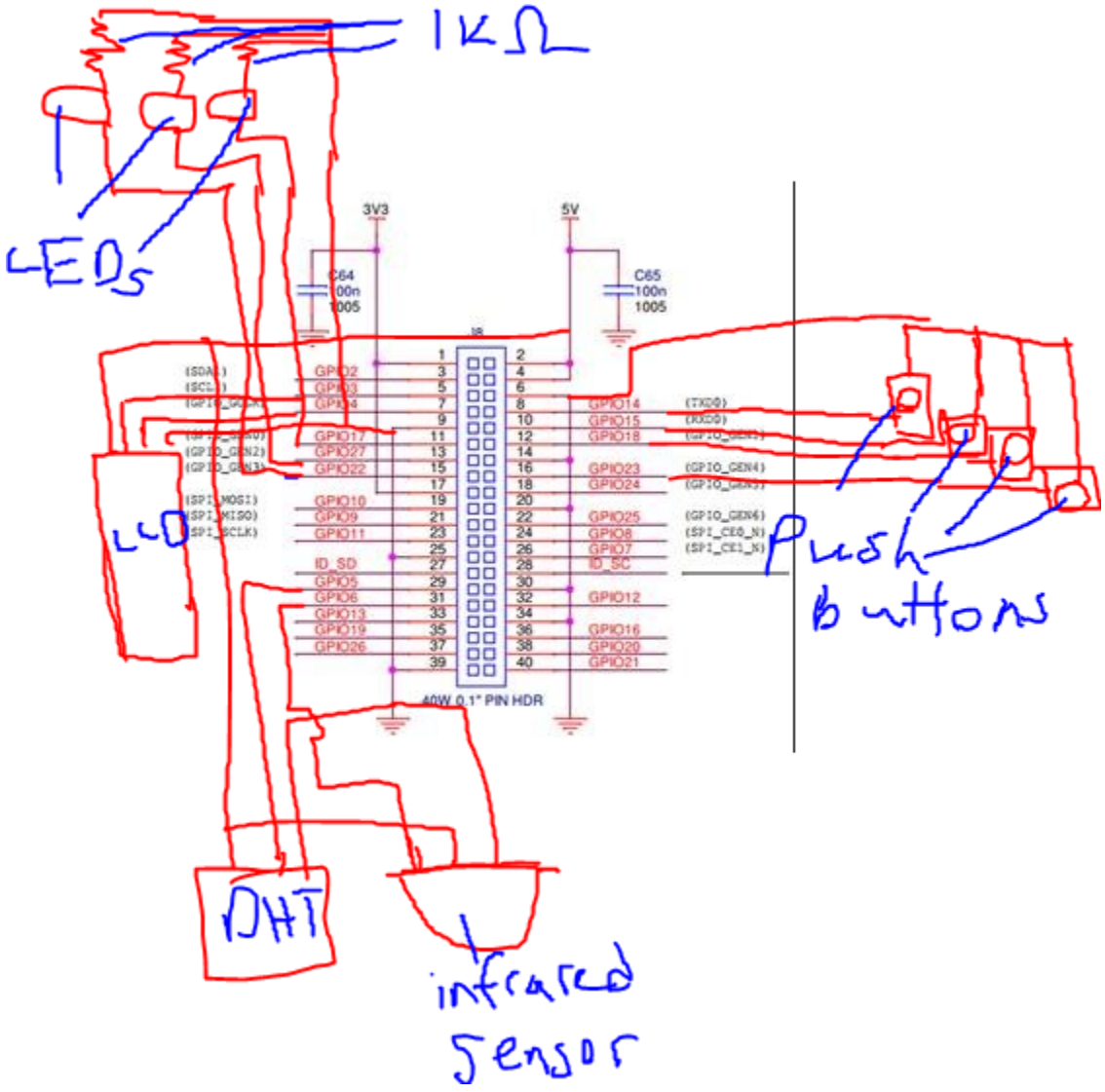
Final Project

EECS 113
University of California, Irvine
Miles Drake
Student ID 18154111
06/08/2021

Schematic:



LEDs:

- These components were very easy to wire, just an LED on a breadboard with a 1k Ohm resistor in between it and ground. By putting the power to the GPIOs, the LEDs are turned on and off by putting the respective pins into output mode and turning the value to HIGH and LOW. This is done in the code by calling the GPIO library, as shown below.
- `gpio.output(greenLED, gpio.LOW)`

Buttons:

- The wiring of these buttons are identical to that of the LEDs, only there are no resistors needed for them to function. On the coding side, the GPIOs had to be changed to input

mode, and the buttons basically are only implemented in the code with interrupts, which call various functions.

```
#event detection for buttons and infrared sensor
gpio.add_event_detect(infraredSensor, gpio.RISING, callback=handle, bouncetime=10000)
gpio.add_event_detect(tempHigher, gpio.FALLING, callback=TempRaise, bouncetime=200)
gpio.add_event_detect(tempLower, gpio.FALLING, callback=TempLower, bouncetime=200)
gpio.add_event_detect(door, gpio.FALLING, callback=DoorwHandler, bouncetime=200)
gpio.add_event_detect(window, gpio.FALLING, callback=DoorwHandler, bouncetime=200)
```

- Above we see the gpio event detect, which calls interrupts whenever any of these buttons are pressed and a falling edge is detected.
- As for the functions, the window and door buttons, for example, call the DoorwHandler() function, which changes the state of doors to be on or off depending on its current state (the buttons act as a toggle, which turn a door state or window state to be on/off.

Infrared Sensor:

- This component is similar to the button, but instead is connected to both power and ground in addition to a GPIO pin. As an input, the sensor sends a 1 or zero. The rising edge represents motion being detected, so the rising edge is used in an interrupt statement to call a handler, which is shown above in the event block.
- Since the light must be on for ten seconds before turning off, it would be very inefficient to have to wait this whole duration before allowing other inputs. Therefore, threading is used to keep the main function looping while the time is measured out in the AmbientLightCon() function.

```
def handle(event = None):  #handler for lights, which is the only functinos that will use threading as 10 seconds is a long time
    sleep(1)
    global lightMsg        #flag to turn on light message in the main loop
    lightMsg = 1
    t = threading.Thread(target=AmbientLightCon)
    t.daemon = True
    t.start()
    sleep(3)
    return

def AmbientLightCon(): #threaded function to turn on lights for 10 seconds
    global globalData
    gpio.output(greenLED, gpio.HIGH)
    globalData.changeLights(1)
    sleep(10)
    gpio.output(greenLED, gpio.LOW)
    globalData.changeLights(0)
    return

def TempRaise(event = None):
```

- It is also important to note that the LCD is not changed during these functions that are called through interrupts, and instead the LCD is only updated in main. This is because if the LCD is changed at the same time in an interrupt and main simultaneously or in quick succession, undefined behavior can occur on the LCD, rendering it useless. Therefore, these functions that are caused by interrupts trigger flags that are checked in the main loop, where each message can occur sequentially to avoid this problem. Here, lightMsg in the handle function represents the flag for this function.

DHT Sensor:

- The DHT sensor is wired nearly identically to the infrared sensor in the way that it is connected to ground, power, and a GPIO pin. This DHT is read from once per loop, where it feeds a humidity reading (unused) and a temperature reading to the variables in the main loop.
- ```
  humidity, temp = Adafruit_DHT.read(Adafruit_DHT.DHT11, DHT)   #read vals from dht11
  ```
- ```
  if temp == None:
      temp = temp1                #if temp reads null, use last recorded value
  temp3 = temp2
  temp2 = temp1
  temp1 = temp
  temp = int((temp3+temp2+temp1)/3.0)                #average last 3 measurements
  temp = temp*(9.0/5.0) + 32 + 0.05*float(cimisHum)
  globalData.changeTemp(temp)
  ```
- Above we can see how the last three measurements are recorded: three extra variables are created, which default at 26 degrees celsius. Each variable takes its next variable in line's value and stores it as its own as each new measurement comes in, after which the three values are averaged and converted to fahrenheit. It is to be noted that the DHT11 does not always output a value, so it is necessary to check if the current reading is null, such that the most recently used value can be copied and the average value does not drop randomly. After this, the temp is added to a fraction of the humidity, as the formula follows.

LCD:

- This build uses an I2C LCD, which has four pins, two of which connect to specific GPIOs on the pi, and the other two connect to power and ground. This LCD used a library which just printed to the pi using a string and a line specifier parameter, and a clear function which cleared the entire LCD, both of which are shown below.
- ```
  display.lcd_clear()
  display.lcd_display_string("   HVAC AC", 1)
  ```

Code Structure:

- The code structure consists of three main parts: A global structure, interrupts, and the main loop. The global structure is basically a global object of a class made in python to hold variables regarding the status of the system. It contains a temperature variable, a desired temperature variable, a variable which represents if a door or window is open or not, a variable that represents if heating, cooling, or nothing is on for HVAC, and a variable which represents if lights are turned on or off. The class comes with a set of functions for changing these variables easily. Below is the class itself in addition to the global object that is used throughout the code:
  ```

```
class GlobalData():       #global data basically acts like a struct, has functions to change these values
    def __init__(self, temp, dtemp, doorw, heat, lights):
        self.temp = temp
        self.dtemp = dtemp
        self.doorw = doorw
        self.heat = heat
        self.lights = lights

    def changeTemp(self, new):
        self.temp = new
    def changeDtemp(self, new):
        self.dtemp = new
    def changeDoorw(self, new):
        self.doorw = new
    def changeHeat(self, new):
        self.heat = new
    def changeLights(self, new):
        self.lights = new

globalData = GlobalData(72, 72, 0, 1, 0) #instantiate global struct first so functions can use them
```

- The interrupts run when the buttons or sensors are activated, changing global variables in addition to flags that are detected in the main function.

```
def handle(event = None):  #handler for lights, which is the only functinos that will use threading as 10 seconds is a long time
    sleep(1)
    global lightMsg        #flag to turn on light message in the main loop
    lightMsg = 1
    t = threading.Thread(target=AmbientLightCon)
    t.daemon = True
    t.start()
    sleep(3)
    return

def AmbientLightCon(): #threaded function to turn on lights for 10 seconds
    global globalData
    gpio.output(greenLED, gpio.HIGH)
    globalData.changeLights(1)
    sleep(10)
    gpio.output(greenLED, gpio.LOW)
    globalData.changeLights(0)
    return

def TempRaise(event = None):
    global globalData
    #print(globalData.dtemp)
    if (globalData.dtemp < 85):  #raise temp with upper limit 85
        globalData.changeDtemp(globalData.dtemp + 1)
    return

def TempLower(event = None):
    global globalData
    if (globalData.dtemp > 65):  #lower desired temp with lower limit 65
        globalData.changeDtemp(globalData.dtemp - 1)
    return

def DoorwHandler(event = None):
    global doorToggle
    global globalData
    global last
    global doorwMsg
    if doorToggle == 0:        #check if door is already open
        doorToggle = 1
        doorwMsg = 1
        globalData.changeHeat(0)     #update doorw and heat values in global struct
        globalData.changeDoorw(1)
        gpio.output(redLED, gpio.LOW)  #update leds
        gpio.output(blueLED, gpio.LOW)
    else:
        globalData.changeDoorw(0)
        doorToggle = 0
        last = 0
```

- As mentioned previously, no function modifies the value on the LCD directly, as this would cause lots of issues when multiple writes happen at the same time.
- The main loop is the bulk of the code, which pulls the humidity from the CIMIS website from location 211, using requests.get(), takes the data from the DHT sensor and averages the last three measurements, checks if any flags have been triggered from interrupts, and if so displays the respective warnings on the LCD, and also deals with the HVAC system. When the temperature readings are detected to have a difference greater than three degrees to the desired temperature, HVAC messages turn on in addition to the LEDs. Below is the code which checks which mode the HVAC is in, if the message for switching in between modes has already been displayed or not, and updates the respective values in the global structure when necessary. Diff represents the difference between measured temperature and desired temperature.

```
if abs(diff) > 3:
    if (diff < 0 and last != 2): #if difference is negative, turn heat, if not, turr
        last = 2
        globalData.changeHeat(2)
        display.lcd_clear()
        display.lcd_display_string("   HVAC HEAT", 1)
        gpio.output(redLED, gpio.HIGH)
        gpio.output(blueLED, gpio.LOW)
        sleep(3)
    elif(diff > 0 and last != 1):
        last = 1
        globalData.changeHeat(1)
        gpio.output(blueLED, gpio.HIGH)
        gpio.output(redLED, gpio.LOW)
        display.lcd_clear()
        display.lcd_display_string("   HVAC AC", 1)
        sleep(3)
else:               # if difference is less than 3, shut off all HVAC
    last = 0
    globalData.changeHeat(0)
    gpio.output(redLED, gpio.LOW)
    gpio.output(blueLED, gpio.LOW)
```

- The last and final function of the main loop is to update the values every iteration based on the current values of the global structure. This is conveniently done using the below function:

```
def displayData():       #display the data on main screen using data from global struct
    global display
    global globalData

    if globalData.doorw == 0:
        formatString = "{0:0.0f}/{1:0.0f}    D:SAFE".format(globalData.temp, globalData.dtemp)
    else:
        formatString = "{0:0.0f}/{1:0.0f}    D:OPEN".format(globalData.temp, globalData.dtemp)
    if globalData.heat == 1:
        heatString = "A/C "
    elif globalData.heat == 0:
        heatString = "OFF "
    else:
        heatString = "HEAT"
    if globalData.lights == 0:
        lightsString = "OFF"
    else:
        lightsString = "ON"
    format2String = "H:{}   L:{}".format(heatString, lightsString)

    display.lcd_display_string(formatString, 1)
    display.lcd_display_string(format2String, 2)
    return
```

- The humidity is pulled from the CIMIS like so:

```
response = requests.get('http://et.water.ca.gov/api/data?appKey=6cc8b450-3c2d-48d9-968d-180f2a278617&targets=211&startDate=2021-06-08&endDate=2021-06-08&dataItems=hly-rel-hu
cimisHum = response['Data']['Providers'][-1]['Records'][0]['HlyRelHum']['Value']
#get humidity value from location 211 (closest to my house)
```

- Response basically receives a nested library with lists and libraries inside, which is navigated to get the relative humidity value from the site.