Abstract

In our day to day life we are connected to the internet on an average of 6 hours and 30 minutes*.The main essence of this mini project was to harness some of time into making our life more efficient and informative.

We have devised a prototype for a smart home that uses ESP32, Firebase and a mobile app letting u control your home's electronic devices.

The Current is measured through an ACS712 and fed to the ESP32 for processing and uploading the data to firebase.

The app acts as a switch allowing the system to run on its own or on input from the user. The uploaded data is accesses firebase and is subjected to a program that visualises the data and displays it as a real time graph.

This project shows an almost perfect way to combine all the aspects of IoT to develop an efficient system.

* Refer to the Appendix for the Source of the information.

Acknowledgement

We would like to express my special thanks of gratitude to my teacher, Dr. Manikandan J, who gave us the opportunity to carry out this Project as our 2 Credit course for IV Semester. This Project helped us in doing a lot of research and we came across a variety of new techniques and topics that we never knew existed.

Secondly, we would also like to thank our parents and friends who helped us in finalising this project within the limited time frame.

Contents

Individual Contributions

Anantha Krishna

1. FireBase Database Configuration

2. Firebase Authentication with E-Mail

3. Rewrote part of the Firebase.set part of the Firebase Library of ESP32 to suit our needs. (Line 82 - 85) of IOXhop_FirebaseESP32.cpp

4. Wrote the Firebase push and get code for the ESP32.

5. Made an App using Swift and Xcode to map the realtime value of the power and to act as a switch for the light with Firebase Authentication built into the App.

Isha Prabhakar

1. Developed the hardware aspect of the project by designing the interface for measuring of current using ACS712 and ESP32.

2. Interfaced all the Hardware Components.

3. Interfaced a LDR , in order to facilitate the automated powering off of the LED Strip.

4. Interfaced a bidirectional logic shifter to convert 5v to 3.3 V and vice-versa.

5. Wrote the code to control all Hardware.

6. Tested and verified the measurements of the components using a multimeter.

.

Chapter 1

1.1 Introduction

A recent study concluded that on Average, a person spends 6.5 hours daily, connected to the internet sending and receiving data. Most of this time spent is through a hand held smart electronic device.

The main essence of this project was to harness some of that time into making one's life more efficient so as to utilise our time in better ways.

This combination of Hardware and software connected over the internet also known as IoT when done using ESP32 (A Powerful developer board) and Firebase (A google backed high end cloud service provider with various APIs) Makes the user experience top notch.

1.2 Problem Statement and Objective of Project

Millions of people around the world use electricity on a daily basis for day to day activities. Doing so consumes a certain amount of power from the national grid.

Many times people forget to switch off appliances or can't switch it on at the right moment. This causes a wastage of power and that's a bad thing as it could have been used by another person for their needs.

If people use electricity properly the grid stays stable and that in turn allows us to use the electricity without any interruption.

Our Project is a solution that combats all these problems as a whole. It enables the user to allow the appliance to run automatically or he can toggle the appliance on/off depending on the need and the best part is that the user gets real time power consumption readings and a graph to go along with it so, they are well informed about how they are consuming power from the grid.

Chapter 2

2.1 Literature Survey

(i) FireBase Database (Documentation)

- Google

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronised in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

Key Capabilities:

- **RealTime:**
  Instead of typical HTTP requests, the Firebase Realtime Database uses data synchronisation—every time data changes, any connected device receives that update within milliseconds. Provide collaborative and immersive experiences without thinking about networking code.

- **Offline**:
  Firebase apps remain responsive even when offline because the Firebase Realtime Database SDK persists your data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronising It with the current server state.

- **Accessible from Client Devices**:
  The Firebase Realtime Database can be accessed directly from a mobile device or web browser; there's no need for an application server. Security and data validation are available through the Firebase Realtime Database Security Rules, expression-based rules that are executed when data is read or written.

- **Scale across multiple databases:**
  With Firebase Realtime Database, you can support your app's data needs at scale by splitting your data across multiple database instances in the same Firebase project. Streamline authentication with Firebase Authentication on your project and authenticate users across your database instances. Control access to the data in each database with custom Firebase Realtime Database Rules for each database instance.

How does it work?

The Firebase Realtime Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, realtime events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime Database synchronises the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

The Realtime Database provides a flexible, expression-based rules language, called Firebase Realtime Database Security Rules, to define how your data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.

The Realtime Database is a NoSQL database and as such has different optimisations and functionality compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly. This enables you to build a great realtime experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your data and then structure it accordingly**.

** Other aspects of the database and how to use it will be covered in Section 4.3

(ii) A Vision of IoT: Applications, Challenges, and Opportunities.

- Shanzhi Chen

Internet of Things (IoT), which will create a huge network of billions or trillions of "Things" communicating with one another, are facing many technical and application challenges.
The IoT is an intelligent network which connects all things to the Internet for the purpose of exchanging information and communicating through the information sensing devices in accordance with agreed protocols. It achieves the goal of intelligent identifying, locating, tracking, monitoring, and managing things [1]. It is an extension and expansion of Internet-based network, which expands the communication from human and human to human and things or things and things.

IoT should have the following three characteristics

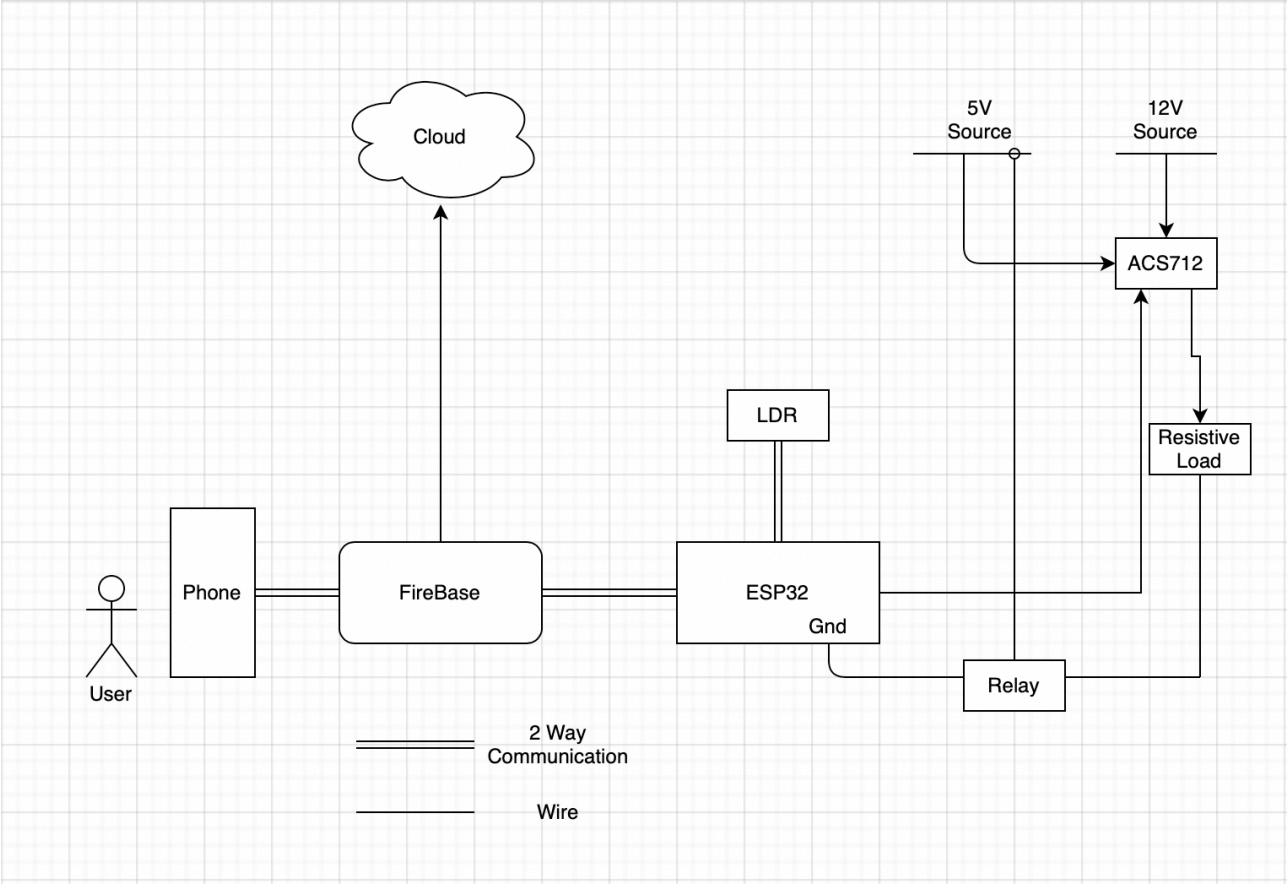
1. **Comprehensive Perception:** Using sensors, and two-dimensional toggles/buttons to obtain the object information at anytime and anywhere, it will be a new opportunity. Using it, information and communication systems can be invisibly embedded in the environment around us. Sensor network will enable people to interact with the real world remotely. Identification technologies mentioned here include objects and location identifications. Identification and recognition of the physical world is the foundation of implementing overall perception.

2. **Reliable Transmission:** Through a variety of available radio networks, telecommunication networks, and Internet, objects information can be available in any time. Communication technology here includes a variety of wired and wireless transmission technologies, switching technologies, networking technologies, and gateway technologies.

3. **Intelligent Processing:** By collecting IoT data into databases, various intelligent computing technologies including cloud computing will be able to support IoT data applications. The network service providers can process tens of millions or even billion pieces of messages instantly through cloud computing. Cloud computing technology will thus be the promoter of IoT.

Chapter 3

3.1 Block Diagram



3.2 Explanation of Each Block

- **User:**
  The User is the one who has the app on his smart device that can control the entire system.

- **Phone:**
  The phone is the physical smart device that is in the hands of the user. It contains the App with the respective Authorisations to access the system.
  The app contacts FireBase to request for the specified data through a dependency manager that is called cocoa pods.

- **FireBase:**
  FireBase is a cross platform cloud service that lets us store data and run analytics on it if needed. For this project we have made use of the Real Time Database, Firebase Authentication and Firebase Analytics.
  Source: 1

- **Cloud:**
  Virtual destination where all the data we send to Firebase is Stored.
  Source: 2

- **ESP32**:
  ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series employs a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations and includes built-in antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management modules.
  Source: 3

- **LDR:**
  Light Dependent Resistor(LDR) works on the principle of photo conductivity. LDRs depend on the light intensity, when light falls on the LDR, the resistance decreases, and the resistance increases in the dark.
  We exploit this property to automate our Appliance.

Source: 4

- **Relay:**

1. Relay used: HELISHUN HLS8L-DC5V-S-C

2. Specifications:7A 250V AC
   5A 30V DC

3. Works for both AC and DC loads.

4. The maximum output voltage of an ESP32 is 3.3V.

5. When we have to control devices that work at higher voltages say a 12V LED strip using an ESP32 we use a relay(here 5V DC relay) to power ON/OFF the 12V LED strip.

6. The relay we have used is a Single Pole Double Throw Relay(SPDT). The Relay consists of a coil(internal), 1 common terminal, 1 normally closed terminal, and one normally open terminal.

Source : 5

- **Resistive Load:**

We used a 12V Blue LED Strip.

Specifications :

Wattage	14.4W per meter
Input Voltage	DC 12V
LED Type	5050 SMD LED
Beam Angle	120 Degree
No. of LEDs	60 LEDs per meter
Size (mm)	5000 x 10 x 2.2

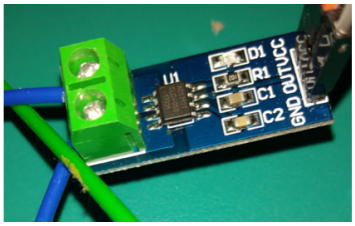
Source: 6

- **ACS712:**

<u>Specifications of the model used for our project:</u>

Allegro ACS712 5A

IC : ACS712TELC-05B 17031



Current sensitivity: 185 mV/A

Features(as mentioned on the Data sheet):

- Low-noise analog signal path
- Fully Integrated, Hall-Effect-Based Linear Current Sensor IC
- Device bandwidth is set via the new FILTER pin
- 5 µs output rise time in response to step input current
- 80 kHz bandwidth
- Total output error 1.5% at TA= 25°C
- Small footprint, low-profile SOIC8 package
- 1.2 mΩ internal conductor resistance
- 2.1 kVRMS minimum isolation voltage from pins 1-4 to pins 5-8
- 5.0 V, single supply operation
- 66 to 185 mV/A output sensitivity
- Output voltage proportional to AC or DC currents
- Extremely stable output offset voltage
- Nearly zero magnetic hysteresis
- Ratio-metric output from supply voltage

Source: 7

- **5V DC Supply:**
  Powered the High Voltage side of the Bi-Directional logic level converter with the 5V DC power supply.
  This in turn powered the VCC & GND of the relay,ACS712 (current sensor) and served as the +5V and GND for the LDR
  Source: 8

- **12V DC Supply:**

  A DC adapter converts AC electricity into DC electricity.

  - It converts AC operating at household mains voltage(230V) to low-voltage DC suitable for powering devices.
  - A 12V DC adapter 230 V AC at 50 Hz into 12 V DC.
  - The DC adapter is connected to the LED strip via a DC female barrel jack.

  Source: 9

Chapter 4

4.1 Current Measurement Mechanism

- Current Sensing is done in two ways – Direct sensing and Indirect Sensing.
- In Direct sensing, to detect current, Ohm's law is used to measure the voltage drop occurred in a wire when current flows through it.
- ACS712 Current Sensor uses Indirect Sensing method to calculate the current.
- The ACS712 senses the current by having current flow between the terminals.
- The Hall sensor is located at the surface of the IC on a copper conduction path.
- The current flow generates a magnetic field around a loop inside the IC that is proportional to the magnitude of the current flowing in the electrical conductor.
- The Hall-Effect sensor senses the magnetic field generated by the current flowing in the conductor, turning the magnetic field into a voltage which can be measured using the AnalogRead() function.
- This analog value is converted to a digital voltage by multiplying with a factor of (3300/4096).
- The difference of the digital voltage and the offset voltage divided by the current sensitivity will give the value of current drawn.
- By default when no current is flowing through the module terminals the output voltage will be +2.5V (Vcc/2)(2500mV being the offset voltage).
- When the current flows in one direction the value will increase from 2.5V and when it flows in other direction the values will decrease from 2.5V.

This way the module enables us to measure both AC current and DC current.

- When DC load takes current the output voltage will deviate from 2.5V to either 0V or 5V depending on polarity of sensing load.
- The proximity of the magnetic signal to the Hall sensor decides the accuracy of the device. Nearer the magnetic signal higher the accuracy.
- The power loss is very low, making the ACS712 highly efficient in sensing current.
- The sensor allows the Hall transducer to be placed right over the portion of the loop where the magnetic field is the strongest.
- While the Hall sensor is close to the current loop, there is an insulating layer of plastic in between the two, providing high-voltage isolation and allowing it to be used in AC tight applications. Source: 10

When the ACS is used to measure the current used by a Motor:

Red Oval : This is the inductor current when the PWM is ON

Blue Oval : This is the current when PWM is off.
The voltage across inductor is so high that it pushes current back through the DC Bus.

Source: 11

4.2 In Depth Explanation of Hardware Used

1. ESP32:

- ESP32 is a microcontroller with a Hybrid Wi-Fi & Bluetooth Chip.
- It has a 32 bit Tensilica Xtensa LX6 as its microprocessor.
- ESP32 is the successor to ESP8266, both created and developed by Espressif Systems.
- The board we have used for our project is the DOIT ESP32 DEVKIT V1.
- It works on 3.3 V DC power supply and has 48 pins out of which 32 are GPIO pins.
- 15 pins are meant for analog to digital conversion(ADC pins): out of which 6 are of type ADC1 and 9 are of type ADC2.
- ADC2 pins are shared with Wi-Fi drivers, so they can be used only when the Wi-Fi drivers are not started.
- For our project we need our calculated values to be transmitted to Firebase by taking the help of the built-in Wi-Fi, which in turn needs the Wi-Fi driver to be activated. So we have used only ADC1 pins for measuring the values recorded by the LDR and ACS712 sensor.
- The analog pins have a 12- bit sensitivity, implying that the analog values measured will vary in the range 0-4095 (as 2^12 is 4096).
- So, any value in this range will map to a voltage level between 0-3.3V(0 corresponding to analog value 0 and 4095 corresponding to 3.3V), as the output of the ESP32 has a maximum value of 3.3V.
- For this project we have used 5 pins of the ESP32 viz. 3.3V, GND, GPIO 23 (for the relay), GPIO 32 (for the LDR), GPIO 33 (for the current sensor ACS712).

2. ACS712 -current sensor 5A module

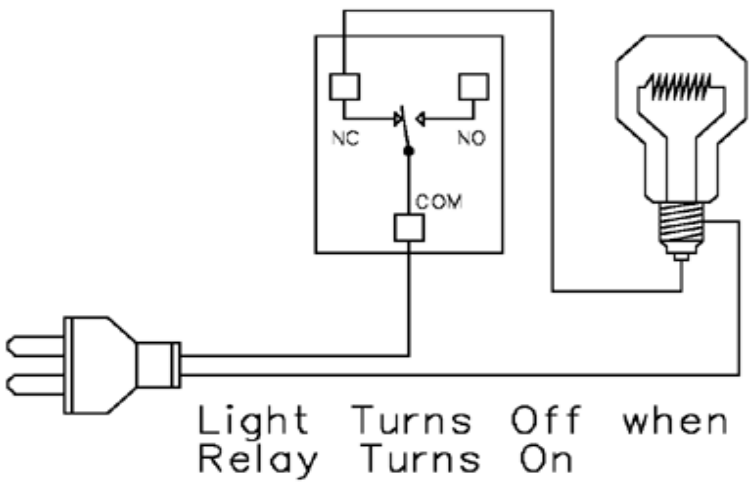
- The ACS712 Module uses the ACS712 IC to measure current using the Hall Effect principle.
- The Hall effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, when a magnetic field is applied in a direction perpendicular to that of the flow of current.
- A Hall effect sensor is used to measure the magnitude of a magnetic field. Its output voltage is directly proportional to the magnetic field strength through it.
- The ACS712 consists of a precise, low-offset, linear Hall circuit with a copper conduction path located near the surface of the die.
- Applied current flowing through this copper conduction path generates a magnetic field which the Hall IC converts into a proportional voltage.
- Device accuracy is optimised through the close proximity of the magnetic signal to the Hall transducer.
- A precise, proportional voltage is provided by the low-offset, chopper-stabilised BiCMOS Hall IC.
- The ACS712 module can measure AC or DC current ranging from +5A to -5A, +20A to -20A and +30A to -30A.
- For our project we have used a 5A model.
- It has to be connected in series as it has to measure current.

3. 5V DC relay

- Relay used: HELISHUN HLS8L-DC5V-S-C
- Specifications:7A 250V AC
  7A 30V DC
- Works for both AC and DC loads.

- The maximum output voltage of an ESP32 is 3.3V.
- When we have to control devices that work at higher voltages say a 12V LED strip using an ESP32 we use a relay(here 5V DC relay) to power ON/OFF the 12V LED strip.
- The relay we have used is a Single Pole Double Throw Relay(SPDT). The Relay consists of a coil(internal), 1 common terminal, 1 normally closed terminal, and one normally open terminal.
- A relay has three high voltage terminals (NC, COM, and NO) which connect to the device we want to control. The other side has three low voltage pins (Ground, VCC, and Signal) which connect to the ESP32(Where NO ->Normally Open, NC->Normally Closed, COM->Common).

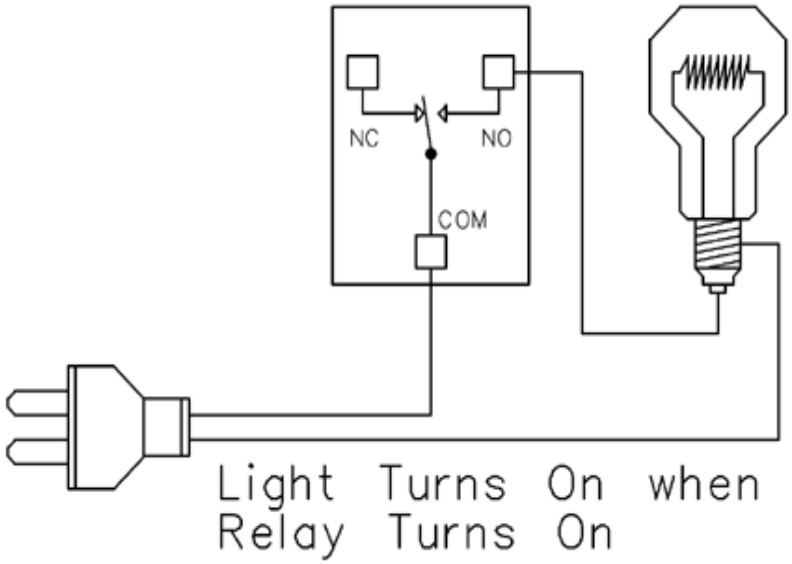
- The signal pin of the relay is connected to a 5V to 3.3V bidirectional logic level converter from which it is connected to the ESP32.
- This is done because the relay works on 5V and the output of the ESP32 is 3.3v, so if the signal pin of the relay is directly connected to the ESP32 the relay will not get energised. Hence the relay action will not take place.
- In the NC configuration of the relay a HIGH signal opens the switch and thus disconnects the circuit from the power supply. A LOW signal closes the switch and allows current to flow from the COM terminal to the NC terminal.



-

- For our project, we need a HIGH signal to activate the relay, thereby switching ON the LED strip.

- Therefore, we use the NO configuration of the relay, wherein the COM terminal of the relay is connected to the positive power supply and the NO terminal is connected to the positive terminal of the LED strip.



- So, when there is a LOW signal on the Signal pin, the relay is deactivated and there is an open circuit condition, so there is no current flowing through the LED strip and it does not light up. When there is a HIGH signal , the relay is energised and there is a closed circuit formation, therefore current flows through the LED strip and it glows. Current values are then measured by the current sensor and displayed on the serial monitor.

4. LDR with 10K Ohm resistor

- Resistance range: fewK Ohms(bright conditions) to 1M Ohm(complete darkness)
- Voltage requirements: 3.3v or 5V
- Light Dependent Resistor(LDR) works on the principle of photo conductivity.

- LDRs depend on the light intensity, when light falls on the LDR then the resistance decreases, and the resistance increases in the dark.
- When light falls on its surface, then the material resistance reduces and also the electrons in the valence band of the LDR are excited to the conduction band.
- The photons in the incident light must have energy greater than the band gap of the semiconductor material. This makes the electrons to jump from the valence band to conduction band thereby reducing the resistance under bright lighting conditions.
- Though the LDR is a resistor, we cannot use it directly and measure the voltage difference across its terminals since there is no reference point except for VCC(3.3V) and GND.
- Also, the entire voltage is dropped across the LDR no matter what resistance it has.
- If we use a voltage divider, we will have a well-defined reference point to measure the voltage changes.
- Therefore, we add a second resistor(10K ohm), in series with the LDR(other valued resistors can also be used depending on the lighting conditions and the resistance of the LDR).
- The second resistor acts like a fixed reference point.
- Using that we can figure out what resistance does the LDR have and thereby measure the intensity of light.
- We measure the intensity of light by connecting one end of the LDR to the +3.3v power supply, one end of the 10K ohm resistor with GND and the junction of other end of LDR and the resistor with the analog pin GPIO 32.
- The analog value obtained is checked and if the value is less than 800(dark), then the relay is given a HIGH signal, else the relay is given a LOW signal(as the surrounding is bright).

5. Blue LED Strip -12V DC (3LEDs*20strips)

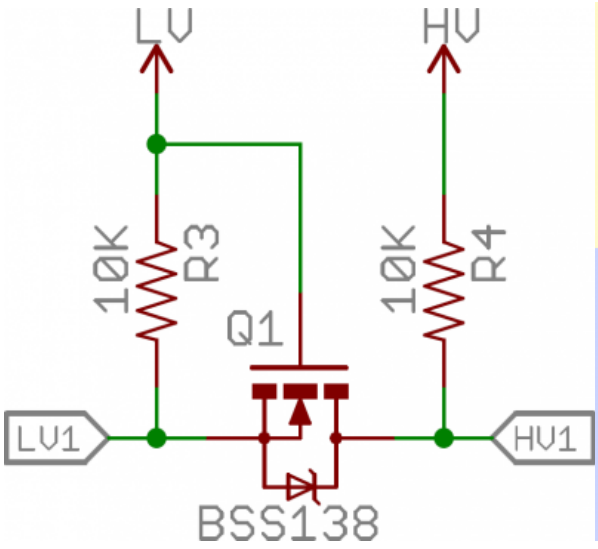
- For this project we have used 20 strips, each strip having 3 LEDs.
- Therefore (20/60)=0.333 metres of LED strip.
- Current per strip ~20mA, therefore for 20 strips it is 0.4 A.
- Power consumption=14.4x0.333=4.8 W

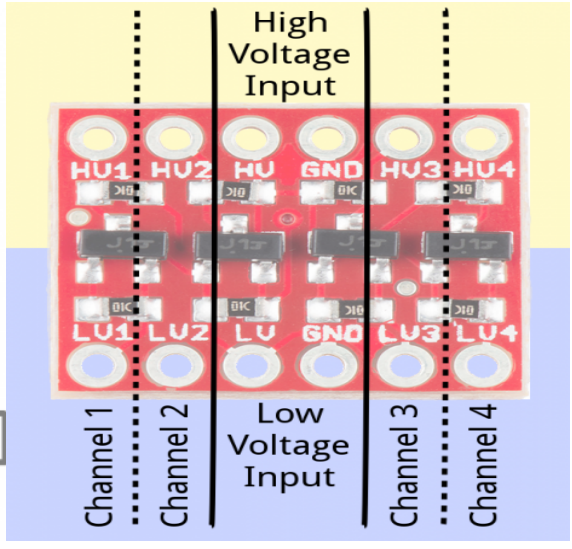
6. 12V adapter(power supply):

- Specifications :
  - RE-PRO special adapter
  - INPUT: 90-270V-0.8A, 50-60Hz
  - OUTPUT:12V 2.0A(MAX) 24 W(MAX)
  - Maximum power output in watts being 24
- A DC adapter converts AC electricity into DC electricity.
- It converts AC operating at household mains voltage(230V) to low-voltage DC suitable for powering devices.
- A 12V DC adapter 230 V AC at 50 Hz into 12 V DC.
- The DC adapter is connected to the LED strip via a DC female barrel jack.

7. Bi-directional logic level converter(3.3V<-> 5V logic converter)

- The sensors and components (relay and ACS712) that we have used in our project work on 5v.
- The output of the Esp 32 is 3.3v, so in order to power the 5v sensors we need to shift the logic level from 3.3v to 5v and vice-versa to enable efficient and ideal functioning of the sensors.
- Considering the 12-bit Esp 32, if this logic level shifting is not done then the sensor will give analog values only from 0-2702. The reason being 3.3v maps to analog value 2702 via the following relation (3.3x4095)/5. Therefore we will not get full range of analog values of 0-4095.
- The circuit of a bi-directional logic level converter uses a single N-channel MOSFET and a couple of pull-up resistors to realise the bi-directional level shifting.





- There are 12 total pins on the bi-directional logic level converter: two parallel rows of six headers. One row contains all of the high voltage inputs and outputs, the other row has all low voltage inputs and outputs.

- The pins labelled *HV*, *LV*, and two *GND*'s provide high and low voltage references to the board. Supplying a steady, regulated voltage to both of these inputs is required.
- In our case we have supplied 5v to the high voltage side and 3.3v to the low voltage side.
- Both the GND pins are then common grounded.
- There are four separate data channels on the bi-directional logic level converter, each capable of shifting data to and from high and low voltages. These pins are labelled as HV1, HV2, HV3, HV4, LV1, LV2, LV3 and LV4.
- A low-voltage signal sent in to LV1 will be shifted up to the higher voltage and sent out HV1. Similarly a high voltage signal sent in HV3(5v) will be shifted down and sent out of LV3(3.3v).
- The logic level shifter is useful for mainly digital voltages. If they are used with analog voltages, they can result in non-linear values. We have used the logic shifter with the analog ACS712, compromising on the linearity of values measured by the ACS712 in order to avoid the limited range of analog values measured by it and to enable the working of the sensor at its specified voltage, thus preventing the malfunctioning of the sensor.

8. 5V DC power supply

- Powered the High Voltage side of the Bi-Directional logic level converter with the 5V DC power supply.
- This in turn powered the VCC & GND of the relay,ACS712 (current sensor) and served as the +5V and GND for the LDR.

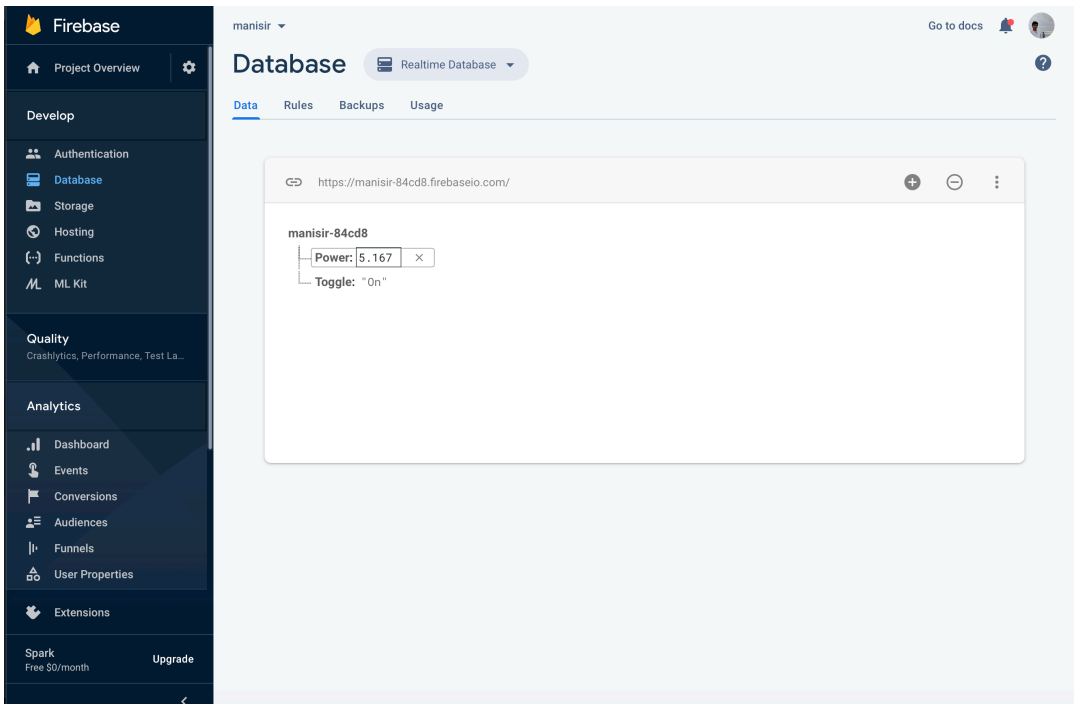
9. Other components: Jumper cables, breadboard

A 10cm jumper cable has a resistance of ~300 milli Ohms. Therefore voltage drop per ampere of current will be ~0.3 ampere.

A breadboard has a large parasitic capacitance:~2 pF between adjacent columns. (These values affect the power reading by ~0.15)

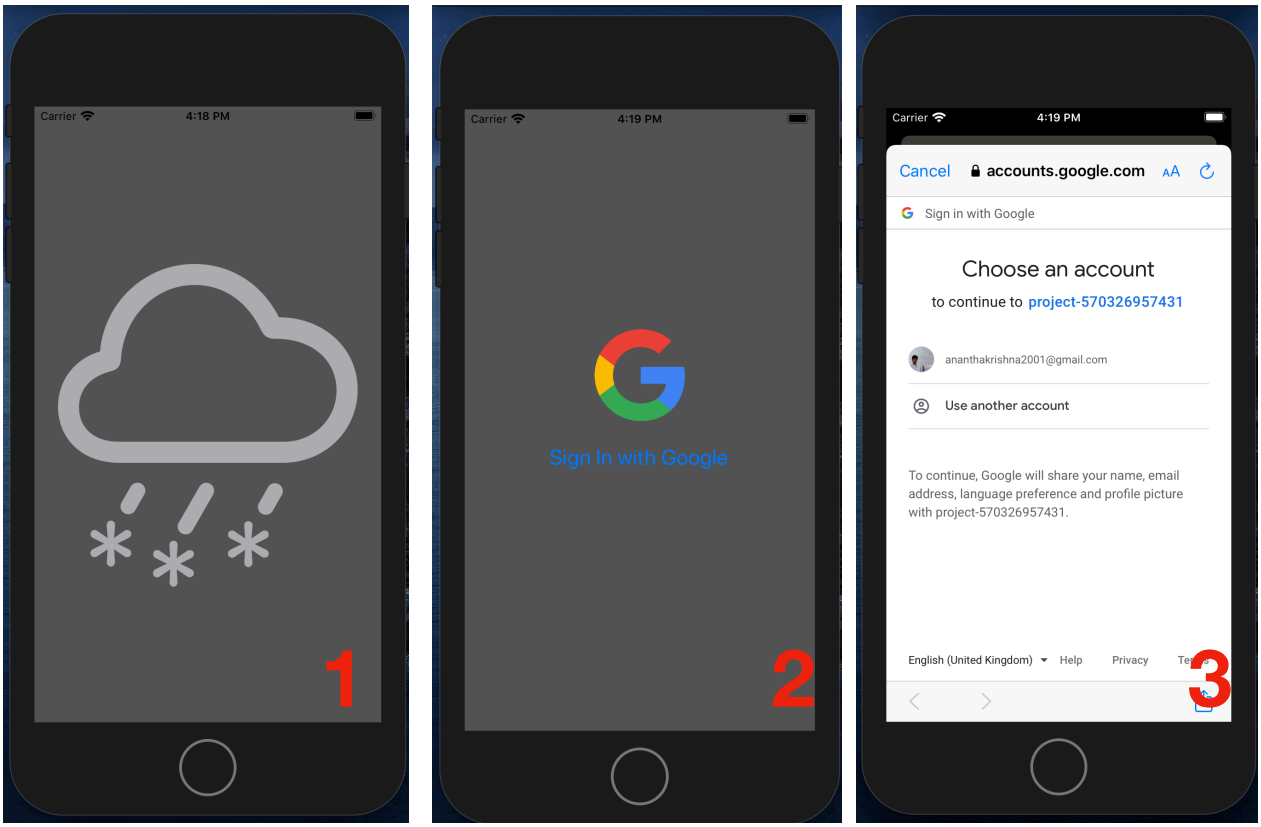
4.3 In Depth Explanation of the Software

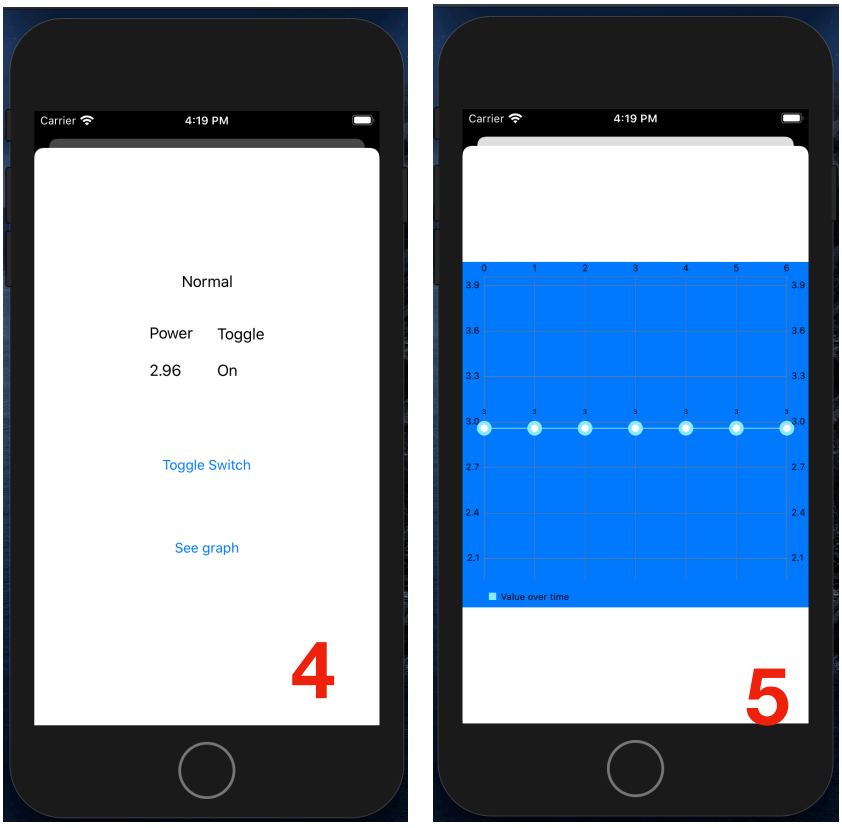
1. The Real-Time Database



- This is the Database Console that the user can view to get the raw data without any UI.
- The database is named Manisir and it has 2 Childs.
- The Childs are named Power and Toggle Respectively.
- The Power child receives info from the ESP32 and sends that info to the app where it is displayed and the real time data is plotted, the info received is the processed raw values i.e the final Power reading of the Appliance connected.
- The Toggle child is helps the ESP understand the mode under which it is working. The App can control the toggle child to let the user change it according to their wish.

2. The App


Carrier
4:18 PM
1
Carrier
4:19 PM
Sign In with Google
2
Carrier
4:19 PM
Cancel
accounts.google.com
AA
Sign in with Google
Choose an account
to continue to project-570326957431
ananthakrishna2001@gmail.com
Use another account
To continue, Google will share your name, email address, language preference and profile picture with project-570326957431.
English (United Kingdom)
Help
Privacy
3


Carrier
4:19 PM
Normal
Power
Toggle
2.96
On
Toggle Switch
See graph
4
Carrier
4:19 PM
Value over time
5

1 This is the App's Landing Page. The User will see that image as soon as he clicks on the App.

2 This is the UI for the FireBase Authentication.

3 This is the FireBase Authentication Taking place.

4 After the Authentication is over the user is led to the main screen where he can see the power value and the status of the system with the 3 condition switch.

5 After pressing on the "See Graph" button the user us directed to this page where they can see a realtime graph of their device.

Functioning:

- The App runs on **Xcode 13.0.1 SDK** Using The **Swift Standard Library** as its Framework.
- Uses Cocoa Pods as its dependancy manager to integrate Swift with FireBase SDK.
- The App natively runs on IOS as it was developed on Xcode Using Swift

App's Functioning File by File

(All the codes will be available in the Annexure_codes file attached with the report)

1. FireBase Authentication Code

```
    func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!, withError error: Error?) {
         // ...
         if let error = error {
            print(error.localizedDescription)
           return
         }

         guard let authentication = user.authentication else { return }
         let credential = GoogleAuthProvider.credential(withIDToken: authentication.idToken,accessToken:
             authentication.accessToken)

        Auth.auth().signIn(with: credential) { (AuthDataResult, Error) in
            if let error = Error{
            print(error)
            }

//            print(user.profile.name)
//            print(user.profile.familyName)
        }


        print("Successfully signed in using google")

        self.dismiss(animated: true) {
         self.performSegue(withIdentifier: "Test", sender: self)
        }

           // MARK: Firebase Sign In

       }
```

-> We used user generated tokens to authenticate users through firebase. This gives us the flexibility to Authenticate users faster while preserving the security of the process.

-> If the token was generated successfully then we get a message prompting us that the sign in was successful.

-> If token wasn't generated due to any issue it redirects it back to function where the system will try to generate another token.

2. Getting the Status of the System

```
func getStatus(){
    if (Double(self.powerString!) == 0.0)
    {
        if(self.toggleString! == "On")
        {
            self.statusLabel.text! = "Fused"
        }
        else if(self.toggleString! == "Off"){
            self.statusLabel.text! = "Normal"
        }
        else {
            self.statusLabel.text! = "Functioning in Auto Mode"
        }
    }
    else
    {
        if(self.toggleString! == "On"){
            self.statusLabel.text! = "Normal"
        }
        else if(self.toggleString! == "Off"){
            self.statusLabel.text! = "Switch Faulty"
        }
        else {
            self.statusLabel.text! = "Functioning in Auto Mode"
        }
    }
}
```

-> This part of the code checks all the test cases and displayed it as a label on the app.

4 Cases:

1. Fused (Switch **ON** but Power **0**)
2. Normal (Normal Functioning)
3. Switch Faulty (Switch **OFF** but Power **not 0**)
4. Functioning in Auto mode (Functioning independently)

3. Virtual 3 condition switch for the system

```
@IBAction func toggleSwitchPressed(_ sender: Any) {
    if(self.toggleString == "On")
    {
        self.ref.child("Power").setValue(0.0)
        self.ref.child("Toggle").setValue("Off")
    }
    else if(self.toggleString == "Off")
    {
        // change that value to float if needed
      self.ref.child("Power").setValue(5.0)
      self.ref.child("Toggle").setValue("Auto")
    }
    else
    {
        // change that value to float if needed
      self.ref.child("Power").setValue(5.0)
      self.ref.child("Toggle").setValue("On")
    }


}
```

-> This Part of the code enables the user to toggle the system to either On, Off or Auto.
-> By default whenever we set the system to On we have given a value of 5 so as to give the system a memory to join to and to avoid an app crash due to that reason.

4. Getting the values from the Realtime Database

```
    func getFirebaseData(_ ref:DatabaseReference)
    {
        ref.child("Power").observeSingleEvent(of: .value) { (DataSnapshot) in
            let val:Double?
//            print(DataSnapshot.value)
            val = DataSnapshot.value as? Double
            if let val = val{
                self.powerString = String(val)
            }

        }
        ref.child("Toggle").observeSingleEvent(of: .value) { (DataSnapshot) in

            let val:String?
//            print(DataSnapshot.value)
            val = DataSnapshot.value as? String
            if let val = val{
                self.toggleString = val
            }
        }
    }
```

-> The DataBase Reference gets the Database key of our realtime DataBase and pulls requests for both the Childs.
-> Both values of the Childs are converted to String type so as to enable easier readability.
-> The observeSingleEvent will keep checking that specific child for a change so as to save it in the memory of the variable assigned

5. Plotting the Realtime Chart for the Power Value

```
func chartValueSelected(_ chartView: ChartViewBase, entry: ChartDataEntry, highlight: Highlight) {
    print(entry)
}
func setData()
{
    let set1 = LineChartDataSet(entries: yValues, label: "Value over time")
    let data = LineChartData(dataSet:set1)
    lineChartView.data = data
}

var yValues:[ChartDataEntry] = []

func getFirebaseData(_ ref:DatabaseReference)
{

    ref.child("Power").observeSingleEvent(of: .value) { (DataSnapshot) in

        var rawData:Double?
        rawData = DataSnapshot.value as? Double

        if let doubleFromString = rawData
        {
            print(doubleFromString)
            self.yValues.append(ChartDataEntry(x:self.counter,y:doubleFromString))
            self.counter+=1
        }
    }

}
```

-> chartValueSelected is the function name we have given for the real time graph.
-> This snippet of code gets the data from firebase and then plot it w.r.t to time and display it real time.

3. ESP32 Code

This part of the code includes the specific libraries needed and initialises the database with the Authentication key.

```
#include <WiFi.h>
#include <IOXhop_FirebaseESP32.h>
#define FIREBASE_HOST "manisir-84cd8.firebaseio.com"
#define FIREBASE_AUTH "ReiazJsk0Tna5G10dYXk0smosv1GH4FN14FwuSW5"
#define WIFI_SSID "vivo V9 1723"
#define WIFI_PASSWORD "esp32devkitv1"
```

This part of the code defines all the pins of the hardware connected and initialises all the variables needed.

```
//------------DEFINING PINS---------
int relayPin = 23;
const int currentPin = 33;//ADC2 PIN
const int ldrPin = 32;//ADC2 PIN
//---------------------------------
const int voltage=12;//DC power supply
int sensitivity = 185;//5A ACS712
int offsetVoltage = 2500;//5000mV/2
//---------------------------------
int adcValue= 0;
double adcVoltage = 0;
double currentValue = 0;
float power=0.0;
String toggle="";
float maxcurrent=0.0,mincurrent=0.0,finalcurrent=0.0;
//---------------------------------
```

This part of the code is the setup loop of the board. It runs only once and does the essential wifi setup and secures a connection between the hardware and the realtime database through FireBase proxies.

```
void setup() {
  pinMode(ldrPin, INPUT);
  pinMode(relayPin, OUTPUT);
  Serial.begin(9600);
  delay(2000);

  WiFi.begin(WIFI_SSID,WIFI_PASSWORD);
  Serial.print("Connecting to..");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
   }
  Serial.println();
  Serial.print("Connected to ");
  Serial.println(WIFI_SSID);
  Serial.print("IP Address is : ");
  Serial.println(WiFi.localIP());//print local IP address
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  check();

}
```

This part of the code sets the toggle variable to the toggle child of the database and gets a reading from the LDR so that the system can run on Auto mode if needed.

```
void loop()
{

    toggle=Firebase.getString("Toggle");
    Serial.println(toggle);
    int ldrStatus = analogRead(ldrPin);
    if (isnan(ldrStatus))
    {
      Serial.println("Failed to read from LDR sensor!");
      return;
    }
```

This part of the code is for the system to function on Auto. This basically means the system functions as an automatic light source based the LDR value.

```
if(toggle =="Auto")
{
   if(ldrStatus>800)//bright -> turn OFF the lights
    {
      Serial.print("Light Intensity ");//printing the intensity of light
      Serial.println(ldrStatus);
      digitalWrite(relayPin, LOW); //deactivating the relay
      Serial.println("Light OFF");
      power =0;
      Serial.print("Power(W)");//printing power
      Serial.println(power);
      Firebase.set("/Power",power);
    }
   else//dark ->turn ON the lights
    {
      Serial.print("Light Intensity ");
      Serial.println(ldrStatus);
      digitalWrite(relayPin, HIGH);
      Serial.println("Light ON");
      finalcurrent=abs(maxcurrent-mincurrent);
      Serial.print("\t Final Current(A) = ");
      Serial.println(finalcurrent,3);
      power=voltage*finalcurrent;
      Serial.print("\t Power(W) = ");
      Serial.println(power,3);
      Firebase.set("/Power",power);
    }
}
```

This part of the code is for the system to function given the toggle value from the user.
Based upon toggle the system functions and calculates the power value
The power calculation code is the snapshot up next.

```
else if(toggle == "On")
{
  Serial.print("Light Intensity ");
  Serial.println(ldrStatus);
  digitalWrite(relayPin, HIGH);
  Serial.println("Light ON");
  finalcurrent=abs(maxcurrent-mincurrent);
  Serial.print("\t Final Current(A) = ");
  Serial.println(finalcurrent,3);
  power=voltage*finalcurrent;
  Serial.print("\t Power(W) = ");
  Serial.println(power,3);
  Firebase.set("/Power",power);
}
else if(toggle == "Off")
{
  Serial.print("Light Intensity ");
  Serial.println(ldrStatus);
  digitalWrite(relayPin, LOW);
  Serial.println("Light OFF");
  power = 0;
  Serial.print("Power(W)");
  Serial.println(power);
  Firebase.set("/Power",power);
}
```

This part of the code shows how the analog read gets the raw voltage value and scales it into a value that is dependent upon the bit rate and the resolution scope of the ESP32 processor and the Current sensor respectively.
After doing that it displays the power as a float value.
The sensitivity of the ACS712 is obtained from the data sheet as mentioned in section 4.1.
The Data sheet snap shots are in the References Section.

```
float calculations()
{
  float Samples=0.0,AcsValue=0.0,AvgAcs=0.0;
  for (int x = 0; x < 150; x++)//Get 150 samples
  {
  AcsValue = analogRead(currentPin);//Reads current sensor values
  Samples = Samples + AcsValue;  //Adds samples together
  delay (3); // lets ADC settle before next sample with a time gap of 3ms
  }
  AvgAcs=Samples/150.0;//Taking Average of 150 samples
  adcVoltage = (AvgAcs / 4096) * 3300;//0-4095 being range sensed by ESP32(12 bit resolution of the analog pins)
  currentValue = abs((adcVoltage - offsetVoltage) / sensitivity);
  Serial.print("Raw Sensor Value = " );
  Serial.print(AvgAcs);
  Serial.print("\t Voltage(mV) = ");
  Serial.print(adcVoltage,3);
  Serial.print("\t Current = ");
  Serial.println(currentValue,3);
  return(currentValue);
}
```

4. The Firebase Library for ESP32

The by default Firebase library as downloaded from an open source developer on GitHub lacked the functionality we needed to push the float value of the Power variable onto the power child of the Database hence we re-wrote the main.cpp file of the library to get it working up to our standards.

Added line 82-85 so as to add the set functionality of this library for our data types.

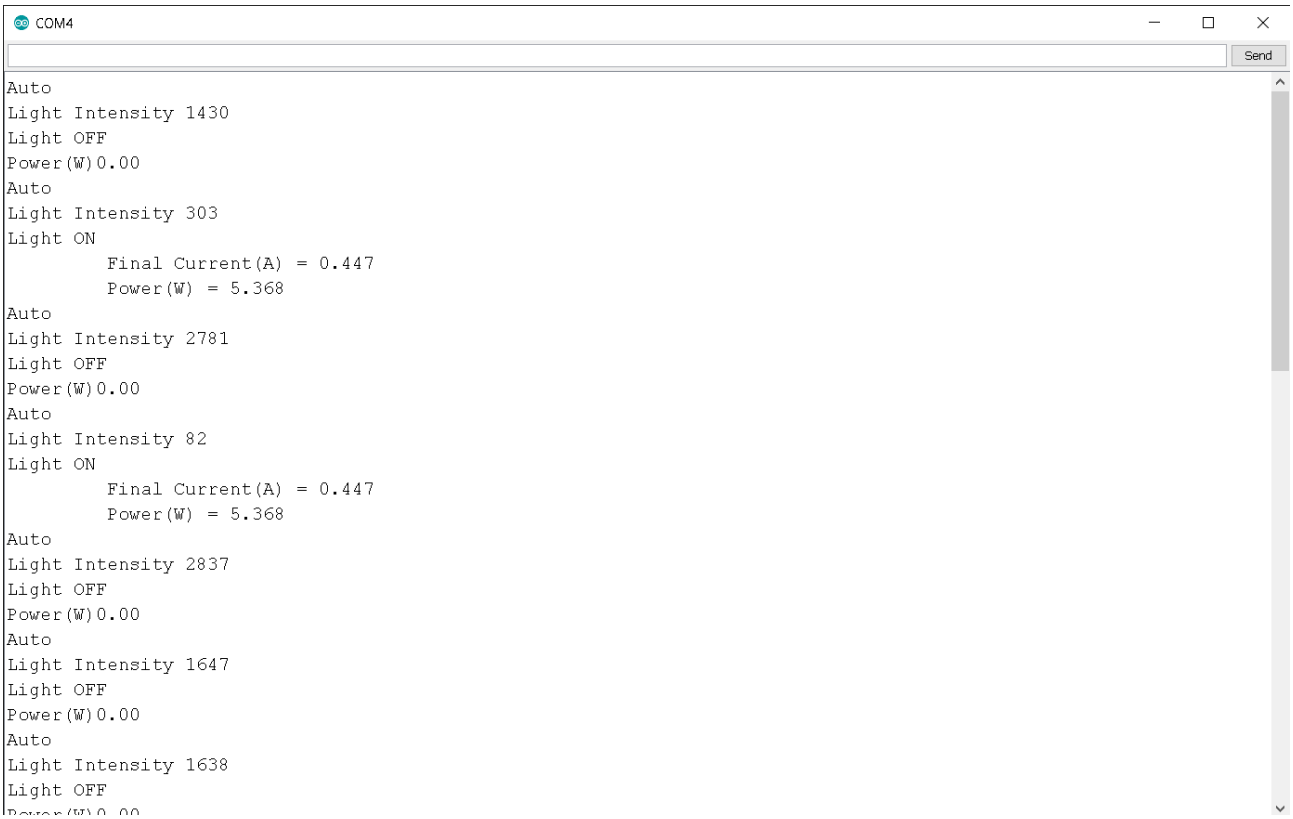
```
82
83  void FirebaseESP32::set(String path, int value) {
84      _http(path, "PUT", String(value));
85  }
```

Results

The Prototype was successfully built and is working efficiently.

The User uses the app to start the system and give his choice of toggle for the switch and the system does the rest all the things from getting the toggle input to calculating the power value of the appliance.

The Snapshot pf the serial monitor to verify. The results is as below.



As Expected the Power values were received with a small error accounting for the voltage drop of the wires, stray capacitance of bread board and the linearity of ACS712

The Power shown = 5.368 W

The error = 0.15W

This matches the readings of the power derived from the measurement of the Multimeter

References

1. https://en.wikipedia.org/wiki/ESP32

   https://randomnerdtutorials.com/ESP32-adc-analog-read-arduino-ide/

   https://www.espressif.com/sites/default/files/documentation/ESP32_datasheet_en.pdf

2. ACS712 Datasheet

SELECTION GUIDE

Part Number	Packing*	T_A (°C)	Optimized Range, I_P (A)	Sensitivity, Sens (Typ) (mV/A)
ACS712ELCTR-05B-T	Tape and reel, 3000 pieces/reel	–40 to 85	±5	185
ACS712ELCTR-20A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±20	100
ACS712ELCTR-30A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±30	66

*Contact Allegro for additional packing options.

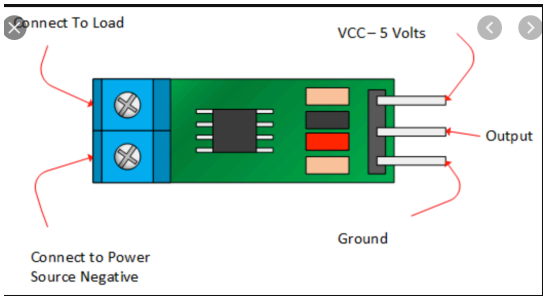
DESCRIPTION

The Allegro™ ACS712 provides economical and precise solutions for AC or DC current sensing in industrial, commercial, and communications systems. The device package allows for easy implementation by the customer. Typical applications include motor control, load detection and management, switch-mode power supplies, and overcurrent fault protection. The device is not intended for automotive applications.

The device consists of a precise, low-offset, linear Hall circuit with a copper conduction path located near the surface of the die. Applied current flowing through this copper conduction path generates a magnetic field which the Hall IC converts into a proportional voltage. Device accuracy is optimized through the close proximity of the magnetic signal to the Hall transducer. A precise, proportional voltage is provided by the low-offset, chopper-stabilized BiCMOS Hall IC, which is programmed for accuracy after packaging.

FEATURES AND BENEFITS

- Low-noise analog signal path
- Device bandwidth is set via the new FILTER pin
- 5 µs output rise time in response to step input current
- 80 kHz bandwidth
- Total output error 1.5% at T_A = 25°C
- Small footprint, low-profile SOIC8 package
- 1.2 mΩ internal conductor resistance
- 2.1 kVRMS minimum isolation voltage from pins 1-4 to pins 5-8
- 5.0 V, single supply operation
- 66 to 185 mV/A output sensitivity
- Output voltage proportional to AC or DC currents
- Factory-trimmed for accuracy
- Extremely stable output offset voltage
- Nearly zero magnetic hysteresis
- Ratiometric output from supply voltage



3. DC relay 5A Manuals and how to use guide - Circuitdigest

https://circuitdigest.com/article/relay-working-types-operation-applications

https://relaypros.com/wiring_diagrams.htm

4. Troubleshooting R Value to attach with an LDR

https://arduino.stackexchange.com/questions/16525/why-should-i-put-a-10k-resistor-with-an-ldr

5. Blue LED strip physical attributes with data sheet

https://www.applelite.net/LED-flexible-strips.html

6. 12V DC Supply the In's and Out's

https://en.wikipedia.org/wiki/Adapter

7. Troubleshooting voltage drops for Logic shifter - SparkFun

https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide/all

8. Voltage drop for jumper cables. -Arduino stackexchnage

https://forum.arduino.cc/index.php?topic=611226.0

9. Breadboard voltage drop and stray capacitance values

https://www.electro-tech-online.com/threads/solderless-breadboard-problems-and-warnings.145975/

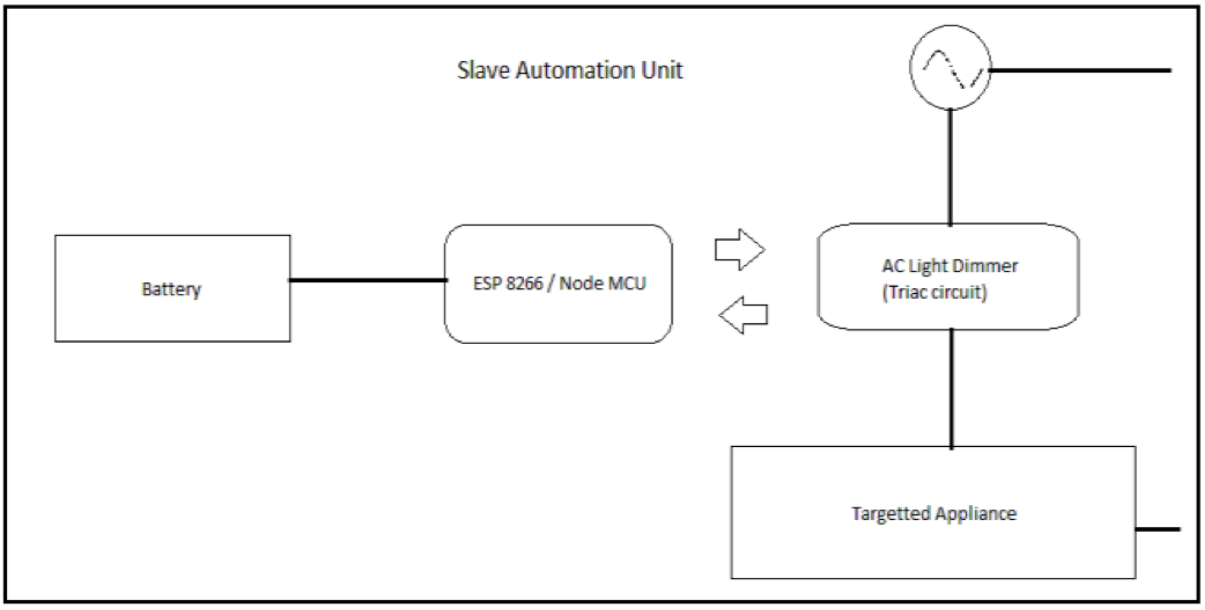
10. ACS712 Datasheet

11. https://www.youtube.com/watch?v=nnXWeecWWlM

12. FireBase Documentation. - Google

13. Swift Documentation - swift.org

14. Smart Home Automation and Security system - Siddharth Wadhwani
http://ijsrcseit.com/paper/CSEIT1952267.pdf



Future Scope

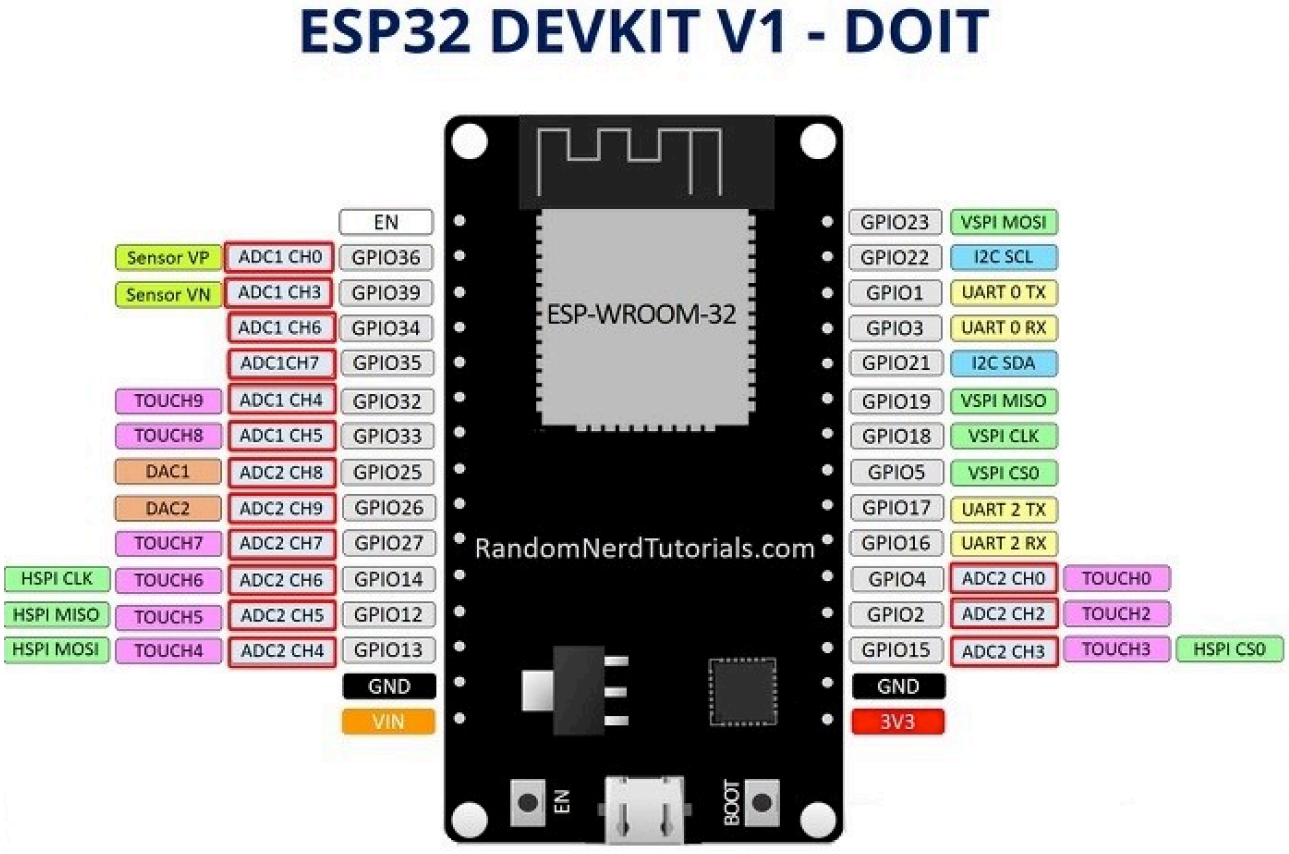
The Future Scope for projects in these domains are vast as their field always has a scope for growth and improvement .

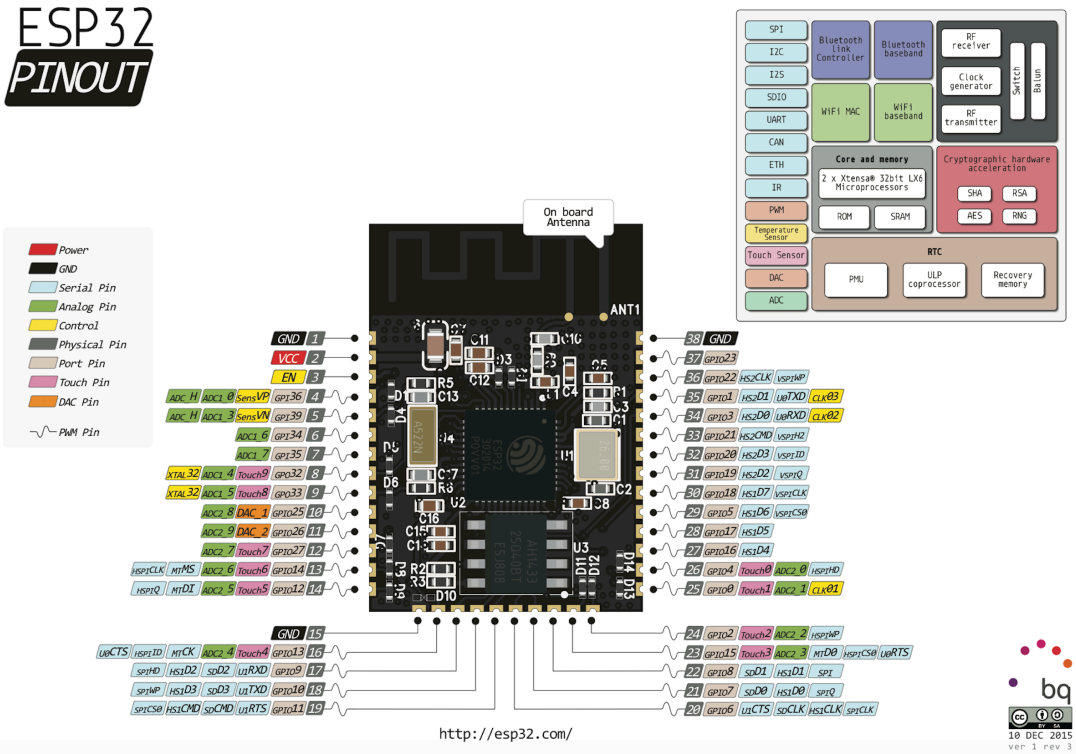
We can improve this project in the future by implementing these ideas/ methodologies:

- Can be expanded to Industrial size automation that can work in line with SCADA systems.
- Can be integrated with existing smart home devices i.e Alexa, HomePod, Google home to enable it to work with Voice commands.
- The app can be used as a backbone to construct more complex applications after upgrading the existing Database.

These are some of the ways(but not limited to) that this Project can be expanded in the upcoming future.

Appendix


ESP32 DEVKIT V1 - DOIT
ESP-WROOM-32
RandomNerdTutorials.com
EN
BOOT
EN
Sensor VP
Sensor VN
TOUCH9
TOUCH8
DAC1
DAC2
TOUCH7
HSPI CLK
HSPI MISO
HSPI MOSI
TOUCH6
TOUCH5
TOUCH4
ADC1 CH0
ADC1 CH3
ADC1 CH6
ADC1CH7
ADC1 CH4
ADC1 CH5
ADC2 CH8
ADC2 CH9
ADC2 CH7
ADC2 CH6
ADC2 CH5
ADC2 CH4
GPIO36
GPIO39
GPIO34
GPIO35
GPIO32
GPIO33
GPIO25
GPIO26
GPIO27
GPIO14
GPIO12
GPIO13
GND
VIN
GPIO23
GPIO22
GPIO1
GPIO3
GPIO21
GPIO19
GPIO18
GPIO5
GPIO17
GPIO16
GPIO4
GPIO2
GPIO15
GND
3V3
VSPI MOSI
I2C SCL
UART 0 TX
UART 0 RX
I2C SDA
VSPI MISO
VSPI CLK
VSPI CS0
UART 2 TX
UART 2 RX
ADC2 CH0
ADC2 CH2
ADC2 CH3
TOUCH0
TOUCH2
TOUCH3
HSPI CS0


ESP32
PINOUT
Power
GND
Serial Pin
Analog Pin
Control
Physical Pin
Port Pin
Touch Pin
DAC Pin
PWM Pin
On board Antenna
ANT1
SPI
I2C
I2S
SDIO
UART
CAN
ETH
IR
PWM
Temperature Sensor
Touch Sensor
DAC
ADC
Bluetooth link Controller
Bluetooth baseband
WiFi MAC
WiFi baseband
RF receiver
Clock generator
RF transmitter
Switch
Balun
Core and memory
2 x Xtensa® 32bit LX6 Microprocessors
ROM
SRAM
Cryptographic hardware acceleration
SHA
RSA
AES
RNG
RTC
PMU
ULP coprocessor
Recovery memory
http://esp32.com/
10 DEC 2015
ver 1 rev 3

* **https://www.digitalinformationworld.com/2019/02/internet-users-spend-more-than-a-quarter-of-their-lives-online.html**

ANNEXURE OF CODES FROM THE NEXT PAGE

```swift
//
//  ViewController.swift
//  specialTopicSem4
//
//  Created by Anantha Krishna on 02/03/20.
//  Copyright © 2020 Anantha Krishna. All rights reserved.
//

import UIKit
import FirebaseDatabase
import Firebase
import GoogleSignIn
import FirebaseAuth

class ViewController: UIViewController,GIDSignInDelegate{


    @IBOutlet weak var signInButton:GIDSignInButton!


    override func viewDidLoad() {
        super.viewDidLoad()
        GIDSignIn.sharedInstance().delegate = self
        // Do any additional setup after loading the view.
    }


    @IBAction func signInButtonPressed(_ sender: Any) {

        GIDSignIn.sharedInstance()?.presentingViewController = self
        GIDSignIn.sharedInstance().signIn()
    }

    func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!,
     withError error: Error?) {
         // ...
         if let error = error {
            print(error.localizedDescription)
           return
         }

         guard let authentication = user.authentication else { return }
         let credential = GoogleAuthProvider.credential(withIDToken:
          authentication.idToken,accessToken: authentication.accessToken)

        Auth.auth().signIn(with: credential) { (AuthDataResult, Error) in
            if let error = Error{
            print(error)
            }

//            print(user.profile.name)
//            print(user.profile.familyName)
        }
```

```
        print("Successfully signed in using google")

        self.dismiss(animated: true) {
         self.performSegue(withIdentifier: "Test", sender: self)
        }

           // MARK: Firebase Sign In

       }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "Test"
        {
// no data required to be passed
//            let destinationVC = segue.destination as!
 AfterSignInViewController

        }


    }
    func moveToNextView()
    {
        self.performSegue(withIdentifier: "Test", sender: self)
    }


}
```

```
//
//  AfterSignInViewController.swift
//  specialTopicSem4
//
//  Created by Anantha Krishna on 22/03/20.
//  Copyright © 2020 Anantha Krishna. All rights reserved.
//

import UIKit
import Firebase
import FirebaseDatabase

class AfterSignInViewController: UIViewController {




    @IBOutlet weak var statusLabel: UILabel!
    @IBOutlet weak var powerValue: UILabel!

    @IBOutlet weak var toggleValue: UILabel!

    var ref:DatabaseReference!
    var data:FirebaseData?
    var powerString:String?
    var toggleString:String?
    var timer:Timer?
    override func viewDidLoad() {
        super.viewDidLoad()
        ref = Database.database().reference()
        getFirebaseData(ref)
        // change the delay to 1s but proceed with caution
        timer = Timer.scheduledTimer(withTimeInterval: 1.0, repeats: true,
         block: { (Timer) in
             print("Inside dispatch ",self.powerString)
                      if let powerString = self.powerString
                      {
                          print("typecasted\n")
                          self.powerValue.text! = powerString
                          self.toggleValue.text! = self.toggleString!
                          self.getStatus()


                      }
             self.getFirebaseData(self.ref)
        })

    }

    func getStatus(){
        if (Double(self.powerString!) == 0.0)
        {
            if(self.toggleString! == "On")
            {
                self.statusLabel.text! = "Fused"
```

```
            }
            else if(self.toggleString! == "Off"){
                self.statusLabel.text! = "Normal"
            }
            else {
                self.statusLabel.text! = "Functioning in Auto Mode"
            }
        }
        else
        {
            if(self.toggleString! == "On"){
                self.statusLabel.text! = "Normal"
            }
            else if(self.toggleString! == "Off"){
                self.statusLabel.text! = "Switch Faulty"
            }
            else {
                self.statusLabel.text! = "Functioning in Auto Mode"
            }
        }
    }


    @IBAction func buttonPressed(_ sender: UIButton) {
        if(sender.titleLabel?.text! != "Button Disabled")
        {
         performSegue(withIdentifier: "ChartViewSegue", sender: self)
        }
    }

    @IBAction func toggleSwitchPressed(_ sender: Any) {
        if(self.toggleString == "On")
        {
            self.ref.child("Power").setValue(0.0)
            self.ref.child("Toggle").setValue("Off")
        }
        else if(self.toggleString == "Off")
        {
            // change that value to float if needed
          self.ref.child("Power").setValue(5.0)
          self.ref.child("Toggle").setValue("Auto")
        }
        else
        {
            // change that value to float if needed
          self.ref.child("Power").setValue(5.0)
          self.ref.child("Toggle").setValue("On")
        }


    }
    @objc func display(){

//                    print("Inside dispatch ",self.powerString)
                  if let powerString = self.powerString
```

```
                    {
//                      print("typecasted\n")
                        self.powerValue.text! = powerString
                        self.toggleValue.text! = self.toggleString!

                    }
    }



    func getFirebaseData(_ ref:DatabaseReference)
    {
        ref.child("Power").observeSingleEvent(of: .value) { (DataSnapshot)
         in
            let val:Double?
//            print(DataSnapshot.value)
            val = DataSnapshot.value as? Double
            if let val = val{
                self.powerString = String(val)
            }

        }
        ref.child("Toggle").observeSingleEvent(of: .value) { (DataSnapshot)
         in

            let val:String?
//            print(DataSnapshot.value)
            val = DataSnapshot.value as? String
            if let val = val{
                self.toggleString = val
            }
        }
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        if (segue.identifier == "ChartViewSegue")
        {
            // no data to be passed from this view to the other so this is
             empty
        }
    }



}
```

```swift
//
//  ChartViewController.swift
//  specialTopicSem4
//
//  Created by Anantha Krishna on 04/04/20.
//  Copyright © 2020 Anantha Krishna. All rights reserved.
//

import UIKit
import Charts
import TinyConstraints
import Firebase

class ChartViewController: UIViewController ,ChartViewDelegate{
    lazy var lineChartView: LineChartView = {
        let chartView = LineChartView()
        chartView.backgroundColor = .systemBlue;
        return chartView
    }()

    var deviceType:String?
    var counter:Double = 0.0

    var ref:DatabaseReference!

    override func viewDidLoad() {
        super.viewDidLoad()
        view.addSubview(lineChartView)
        lineChartView.centerInSuperview()
        lineChartView.width(to: view)
        lineChartView.heightToWidth(of: view)
        ref = Database.database().reference()
        getFirebaseData(ref)
        setData()
        DispatchQueue.main.asyncAfter(deadline: .now()+1) {
            self.viewDidLoad()
        }

        // Do any additional setup after loading the view.
    }

    func chartValueSelected(_ chartView: ChartViewBase, entry:
     ChartDataEntry, highlight: Highlight) {
        print(entry)
    }
    func setData()
    {
        let set1 = LineChartDataSet(entries: yValues, label: "Value over
         time")
        let data = LineChartData(dataSet:set1)
        lineChartView.data = data
    }

    var yValues:[ChartDataEntry] = []
```

```
    func getFirebaseData(_ ref:DatabaseReference)
    {

        ref.child("Power").observeSingleEvent(of: .value) { (DataSnapshot)
         in

            var rawData:Double?
            rawData = DataSnapshot.value as? Double

            if let doubleFromString = rawData
            {
                print(doubleFromString)
                self.yValues.append(ChartDataEntry(x:self
                 .counter,y:doubleFromString))
                self.counter+=1
            }
        }

    }

    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a
     little preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        // Get the new view controller using segue.destination.
        // Pass the selected object to the new view controller.
    }
    */

}
```

```swift
//
//  AppDelegate.swift
//  specialTopicSem4
//
//  Created by Anantha krishna on 02/03/20.
//  Copyright © 2020 Anantha Krishna. All rights reserved.
//

import UIKit
import Firebase
import GoogleSignIn

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {


     func application(_ application: UIApplication,
      didFinishLaunchingWithOptions launchOptions:
      [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        FirebaseApp.configure()
        GIDSignIn.sharedInstance().clientID =
         FirebaseApp.app()?.options.clientID

        return true
    }

    // MARK: UISceneSession Lifecycle

    func application(_ application: UIApplication,
     configurationForConnecting connectingSceneSession: UISceneSession,
     options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
        // Use this method to select a configuration to create the new
         scene with.
        return UISceneConfiguration(name: "Default Configuration",
         sessionRole: connectingSceneSession.role)
    }

    func application(_ application: UIApplication, didDiscardSceneSessions
     sceneSessions: Set<UISceneSession>) {
        // Called when the user discards a scene session.
        // If any sessions were discarded while the application was not
         running, this will be called shortly after
         application:didFinishLaunchingWithOptions.
        // Use this method to release any resources that were specific to
         the discarded scenes, as they will not return.
    }

    func application(_ app: UIApplication, open url: URL, options:
     [UIApplication.OpenURLOptionsKey : Any]) -> Bool {

        return GIDSignIn.sharedInstance().handle(url)
    }
```

```
}
```

```swift
//
//  SceneDelegate.swift
//  specialTopicSem4
//
//  Created by Anantha Krishna on 02/03/20.
//  Copyright © 2020 Anantha Krishna. All rights reserved.
//

import UIKit

class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?


    func scene(_ scene: UIScene, willConnectTo session: UISceneSession,
     options connectionOptions: UIScene.ConnectionOptions) {
        // Use this method to optionally configure and attach the UIWindow
         `window` to the provided UIWindowScene `scene`.
        // If using a storyboard, the `window` property will automatically
         be initialized and attached to the scene.
        // This delegate does not imply the connecting scene or session are
         new (see `application:configurationForConnectingSceneSession`
         instead).
        guard let _ = (scene as? UIWindowScene) else { return }
    }

    func sceneDidDisconnect(_ scene: UIScene) {
        // Called as the scene is being released by the system.
        // This occurs shortly after the scene enters the background, or
         when its session is discarded.
        // Release any resources associated with this scene that can be
         re-created the next time the scene connects.
        // The scene may re-connect later, as its session was not
         neccessarily discarded (see `application:didDiscardSceneSessions`
         instead).
    }

    func sceneDidBecomeActive(_ scene: UIScene) {
        // Called when the scene has moved from an inactive state to an
         active state.
        // Use this method to restart any tasks that were paused (or not
         yet started) when the scene was inactive.
    }

    func sceneWillResignActive(_ scene: UIScene) {
        // Called when the scene will move from an active state to an
         inactive state.
        // This may occur due to temporary interruptions (ex. an incoming
         phone call).
    }

    func sceneWillEnterForeground(_ scene: UIScene) {
        // Called as the scene transitions from the background to the
         foreground.
```

```
        // Use this method to undo the changes made on entering the
         background.
    }

    func sceneDidEnterBackground(_ scene: UIScene) {
        // Called as the scene transitions from the foreground to the
         background.
        // Use this method to save data, release shared resources, and
         store enough scene-specific state information
        // to restore the scene back to its current state.
    }


}
```

```
//
//  FirebaseData.swift
//  specialTopicSem4
//
//  Created by Anantha krishna on 24/03/20.
//  Copyright © 2020 Anantha Krishna. All rights reserved.
//

import Foundation
struct FirebaseData{
    var devicePower:String
    var deviceToggle:String
}
```

```
# Uncomment the next line to define a global platform for your project
platform :ios, '13.0'

target 'specialTopicSem4' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for specialTopicSem4
pod 'Firebase/Database'
pod 'Firebase/Auth'
pod 'GoogleSignIn'
pod 'Charts'
pod 'TinyConstraints'
# add pods for any other desired Firebase products
# https://firebase.google.com/docs/ios/setup#available-pods

end
```

```
#include <WiFi.h>
#include <IOXhop_FirebaseESP32.h>
#define FIREBASE_HOST "manisir-84cd8.firebaseio.com"
#define FIREBASE_AUTH "ReiazJsk0Tna5G10dYXk0smosv1GH4FN14FwuSW5"
#define WIFI_SSID "vivo V9 1723"
#define WIFI_PASSWORD "esp32devkitv1"
//------------DEFINING PINS---------
int relayPin = 23;
const int currentPin = 33;//ADC2 PIN
const int ldrPin = 32;//ADC2 PIN
//---------------------------------
const int voltage=12;//DC power supply
int sensitivity = 185;//5A ACS712
int offsetVoltage = 2500;//5000mV/2
//---------------------------------
int adcValue= 0;
double adcVoltage = 0;
double currentValue = 0;
float power=0.0;
String toggle="";
float maxcurrent=0.0,mincurrent=0.0,finalcurrent=0.0;
//---------------------------------
void setup() {
  pinMode(ldrPin, INPUT);
  pinMode(relayPin, OUTPUT);
  Serial.begin(9600);
  delay(2000);

  WiFi.begin(WIFI_SSID,WIFI_PASSWORD);
  Serial.print("Connecting to..");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
   }
  Serial.println();
  Serial.print("Connected to ");
  Serial.println(WIFI_SSID);
  Serial.print("IP Address is : ");
  Serial.println(WiFi.localIP());//print local IP address
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  check();

}

void loop()
{
```

```
toggle=Firebase.getString("Toggle");
Serial.println(toggle);
int ldrStatus = analogRead(ldrPin);
if (isnan(ldrStatus))
{
  Serial.println("Failed to read from LDR sensor!");
  return;
}
if(toggle =="Auto")
{
   if(ldrStatus>800)//bright -> turn OFF the lights
    {
      Serial.print("Light Intensity ");//printing the intensity of light
      Serial.println(ldrStatus);
      digitalWrite(relayPin, LOW); //deactivating the relay
      Serial.println("Light OFF");
      power =0;
      Serial.print("Power(W)");//printing power
      Serial.println(power);
      Firebase.set("/Power",power);
    }
   else//dark ->turn ON the lights
    {
      Serial.print("Light Intensity ");
      Serial.println(ldrStatus);
      digitalWrite(relayPin, HIGH);
      Serial.println("Light ON");
      finalcurrent=abs(maxcurrent-mincurrent);
      Serial.print("\t Final Current(A) = ");
      Serial.println(finalcurrent,3);
      power=voltage*finalcurrent;
      Serial.print("\t Power(W) = ");
      Serial.println(power,3);
      Firebase.set("/Power",power);
    }
}
else if(toggle == "On")
{
  Serial.print("Light Intensity ");
  Serial.println(ldrStatus);
  digitalWrite(relayPin, HIGH);
  Serial.println("Light ON");
  finalcurrent=abs(maxcurrent-mincurrent);
  Serial.print("\t Final Current(A) = ");
  Serial.println(finalcurrent,3);
  power=voltage*finalcurrent;
  Serial.print("\t Power(W) = ");
```

```
    Serial.println(power,3);
    Firebase.set("/Power",power);
   }
   else if(toggle == "Off")
   {
    Serial.print("Light Intensity ");
    Serial.println(ldrStatus);
    digitalWrite(relayPin, LOW);
    Serial.println("Light OFF");
    power = 0;
    Serial.print("Power(W)");
    Serial.println(power);
    Firebase.set("/Power",power);
   }

  delay(2000);
 }
float calculations()
{
  float Samples=0.0,AcsValue=0.0,AvgAcs=0.0;
  for (int x = 0; x < 150; x++)//Get 150 samples
  {
  AcsValue = analogRead(currentPin);//Reads current sensor values
  Samples = Samples + AcsValue;  //Adds samples together
  delay (3); // lets ADC settle before next sample with a time gap of 3ms
  }
  AvgAcs=Samples/150.0;//Taking Average of 150 samples
  adcVoltage = (AvgAcs / 4096) * 3300; //0-4095 being range sensed by ESP32(12 bit resolution of
the analog pins)
  currentValue = abs((adcVoltage - offsetVoltage) / sensitivity);
  Serial.print("Raw Sensor Value = " );
  Serial.print(AvgAcs);
  Serial.print("\t Voltage(mV) = ");
  Serial.print(adcVoltage,3);
  Serial.print("\t Current = ");
  Serial.println(currentValue,3);
  return(currentValue);
}
float check()
{
  digitalWrite(relayPin,HIGH);
  maxcurrent = calculations();
  delay(500);
  digitalWrite(relayPin,LOW);
  mincurrent = calculations();
  delay(500);
}
```

CODE FOR THE GRAPH

```
from firebase import firebase #importing the Firebase REST API
import matplotlib.pylab as plt # Importing the math plotting libraries
import datetime as datetime # Importing the date and time library to plot graph
firebase1 = firebase.FirebaseApplication('https://specialtopic-4e64d.firebaseio.com/', None)
#declaring a firebase variable
while(1):

	result = firebase1.get('/Status', None) # Fetching the power child
	print(result)
	print(result1)

	offcount = 0 #count the no of times the power is 0
	oncount = 0 # count the no of times the power is non zero

	for keys in result.keys():
		if(result[keys] == 'Off'):

			offcount = offcount+1 # Counting the Off's

		elif(result[keys] == 'On'):
			oncount  = oncount+1 # Counting the On's

	x = list(result.keys())
	x = x[-5:]
	y = list(result.values())
	y = y[-5:]

	labels1 = 'Off','On'
	sizes1 = [offcount1,oncount1]
	colors1 = ['lightcoral', 'lightskyblue']
	explode1 = (0.1, 0.1,)



	fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)

	ax1.plot(x, y,'ro') #plotting the value vs Time graph

	ax2.pie(sizes, explode=explode, labels=labels, colors=colors,autopct='%1.1f%%',
shadow=True, startangle=140)
	ax2.axis('equal')
```