# Week 2 Introductory projects on Raspberry Pi

## Distance measuring ultrasonic sensor using Raspberry Pi

### Introduction

The objective of this assignment is to build a circuit using the HC-SR04 Ultrasonic Sensor to build a circuit which uses the GPIO pins of the Raspberry Pi to be able to detect the distance between the sensor and the object. The HC-SR04 Ultrasonic Sensor is marketed as a Ranging Module as it can be accurately used for measuring distances in the range of 2cm to 400cm with an accuracy of 3mm (Gus, 2019).

The range of this Ultrasonic Sensor seems very less but it is sufficient for the applications it is implemented in i.e. Proximity Detection and Obstacle Avoiding, for example. The output of the HC-SR04 is at 5V, but Raspberry Pi only accepts an input of 3.3V so the circuit must be thoughtfully planned out to avoid damage to the GPIO of the Raspberry Pi. The HC-SR04 Ultrasonic Sensor, works on the principle that is similar to RADAR and SONAR i.e. transmits a signal and analyzes the target by capturing the reflected signals.

The HC-SR04 consists of three parts: an ultrasonic transmitter, a control circuit and an ultrasonic receiver. Coming to the pins of the HC-SR04 Sensor, it has only four pins namely VCC, TRIG (Trigger), ECHO (Echo) and GND. The Ultrasonic Transmitter in the Sensor generates a 40 KHz Ultrasound (Gus, 2019). This signal then propagates through air and if there is any obstacle in its path, the signal hits the object and bounces back. This bounced signal is then collected by the Ultrasonic Receiver. Based on the signal’s time of travel, you can calculate the distance of the object as you already know the speed of sound.

To transmit the ultrasonic sound, the TRIG pin of the sensor has to be HIGH for 10µs (Gus, 2019). After this, the Ultrasonic Transmitter, will transmit a burst of 8-pulses of ultrasound at 40 KHz. Immediately, the control circuit in the sensor will change the state of the ECHO pin to HIGH. This pin stays HIGH until the ultrasound hits an object and returns to the Ultrasonic Receiver.

Since the speed of sound is 340m/s. The distance can be calculated by using the formula below. One thing to note is the time returned by the sensor has to be divided by 2 because that is the time for the ultrasonic wave to travel to and fro, which is twice the distance between the object and the sensor.



Figure: Formula used to calculate distance between object and sensor

### Discussion

First, we have to start by interfacing the sensor into our circuit. The plan is to use a pushbutton to issue an input, which then triggers the ultrasonic sensor to start measuring. For the first step we would build our circuit first. We connect the VCC to the 5V output pin #2 of the Raspberry Pi. Next, we have to connect the TRIG pin to GPIO pin #36 on the board. This GPIO pin #36 will output the signal to trigger the sensor. Next, ECHO pin is connected to GPIO pin #35, but since the sensor outputs a signal of 5V, we have to use a voltage divider of two resistors, 1㏀ and 2㏀ respectively to step down the voltage to 2.5V approximately. The GPIO pi #35 will receive the input from ECHO, then it is able to stop the sensor and go on calculating the distance. The last pin GND is connected to GROUND pin #6. Thus our HC-SR04 is successfully connected to the Raspberry Pi.

After that, a push button is connected to GPIO pin #40 and GROUND pin #6 with a capacitor in between to eliminate debouncing. Inside the push button is made up of two metal contacts. When a pushbutton or any switch's position is changed, noise is generated. The noise will cause extra “button presses” to be registered when the button is only pressed once. Some noise (contact) occurs because the switch contact is metal and it has elasticity. When the switch is moved to a new position it strikes a metal contact and physically bounces a few times. We call this contact bounce.

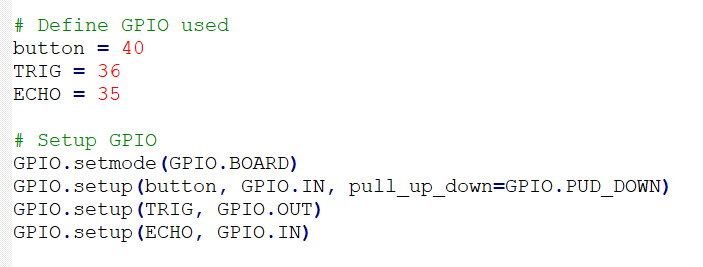


Figure: Setup the GPIO pins

To control the GPIO pins, we will be using Thonny Python IDE in the Raspberry Pi to write our program. First, we have to define the GPIO pins used so it is easier to program the pins later. GPIO.setmode( GPIO.BOARD) allows the program to define the pin according to the physical placement on the board rather than their respective functions. The button pin is set up as an input and is initially set as DOWN or LOW. The TRIG and ECHO pins are also set up before being used as output pin and input pin respectively.

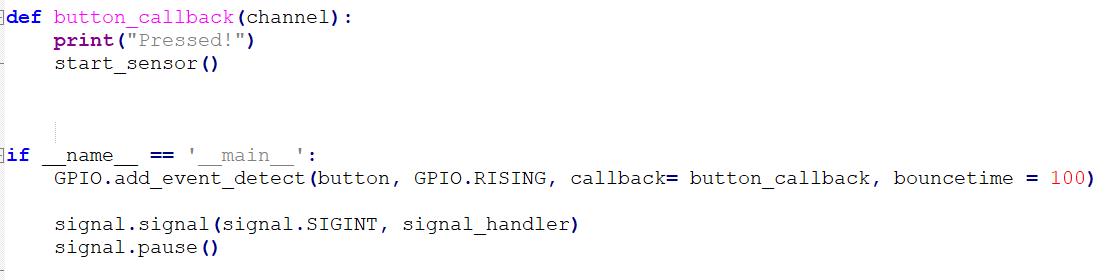


Figure : Code snippet for the button interrupt to work

First, button\_callback function is defined so that when the interrupt is triggered, this is the function that would run. The function prints “Pressed” and starts measuring distance using the ultrasonic sensor. In the main function, the interrupt is added by using the function GPIO.add\_event\_detect, which in this case is used to detect a RISING signal from the button pin. This signal can be detected when the button is pressed, when the button pin input is changed from LOW to HIGH. When this RISING signal is triggered, the function button\_callback will be called. The bouncetime also aids in debouncing as it waits for 100 mS before triggering the interrupt.

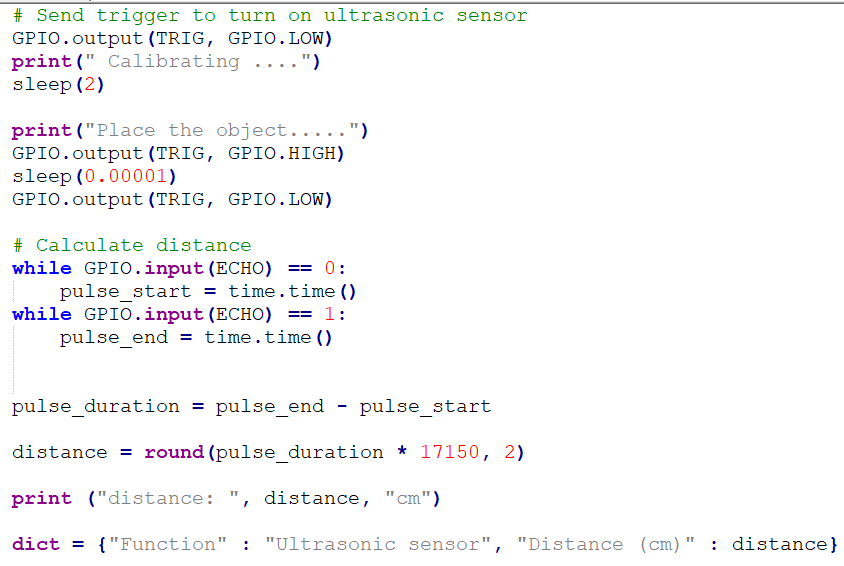


Figure : Code snippet to use the ultrasonic sensor

The trigger is placed as low and is paused for 2 seconds to give some time for the sensor to calibrate. During that time the object should be placed in front of the sensor. Then, the TRIG pin is set to HIGH for 10µS to trigger the sensor to transmit the ultrasonic wave. The program waits until the ECHO pin receives an HIGH input from the sensor to signal that the distance has already measured. The time taken for the ultrasonic wave to travel to and fro is the time when the distance has been measured, subtracting the time when the transmission signal is sent out. THe distance is calculated using the formula in Figure . Finally, the distance is printed.

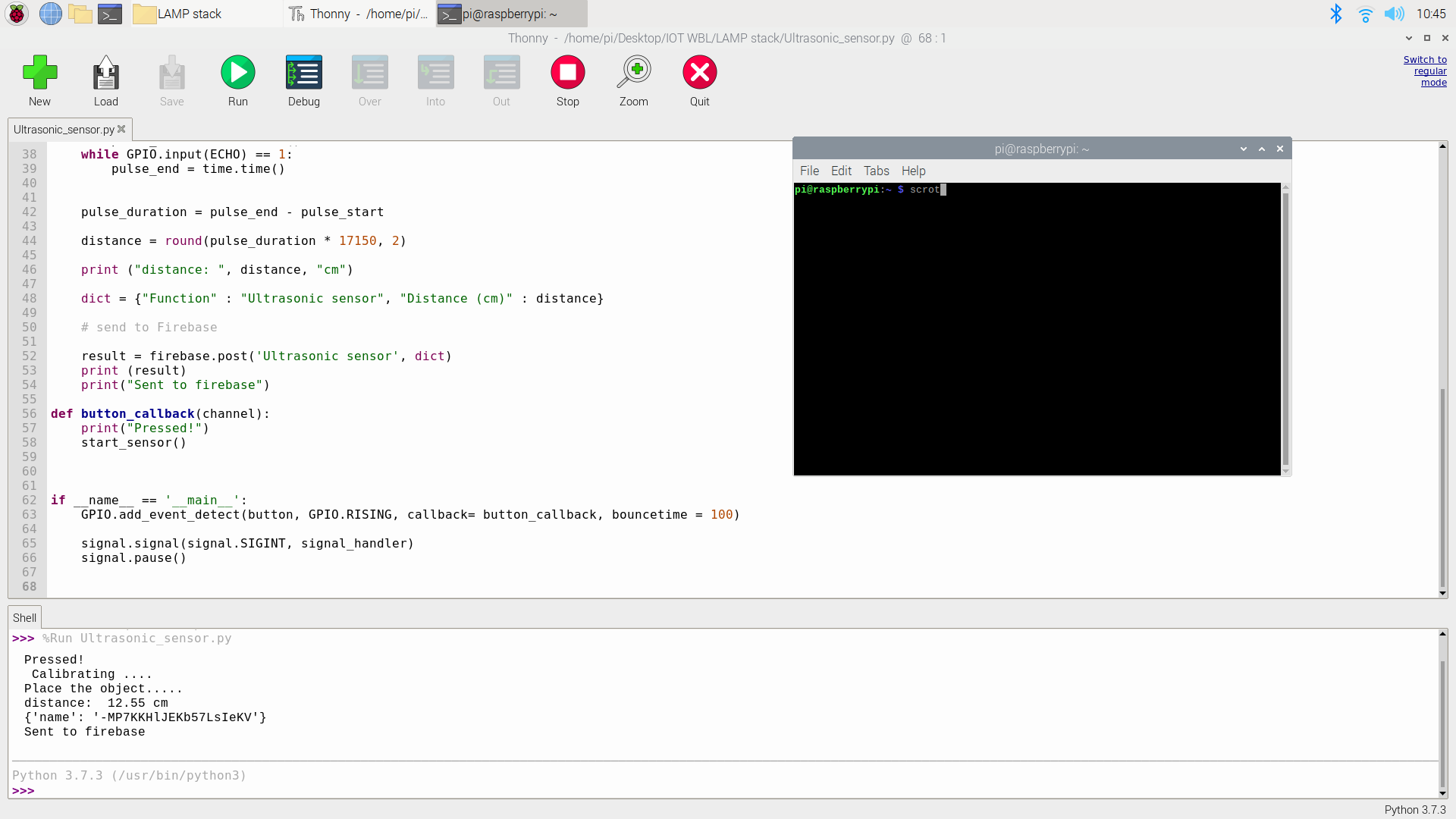


Figure : The result

### Reference

1. Gus. (2019). Raspberry Pi Distance Sensor using the HC-SR04. Retrieved from <https://pimylifeup.com/raspberry-pi-distance-sensor/>.

# Week 3 IoT Projects on Raspberry Pi Pt. 1

## REST/ HTTP communication using LAMP in Raspberry Pi

### Introduction

The objective of the assignment is to set up Raspberry to act as host server in a LAMP stack, and upload sensor data in JSON format, Display data over the HTML page, and understand the Client-Server relation via this assignment.

A LAMP Stack is a set of open-source software that can be used to create websites and web applications. LAMP is an acronym, and these stacks typically consist of the Linux operating system, the Apache HTTP Server, the MySQL relational database management system, and the PHP programming language.

The operating system (OS) makes up our first layer. Linux sets the foundation for the stack model. All other layers run on top of this layer. Linux is the lowest-level layer and provides the operating system. Linux actually runs each of the other components (“IBM”, 2019).

The second layer consists of web server software, typically Apache Web Server. This layer resides on top of the Linux layer. Web servers are responsible for translating from web browsers to their correct website. Apache processes requests and serves up web assets via HTTP so that the application is accessible to anyone in the public domain over a simple web URL. Apache provides the mechanics for getting a Web page to a user. Apache is a stable, mission-critical-capable server, and it runs more than 65 percent of all Web sites on the Internet. The PHP component actually sits inside Apache, and you use Apache and PHP together to create your dynamic pages (“IBM”, 2019).

Our third layer is where databases live. MySQL stores details that can be queried by scripting to construct a website. MySQL usually sits on top of the Linux layer alongside Apache/layer 2. In high-end configurations, MySQL can be offloaded to a separate host server. MySQL provides the data-storage side of the LAMP system. With MySQL, you have access to a very capable database suitable for running large and complex sites. Within your Web application, all your data, products, accounts, and other types of information will reside in this database in a format that you can easily query with the SQL language. However since in this assignment the main aim is to use HTTP requests, thus a database is not set up for this assignment (“IBM”, 2019).

Sitting on top of them all is our fourth and final layer. The scripting layer consists of PHP and/or other similar web programming languages. Websites and Web Applications run within this layer. PHP is a simple and efficient programming language that provides the glue for all the other parts of the LAMP system. You use PHP to write dynamic content capable of accessing the data in the MySQL database and some of the features that Linux provides (“IBM”, 2019).

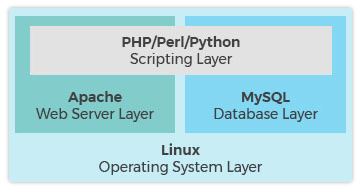


Figure : Overview of the LAMP stack

To implement this used a push button counter instead of the ultrasonic sensor because the sensor spoilt when I was doing this assignment. The idea is to run a program on Python which is able to count the number of times I pressed the push button, then uploads that data in JSON format to the apache web server using HTTP POST. Since Raspberry Pi already runs on Raspberry Pi OS which is based on Linux, we just have to install the Apache server on the Raspberry Pi using the command in Terminal ( Figure ).

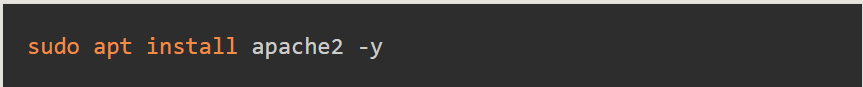


Figure : Command in terminal to install apache server

After installing the apache server, we can just type in <http://localhost/> or <http://127.0.0.1> in the browser in the Raspberry Pi to access the html file in the **/var/www/html/index.html** directory. Next, we have to install PHP into our Raspberry Pi to allow your Apache server to process PHP files.



Figure : Command in terminal to install latest version of PHP

We can delete the index.html file in **/var/www/html/index.html** and replace it with index.php and we can write our script in PHP to display on the browser. The script file can be viewed in the browser of any device connecting to the same WiFi network as the Raspberry Pi by entering <http://raspberrypi.local> in the browser.

### Discussion

First, I learned some basic PHP through some tutorial videos on youtube and LinkedIn Learning. After that I was able to write the code in index.php to receive the push button count data in JSON format through HTTP POST from the python program. Below is the python program and the index.php script.

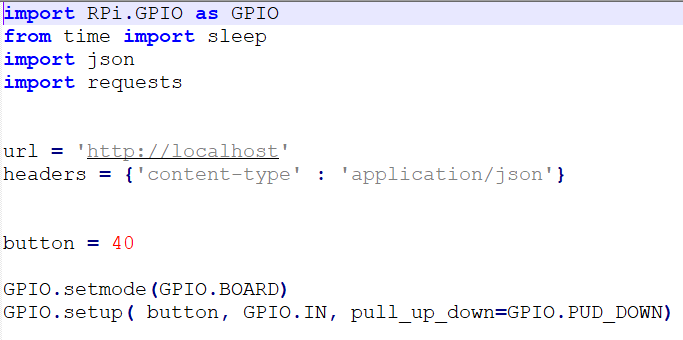


Figure : Code snippet to import library and define GPIO, URL

First, we have to import the RPi.GPIO library for the GPIOs to work, the time library for time control, json library to convert our data into JSON format and requests library to send or receive HTTP requests. Below we define our URL as <http://localhost> which is the URL to our index.php file. We define pin #40 as button pin as it is connected to our push button. And lastly we setup the GPIO button pin as initially LOW.

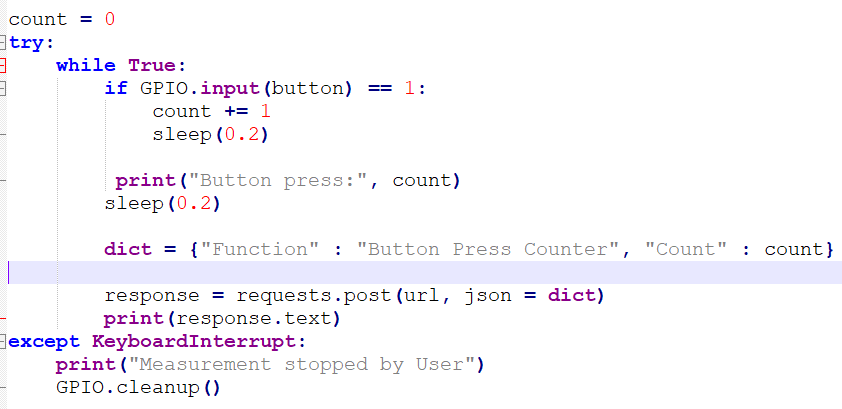


Figure : Code snippet for push button count

When the push button is pressed, the button pin becomes HIGH, and this increases the variable count by 1. The dict variable creates a dictionary which lists the function of the program and the variable count. Then, the program sends a HTTP POST request to the apache web server on the Raspberry Pi with the data in the dict variable as JSON format. The response from the server is sent back to the program and is printed out. The program will keep looping until CTRL + C is pressed to interrupt the loop. The final result is shown below.

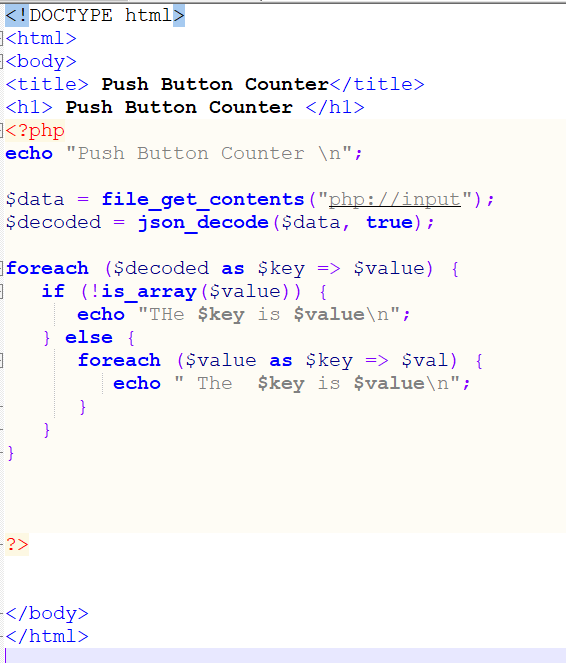


Figure : index.php file

The **data** variable uses the function **file\_get\_contents** to save the JSON data from the python program. After that, the function **json\_decode** decodes the data from JSON format into an array format which can be displayed and used in the PHP program. Each key and value pair in the array will be sent back as a response to the client, which is the python program.

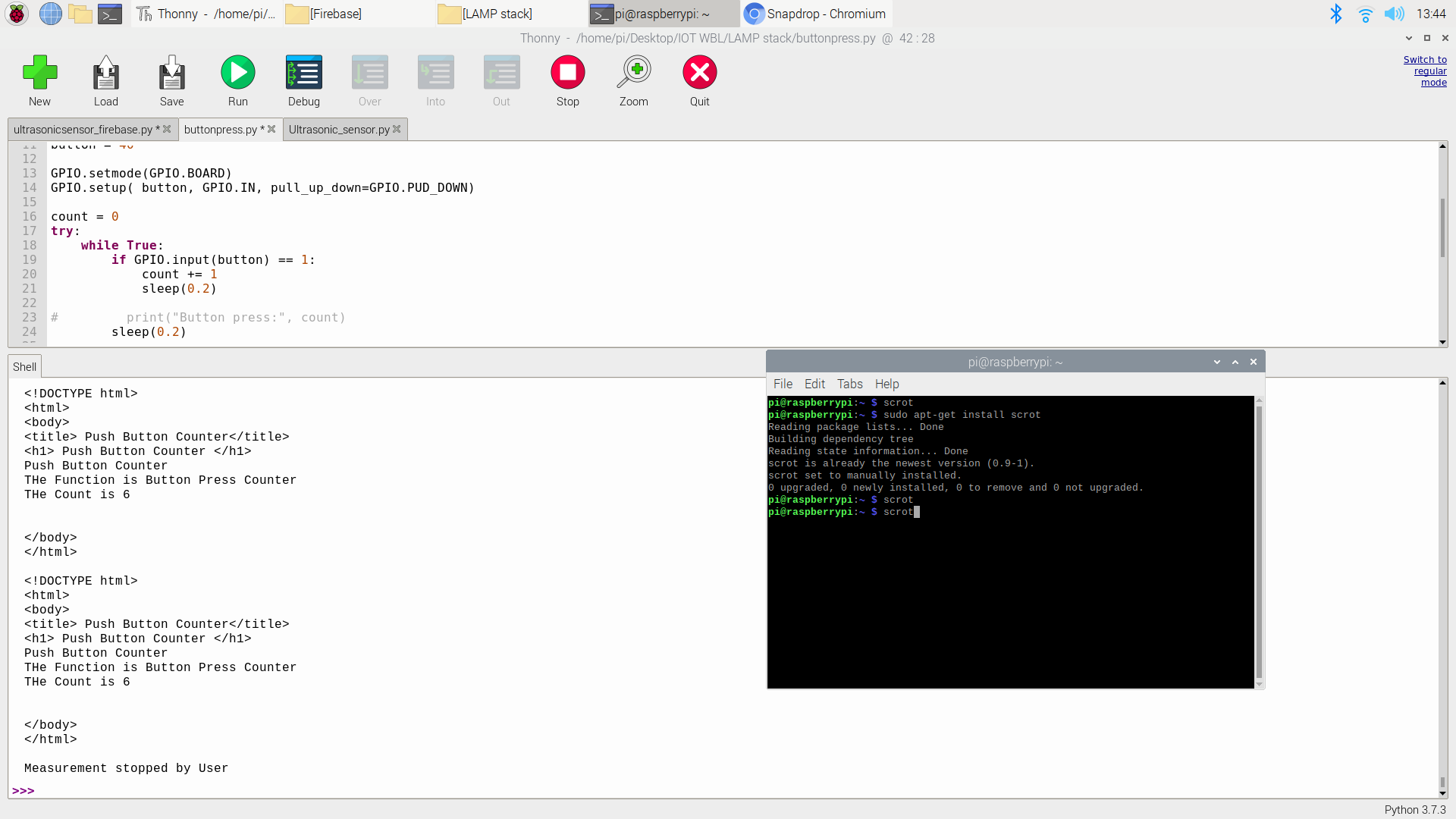


Figure : Output of the program

Therefore, the python program, the client, successfully sent a HTTP POST request and the apache server successfully responds to the client.

Reference

1. “IBM”. (2019). LAMP Stack. Retrieved from https://www.ibm.com/cloud/learn/lamp-stack-explained.

# Week 4 IoT Projects on Raspberry Pi Pt. 2

## MQTT application using Raspberry Pi and PC

### Introduction

MQTT or MQ Telemetry Transport is a lightweight messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information. MQTT uses a publish-subscribe way to communicate. A centralized broker or server is the centrepoint where clients send and receive messages to each other. Clients can both publish or subscribe to message channels called topics, any data that passes on through the broker will be tagged with a topic label. For example, if a smart home uses a smart thermosensor, the temperature data can be stored under several topic levels, such as **home/living room/temperature**. It is very simple as the clients just have to point to the broker’s IP address and no other system configuration has to be set up (Cook, 2019). It is usually used for machine-to-machine communication. Each client does not have any info about other clients on the network. The broker just takes care of data distribution.

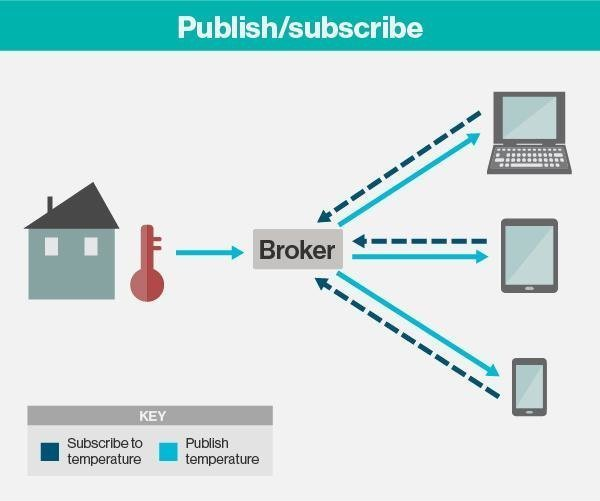


Figure: General concept overview of MQTT

For this assignment we will be using the ultrasonic sensor program from before and apply MQTT communication. The idea is to start a Mosquitto MQTT broker on the Raspberry Pi, then the Raspberry Pi and my Windows PC will both act as clients. The Raspberry Pi would publish the distance data to the broker, then my PC would subscribe to the same topic and will receive the data sent by the Raspberry Pi.

### Discussion

First, we have to set up our mosquitto MQTT broker on our Raspberry Pi. First, we have to go to the terminal and enter the following command.



Figure : Command to install mosquitto and mosquitto clients for Raspberry Pi

When Mosquitto is successfully installed on the Raspberry Pi, it automatically runs, even when the Raspberry Pi reboots. To connect to the mosquitto broker we require the ip address of the Raspberry Pi which can be found by entering the following command in the terminal.



Figure : IP address of Raspberry Pi will be returned

Once we set up the broker and know the IP address of the Raspberry Pi, we can proceed to setting up the MQTT clients on Windows and Raspberry Pi. I will be using Python to set up both clients, and I will be using PyCharm on PC, and Thonny Python IDE on the Raspberry Pi. First, we will set up the publishing client which is the Raspberry Pi.

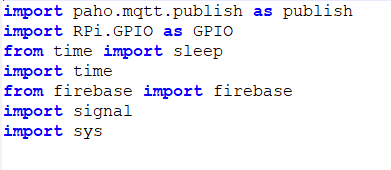


Figure : Import paho mqtt library ( Raspberry Pi)

Paho MQTT library is the most used python MQTT library, thus we will import paho.mqtt.publish to use MQTT publish functions in our python program.

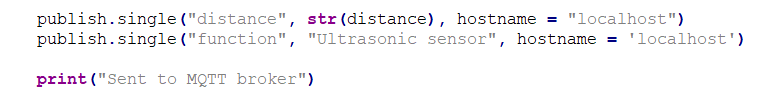


Figure : Code snippet to publish message to the broker

In this code snippet, we will publish a message to the broker using the publish.single function. The first parameter of this function is the message topic. The second parameter is the message payload, which is the contents of the message we want to send to the broker. The last parameter states the IP address of the broker, which is localhost since the client and broker runs on the same device. The first line publishes the distance data with the topic label “distance”. The second line publishes the function of the program with a topic tag “function”. The rest of the python program is exactly the same as the ultrasonic sensor python program. Next, we will look at the subscribing client using a PC to see how a message can be received using MQTT.

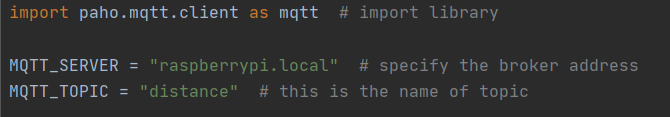


Figure : Code Snippet 1 (Subscribing Client PC)

For the subscribing client, we have to import paho.mqtt.client so that we are able to use it’s functions to receive MQTT messages. Here, raspberrypi.local gives the IP address of the Raspberry Pi to other devices as long as they are on the same WiFi connection. If a different WiFi connection is used, entering the IP address of the Raspberry Pi also works the same here. I also defined the topic “distance” to MQTT\_TOPIC so that it is easier to use later. In this case I want to receive distance data.

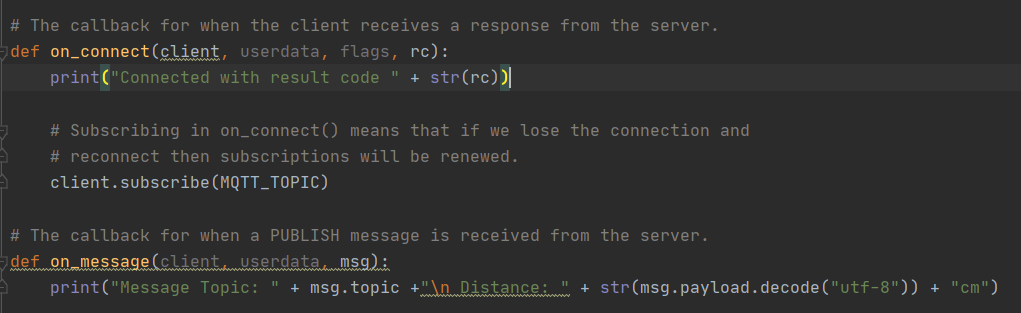


Figure : Code Snippet 2 (Subscribing Client PC)

First, we have to define two functions. The on\_connect function will be called when a response is received from the server, then it will print the result code. A result code of 0 means the broker and the client successfully connected to each other. After successfully connecting, this client will subscribe to the message topic defined above. Next, the on\_message will be called when a PUBLISH message is received from the server. This function will display the message topic and the message payload in the program. The message payload has to be decoded since MQTT automatically encodes the message payload in UTF-8.

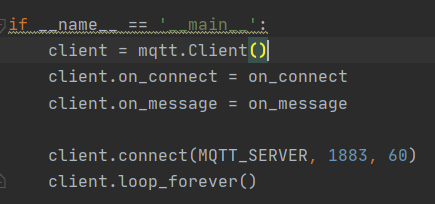


Figure : Code Snippet 3 (Subscribing Client PC)

A new client instance is created with the function **mqtt.Client**. The client.connect function connects to the MQTT broker with the IP address, port = 1883, and keeps alive for 60 seconds. The client will keep looping so that when a message is sent, it will receive in real time. The results are shown below.

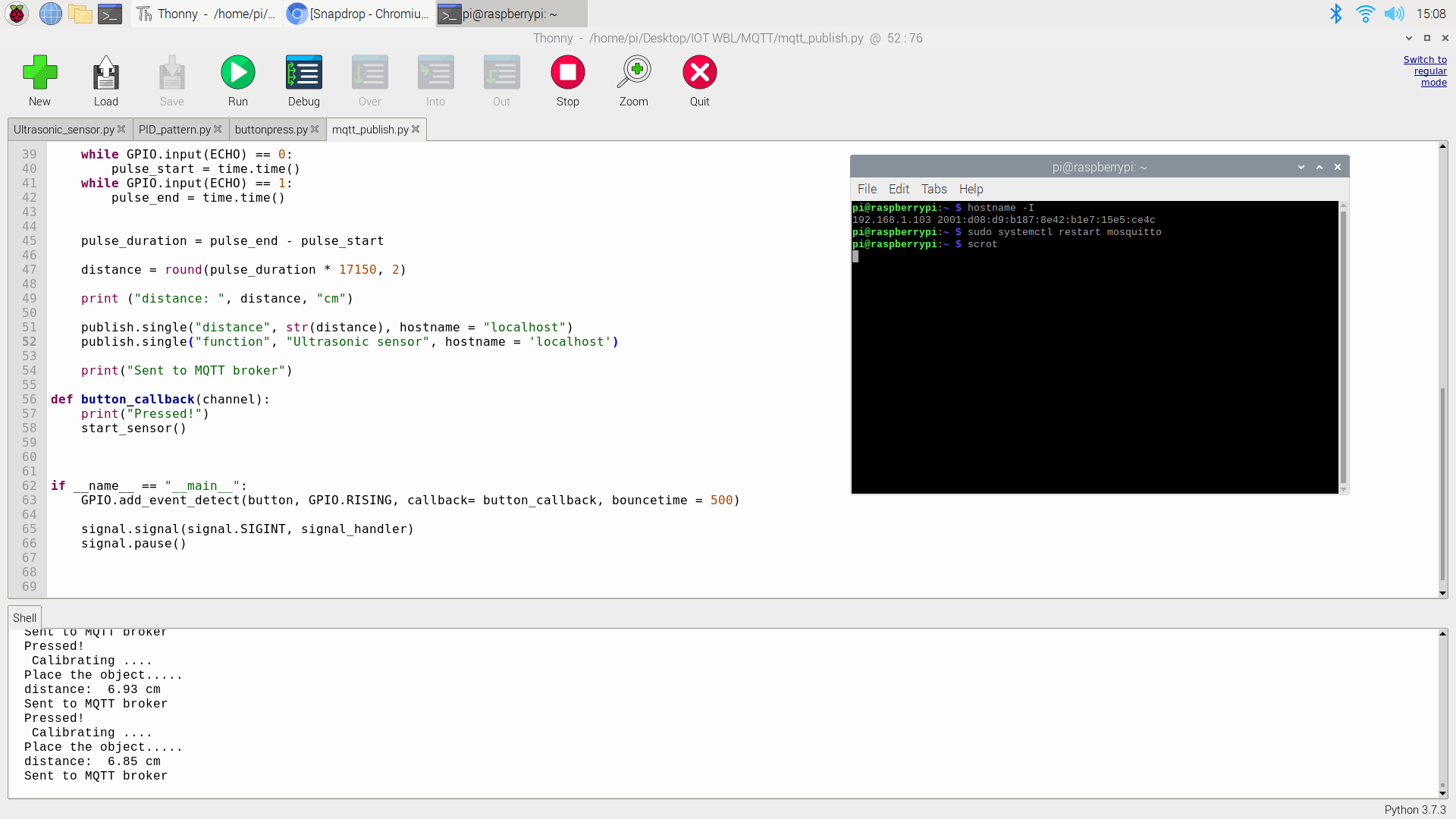


Figure : When the pushbutton is pressed

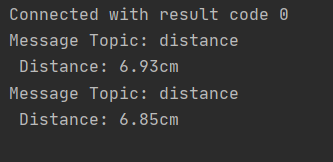


Figure: “distance” is successfully transferred using MQTT

We can repeat the process but this time, the client subscribes to the topic “function” instead of “distance”.

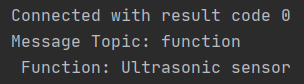


Figure: “Function” is successfully transferred using MQTT

### Reference

1. J. S. Cook. (2019). MQTT Tutorial: Setup MQTT Server on Raspberry Pi Zero. Retrieved from https://www.arrow.com/en/research-and-events/articles/mqtt-tutorial

## Firebase application in Raspberry Pi

### Introduction

In this assignment, the objective is to implement Google’s Firebase realtime database to use REST to communicate between the database and the clients. The idea for this assignment is to use the ultrasonic sensor circuit built in Week 2, and use the distance data to transmit it from the client (Raspberry Pi) to the database. The data should be able to be viewed in Firebase’s database console that can be accessed in the browser. Other than that, I also built a simple android app using MIT’s app inventor which will be able to control the ultrasonic sensor remotely as long as the Raspberry Pi and the android phone is connected to the internet.

Firebase is a Cloud-hosted, NoSQL database that uses a document-model. It can be horizontally scaled while letting you store and synchronize data in real-time among users. This is great for applications that are used across multiple devices such as mobile applications. Firebase is optimized for offline use with strong user-based security that allows for serverless based apps as well. Firebase is built on the Google infrastructure and is built to scale automatically. In addition to standard NoSQL database functionality, Firebase includes analytics, authentication, performance monitoring, messaging, crash reporting and much more. Because it is a Google product, there is also integration into a lot of other products. This includes integration with Google Ads, AdMob, Google Marketing Platform, the Play Store, Data Studio, BigQuery, Slack, Jira, and more (Jones, 2020).

### Discussion

First, we have to create an account in Firebase. A project is then created with the name Ultrasonic sensor. The code for the python program is shown below.

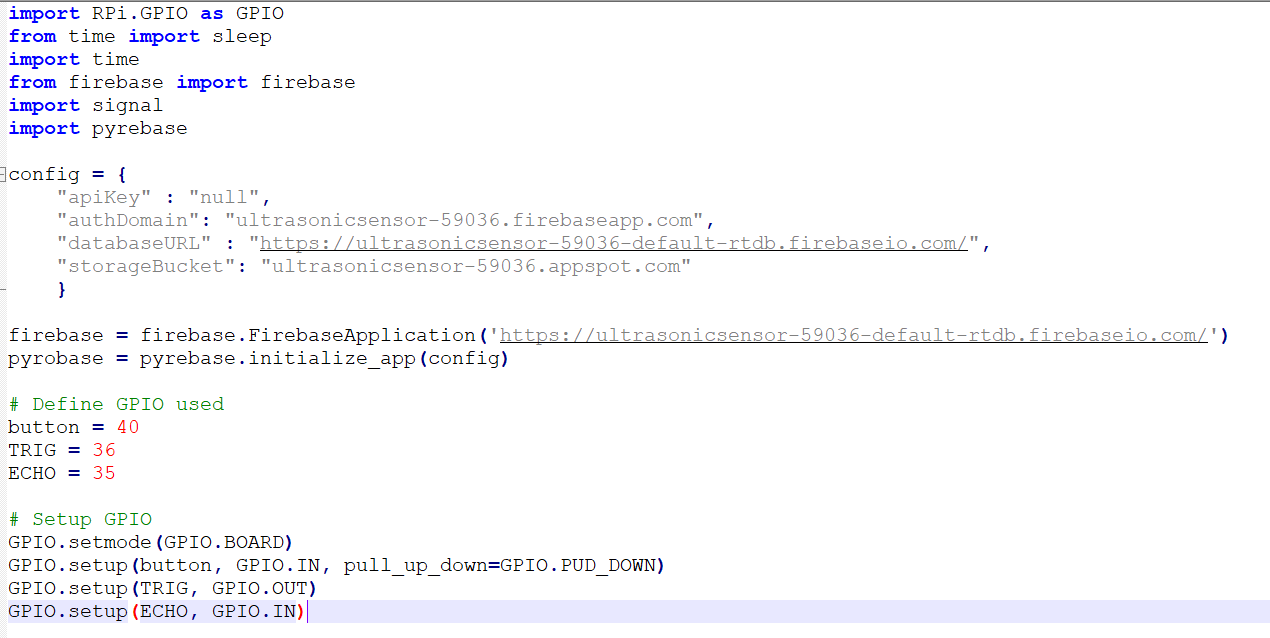


Figure : Python Program Code Snippet 1

We need to import the usual RPi.GPIO, time, signal library. The firebase and pyrebase library are imported so that we have access to functions that are able to interact with the firebase realtime database. Next, we have to set our config dictionary which includes our API key, authorised domain, database URL and storage bucket to use in conjunction with the pyrebase library. **pyrebase.initialize\_app(config)** allows pyrebase functions to be able to interact with the firebase database. Next we have to paste our databaseURL into the function **firebase.FirebaseApplication** to allow access to the database for firebase functions.

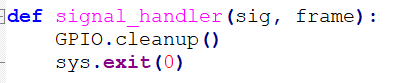


Figure : Python Program Code Snippet 2

This function turns off all the GPIO pins and exits the program when called.

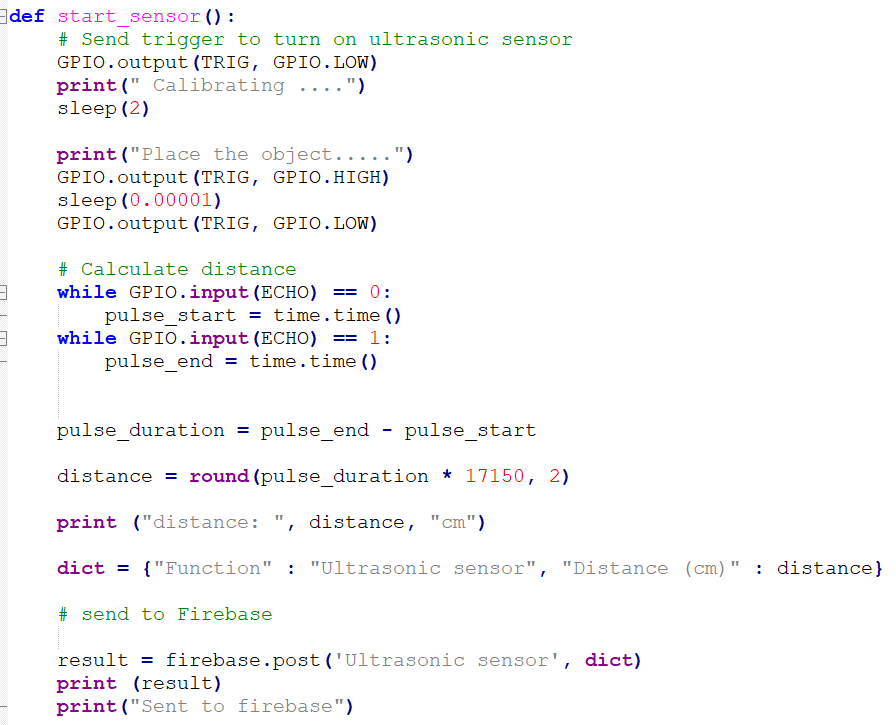


Figure : Python Program Code Snippet 3

The start\_sensor function measures the distance between the object and the sensor, then uses POST to send the distance and the function of the program to the firebase real time database. If sent successfully the program will display “ Sent to firebase”.

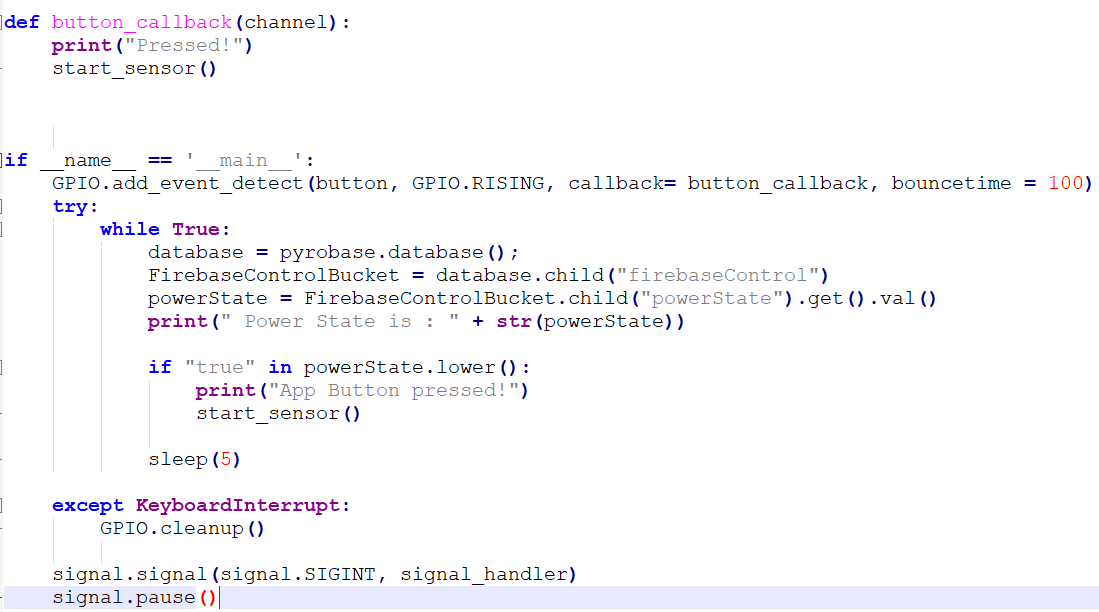


Figure : Python Program Code Snippet 4

**Button\_callback** function works the same as in the ultrasonic sensor assignment. In the main function, **pyrobase.database** function is defined by the variable **database**. Then, database.child function is used to create a bucket under the project to store boolean value TRUE or FALSE. Next, the **powerState** variable gets the boolean variable from the firebase real time database, then prints it in the program. If powerState is true, then the ultrasonic sensor starts to measure the distance via the start\_sensor function.

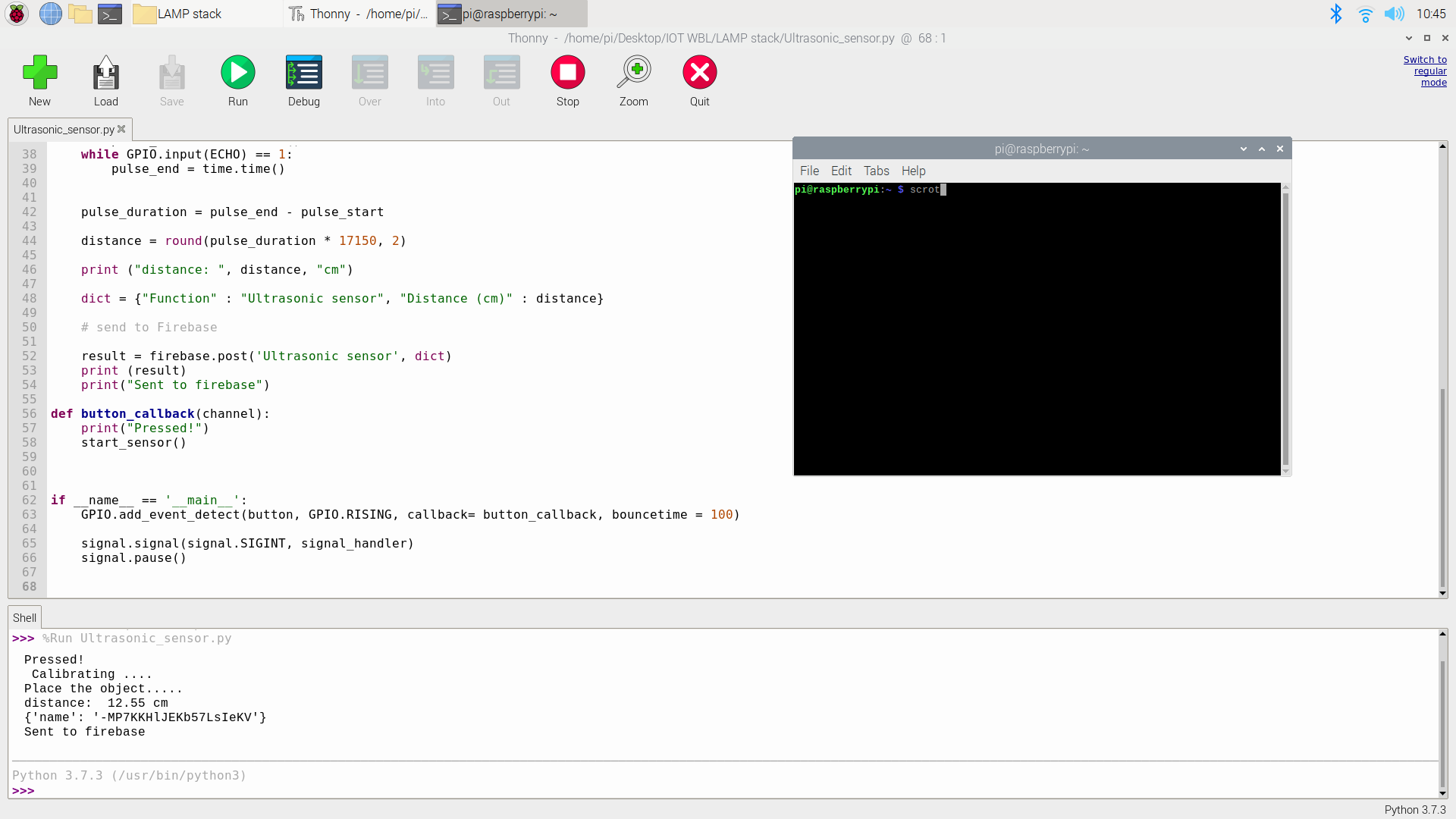


Figure : Distance sent to firebase

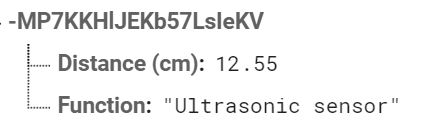


Figure : Distance data in firebase real time database

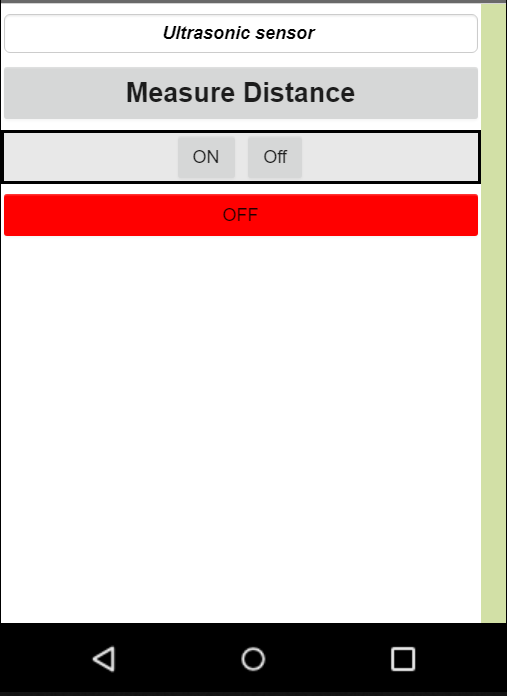


Figure : The app interface in the MIT App Inventor website

The app is simple and easy to operate. Just press on and the ultrasonic sensor will start measuring the distance which can be seen in the firebase real time database or in the python program. When the off button is pressed, the ultrasonic sensor will also stop measuring distance. MIT App Inventor makes it easy to design a simple app as you just have to drag and drop what button, text or image you want to place into your app. However, the customize and layout choices are limited. After I placed all the buttons and display I wanted, MIT app inventor provided easy block programming to program the logic and the function of each component.

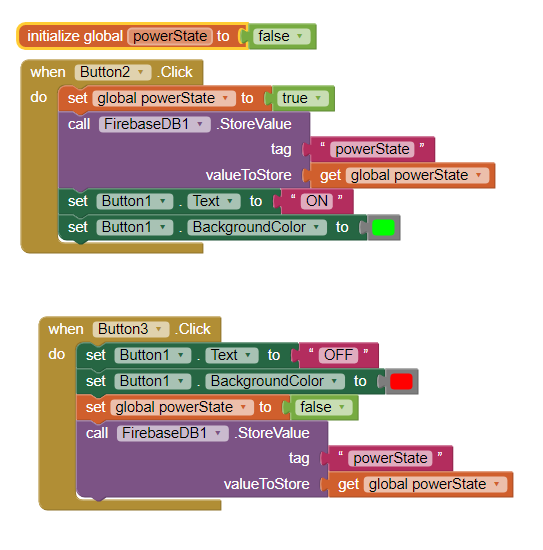


Figure : Block logic programming in MIT App Inventor

Button 2 is the ON button, while Button 3 is the OFF button. When Button 2 is clicked, it will set the global variable powerState to true, then it will call the Firebase database to store the value of powerState in the bucket with a variable of the same name. After that, the display should turn green and displays ON. Button 3 is the OFF button and it changes powerState to false, and changes the display colour to red and OFF.

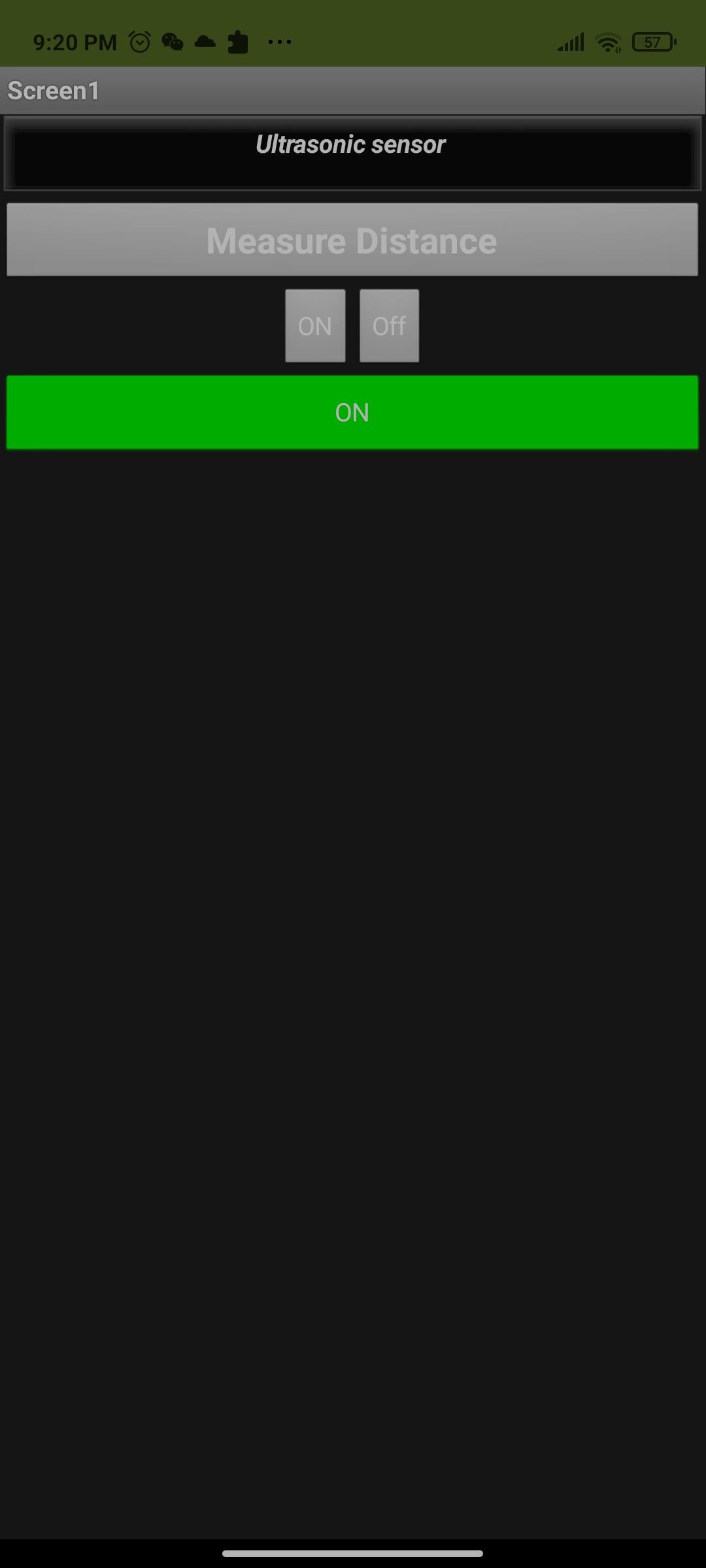


Figure : App on my phone ( When pressed ON)

When the on button is pressed the OFF becomes ON and turns GREEN. Then it changes the variable powerState in the firebaseControl bucket into TRUE. The python program gets the value of powerState, and acts accordingly. If TRUE, the ultrasonic sensor will start. If FALSE, the ultrasonic sensor stops.



Figure: database when ON pressed

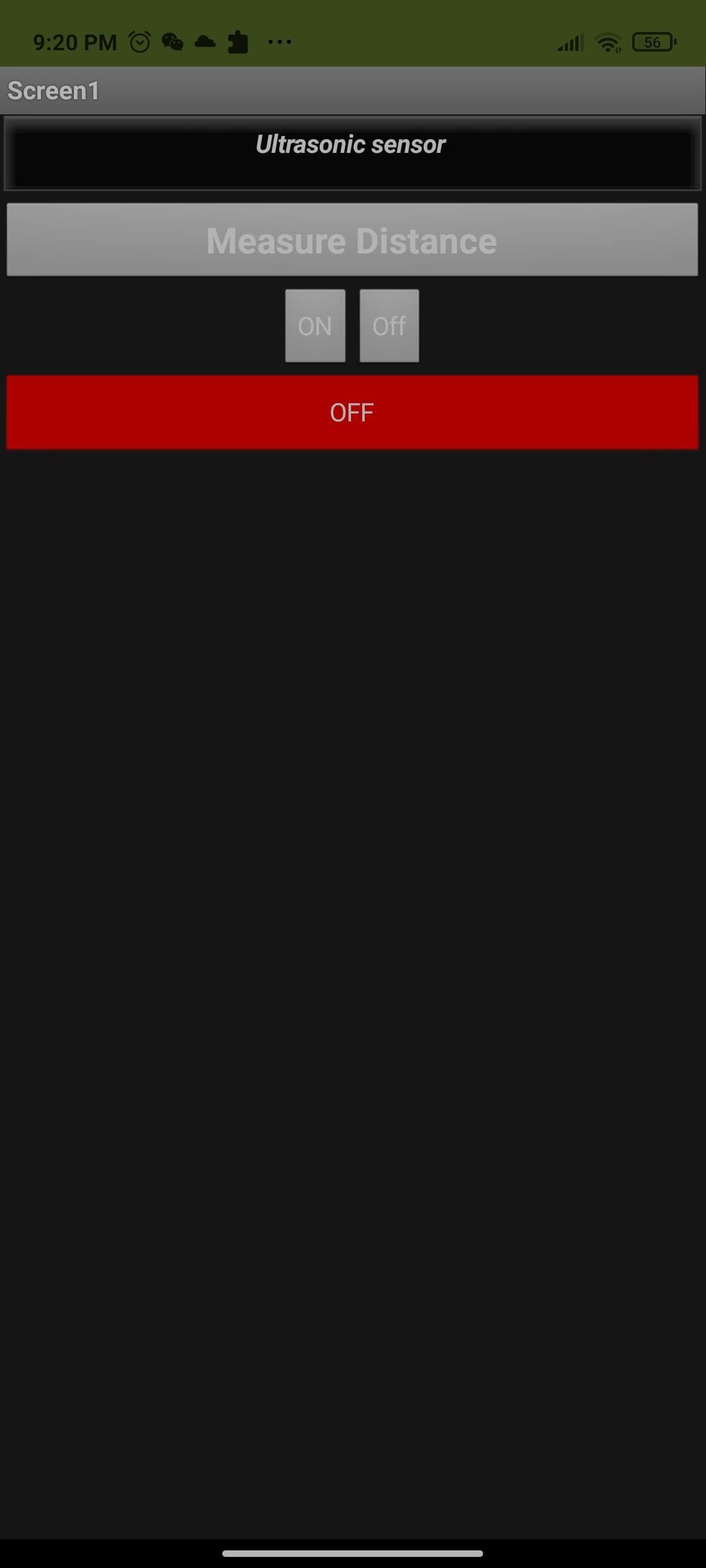


Figure : App on my phone ( When OFF is pressed)



Figure : Database when OFF is pressed

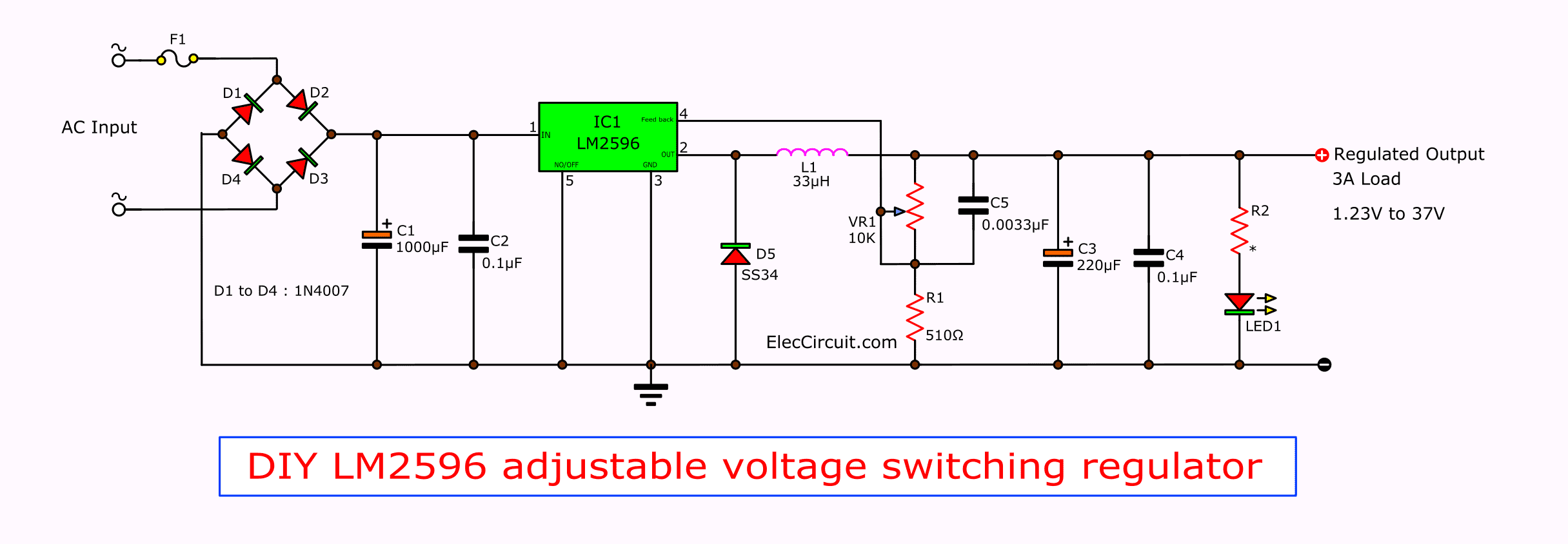
Reference

1. B. L. Jones. (2020). Introduction to Firebase. Retrieved from https://www.databasejournal.com/features/mysql/introduction-to-firebase.html

## PCB Designing

### Introduction

In this project, I will be using the circuit below to design my own PCB. The PCB designer program that I will be using is EAGLE 9.6.2. The regular below supports AC input of 3.2V to 40V, while the DC output ranges from 1.23V to 37V.



**Figure : LM2596 adjustable voltage switching regulator**

To understand how this circuit works, we have to see a basic buck converter.

# 

Figure 8: Basic Circuit of a Buck Converter

Zener diodes and voltage divider circuits are also able to convert high voltage input into low voltage output. However, these circuits lower voltage by dissipating power as heat energy. Due to this reason it is not efficient to use these methods as power will be lost. Another alternative method which provides better efficiency is the buck converter. The buck converter lowers the voltage by increasing the current.

When the switch is ON, current flows through the load via the inductor. The inductor stores energy and magnetic field increases, which forms a potential difference across the inductor. The voltage across the load is the difference between the voltage of voltage source and the voltage across the inductor. The cathode is in reverse biased so no current flows through it. The capacitor is also charging.

When the switch is OFF the energy stored in the magnetic field around the inductor is released back into the circuit. The voltage across the inductor is in reverse polarity, and it is enough to keep current flowing when the switch is open. Current flows via the load and the cathode until the load voltage begins to fail, then the charge stored in the capacitor becomes the main source of current.

The overall effect is a ripple waveform. The load voltage is able to be calculated using the formula below:



|  |
| --- |
| On time of switching waveform (tON) , period of switching waveform(T) |

Figure: Formula to calculate output voltage of buck converter

### Discussion

There are several steps in PCB drawing. First, we have to insert the schematics of our circuit into the program. In the EAGLE, there is a wide variety of component libraries which contain different components from different manufacturers. If the required components are not found, EAGLE supports importing custom libraries or 3rd party libraries made by other people. The schematic is shown in Figure.

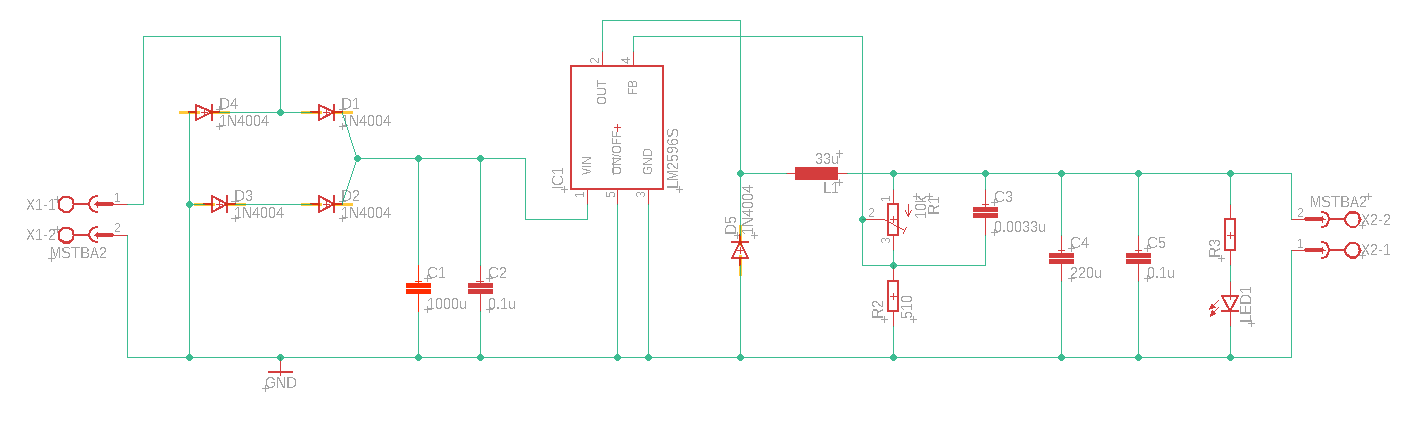


Figure : EAGLE Schematic for the adjustable voltage switching regulator

After placing all the components, wires and connectors, the schematic is complete. After that, we have to convert the schematic diagram into a board form.

EAGLE places all the components in a canvas. The first and crucial step is to determine the board size needed. After that, it is crucial to arrange the components based on their operation and function in the circuit. We should first place the critical components first, then only proceed to placing the less important components. For the circuit above, there are a few critical segments of the circuit that have to be wary of when arranging the components. In Figure , the connector receives AC input from an external AC source. Tracks and circuits running AC have to be placed together away from the DC circuit as there will be electromagnetic interference if the tracks are placed too close and since the voltage may be high, the copper track width has to be thick enough to accommodate high current. This is critical as high current density in the track will cause overheating issues, or in some cases the copper tracks will just break apart. Thus, I have made the tracks in the blue square in Figure to be thicker. Next, another crucial part of the circuit is the connection from the voltage divider to the feedback port of the IC LM2596S. This connection has to be as close as possible to the IC because the longer the copper track, the higher the resistance between the voltage divider and the feedback port. After arranging the components, I added the ground plane which will connect all the ground using a plane in the PCB board. This simplifies things by eliminating the need of connecting the grounds of components together. Next comes the important part which is routing. For this PCB design, we use two layers, the top and bottom layer. Thus, we are able to place tracks on both these layers. Another thing to note is that through the hole components ( all the components used here except for the variable resistor, inductor and LM2596S) can be connected by the tracks from both layers, while the surface mounted components can be connected either on the top layer or the bottom layer depending on your preference. Since PCB is very compact, there are times when you do not have enough space to connect the track on the top layer. In this scenario, you are able to continue the track in the bottom layer by placing a via. Via is a hole which connects tracks on different layers or planes. Moreover, it is best that the traces can be straight, as this provides the least signal loss. The second best is curve, however curve traces are more expensive to print out, thus it is usually used when high speed is a necessity. If both are not possible then a 45 degree trace is acceptable, while a 90 degree turn is to be avoided. This is because right-angle or any sharp angle corners presented a realistic threat to manufacturability for older PCB manufacturing techniques. Sharp corners could cause acid traps, where some of the acids used in etching linger and continue to corrode the copper at sharp corners. Back then, engineers were also concerned that right-angled corners aren’t as sturdy as 45° ones and could be peeled off easily. Also, at high frequencies of 10GHz and more, Electromagnetic Interference (EMI) may be radiated at sharp corners (90°). So after connecting all the traces to each component we will get our final PCB design. Below shows the three attempts of PCB design.



Figure : Critical segments of the adjustable voltage switching regulator

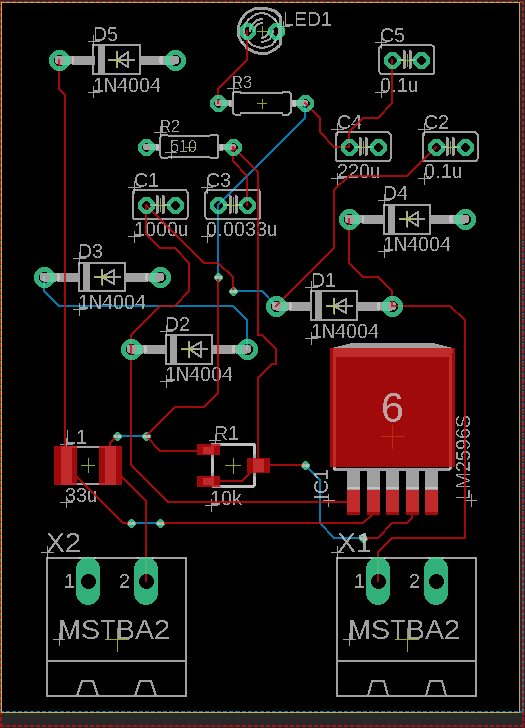
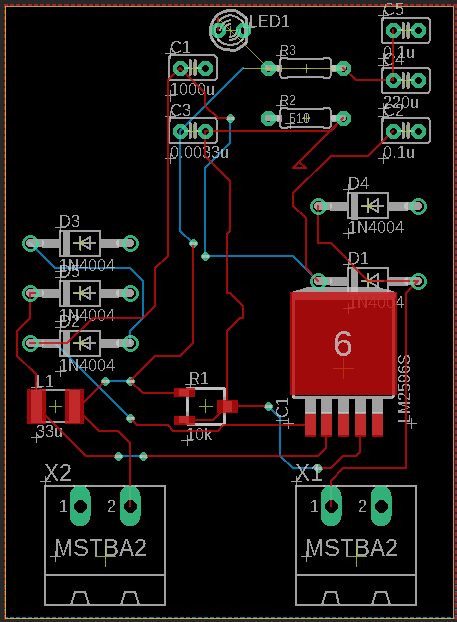


Figure : First attempt PCB design

The traces are all well connected. However the components are not placed neatly and in groups based on their operation and their function.



**Figure: Second attempt PCB Design**

In this second attempt, the components are arranged more neatly, and the traces look neater compared to the first one. However, components and traces that have higher AC and DC currents are near to traces with small Dc currents. This would lead to some problems due to Electromagnetic interference. Thus, this design has to be modified to separate the high current AC, DC circuit from the low current DC circuit. The IC is also far from the voltage divider, so the voltage feedback from the voltage divider may show inaccurate results.

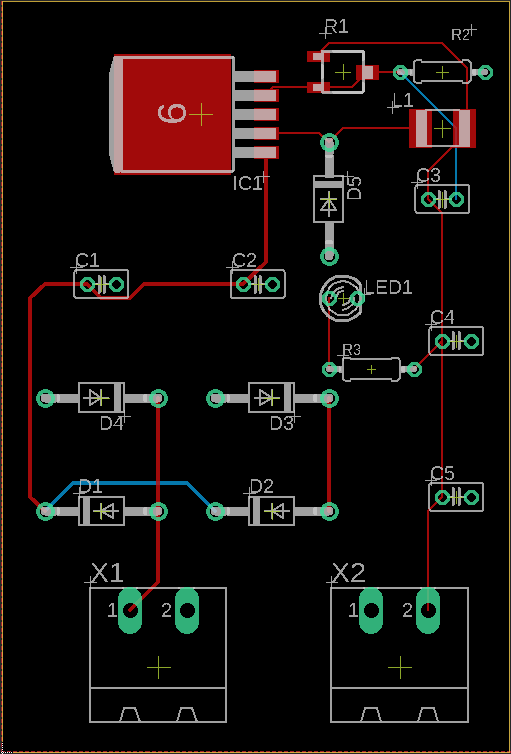


Figure : Third and final attempt PCB Design

The final design of the PCB is much more neat, with many straight traces connecting components for optimum connection. The components have also been grouped together based on their function and operation in the circuit. On the left side above the 2 pin connector, the four diodes that make up the rectifier are placed together, and the connection continues to the two capacitors C1, C2 to the VIN port of the LM2596S. This part of the circuit runs AC until it is rectified into DC by the four diodes. The feedback port is placed beside the voltage divider which consists of the variable resistor and resistor, thus voltage feedback is more accurate. The circuit running DC and AC are separated from each other. Thus, the PCB drawing is complete and can be printed out. The components have to be soldered on the board and the board will work as intended.