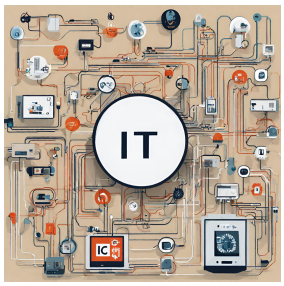


PASCHAL CHUKWUBUIKEM IFEWULU

(R0961282)

Individual project (IOT Essentials)

End Goals:

1. Show temperature and lux values on terminal 2 points

2. Show temperature and lux values on cloud dashboard 4 points

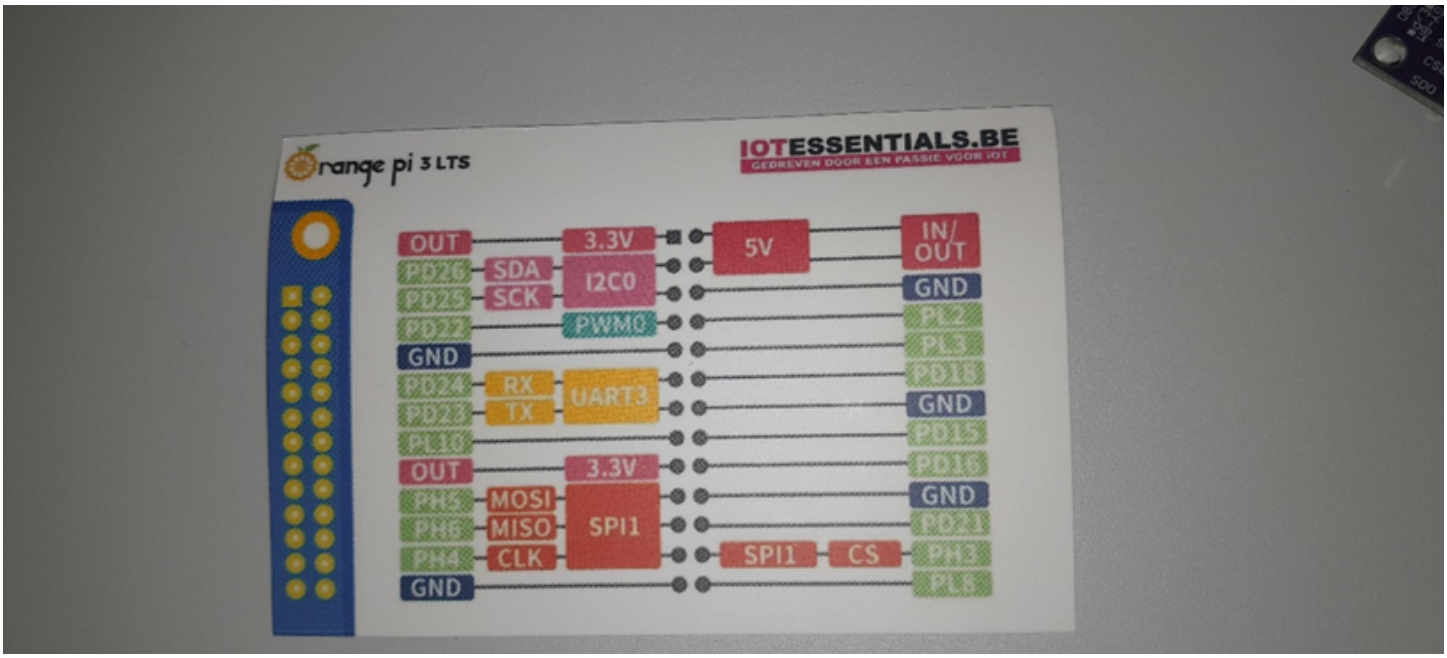
3. Set goal for temperature and lux and show on dashboard 2 points

4. Control light intensity 4 points

5. Control temperature 4 points

6. Automatically adjust light and temperature 4 points

You use this image below as a guide for the connection.



Youtube link

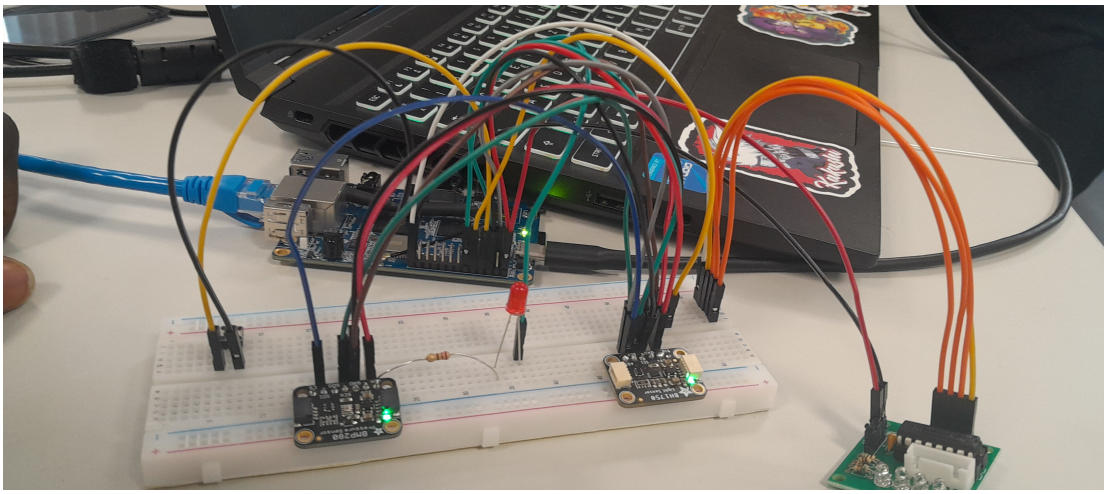
https://youtu.be/B0nHsEzsgas

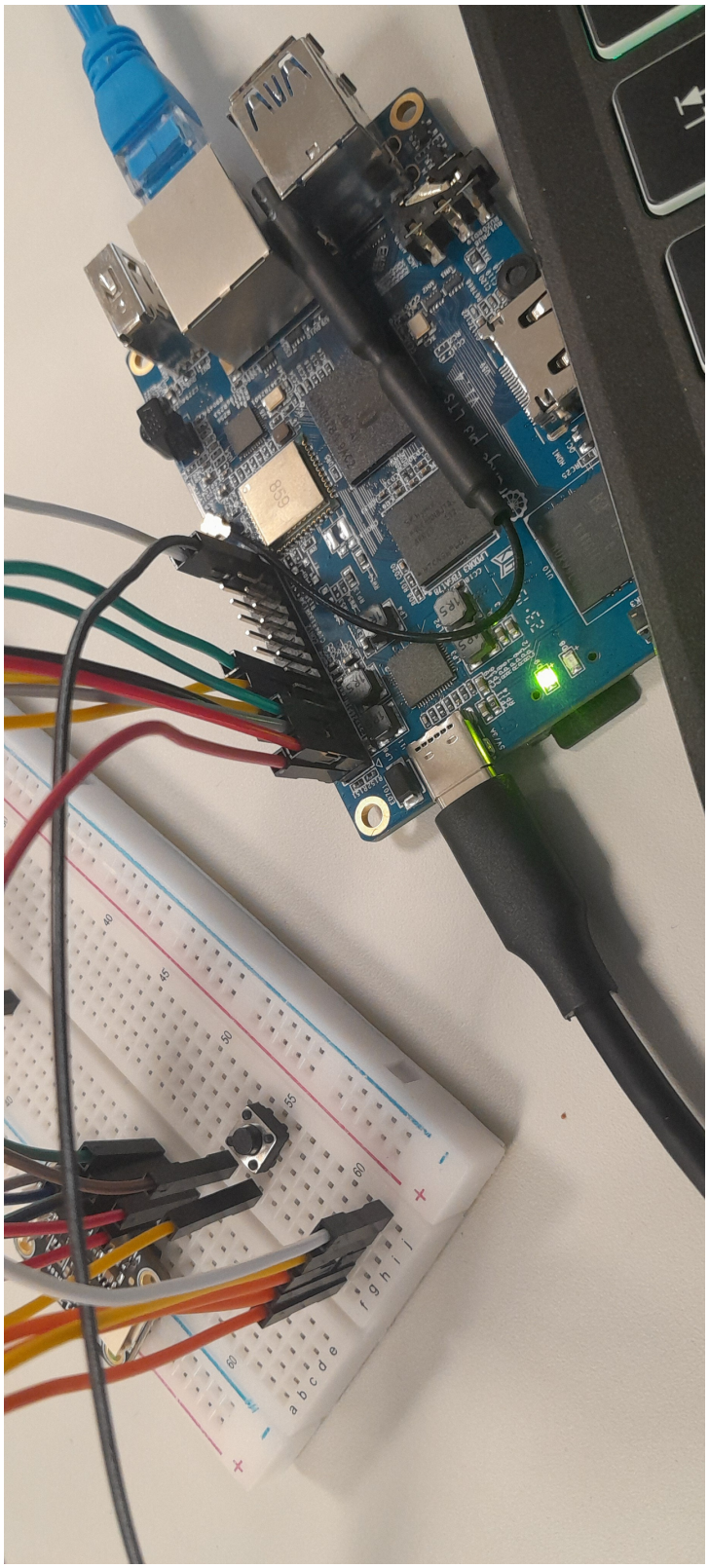
The equipments that were used in this project are: BMP280 sensor, BH1750 sensor, Breadboard, Orangepi, Wires, LED, resistors, Driver ULN2003 and pushbuttons.

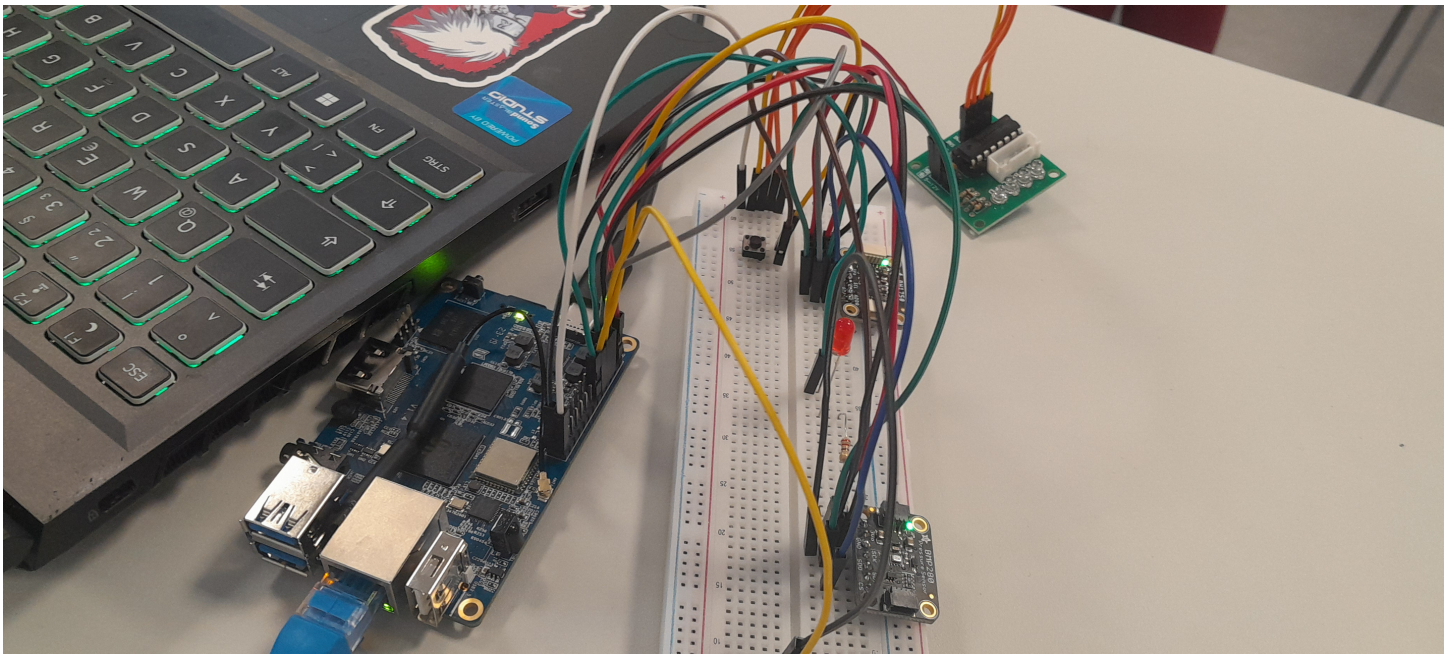
And the application terminal that i used was visual studio code.

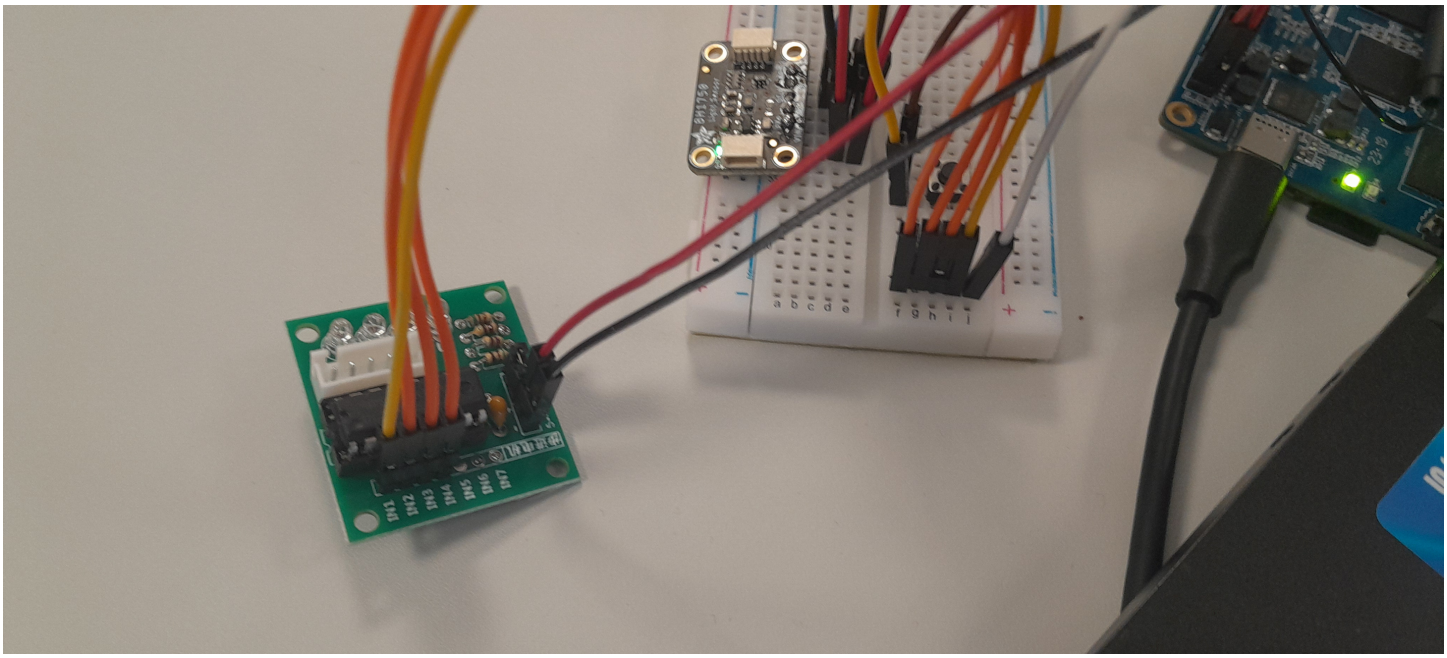
MY WIRING IMAGE

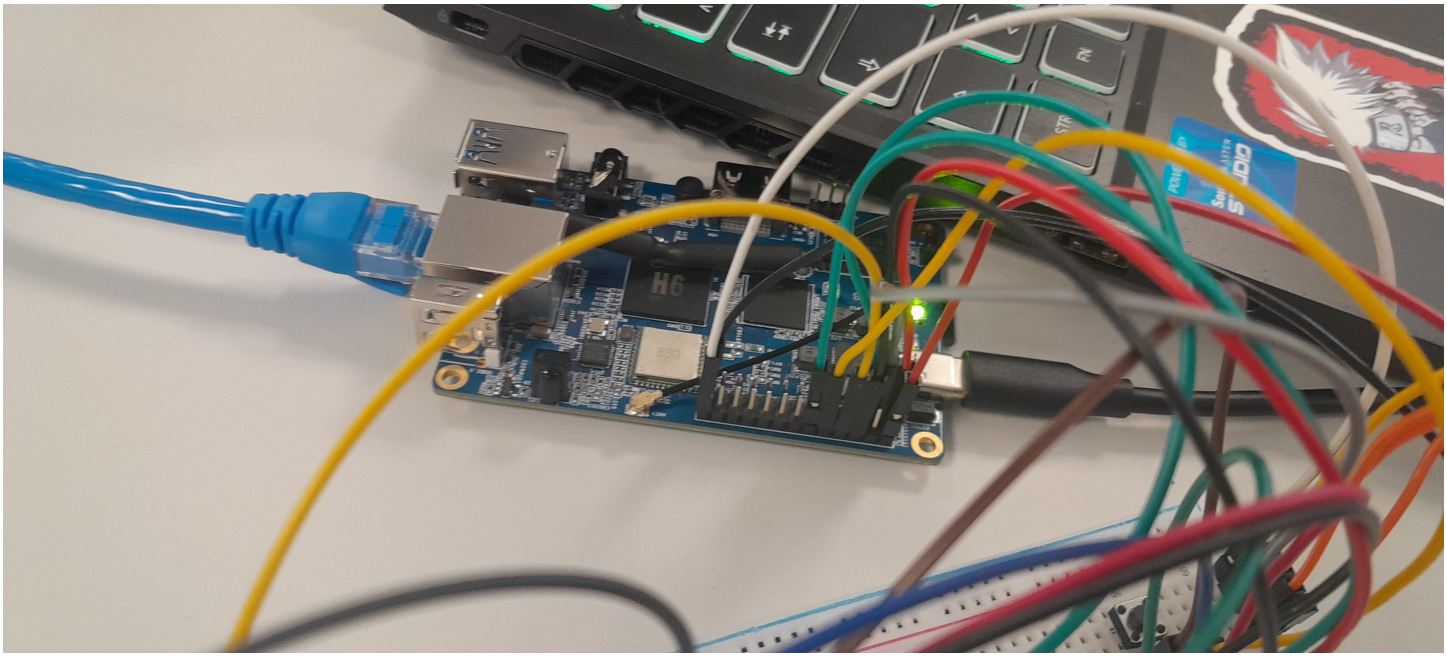
Taken from different angles for understanding

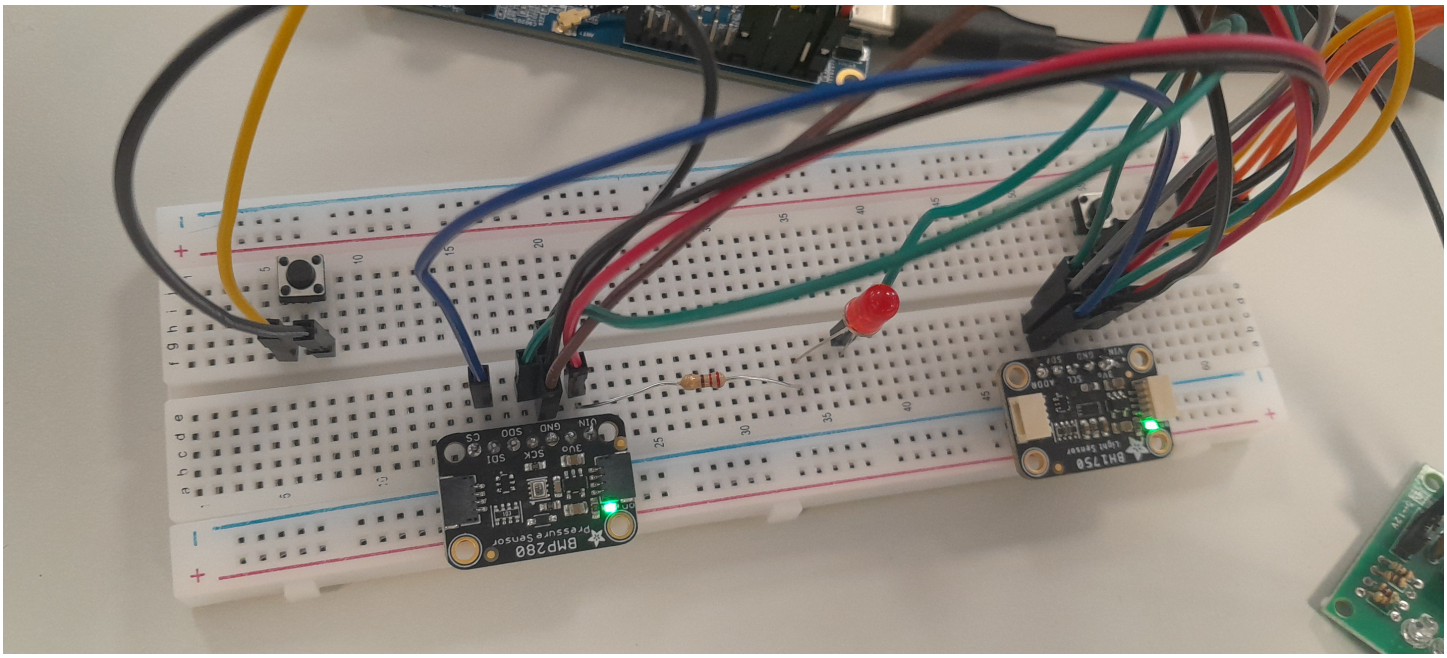


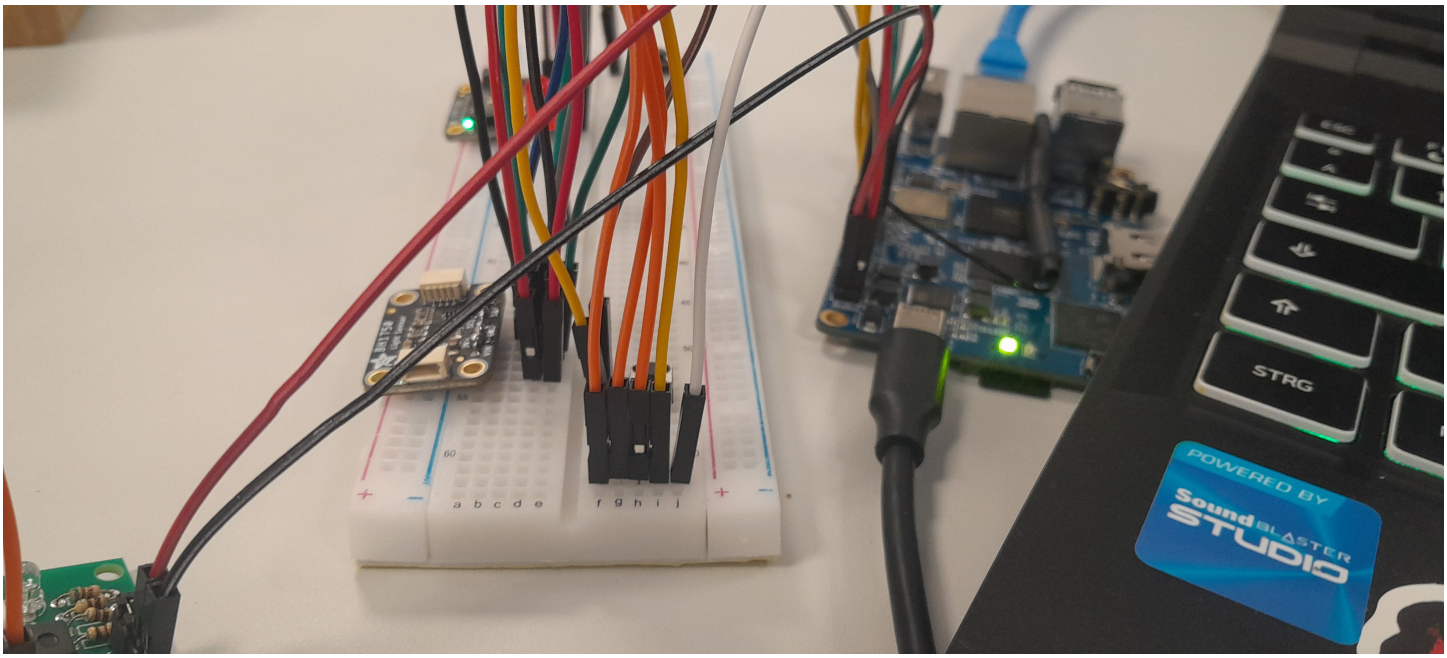


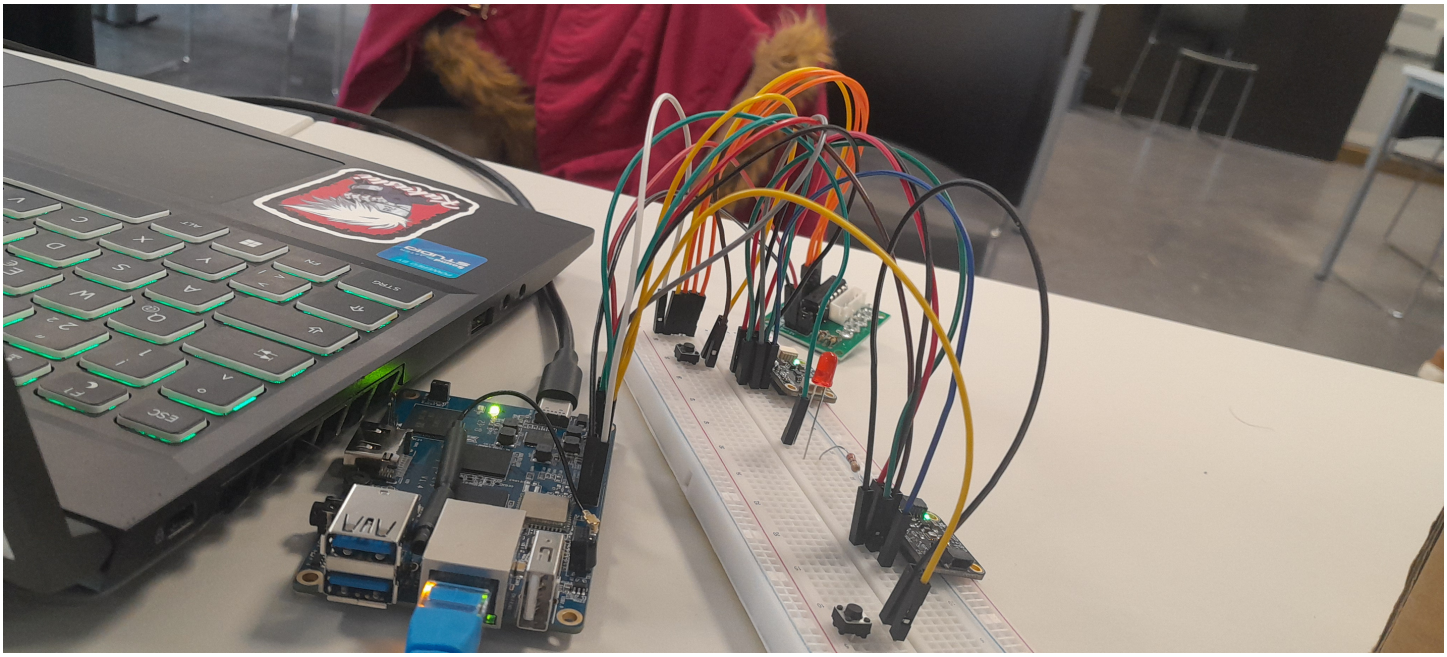


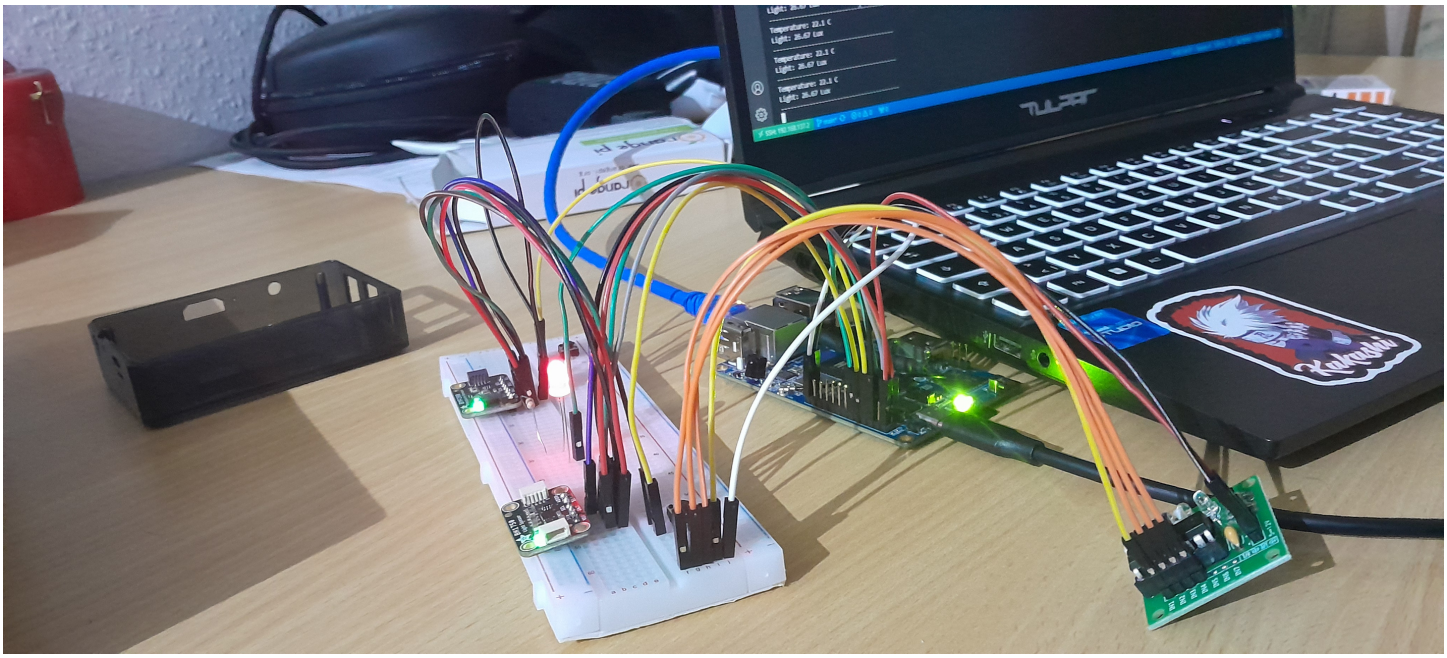





BMP280
Pressure Sensor
BH1750
Light Sensor

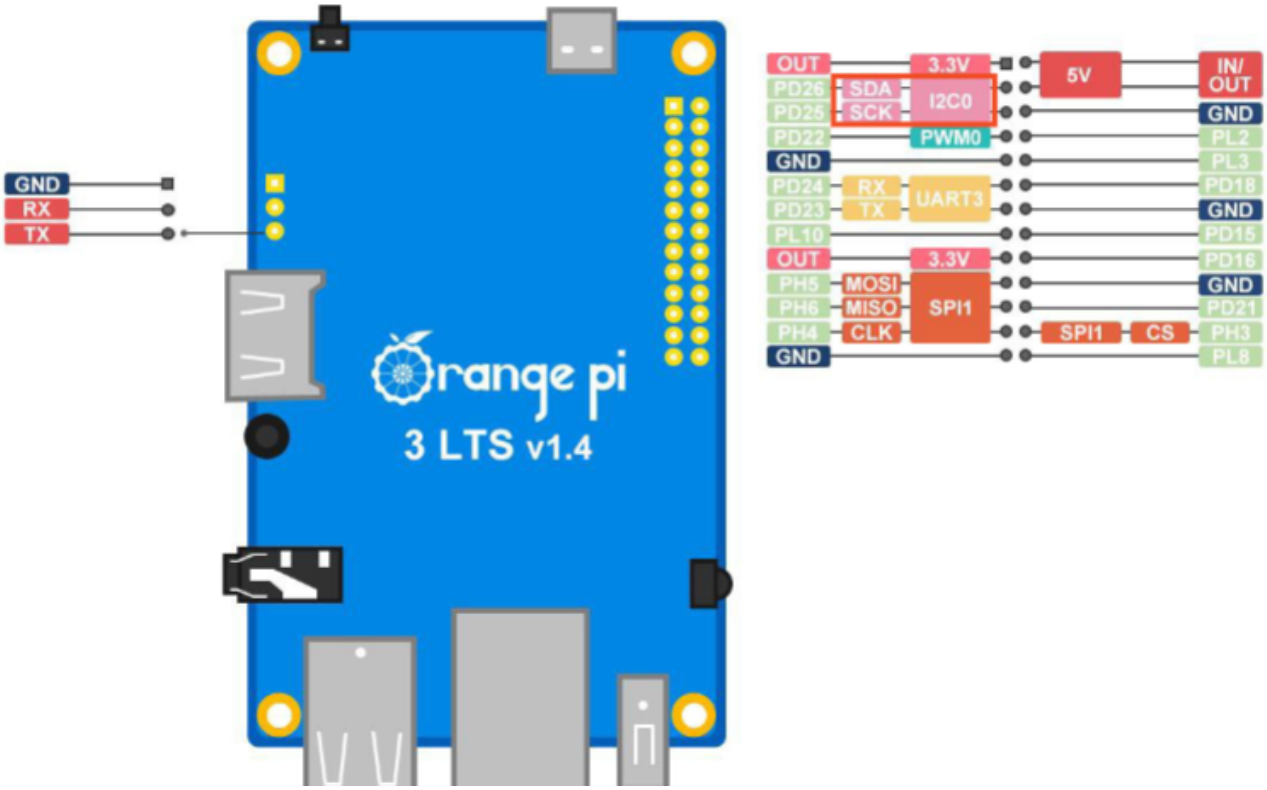

STRG
POWERED BY
Sound BLASTER
STUDIO



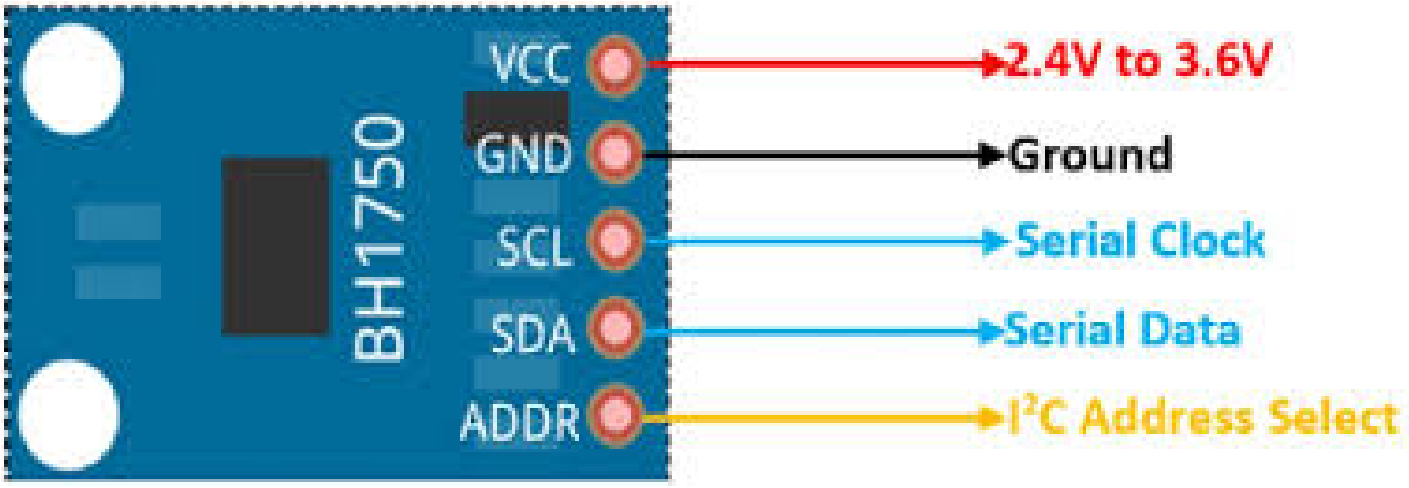


Explanation of the connections

1.Firstly, i connected the sensors to the orangePi through the sensors. Using this image guide



You need the BMP280(to check temperature) and BH1750(to check light) sensors. Let say we start the connection from the Bh1750 sensor.



As you can see in the image above we need to connect to these four places from the right places from the orangePi.

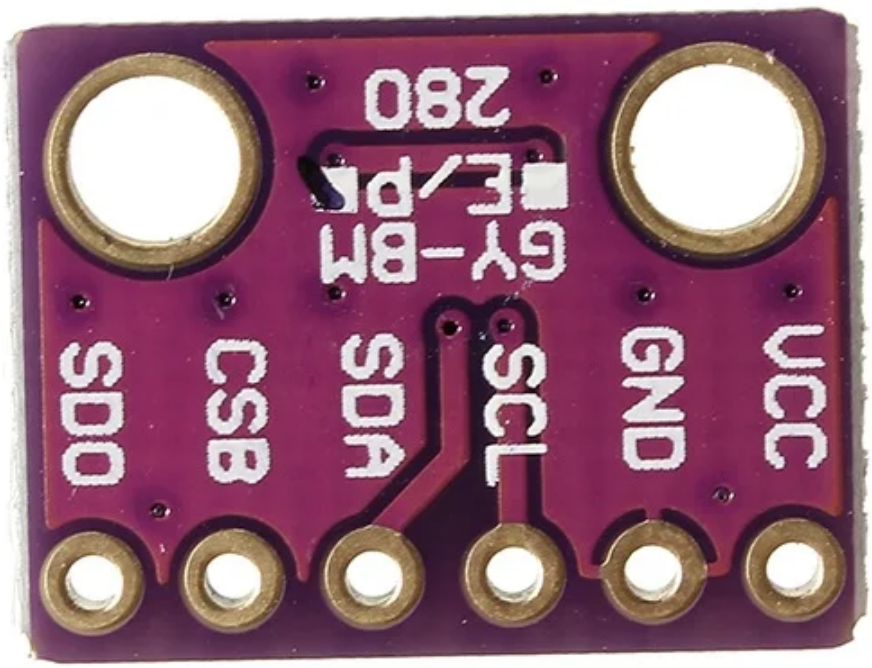
Vcc(voltage) = use 3.3v in the orangePi(pin 1) not 5v. so it should be connected from pin1 in the orangepi.

GND(ground)= There are 5 pins for ground in the orangePi. So i chose pin6(the first ground pin) for my connection.

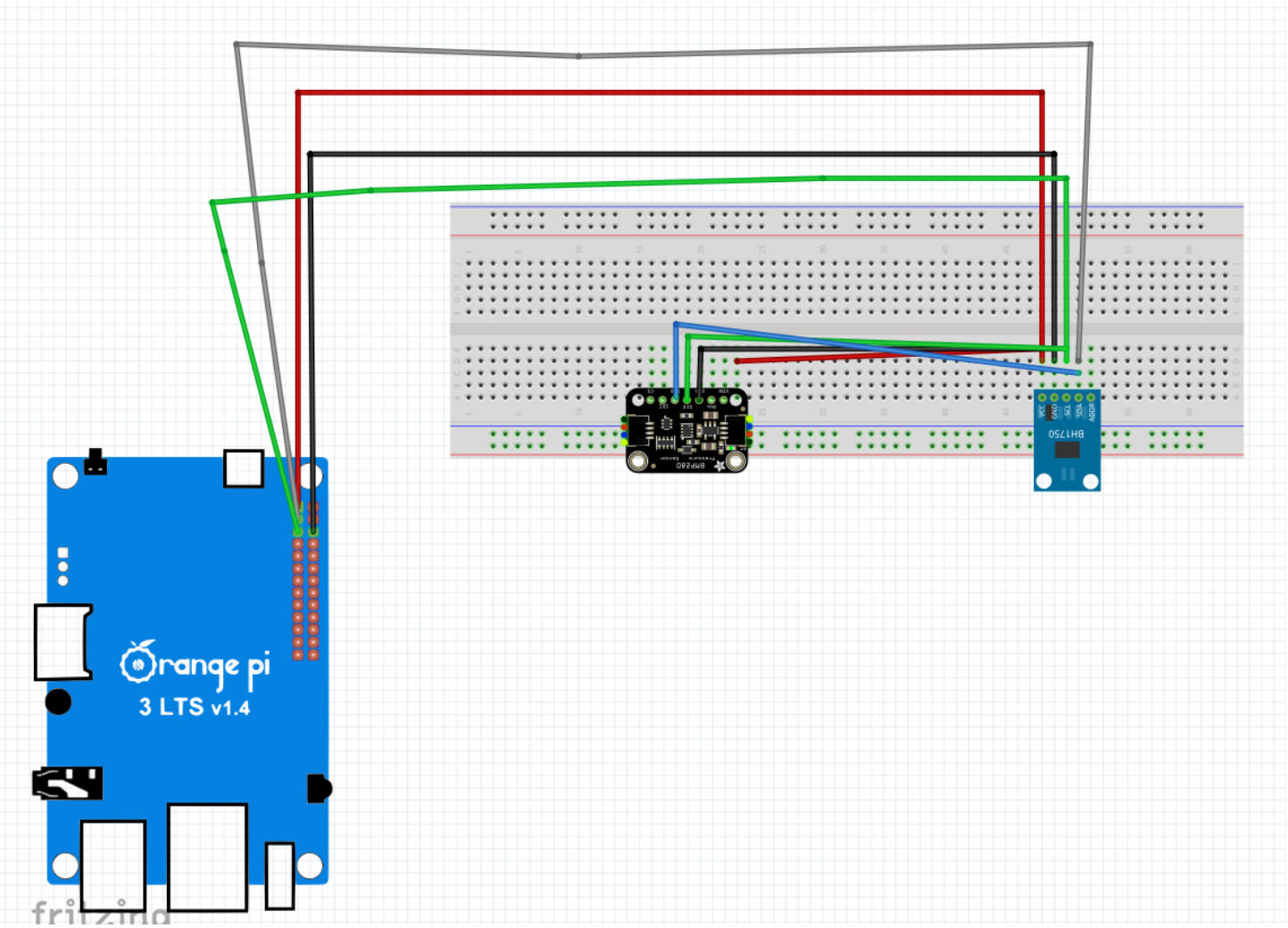
SCL(serial clock) = so it should be connected from pin5 in the orangepi.

SDA(serial Data) =so it should be connected from pin3 in the orangepi.

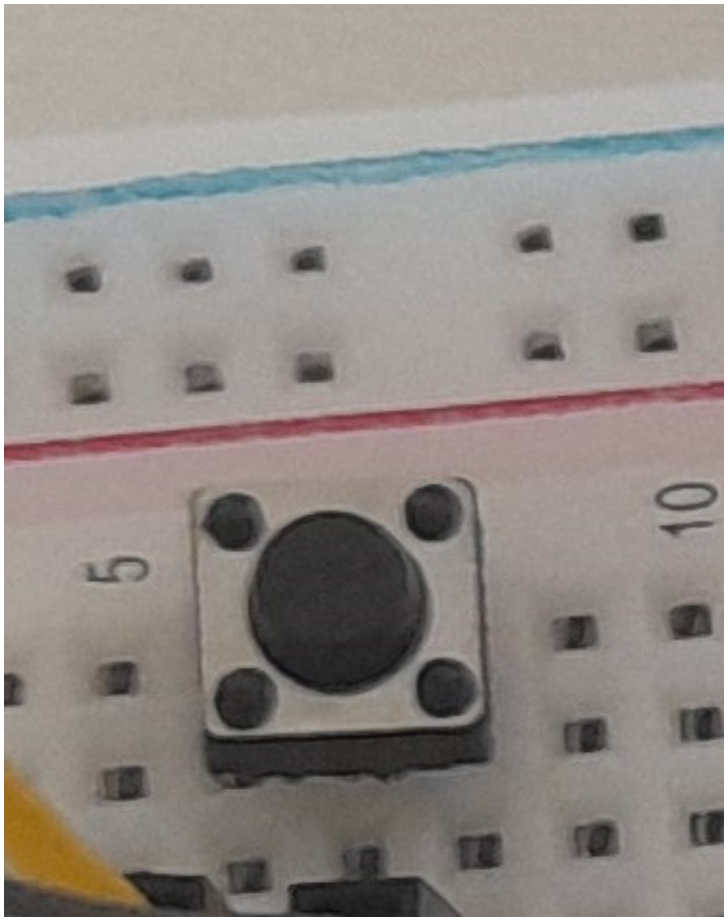
After that I connected from the BH1750 sensor to the BMP280 sensor with another wire using the sensor image guide below.



Blueprint(Note i only saw the blue BH1750 sensor, i didn’t see the black one)



2.Secondly, i connected two buttons(pushbuttons), one for each of the two sensors.



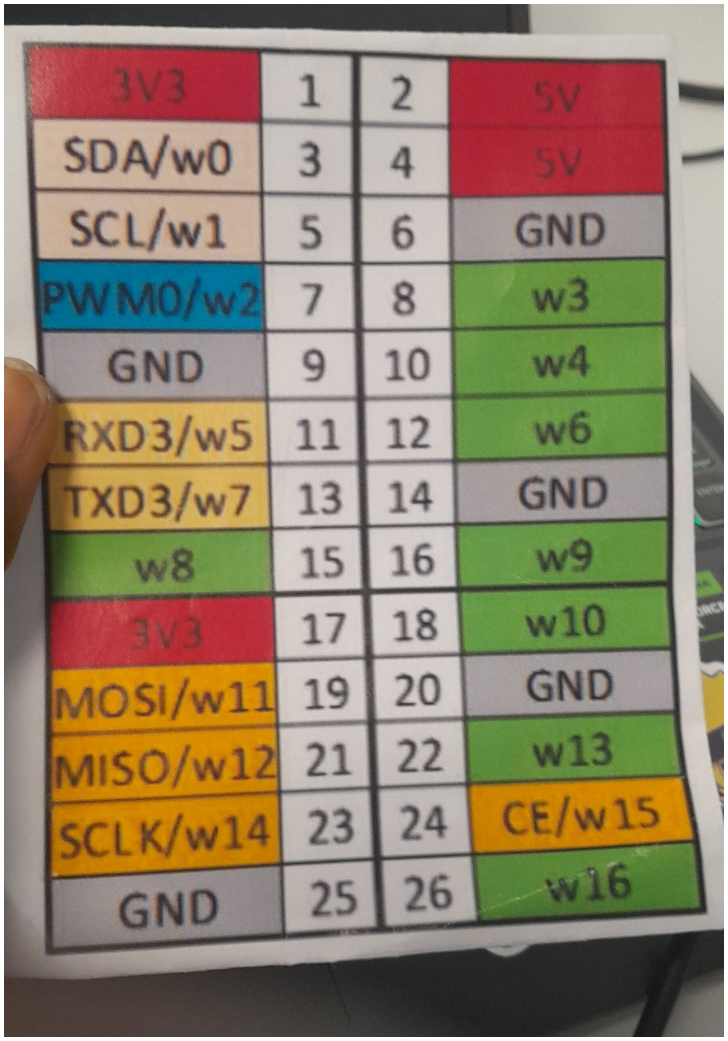
This is used to implement on the values of the lux value and temperature when it is been clicked. In my code(Which i will explain later) When button1 is clicked it adds 60 to the lux value and when button2 is clicked it removes 60 from the lux value.

```
Temperature: 20.7 C
Light: 21.67 Lux
---------------------------------------
btn 1 pressed
Temperature: 20.7 C
Light: 81.67 Lux
---------------------------------------
Temperature: 20.7 C
Light: 20.83 Lux
---------------------------------------
```

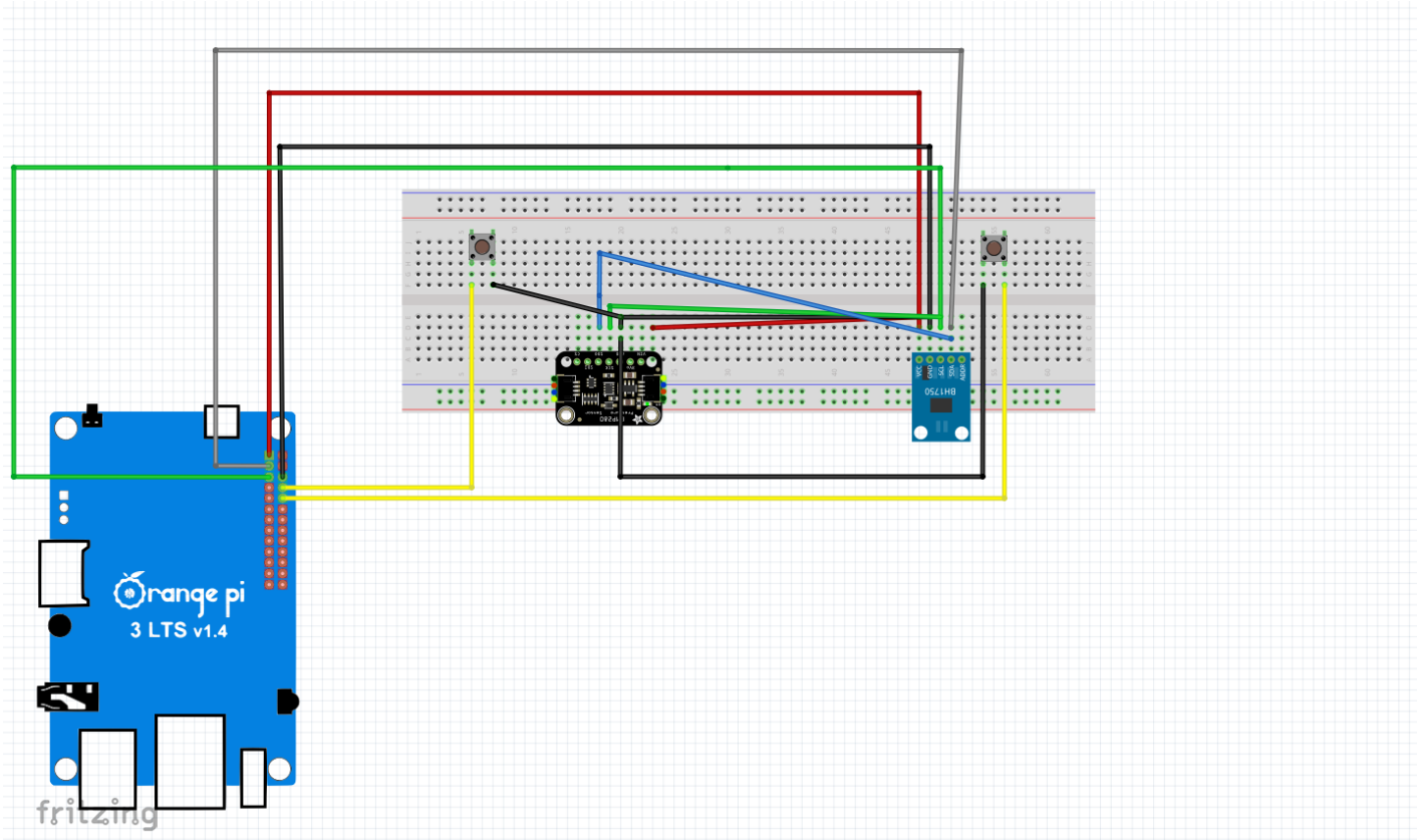
```
btn 2 pressed
Temperature: 20.9 C
Light: -40.83 Lux
----------------------------------------
Temperature: 20.9 C
Light: 17.50 Lux
----------------------------------------
```

How does it work?

I connected the GND(ground to both buttons) and connected to the button from w3(pin8) and w4(pin10) in the orangePi. The image below will show more what i mean by W3, w4.



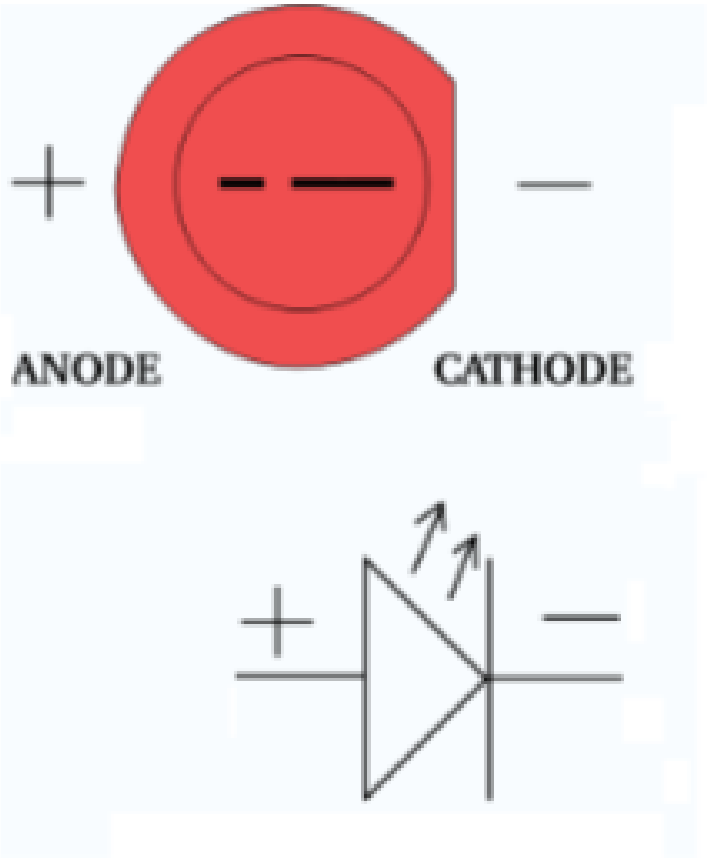
Blueprint

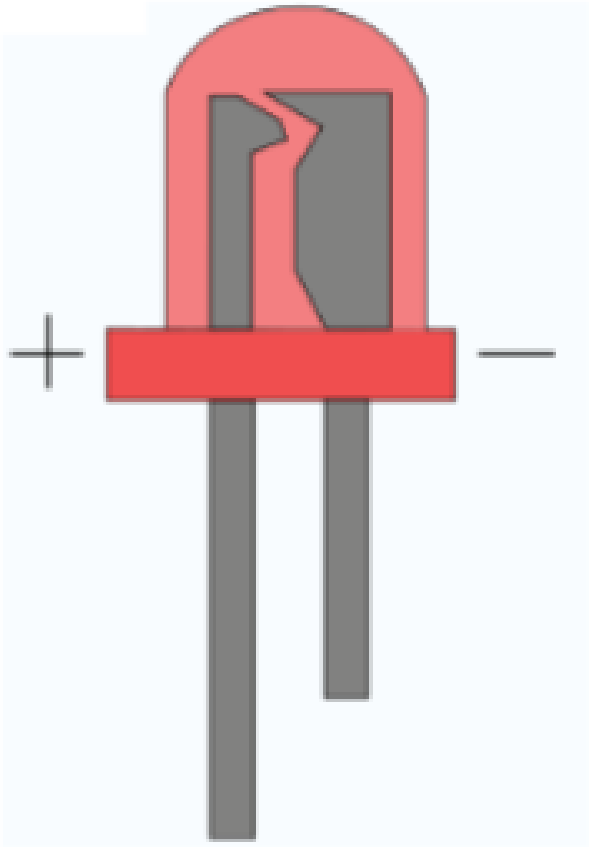


3.Thirdly, used the LED (light emitting diode) construction and resistor.

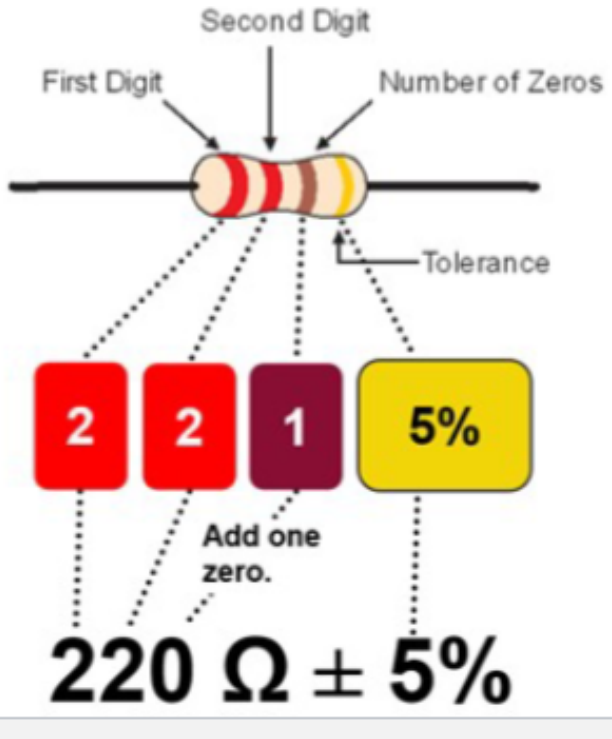
FOR the led it was used to indicate light in the electric device which is the BH1750. The led goes on when the sensor reads light. But when the lux value passes 50 it goes of.

For the resistor Control the amount of current flowing through from the sensor. It Divide voltage in the sensor.



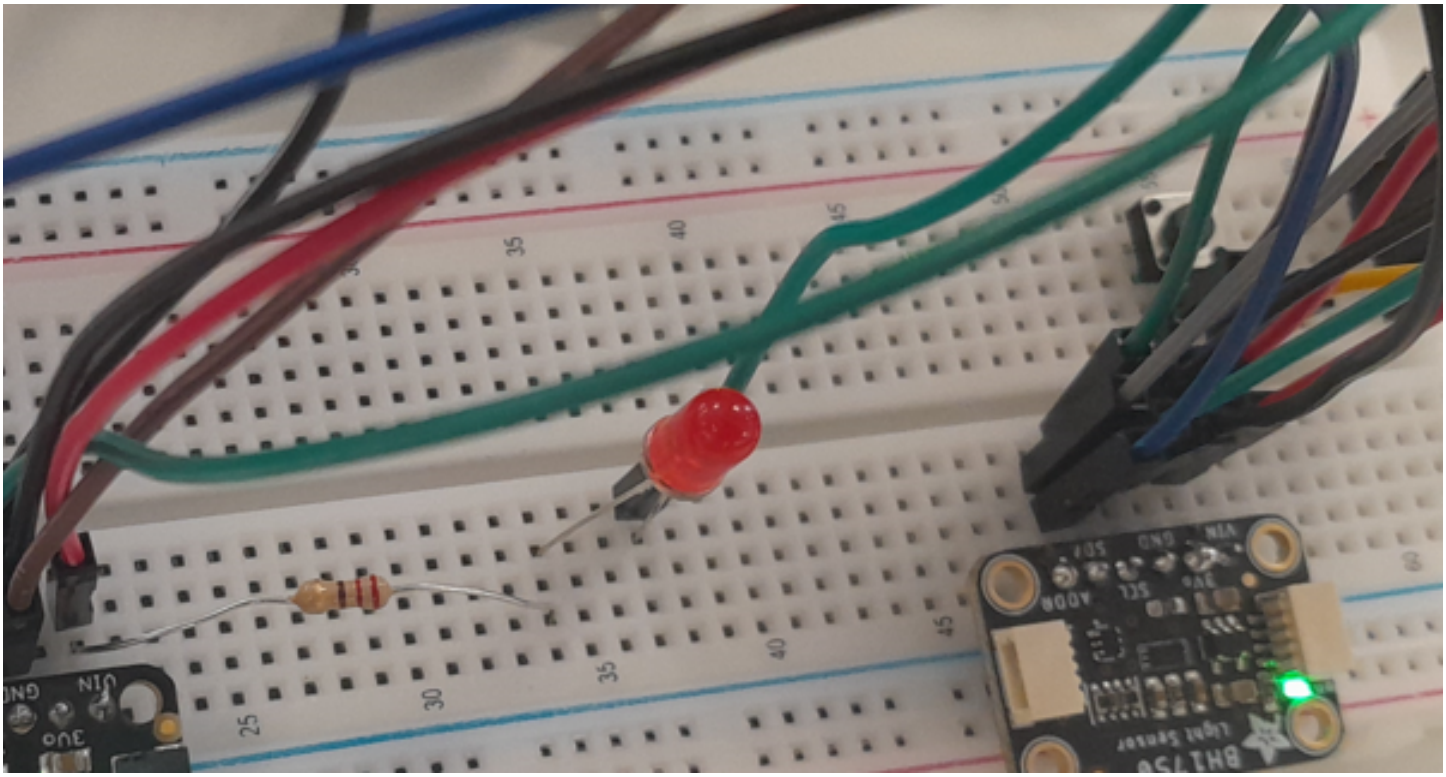


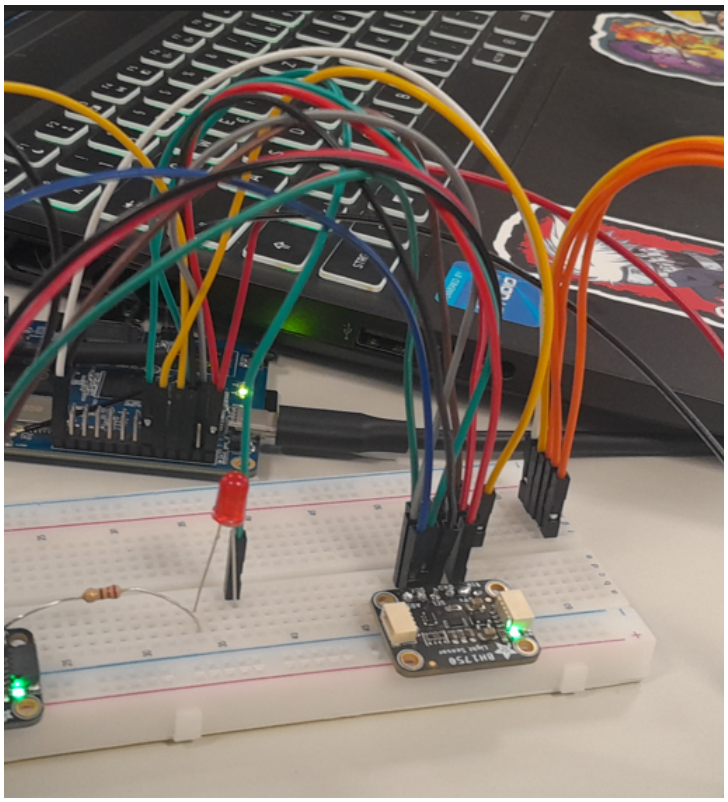
SENTIALS

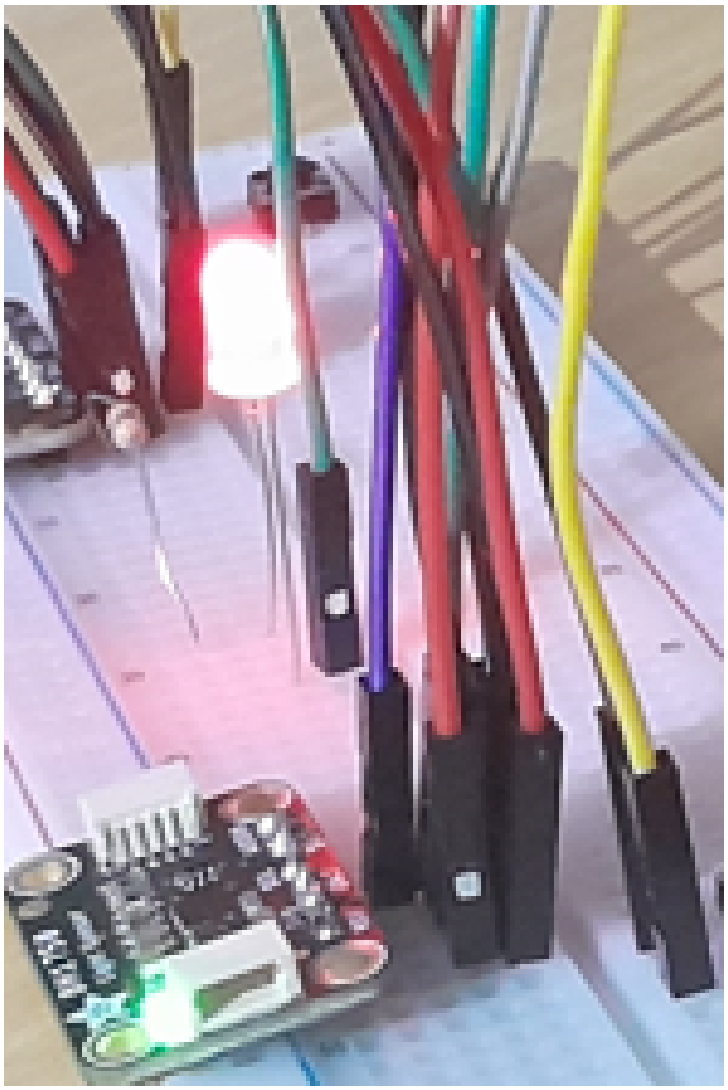


so, the resistor was connected from the voltage area in the sensor to the anode(positive side)

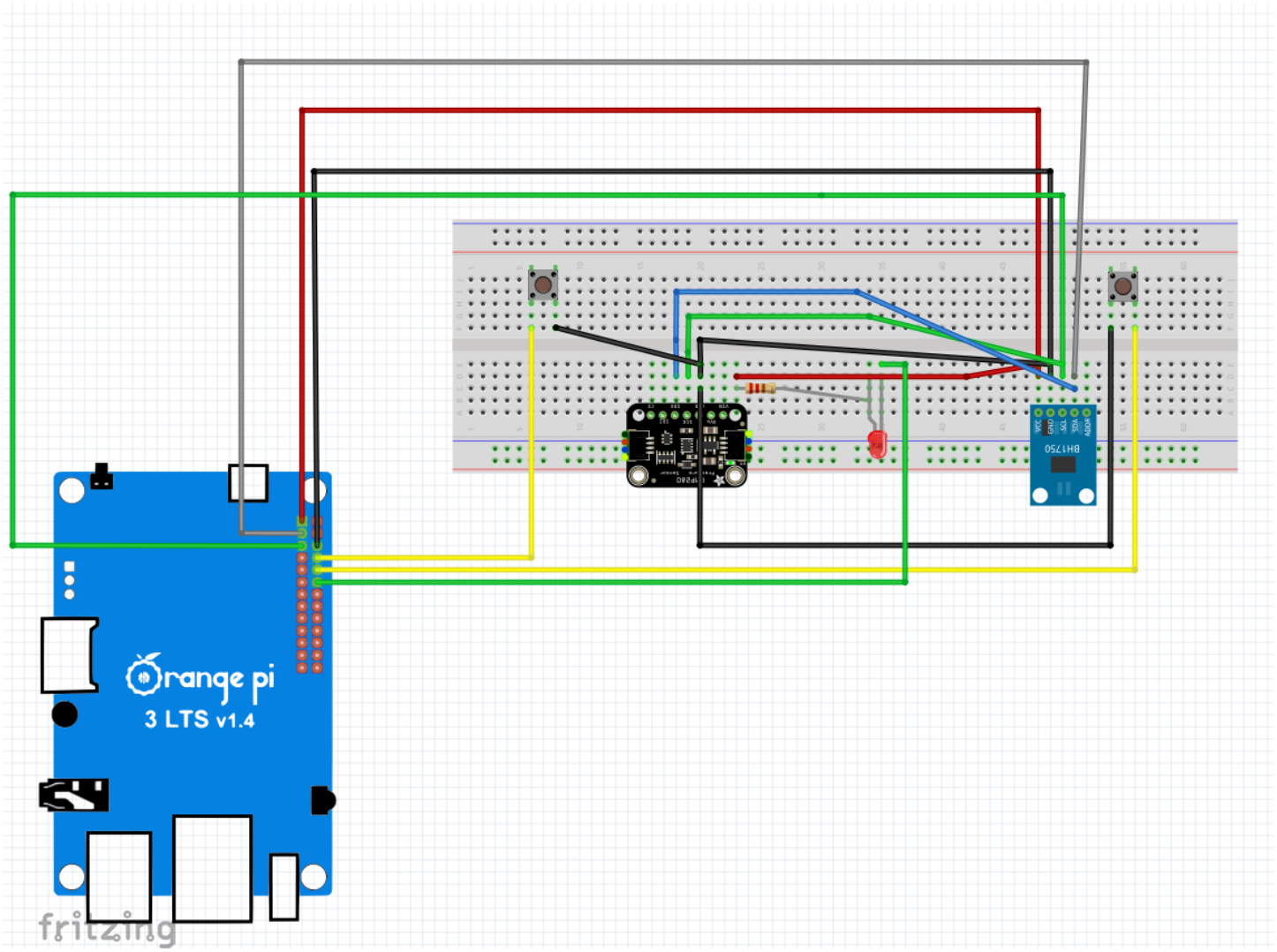
of the LED. And the W6(pin12) from the orange pi was connected to the cathode of the Led.







Blueprint



4.Lastly, The ULN2003 was connected in the aim to check the temperature.

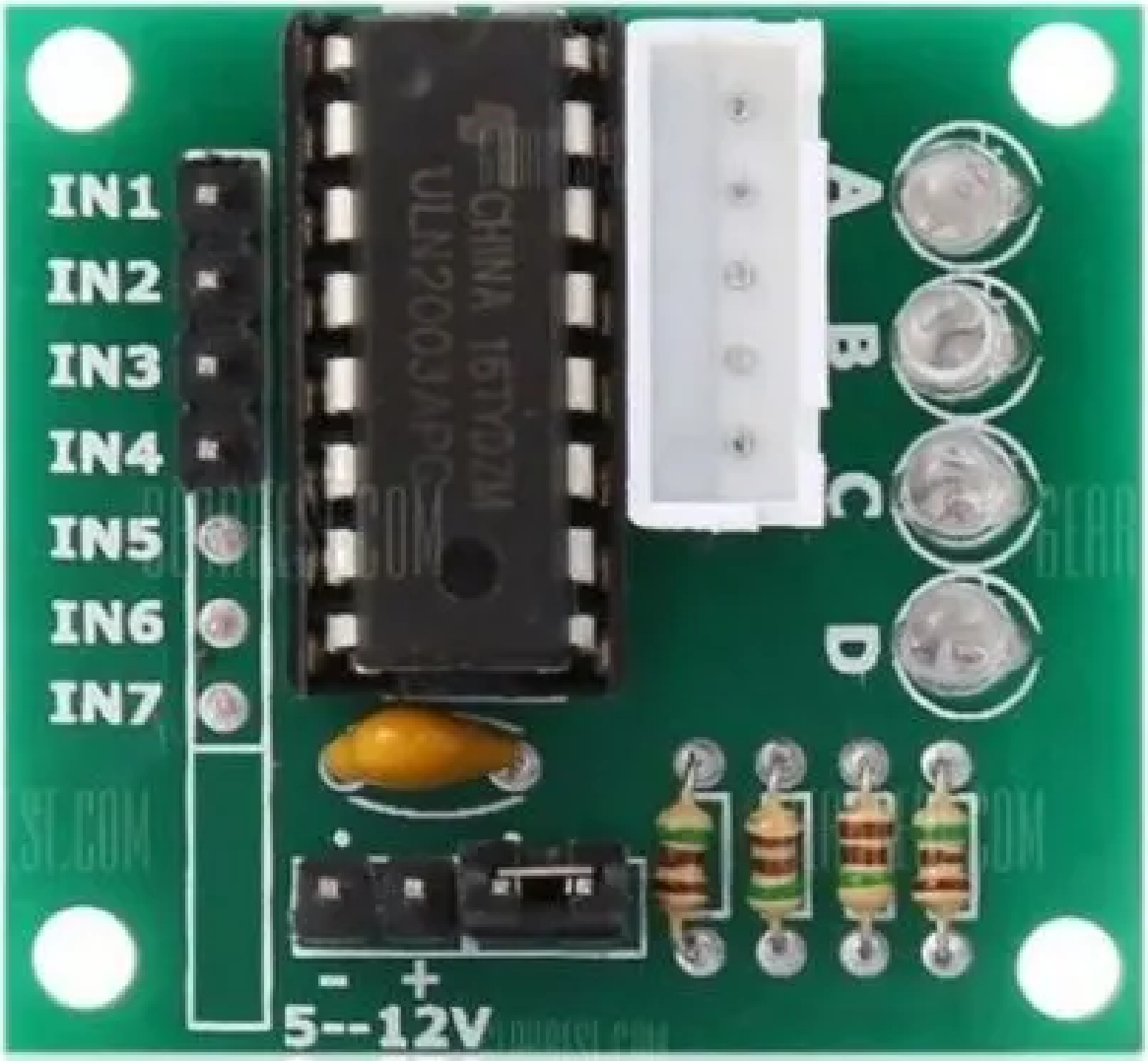
In the code when the temperature is >= 26 it goes on.(Better view in the video)

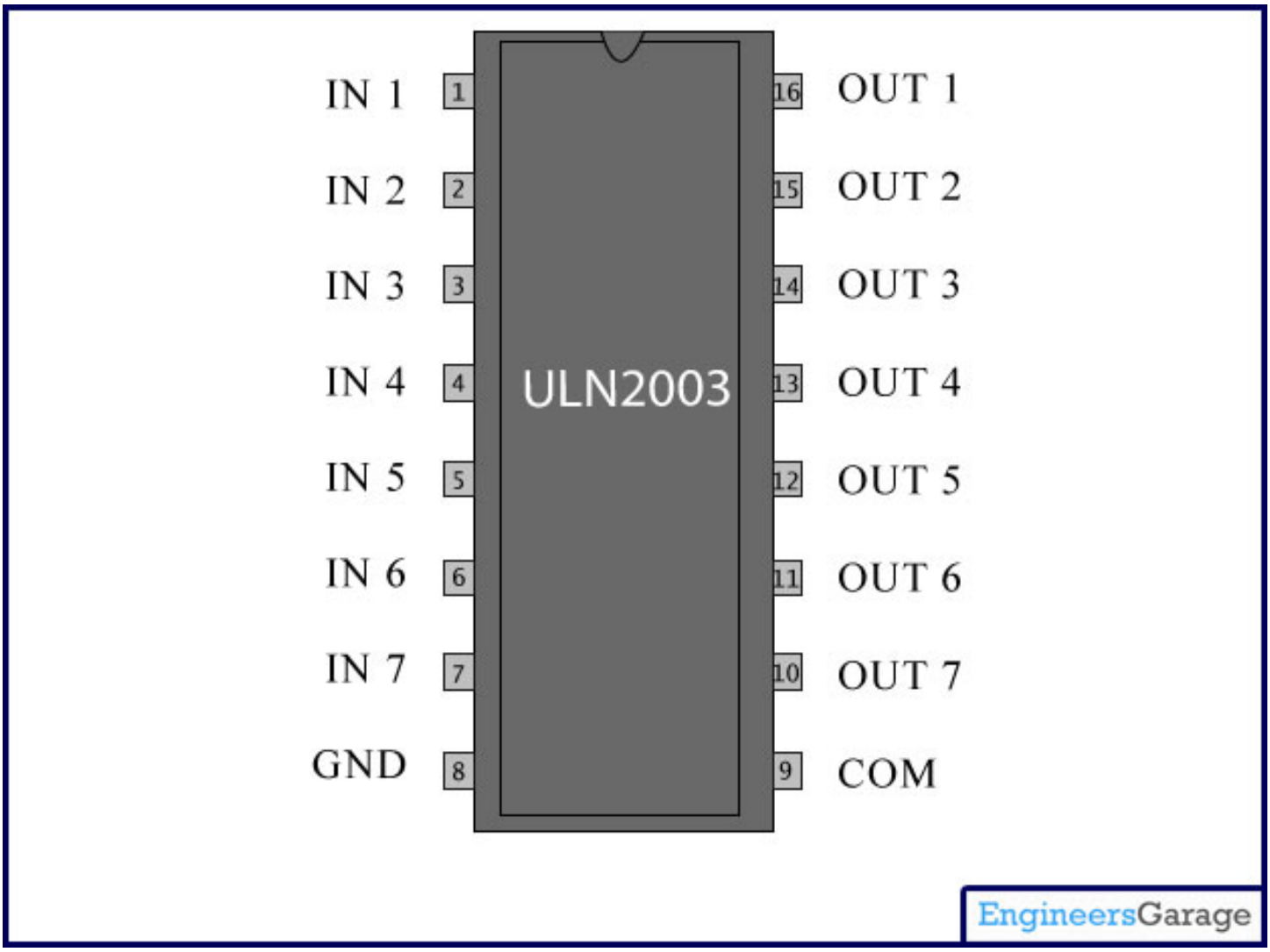
pins connected from the orangepi:-

-5v(pin2) to the ULN positive side.

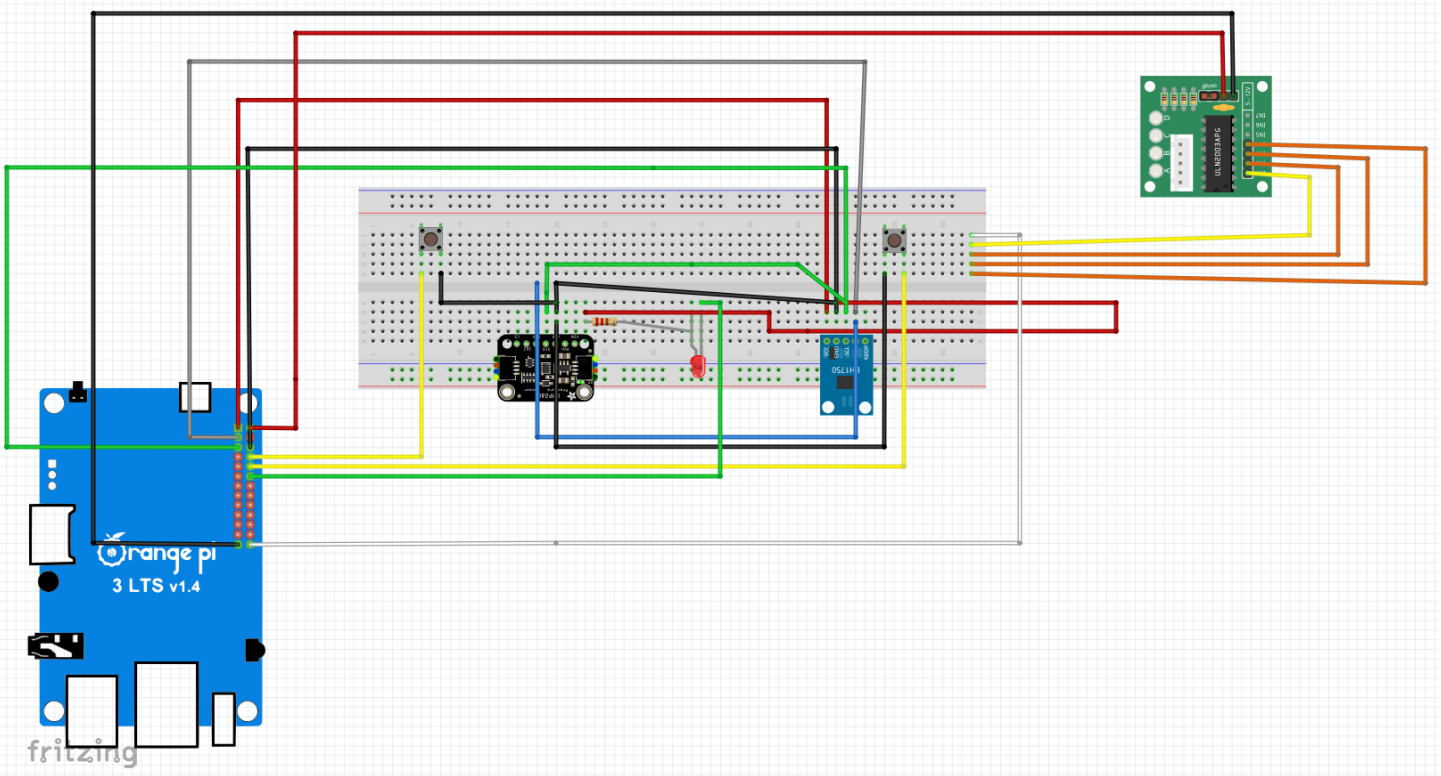
-GND(pin25) to the ULN negative side.

-And connect wire from 4 input area to the breadboard and another wire is been connected from the same area of the 4 inputs in the breadboard to the w16(pin26) in the orangePi.


IN1
IN2
IN3
IN4
IN5
IN6
IN7
ULN2003APG
A
B
C
D
- +
5--12V


IN 1
IN 2
IN 3
IN 4
IN 5
IN 6
IN 7
GND
1
2
3
4
5
6
7
8
ULN2003
16
15
14
13
12
11
10
9
OUT 1
OUT 2
OUT 3
OUT 4
OUT 5
OUT 6
OUT 7
COM
EngineersGarage

Blueprint


Orange Pi
3 LTS v1.4
fritzing

CODE

```
import time

from smbus2 import SMBus, i2c_msg

from bmp280 import BMP280

import wiringpi

# from smbus2 import SMBus

import paho.mqtt.client as mqtt


# Create an I2C bus object

bus = SMBus(0)

address_bh170 = 0x23 # i2c address


# ================== BH1750 Sensor

bus.write_byte(address_bh170, 0x10)

bytes_read = bytearray(2)


def get_value(bus, address):

    write = i2c_msg.write(address, [0x10]) # Command for continuous measurement in high-resolution mode

    read = i2c_msg.read(address, 2)

    bus.i2c_rdwr(write, read)

    bytes_read = list(read)

    # Corrected formula to calculate lux

    return (bytes_read[0] << 8 | bytes_read[1]) / 1.2



# ================== BMP208 Sensor


# Create an I2C bus object
```

```
bus = SMBus(0)
address = 0x77

# Setup BMP280
bmp280 = BMP280(i2c_addr= address, i2c_dev=bus)
interval = 1 # Sample period in seconds

changeValBtns = [3, 4]

def setup(button):
    wiringpi.wiringPiSetup()  # Initialize wiringPi setup
    wiringpi.pinMode(button, wiringpi.INPUT)  # Set pin to input mode
    wiringpi.pullUpDnControl(button, wiringpi.PUD_UP)  # Enable pull-up resistor

for button in changeValBtns:
    setup(button)

# ====== Change the value

def changeLuxValue(b, l):
    for index, button in enumerate(b):
        if wiringpi.digitalRead(button) == wiringpi.LOW:
            if index == 0:  # Assuming button 1 is for the 'button1' data
                l+=60;
                print('btn 1 pressed')
            elif index == 1:  # Assuming button 1 is for the 'button1' data
                l-=60;
                print('btn 2 pressed')
            # Add conditions for other buttons if needed
```

```
            time.sleep(0.5)  # Simple debounce
    return l

wiringpi.wiringPiSetup()
# Light up the LEDs
ledPin = 6;
ledTemp = 16;

wiringpi.pinMode(ledPin, 1)
wiringpi.pinMode(ledTemp, 1)

def lightUp(pin):
    wiringpi.digitalWrite(pin, 1)

def lightDown(pin):
    # wiringpi.wiringPiSetup()
    wiringpi.digitalWrite(pin, 0)

# ================== Loop To Render the values

while True:
    # Measure data
    lux = get_value(bus, address_bh170)
    # Measure temperature and pressure from BMP280
    temperature = bmp280.get_temperature()
    pressure = bmp280.get_pressure()

    # changing lux here with push button

    lux = changeLuxValue(changeValBtns, lux)
```

```
if lux < 50:
    lightDown(ledPin)
else:
    lightUp(ledPin)

if temperature > 26:
    lightUp(ledTemp)
else:
    lightDown(ledTemp)

# Print each measured value on a separate line
print("Temperature: {:.1f} C".format(temperature))
# print("Pressure: {:.1f} hPa".format(pressure))
print("Light: {:.2f} Lux".format(lux))
print("---------------------------------------")

time.sleep(0.5)
```

CODE EXPLANATION😋

```
import time

from smbus2 import SMBus, i2c_msg

from bmp280 import BMP280

import wiringpi

# from smbus2 import SMBus

import paho.mqtt.client as mqtt
```

👆**This part imports the necessary libraries for handling the i2c communication, sensor data,time, GPIO pins and MQTT communication**

```
# Create an I2C bus object

bus = SMBus(0)

address_bh170 = 0x23 # i2c address


# ================== BH1750 Sensor

bus.write_byte(address_bh170, 0x10)

bytes_read = bytearray(2)
```

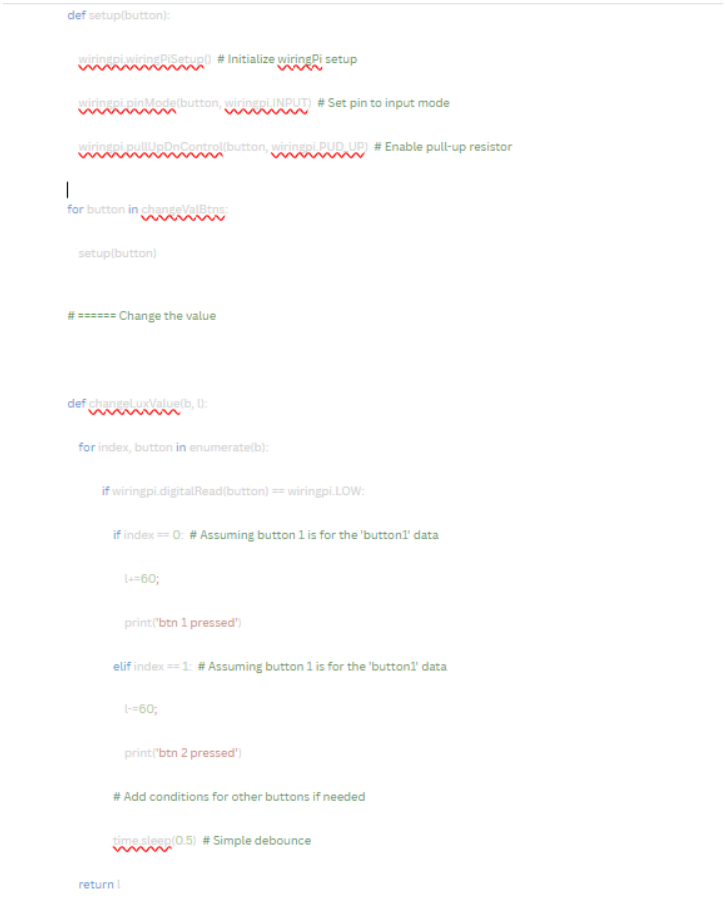
👆**This part an i2c object named “bus” is created using the SMBus class from the smbus module. So the number 0 indicates the i2c bus number to be used. The i2c address which will be used to communicate with the BH1750 sensor will be assigned. And also a single byte is been sent to the i2c address using the method bus object.**

```
def get_value(bus, address):
    write = i2c_msg.write(address, [0x10]) # Command for continuous measurement in high-resolution mode
    read = i2c_msg.read(address, 2)
    bus.i2c_rdwr(write, read)
    bytes_read = list(read)
    # Corrected formula to calculate lux
    return (bytes_read[0] << 8 | bytes_read[1]) / 1.2
```

👆**This function communicates with an i2c device(BH1750 sensor) to get the light intensity value in lux using the correct formula and returns the calculated lux value.**

```
# ================== BMP208 Sensor


# Create an I2C bus object

bus = SMBus(0)

address = 0x77


# Setup BMP280

bmp280 = BMP280(i2c_addr= address, i2c_dev=bus)

interval = 1 # Sample period in seconds


changeValBtns = [3, 4]
```

👆**This code initializes the communication with the BMP280 sensor using the i2c, sets up the sensor object, defines the interval and the specified pins connected to the button used for changing value**



```
def setup(button):
    wiringpi.wiringPiSetup()  # Initialize wiringPi setup
    wiringpi.pinMode(button, wiringpi.INPUT)  # Set pin to input mode
    wiringpi.pullUpDnControl(button, wiringpi.PUD_UP)  # Enable pull-up resistor

for button in changeValBtns:
    setup(button)

# ====== Change the value


def changeLuxValue(b, l):
    for index, button in enumerate(b):
        if wiringpi.digitalRead(button) == wiringpi.LOW:
            if index == 0:  # Assuming button 1 is for the 'button1' data
                l+=60;
                print("btn 1 pressed")
            elif index == 1:  # Assuming button 1 is for the 'button1' data
                l-=60;
                print("btn 2 pressed")
            # Add conditions for other buttons if needed
            time.sleep(0.5)  # Simple debounce
    return l
```

👆This code sets up the button using the wiringPi library, initializes them for input, enables pull up resistors, and defines a function to change the lux value based on button pressed. The function increases the lux value by 60 when button 1 is pressed and decreases it by 60 when button 2 is pressed, with a debounce time of 0.5 seconds.

```
wiringpi.wiringPiSetup()

# Light up the LEDs

ledPin = 6;

ledTemp = 16;


wiringpi.pinMode(ledPin, 1)

wiringpi.pinMode(ledTemp, 1)


def lightUp(pin):

   wiringpi.digitalWrite(pin, 1)


def lightDown(pin):

   # wiringpi.wiringPiSetup()

   wiringpi.digitalWrite(pin, 0)
```

👆 **This code initializes the wiringPi, sets up two LED pins, defines the functions to light up and light off the led connected to a specific pin.**

```
while True:
    # Measure data
    lux = get_value(bus, address_bh170)
    # Measure temperature and pressure from BMP280
    temperature = bmp280.get_temperature()
    pressure = bmp280.get_pressure()

    # changing lux here with push button

    lux = changeLuxValue(changeValBtns, lux)

    if lux < 50:
        lightDown(ledPin)
    else:
        lightUp(ledPin)
```

👆**The code continuously measures the light intensity, pressure and temperature from sensors while updating the lux value based on the button pressed. And then control the lead based on a lux value, turning it off if lux value is less than 50 and turning on otherwise.**

```
if temperature > 26:
    lightUp(ledTemp)
else:
    lightDown(ledTemp)

# Print each measured value on a separate line
print("Temperature: {:.1f} C".format(temperature))
# print("Pressure: {:.1f} hPa".format(pressure))
print("Light: {:.2f} Lux".format(lux))
print("-------------------------------------")


time.sleep(0.5)
```

👆The code checks the value and turns on another led which in this case is the ULN2003, checking if the temperature is 26 degree celsius or above, then the light goes on. It the prints the temperature, lux value(light intensity) and a separator line, and pauses for 0.5secs for the next iteration.

SHOW ON THE CLOUD DASHBOARD(MQTT)

I used ThingSpeak to achieve this. BY connecting and creating two fields, one for temperature and other for lux value.

I added commands for the MQTT to the normal code.

THIS TO THE START OF THE CODE:

```
# MQTT settings
MQTT_HOST ="mqtt3.thingspeak.com"
MQTT_PORT = 1883
MQTT_KEEPALIVE_INTERVAL =60
MQTT_TOPIC = "channels/2463672/publish"
```

```
MQTT_CLIENT_ID ="Bh4WNyg3LRAuADAaBiYkNxQ"

MQTT_USER ="Bh4WNyg3LRAuADAaBiYkNxQ"

MQTT_PWD = "0zDCWCrI+SRDwL0WqYXPHN0o"

# ========== MQTT error handling

def on_connect(client, userdata, flags, rc):

  if rc==0:

    print("Connected OK with result code "+str(rc))

  else:

    print("Bad connection with result code "+str(rc))

def on_disconnect(client, userdata, flags, rc=0):

  print("Disconnected result code "+str(rc))

def on_message(client,userdata,msg):

  print("Received a message on topic: " + msg.topic + "; message: " + msg.payload)

# Set up a MQTT Client

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, MQTT_CLIENT_ID)

client.username_pw_set(MQTT_USER, MQTT_PWD)

# Connect callback handlers to client

client.on_connect= on_connect

client.on_disconnect= on_disconnect

client.on_message= on_message

print("Attempting to connect to %s" % MQTT_HOST)

client.connect(MQTT_HOST, MQTT_PORT)

client.loop_start() #start the loop
```

And this to the end of the code:

```
MQTT_DATA =
"field1="+str(lux)+"&field2="+str(temperature)+"&field3="+str(pressure)+"&status=MQTTPUBLISH"

    client.publish(topic=MQTT_TOPIC, payload=MQTT_DATA, qos=0, retain=False, properties=None)
```

FULL CODE OVERVIEW

```
import time

from smbus2 import SMBus, i2c_msg

from bmp280 import BMP280

import wiringpi

# from smbus2 import SMBus

import paho.mqtt.client as mqtt

# MQTT settings

MQTT_HOST ="mqtt3.thingspeak.com"

MQTT_PORT = 1883

MQTT_KEEPALIVE_INTERVAL =60

MQTT_TOPIC = "channels/2463672/publish"

MQTT_CLIENT_ID ="Bh4WNyg3LRAuADAaBiYkNxQ"

MQTT_USER ="Bh4WNyg3LRAuADAaBiYkNxQ"

MQTT_PWD = "0zDCWCrI+SRDwL0WqYXPHN0o"


# ========== MQTT error handling

def on_connect(client, userdata, flags, rc):

    if rc==0:
```

```
        print("Connected OK with result code "+str(rc))
    else:
        print("Bad connection with result code "+str(rc))

def on_disconnect(client, userdata, flags, rc=0):
    print("Disconnected result code "+str(rc))

def on_message(client,userdata,msg):
    print("Received a message on topic: " + msg.topic + "; message: " + msg.payload)


# Set up a MQTT Client
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, MQTT_CLIENT_ID)
client.username_pw_set(MQTT_USER, MQTT_PWD)

# Connect callback handlers to client
client.on_connect= on_connect
client.on_disconnect= on_disconnect
client.on_message= on_message
print("Attempting to connect to %s" % MQTT_HOST)
client.connect(MQTT_HOST, MQTT_PORT)
client.loop_start() #start the loop

# Create an I2C bus object
bus = SMBus(0)
address_bh170 = 0x23 # i2c address

# ================== BH1750 Sensor
bus.write_byte(address_bh170, 0x10)
bytes_read = bytearray(2)
```

```
def get_value(bus, address):
    write = i2c_msg.write(address, [0x10]) # Command for continuous measurement in high-resolution mode
    read = i2c_msg.read(address, 2)
    bus.i2c_rdwr(write, read)
    bytes_read = list(read)
    # Corrected formula to calculate lux
    return (bytes_read[0] << 8 | bytes_read[1]) / 1.2


# ================== BMP208 Sensor

# Create an I2C bus object
bus = SMBus(0)
address = 0x77

# Setup BMP280
bmp280 = BMP280(i2c_addr= address, i2c_dev=bus)
interval = 1 # Sample period in seconds

changeValBtns = [3, 4]

def setup(button):
    wiringpi.wiringPiSetup()  # Initialize wiringPi setup
    wiringpi.pinMode(button, wiringpi.INPUT)  # Set pin to input mode
    wiringpi.pullUpDnControl(button, wiringpi.PUD_UP)  # Enable pull-up resistor

for button in changeValBtns:
    setup(button)
```

```
# ====== Change the value


def changeLuxValue(b, l):
    for index, button in enumerate(b):
        if wiringpi.digitalRead(button) == wiringpi.LOW:
            if index == 0:  # Assuming button 1 is for the 'button1' data
                l+=60;
                print('btn 1 pressed')
            elif index == 1:  # Assuming button 1 is for the 'button1' data
                l-=60;
                print('btn 2 pressed')
            # Add conditions for other buttons if needed
            time.sleep(0.5)  # Simple debounce
    return l

wiringpi.wiringPiSetup()
# Light up the LEDs
ledPin = 6;
ledTemp = 16;

wiringpi.pinMode(ledPin, 1)
wiringpi.pinMode(ledTemp, 1)

def lightUp(pin):
    wiringpi.digitalWrite(pin, 1)

def lightDown(pin):
    # wiringpi.wiringPiSetup()
```

```
    wiringpi.digitalWrite(pin, 0)

# ================== Loop To Render the values

while True:
    # Measure data
    lux = get_value(bus, address_bh170)
    # Measure temperature and pressure from BMP280
    temperature = bmp280.get_temperature()
    pressure = bmp280.get_pressure()

    # changing lux here with push button

    lux = changeLuxValue(changeValBtns, lux)

    if lux < 50:
        lightDown(ledPin)
    else:
        lightUp(ledPin)

    if temperature > 26:
        lightUp(ledTemp)
    else:
        lightDown(ledTemp)

    # Print each measured value on a separate line
    print("Temperature: {:.1f} C".format(temperature))
    # print("Pressure: {:.1f} hPa".format(pressure))
    print("Light: {:.2f} Lux".format(lux))
```

```
    print("---------------------------------------")


    MQTT_DATA =
"field1="+str(lux)+"&field2="+str(temperature)+"&field3="+str(pressure)+"&status=MQTTPUBLISH"
    client.publish(topic=MQTT_TOPIC, payload=MQTT_DATA, qos=0, retain=False, properties=None)


    time.sleep(0.5)
```

MQTT CODE EXPLANATION

```
# MQTT settings
MQTT_HOST ="mqtt3.thingspeak.com"
MQTT_PORT = 1883
MQTT_KEEPALIVE_INTERVAL =60
MQTT_TOPIC = "channels/2463672/publish"
MQTT_CLIENT_ID ="Bh4WNyg3LRAuADAaBiYkNxQ"
MQTT_USER ="Bh4WNyg3LRAuADAaBiYkNxQ"
MQTT_PWD = "0zDCWCrI+SRDwL0WqYXPHN0o"

# ========== MQTT error handling
def on_connect(client, userdata, flags, rc):
    if rc==0:
        print("Connected OK with result code "+str(rc))
    else:
        print("Bad connection with result code "+str(rc))

def on_disconnect(client, userdata, flags, rc=0):
    print("Disconnected result code "+str(rc))

def on_message(client,userdata,msg):
```

```
    print("Received a message on topic: " + msg.topic + "; message: " + msg.payload)


# Set up a MQTT Client

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, MQTT_CLIENT_ID)

client.username_pw_set(MQTT_USER, MQTT_PWD)

# Connect callback handlers to client

client.on_connect= on_connect

client.on_disconnect= on_disconnect

client.on_message= on_message

print("Attempting to connect to %s" % MQTT_HOST)

client.connect(MQTT_HOST, MQTT_PORT)

client.loop_start() #start the loop
```

This code sets up the MQTT connection settings to link with a broker. It defines functions to handle when the connection is made, broken, and when messages are received. An MQTT client is created, and its username and password are set. The client's callback functions are connected, and then the client tries to connect to the MQTT broker. Finally, the client's loop is started to keep the connection active.

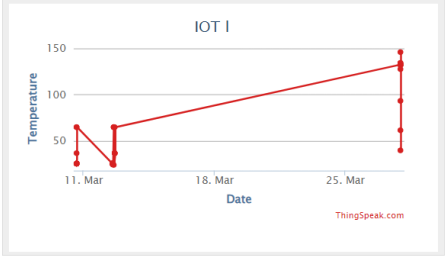
```
MQTT_DATA =
"field1="+str(lux)+"&field2="+str(temperature)+"&field3="+str(pressure)+"&status=MQTTPUBLISH"

    client.publish(topic=MQTT_TOPIC, payload=MQTT_DATA, qos=0, retain=False, properties=None)
```

This code shares sensor data, including light, temperature, and air pressure, along with a status message, to an MQTT topic. The data is formatted as a string with labels and values, then published with the lowest quality of service setting and without saving the message for future use.

Channel Stats

Created: 17 days ago
Last entry: about 2 hours ago
Entries: 177

Field 1 Chart



Field 2 Chart