPin, False)
            lcdLock.acquire()
            lcd.setCursor(0, 1)
            lcd.message('H: OFF')
            lcdLock.release()
        if setTemp <= (weather - 3) and GPIO.input(HVAC_buttonPin):
            GPIO.output(HVAC_ledPin, True)
            GPIO.output(heater_ledPin, False)
            lcdLock.acquire()
            lcd.clear()
            lcd.setCursor(0, 0)
            lcd.message('AC ON')
            time.sleep(3)
            lcd.clear()
            lcdLock.release()
            while setTemp <= (weather - 3) and GPIO.input(door_buttonPin):
                if (not GPIO.input(HVAC_buttonPin)) or (not
GPIO.input(heater_buttonPin)):
                    break
                lcdLock.acquire()
                lcd.setCursor(0, 1)
```

```
                lcd.message('H:COOL')
                lcd.setCursor(0, 0)
                lcd.message('' + str(weather) + '/' + str(setTemp))
                lcdLock.release()
```

This function is responsible for manipulating our weather, temperature, and humidity data alongside controlling the entire HVAC system. We can break down the code into parts. The first section just before the first while loop will initialize our data and set up our initial temperature values based on the first read of the DHT sensor and will then display that data to the LCD.

We then enter the ‘while True’ loop which repeats this initial process. Then we poll for certain scenarios in order to display/change the data displayed on the LCD. The first scenario we are checking to see if the temperature has been lowered/increased by pressing the blue/red button from the schematic above. If the button is pressed then we decrement/increment the desired temperature value and display it on the LCD.

Next we are polling if the temperature has been increased so high or so low that would trigger the Heater/AC system. If our desired temperature is 3 degrees greater than the current weather value then we will trigger the heating system while the door/window remains closed. The AC system check works vice versa; we check to see if the desired temperature is 3 degrees less than the current weather value and then trigger the AC system to ON so long as the door/window remains closed. After each check has happened we will send a message to the LCD that indicates our HVAC mode just as before. If we detect the door is open while either system is on, we then halt the HVAC system which would turn off the heater/AC system.

5 Ambient Lighting System

```
def ambientLighting():  # Motion detector sensing for lighting
   while True:
       if GPIO.input(IR_sensorPin) == GPIO.HIGH:
           GPIO.output(IR_ledPin, GPIO.input(IR_sensorPin))
           lcdLock.acquire()
           lcd.setCursor(11, 1)
           lcd.message('L: ON')
           lcdLock.release()
           time.sleep(10)
           GPIO.output(IR_ledPin, False)
       else:
           GPIO.output(IR_ledPin, GPIO.LOW)
           lcdLock.acquire()
           lcd.setCursor(11, 1)
           lcd.message('L:OFF')
           lcdLock.release()
```

The ambient lighting system function is really simple. It is essentially a ‘while True’ loop that polls the IR sensor input, if the sensor is triggered then we will flash the green LED on the schematic for 10 seconds and then turn it off. We would then display the status of the LED to the LCD.

6 Door Security System

```
def doorSecurity():  # security system/hvac control
   while True:
       if not GPIO.input(door_buttonPin):
           GPIO.output(HVAC_ledPin, False)
           GPIO.output(heater_ledPin, False)
           lcdLock.acquire()
           lcd.clear()
           lcd.setCursor(0, 0)
           lcd.message("Door/WINDOW OPEN \n HVAC HALTED")
           time.sleep(3)
           lcd.clear()
           lcdLock.release()
           while not GPIO.input(door_buttonPin):
               GPIO.output(HVAC_ledPin, False)
               GPIO.output(heater_ledPin, False)
               lcdLock.acquire()
               lcd.setCursor(8, 0)
               lcd.message("D/W: OPEN")
               lcd.setCursor(0, 1)
               lcdLock.release()
       else:
           lcdLock.acquire()
           lcd.setCursor(8, 0)
           lcd.message("D/W:SAFE")
           lcdLock.release()
```

Our door security system function simply opens/closes the door. If the green button is pressed on the circuit we will open the door and display the message “Door/WINDOW OPEN, HVAC HALTED” onto the LCD. While the button is pressed we show the status of the door/window as open on the LCD.

7 CIMIS Data

```
def cimisData():
    # Get most recent humidity value from CIMIS
    today = date.today()
    dateStamp = today.strftime("%Y-%m-%d")

    parameters = {
        "appKey": "fa3d0d05-8375-49a2-b53a-f764ec9817ce",
        "targets": "92602",
        "startDate": dateStamp,
        "endDate": dateStamp,
        "unitOfMeasure": "M",
        "dataItems": "hly-rel-hum"
    }
    url = "https://et.water.ca.gov/api/data"
    response = requests.get(url, params=parameters)

    #  parsing the data from the json file and assigning corresponding data
    jsonHumidityLoaded =
json.loads(response.content)['Data']['Providers'][0]['Records']

    for x in range(len(jsonHumidityLoaded)):
        if jsonHumidityLoaded[len(jsonHumidityLoaded) - x - 1]['HlyRelHum']['Value']
is not None:
            humidity = jsonHumidityLoaded[len(jsonHumidityLoaded) - x -
1]['HlyRelHum']['Value']
            break
        else:
            continue
        break

    return humidity
```

We gather the CIMIS data using their API. First we collect the data based on our generated app key that is provided with registering for a CIMIS account. We request the data from the URL with the given parameters { appKey, targets, start date, end date, unit of measure, data items }. We then take our data and retrieve the latest humidity value. This function will return the humidity value to our hvacSystem() thread.

8 Main

```
if __name__ == '__main__':
    print('Welcome to the Building Management System! ')
    # GPIO Setup
    setup()
    mcp.output(3, 1) # enable backlight on the LCD
    lcd.begin(16, 2) # set columns/rows on the LCD

    hvacThread = threading.Thread(target=hvacSystem)
    # hvacThread.daemon = True
    ambientThread = threading.Thread(target=ambientLighting)
    # ambientThread.daemon = True
    securityThread = threading.Thread(target=doorSecurity)
    # securityThread.daemon = True

    try:
        ambientThread.start()
        hvacThread.start()
        securityThread.start()

    except KeyboardInterrupt:
        destroy()
```

The main function will simply set up our LCD, create our threads and then run them. If we run into a keyboard interrupt (Ctrl + C) we cleanup the GPIO upon exit.

*If any explanation is unclear it will be demonstrated in the video submission.