

Anthony Chavez

EEE 174-CpE 185 Summer 2020

Monday, Wednesday

Lab 6

Data Logging and Garage Web Server

Dennis Dahlquist

Required Hardware:

- Raspberry Pi 4
- Micro SD Card with OS loaded
- Simple Keyboard that is linux compatible
- USB mouse
- HDMI Micro to HDMI
- Monitor
- USB C Power Cable
- Three LEDs (Red, Yellow, Green)
- Two Push Buttons
- Five Resistors (1K ohms)
- Nucleo Microcontroller (will be using L432KC model)
- Wires (male-to-male and female-to-male)

Required Software:

- STM32CubeMX software
- Atollic TrueSTUDIO for STM32 IDE
- Python3
- Flask Framework

Part 1: Setting Up Nucleo MicroController and BreadBoard

First, open the STM32CubeMX software, click on 'File', and select 'New Project'. Second, find the microcontroller you have, in this case we will be using the L432KC model, and click Start Project. Third, set pins PB1, PB3, and PB6 to GPIO_Output and pin PB5 to GPIO_Input. Fourth, open the GPIO panel, change the User Labels, and verify your configuration matches the image shown below.

The image shows the STM32CubeMX software interface. On the left, the 'GPIO Mode and Configuration' window is open, displaying a table of pin configurations. On the right, the 'Pinout view' shows the physical pinout of the STM32L432KC microcontroller with components connected to specific pins.

GPIO Configuration Table:

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/P	Maximum outp.	Fast Mode	User Label	Modified
PB1	n/a	Low	Output Push Pull No pull-up and ...	Low	n/a	n/a	Yellow_LED	<input checked="" type="checkbox"/>
PB3 (JTDO-TR...	n/a	Low	Output Push Pull No pull-up and ...	Low	n/a	n/a	Green_LED	<input checked="" type="checkbox"/>
PB5	n/a	n/a	Input mode	No pull-up and ...	n/a	n/a	Push_Button	<input checked="" type="checkbox"/>
PB6	n/a	Low	Output Push Pull No pull-up and ...	Low	Disable	Disable	Red_LED	<input checked="" type="checkbox"/>

Pinout Diagram:

The pinout diagram shows the STM32L432KC microcontroller with the following components connected:

- Red_LED connected to PB6
- Push_Button connected to PB5
- Green_LED connected to PB3
- Yellow_LED connected to PB1

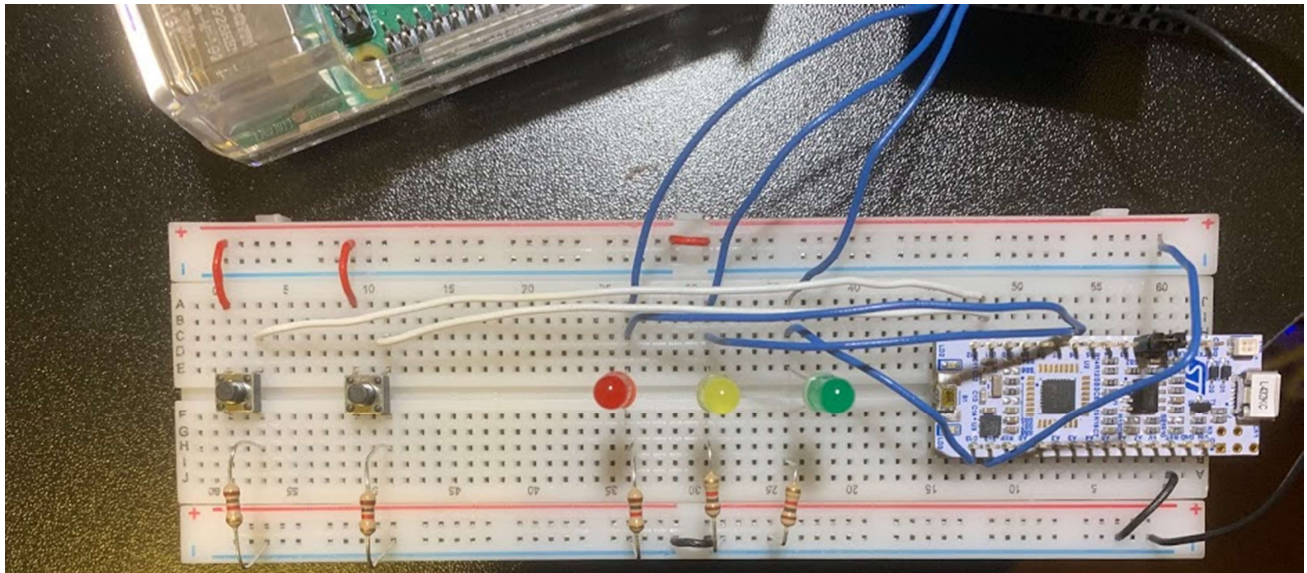
Next, click on 'Generate Code' and 'Open the Project'. Then, navigate to the main.c file under the Core folder in the Src folder. Now add the following code into the infinite while loop in the main() function, build the project, and upload the code to the board.

```

if(HAL_GPIO_ReadPin(GPIOB, Push_Button_Pin)) {
    HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_SET);           // Red LED "ON", signify door closed
    HAL_Delay(3000); // Delay 3 seconds
    HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_RESET);         // Red LED "OFF", signify door closed
    HAL_Delay(3000); // Delay 3 second
    HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_SET);         // Yellow LED "ON", signify door
opening/closing
    HAL_Delay(3000); // Delay 3 seconds
    HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_RESET);       // Yellow LED "OFF", signify door not
opening/closing
    HAL_Delay(3000); // Delay 3 seconds
    HAL_GPIO_WritePin(GPIOB, Green_LED_Pin, GPIO_PIN_SET);          // Green LED "ON", signify door opened
    HAL_Delay(3000); // Delay 3 seconds
    HAL_GPIO_WritePin(GPIOB, Green_LED_Pin, GPIO_PIN_RESET);        // Green LED "OFF", signify door no opened
    HAL_Delay(3000); // Delay 3 seconds
    HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_SET);         // Yellow LED "ON", signify door
opening/closing
    HAL_Delay(3000); // Delay 3 seconds
    HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_RESET);       // Yellow LED "OFF", signify door done
opening/closing
    HAL_Delay(3000); // Delay 3 seconds
    HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_SET);            // Red LED "ON", signify door closed
}
else{}

```

As for the breadboard, connect the wires to the nucleo as shown in the image below. The three blue wires connected to the LEDs and the black wire in the bottom, right corner are connected to the Raspberry Pi's GPIO. The Red LED is connected to BCM 17 (Board Pin 11), the Yellow LED is connect to BCM 27 (Board Pin 13), the Green LED is connected to BCM 22 (Board Pin 15), and the Black wire is connect to Ground (Board Pin 9).



Part 2: Python Script Logger

First, create a new folder and name it 'RPi_Logging'. Then, create a new file in a text editor of your choice, copy the code below, and save the file as 'log.py'.

```
import os
import RPi.GPIO as GPIO
import time
from datetime import datetime

logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt", "a")
logfile.write(datetime.now().strftime("  Program Starting -- %Y/%m/%d -- %H:%M -- Hello! \n"))
logfile.close()
print(datetime.now().strftime("  Program Starting -- %Y/%m/%d -- %H:%M -- Hello! \n"))

print(" Control + C to exit Program")

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Green LED
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Yellow LED
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Red LED
time.sleep(1)

try:
    while 1 >= 0:
        time.sleep(1)

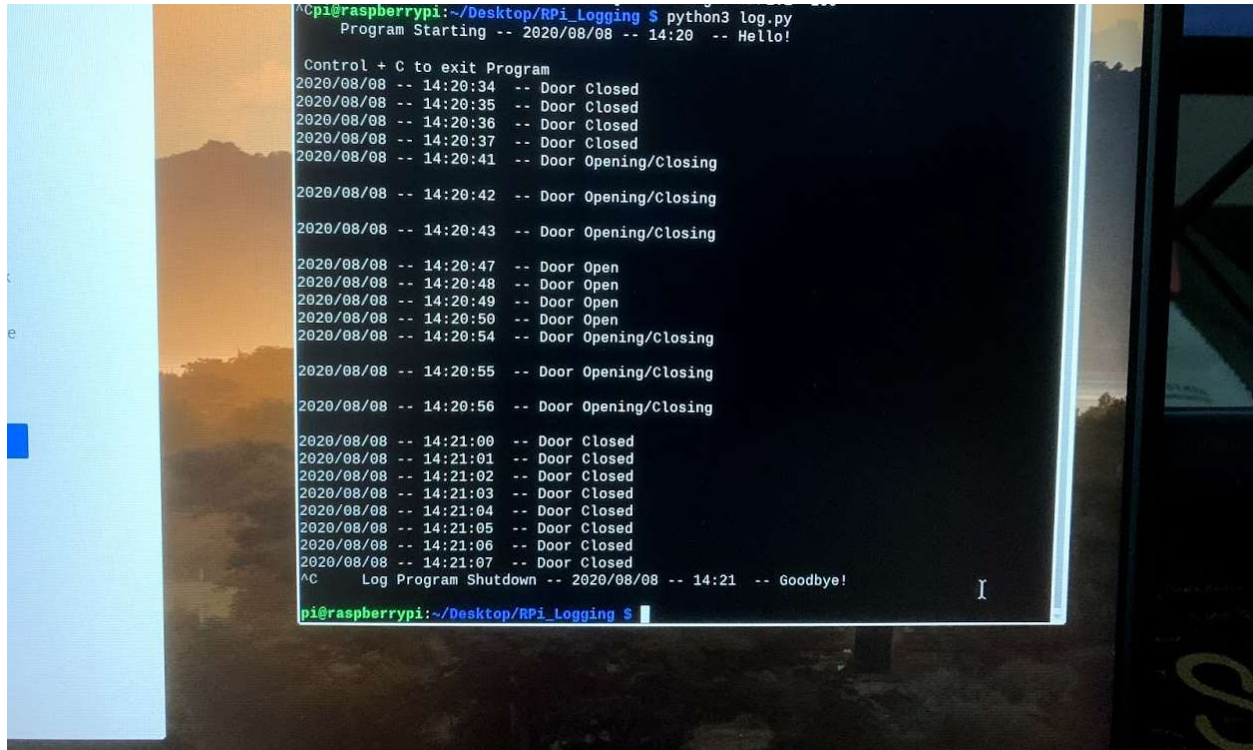
        if GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.HIGH and GPIO.input(22) == GPIO.LOW: # Door Closing/Opening
            logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt", "a")
            logfile.write(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S -- Door Opening/Closing \n"))
            logfile.close()
            print(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S -- Door Opening/Closing \n"))

        elif GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.HIGH: # Door is Closed
            logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt", "a")
            logfile.write(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S -- Door Closed \n"))
            logfile.close()
            print(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S -- Door Closed"))

        elif GPIO.input(17) == GPIO.HIGH and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.LOW: # Door is Open
            logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt", "a")
            logfile.write(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S -- Door Open \n"))
            logfile.close()
            print(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S -- Door Open"))

except KeyboardInterrupt: # Handle Ctrl + C
    logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt", "a")
    logfile.write(datetime.now().strftime("  Log Program Shutdown -- %Y/%m/%d -- %H:%M -- Goodbye! \n"))
    logfile.close()
    print(datetime.now().strftime("  Log Program Shutdown -- %Y/%m/%d -- %H:%M -- Goodbye! \n"))
    GPIO.cleanup()
```

To test if the 'log.py' script is working properly, make sure the nucleo has the created in Part 1 downloaded. Once the code is running in the terminal, press one of the push buttons to start the button sequence. The messages on the terminal should match the LED color it corresponds to on the breadboard. You should have a similar output shown in the image below.



```
pi@raspberrypi:~/Desktop/RPi_Logging $ python3 log.py
Program Starting -- 2020/08/08 -- 14:20 -- Hello!

Control + C to exit Program
2020/08/08 -- 14:20:34 -- Door Closed
2020/08/08 -- 14:20:35 -- Door Closed
2020/08/08 -- 14:20:36 -- Door Closed
2020/08/08 -- 14:20:37 -- Door Closed
2020/08/08 -- 14:20:41 -- Door Opening/Closing
2020/08/08 -- 14:20:42 -- Door Opening/Closing
2020/08/08 -- 14:20:43 -- Door Opening/Closing
2020/08/08 -- 14:20:47 -- Door Open
2020/08/08 -- 14:20:48 -- Door Open
2020/08/08 -- 14:20:49 -- Door Open
2020/08/08 -- 14:20:50 -- Door Open
2020/08/08 -- 14:20:54 -- Door Opening/Closing
2020/08/08 -- 14:20:55 -- Door Opening/Closing
2020/08/08 -- 14:20:56 -- Door Opening/Closing
2020/08/08 -- 14:21:00 -- Door Closed
2020/08/08 -- 14:21:01 -- Door Closed
2020/08/08 -- 14:21:02 -- Door Closed
2020/08/08 -- 14:21:03 -- Door Closed
2020/08/08 -- 14:21:04 -- Door Closed
2020/08/08 -- 14:21:05 -- Door Closed
2020/08/08 -- 14:21:06 -- Door Closed
2020/08/08 -- 14:21:07 -- Door Closed
^C Log Program Shutdown -- 2020/08/08 -- 14:21 -- Goodbye!
pi@raspberrypi:~/Desktop/RPi_Logging $
```

Part 2: Python Webserver Using Flask

Create a new file in a text editor of your choice, copy the code below, and save the file as 'GarageWebServer.py' in the same folder as 'log.py'.

```
import time
from datetime import datetime
from flask import Flask, render_template, request

import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(17, GPIO.IN, GPIO.PUD_DOWN) # set up pin 17 as an input with a pull-down resistor
GPIO.setup(27, GPIO.IN, GPIO.PUD_DOWN) # set up pin 27 as an input with a pull-down resistor
GPIO.setup(22, GPIO.IN, GPIO.PUD_DOWN) # set up pin 22 as an input with a pull-down resistor

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    if GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.HIGH:
        print ("Garage is Closed")
        return app.send_static_file('Closed.html')
    elif GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.HIGH and GPIO.input(22) == GPIO.LOW:
        print("Garage is Opening/Closing")
        return app.send_static_file('Closing-Opening.html')
    elif GPIO.input(17) == GPIO.HIGH and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.LOW:
        print ("Garage is Open")
        return app.send_static_file('Open.html')

@app.route('/stylesheet.css')
def stylesheet():
    return app.send_static_file('stylesheet.css')

@app.route('/Log')
def logfile():
    return app.send_static_file('log.txt')

@app.route('/images/<picture>')
def images(picture):
    return app.send_static_file('images/' + picture)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Now we need to set up our static html files. First, create a new folder in our current RPi_Logging folder, name it 'static', create another folder and name it 'images'. We will be using the following images below to represent the status of the garage door on the webserver. Save them into the images folder as Closed.jpg, Closing-Opening.jpg, Opened.jpg (from left to right).



Second, create a html file for each state, 'Closed.html', 'Closing-Opening.html', and 'Open.html' under the static folder. Copy the following codes for the static html files.

```
<html><head>
<meta name="viewport" content="width=device-width">
<link rel="stylesheet" type="text/css" href="stylesheet.css">

</head>
<body bgcolor="red">
<center>
<br>

<br><br><a href="/Log">Click/Tap to see garage door status</a><br><br>
</center>
</body></html>
```

```
<html><head>
<meta name="viewport" content="width=device-width">
<link rel="stylesheet" type="text/css" href="stylesheet.css">

</head>
<body bgcolor="orange">
<center>
<br>

<br><br><a href="/Log">Click/Tap to see garage door status</a><br><br>
</center>
</body></html>
```



```
<html><head>
<meta name="viewport" content="width=device-width">
<link rel="stylesheet" type="text/css" href="stylesheet.css">

</head>
<body bgcolor="green">
<center>
<br>

<br><br><a href="/Log">Click/Tap to see garage door status</a><br><br>
</center>
</body></html>
```

Finally, create a 'stylesheet.css' file and copy the code below and save it under the static folder.

```

#container {
margin-left: auto;
margin-right: auto;
}

#backtomain {
vertical-align: text-middle;
text-align: center;
float: left;
width: 40px;
height: 40px;
background-color: #88B655;
-moz-border-radius: 15px;
-webkit-border-radius: 15px;
padding: 5px;
}

table, td {
border: 0px;
padding: 0px;
border-spacing: 0px;
}

body {
cursor: crosshair;
margin-left: 0pt;
margin-left: 0px;
margin-right: 0px;
margin-top: 0px;
margin-bottom: 0px;
font-family: Trebuchet MS, Verdana, Arial, Helvetica, sans-serif;
font-size: 18px;
}

a:link {font-family: Arial; font-size: 9pt; font-weight: bold; color: #1D1D1D; text-decoration: none;}
a:visited {font-family: Arial; font-size: 9pt; font-weight: bold; color: #1D1D1D; text-decoration: none;}
a:active {font-family: Arial; font-size: 9pt; font-weight: bold; color: #1D1D1D; text-decoration: none;}
a:hover {font-family: Arial; font-size: 9pt; font-weight: bold; color: #6E6E6E; text-decoration: none; cursor: hand;}

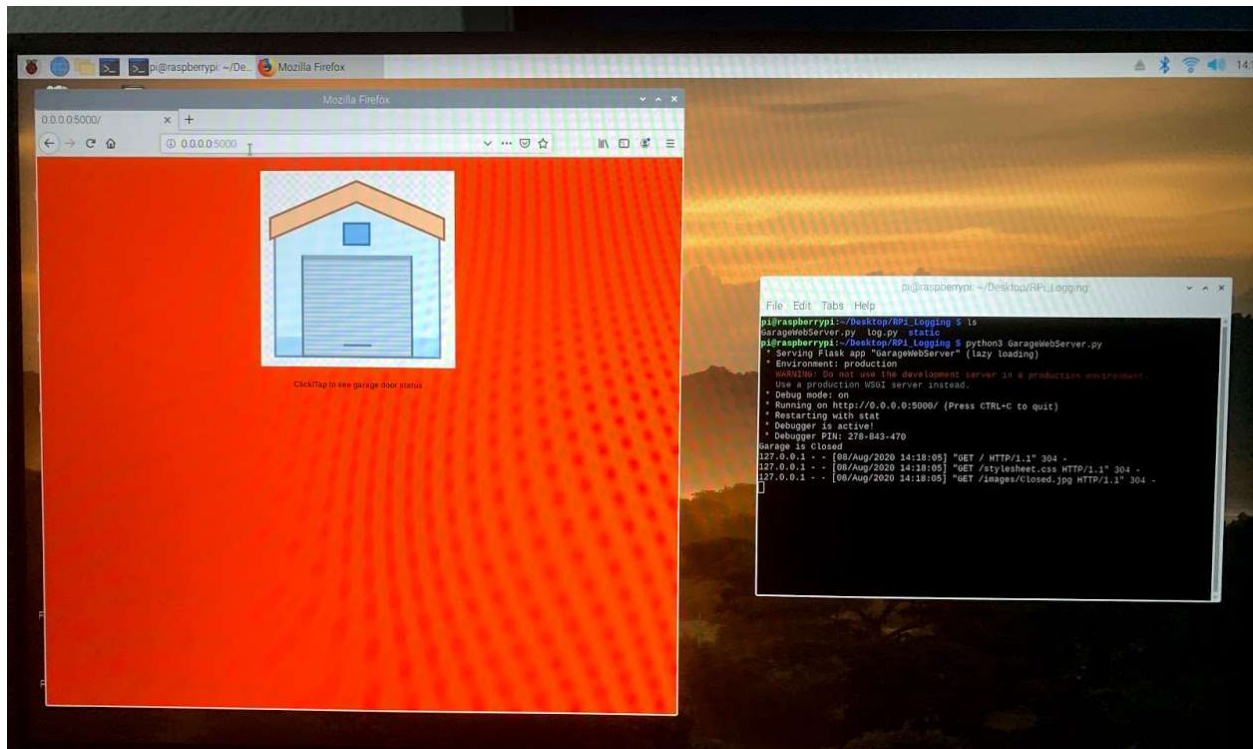
container {
width: 399px;
border: 1px solid black;
display: inline-block;
}

block {
display: inline-block;
width: 123px;
height: 100px;
margin: 5px;
background: #6c6969;
float: right;
font-size: 35px;
}

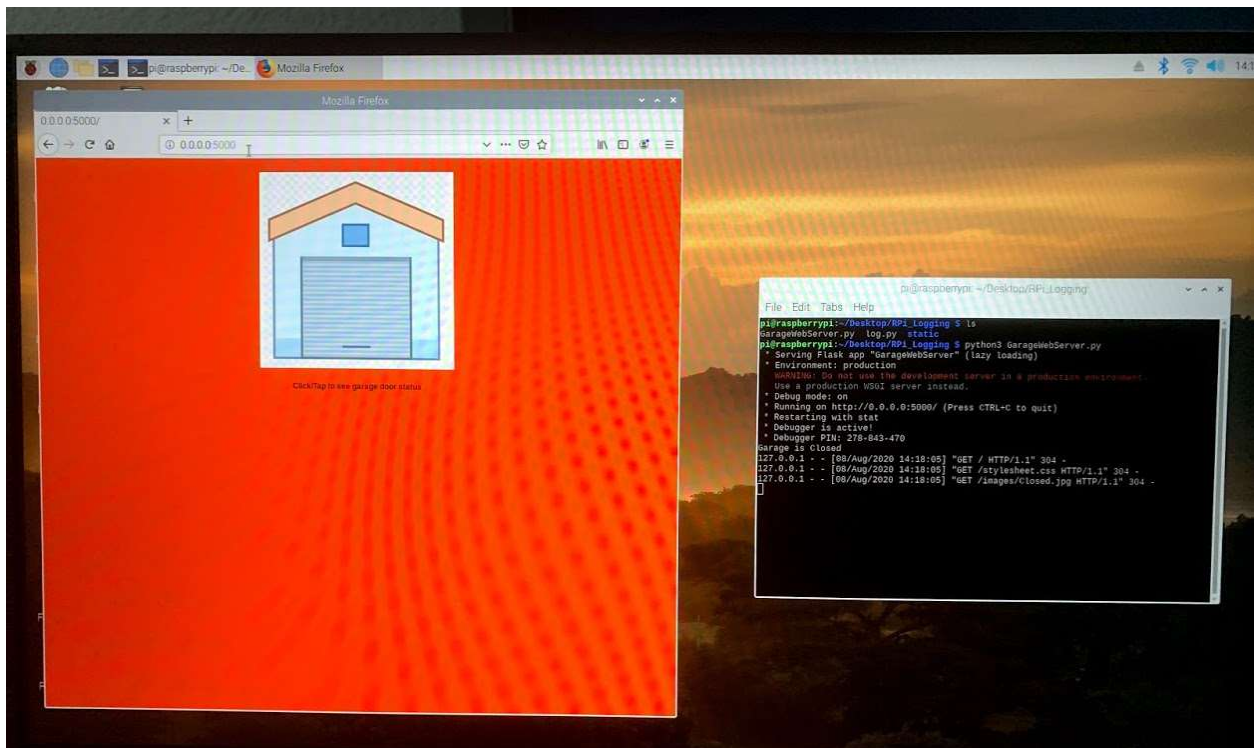
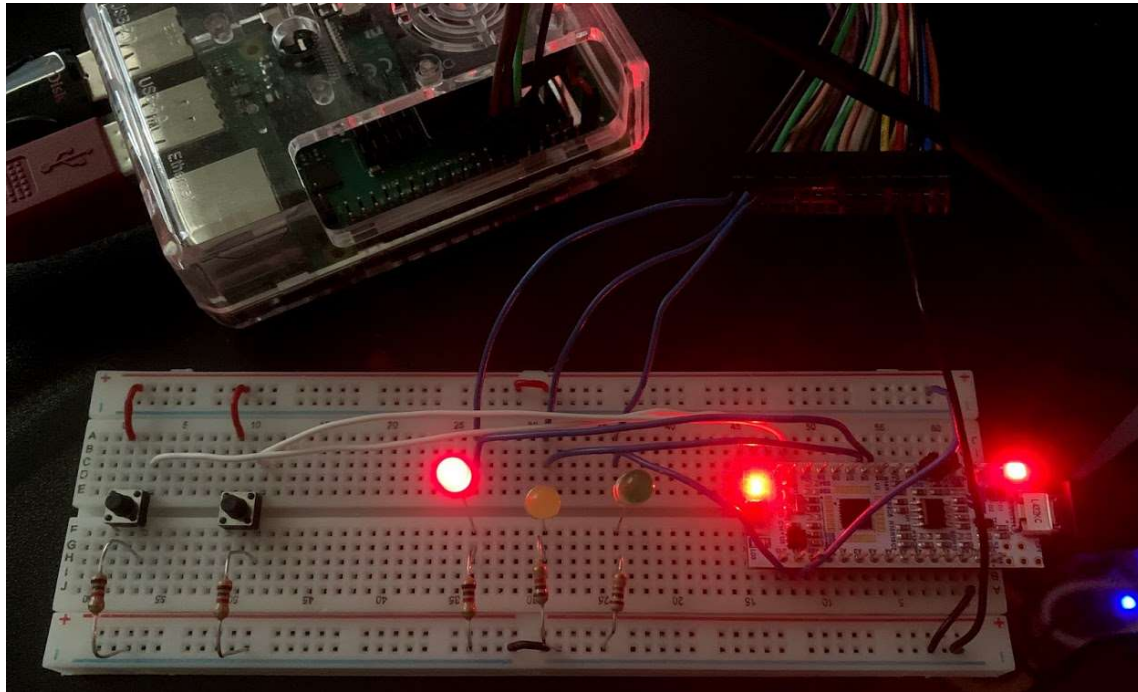
block[h2] {
width: 256px;
}

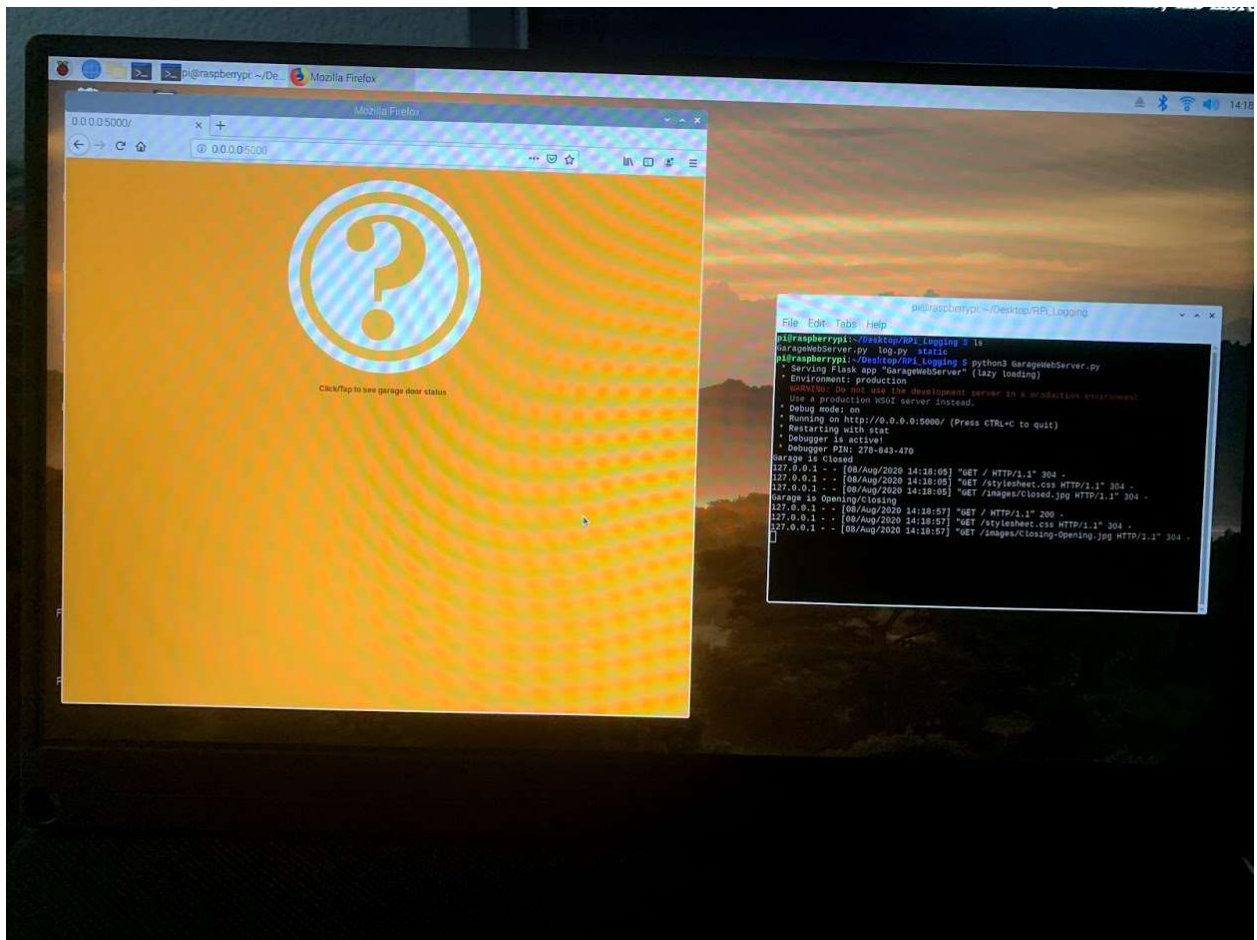
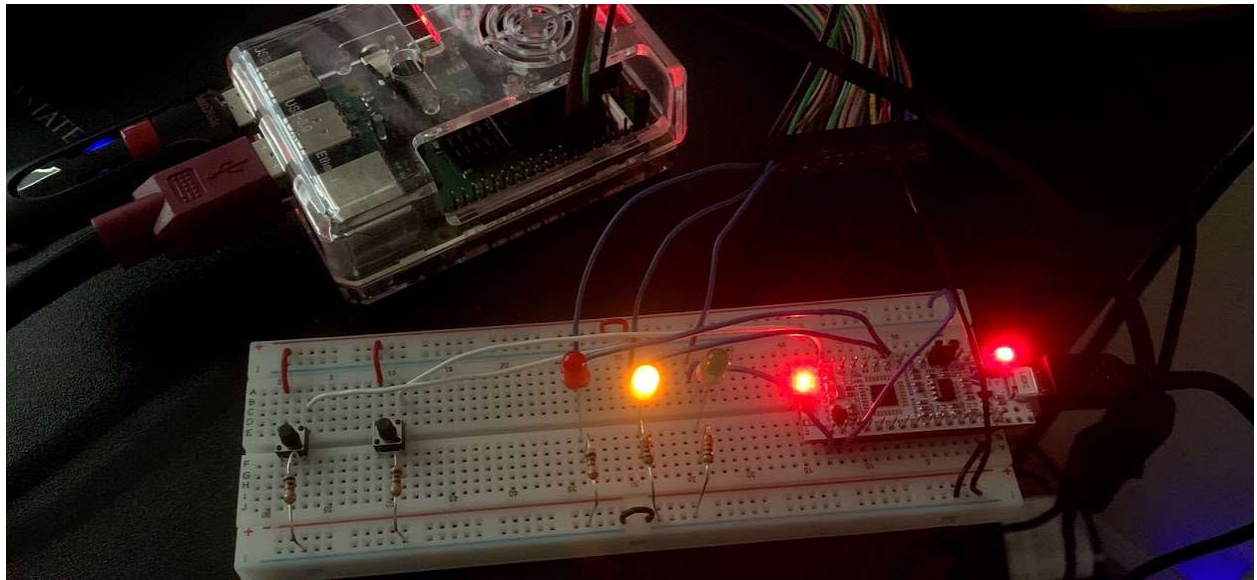
```

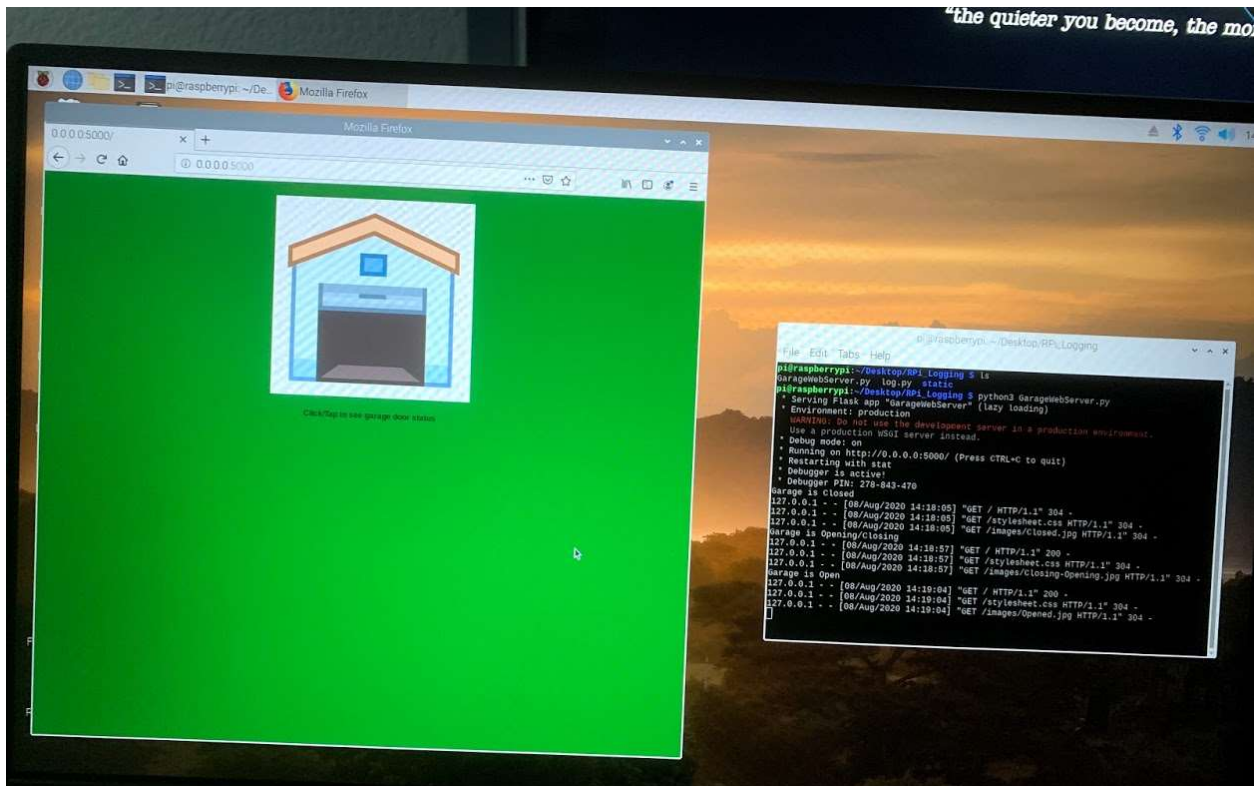
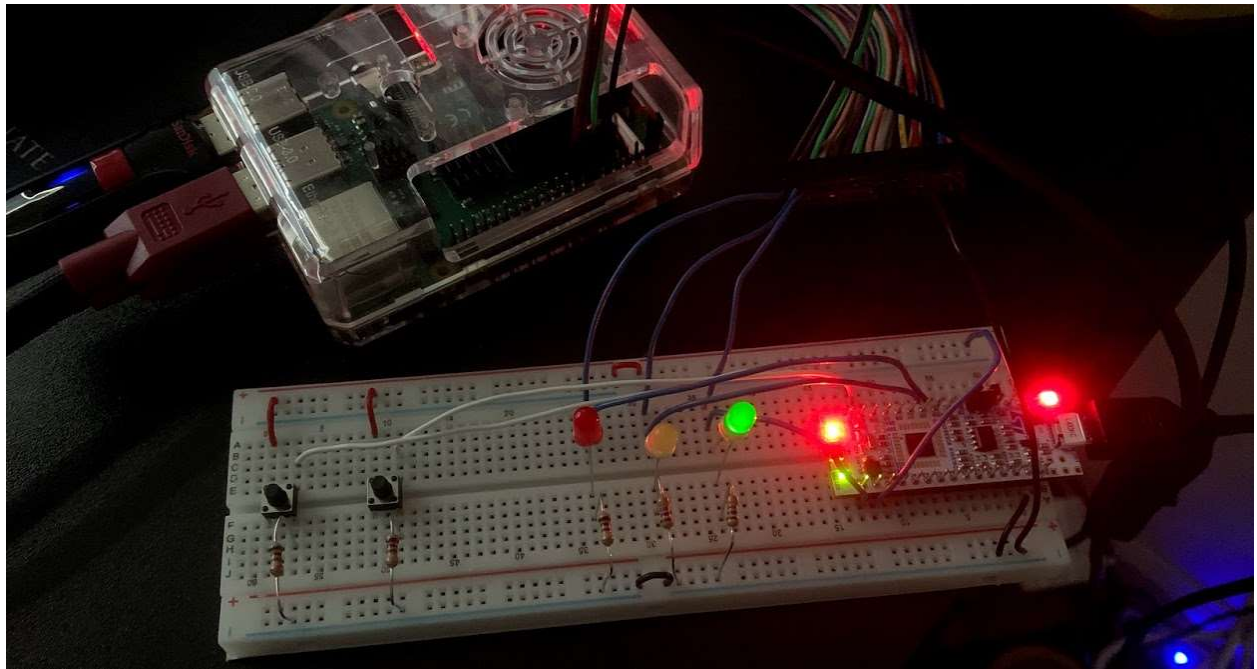
Now that everything is set up, we can run the GarageWebServer.py program to host the webserver. We will be using Firefox to prevent any cache errors which Chrome is known to have. In the address bar, enter <http://0.0.0.0:5000/> and you should have a screen like the image below.



To test the webserver, run the nucleo code by pushing one of the push buttons and hit the refresh button (F5) on the keyboard to see the status of the garage door in the browser when one of the LEDs is on. Also, by clicking on the hyper link, "Click/tap to see garage door status" will open the log.txt file created by the log.py program we made earlier in the browser. You can run this in a separate terminal to keep the log updated. See the following images for correct output. NOTE: You can view the webserver from a phone, but you must open the 5000 port on your router which is not covered in this lab.







References:

<https://github.com/shrocky2/GarageWeb>

<https://pinout.xyz/#>

<https://os.mbed.com/platforms/ST-Nucleo-L432KC/>

https://github.com/Mushuooshu/RPi_Logging, my repository with all code shown in this lab report