1.1 Components

We begin by exploring the core hardware elements that form the backbone of a smart plug system. This includes microcontrollers, sensors, and communication modules, each selected for their unique capabilities and compatibility with the overall design.

1.1.1 ESP32

The ESP32 is a series of low-cost, low-power system-on-a-chip microcontrollers featuring integrated Wi-Fi and dual-mode Bluetooth capabilities. The ESP32 can establish connections with other systems to offer both Wi-Fi and Bluetooth functionality. It outperforms the Arduino as a more potent and capable microcontroller board. With built-in dual Wi-Fi and Bluetooth support, the ESP32 provides comprehensive TCP/IP support for complete internet connectivity. Thanks to its Wi-Fi module, it can operate as both an access point and a Wi-Fi station. The ESP32 pins are illustrated in figure 3.1. The specifications are given in table 3.1.

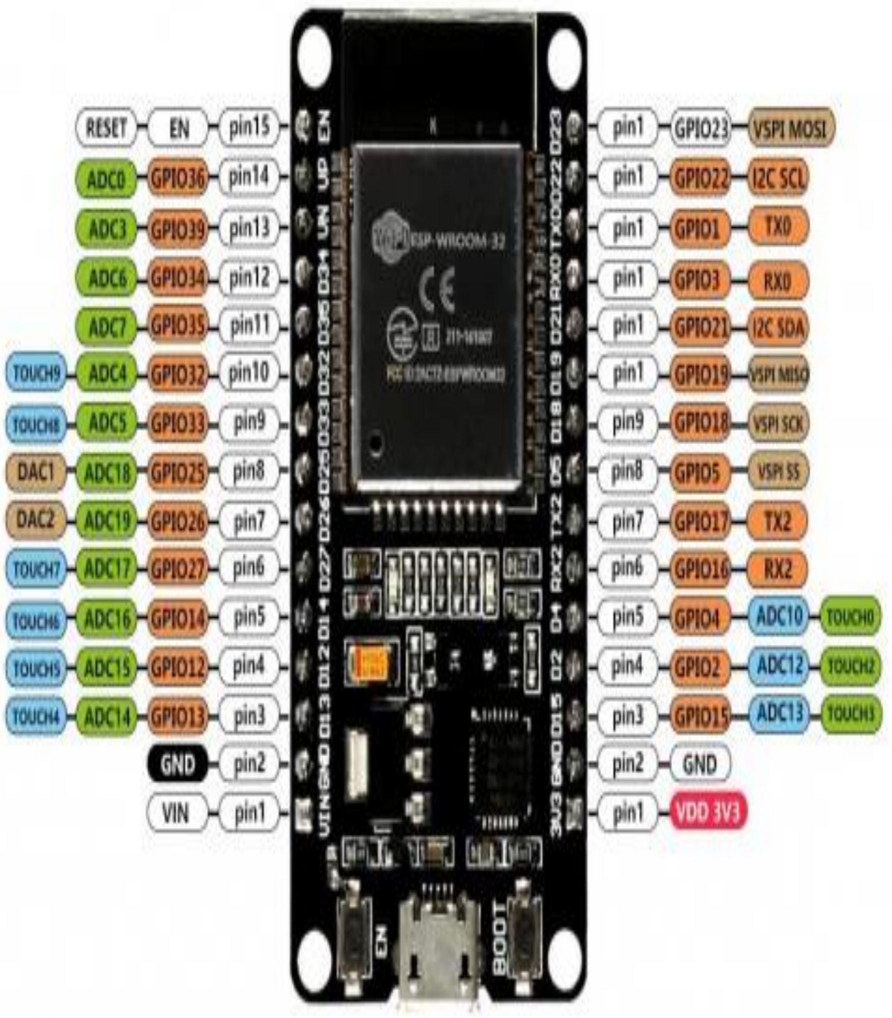

Figure ***Error! No text of specified style in document.****.1: ESP32 pins*

Table ***Error! No text of specified style in document.****.1: ESP-WROOM-32 Specifications*

Categories	Items	Specifications
	Module interface	SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR
		GPIO, capacitive touch sensor, ADC, DAC
Hardware	On-chip sensor	Hall sensor, temperature sensor

	On-board clock	40 MHz crystal
	Operating voltage/Power supply	2.7 ~ 3.6V
	Operating current	Average: 80 Ma
	Minimum current delivered by power supply	500 mA
	Operating temperature range	-40°C ~ +85°C
	Ambient temperature range	Normal temperature
	Package size	18±0.2 mm x 25.5±0.2 mm x 3.1±0.15 mm
Software	Wi-Fi mode	Station/SoftAP/SoftAP+Station/P2P
	Wi-Fi Security	WPA/WPA2/WPA2-Enterprise/WPS
	Encryption	AES/RSA/ECC/SHA
	Firmware upgrade	UART Download / OTA (download and write firmware via network or host)
	Software development	Supports Cloud Server Development / SDK for custom firmware development
	Network protocols	IPv4, IPv6, SSL, TCP/UDP/HTTP/FTP/MQTT
	User configuration	AT instruction set, cloud server, Android/iOS app

1.1.2 Temperature Sensor DHT11

The DHT11 shown in Figure 3.2 is a digital Humidity and Temperature Sensor comprising a resistive-type Humidity Sensor, an NTC-type Temperature Sensor, and an 8-bit Microcontroller. It can measure humidity in the range of 20% to 80% Relative Humidity and temperatures in the range of 0°C to 50°C. The microcontroller within the DHT11 Sensor handles all ADC-related functions and delivers the Digital data through a single wire. Furthermore, the DHT11

Temperature and Humidity Sensor can accommodate a cable length of up to 20 meters. This capability allows for the easy implementation of a wired sensor system in extended locations.

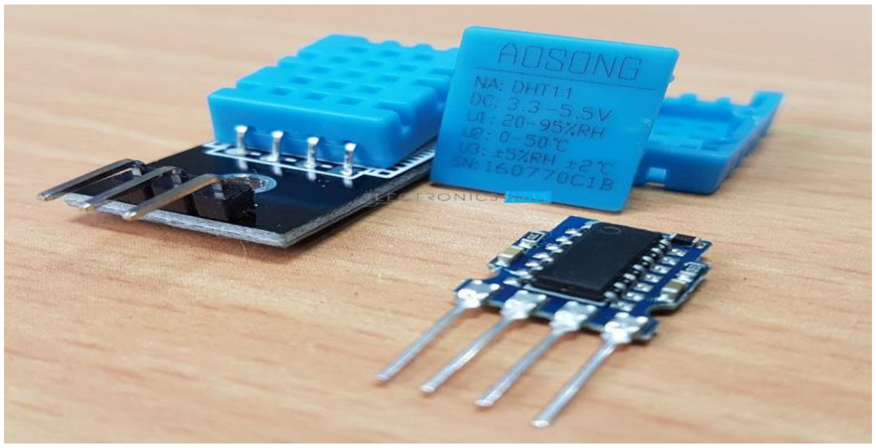

Figure ***Error! No text of specified style in document.****.2: DHT11 Temperature Sensor*

1.1.3 **ZMCT103C Current Sensor**

The ZMCT101C current sensor (Figure 3.3) is a 5A current transformer module designed for measuring AC current in various applications. It is based on the ZMCT103C IC, featuring a turn ratio of 1:2500. Equipped with small high-precision current transformers and precise op-amp circuits, this module ensures accurate sampling and proper signal compensation. It can interface with microcontrollers or Arduino UNO boards via VCC, OUT, and GND pins, and can also be used with a potentiometer to adjust the amplification factor. The module generates analog output signals proportional to the AC current flowing through the wire, making it suitable for current measurement, power monitoring, motor control, automation, and safety systems. Its features include high precision, wide range, low noise, easy usability, epoxy encapsulation, and operating voltage options of 5V or 3.3V. The ZMCT103C module's specifications include a rated input current of 5A, turn ratio of 1:2500, output voltage of 2.5mV/A, precision of 0.2%, noise level of 20µVrms, and compact dimensions of 18.3 x 17mm.

Figure ***Error! No text of specified style in document.****.3: ZMCT103C Current Sensor*

The technical parameters of this sensors are given in table 3.2.

Table ***Error! No text of specified style in document.****.2: The Main Technical Parameters of ZMCT103C*

Model	ZMCT103C
input current	0-10A（50Ω）
Rated output current	5mA at input 5A
turns ratio	1000:1
phase angle error	≤15′ (input 5A，sampling resistor 50Ω）
Accuracy class	0.2
Linearity	≤0.2%（5%dot~120%dot）
Permissible error	-0.2%≤f≤+0.2% (input 5A，sampling resistor50Ω）
isolation voltage	4500V
Application	Precise measurement of current and power
Encapsulation	Epoxy
Installation	PCB mounting Pin Length>3mm）
operating temperature	0℃~+85℃

<u>Direction for use:</u>

Figure 3.4 illustrates active output configuration, the ZMCT103C module can be interfaced with operational amplifiers (op-amps) to provide precise control and amplification of the output signal. This setup allows for low output impedance, which makes it less sensitive to load variations and improves signal quality. By using op-amps, the output signal can be adjusted and tailored to meet the needs of the system. Additionally, active output configurations can offer higher levels of precision and accuracy, making them suitable for applications where precise current measurements are critical. However, it's essential to note that designing and implementing an active output configuration may require more complex circuitry and careful consideration of components and circuit design.

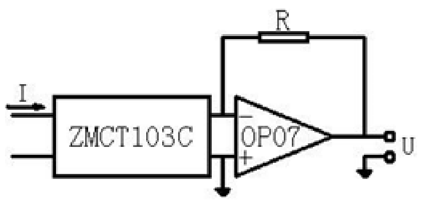

Figure ***Error! No text of specified style in document.****.4: Active Output*

Figure 3.5 illustrates passive output configuration. In this setup, the module's output is directly connected to the measuring or monitoring system without additional amplification or signal conditioning. Passive output configurations are relatively simpler to implement and are cost-effective. They are suitable for applications where high precision and amplification are not required, and the focus is on straightforward current measurements. However, passive output configurations may be more susceptible to load variations and noise compared to active configurations. They are typically used in applications where simplicity and cost efficiency are prioritized over precise signal control and amplification. (Figure 3.5).

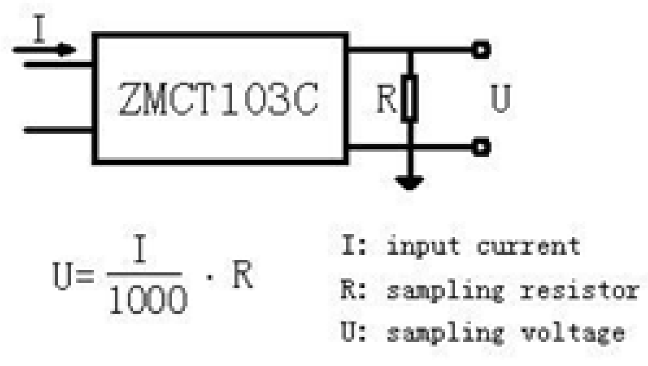

Figure ***Error! No text of specified style in document.****.5: The Passive Output*

1.1.4 **ZMPT101B AC Single Phase Voltage Sensor Module**

The ZMPT101B AC Single Phase voltage sensor module shown in figure 3.6 is built around the high-precision ZMPT101B voltage transformer, which accurately measures AC voltage using a voltage transformer. It's an excellent option for measuring AC voltage with Arduino or ESP32.

The module can measure voltages up to 250V AC, and the corresponding analogue output is adjustable. It's user-friendly, featuring a multi-turn trim potentiometer for calibrating and adjusting the AC output

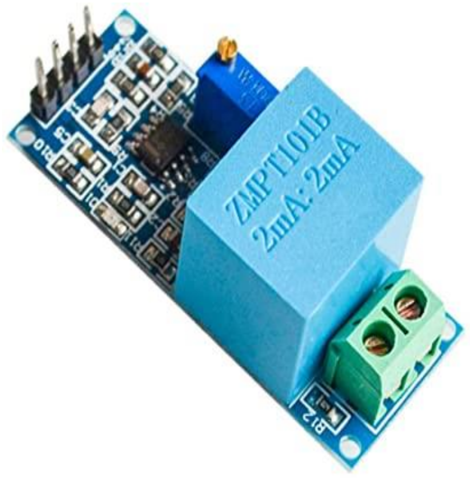

Figure ***Error! No text of specified style in document.****.6: The ZMPT101B AC Single Phase Voltage Sensor Module*

Structural Parameters：

Table ***Error! No text of specified style in document.****.3: Specifications for ZMPT101B sensor*

Model	ZMPT101B
Rated input current	2mA
Rated output current	2mA
turns ratio	1000:1000
phase angle error	≤20′ （input 2mA，sampling resistor 100Ω）
operating range	**0~1000V0~10mA（sampling resistor 100Ω）**
Linearity	≤0.2%(20%dot~120%dot)
Permissible error	-0.3%≤ f ≤+0.2%（input 2mA，sampling resistor 100Ω）
isolation voltage	4000V
Application	voltage and power measurement
Encapsulation	Epoxy
Installation	PCB mounting Pin Length>3mm）
Operating temperature	-40℃~+60℃
Case Material	**ABS (Note: ABS CASE is NOT available for wave-soldering)**

Direction for Use：

Figure 3.8 illustrates active output circuits, often employing operational amplifiers (op-amps), offer precise control, amplification capabilities, and low output impedance. They excel in driving heavy loads and performing complex signal conditioning tasks. However, active circuits are more complex to design, consume more power, and can be costlier due to the use of active components. The choice between passive and active output depends on factors such as required precision, power consumption, complexity, and budget considerations in specific applications.

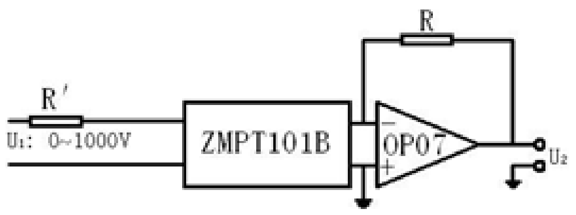

*Figure **Error! No text of specified style in document.**.7: Active Output*

Figure 3.9 illustrates Passive output circuits are known for their simplicity, cost-effectiveness, and stability in electronic circuits, especially in sensor interfacing and signal processing. These circuits, typically composed of resistors and capacitors, provide accurate outputs and are relatively straightforward to implement. However, they have limitations in terms of output range, sensitivity to load changes, and potential phase shifts with varying load resistance.

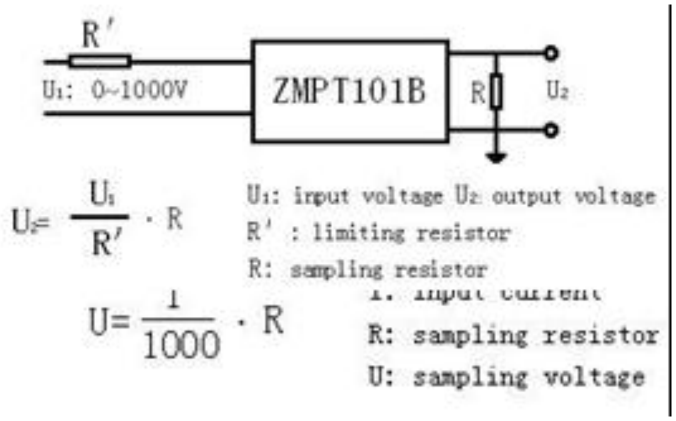

*Figure **Error! No text of specified style in document.**.8: Passive Output*

1.1.5 **2 Channel Relay Module**

Relay supports High-level Trigger or Low-Level Trigger selectable by a jumper.

Relay with Opti coupler Isolation.

Relay with Terminal Blocks for both Input and Output Interface.

Relay with two LED Indicators: power (green LED), the relay status (red LED).

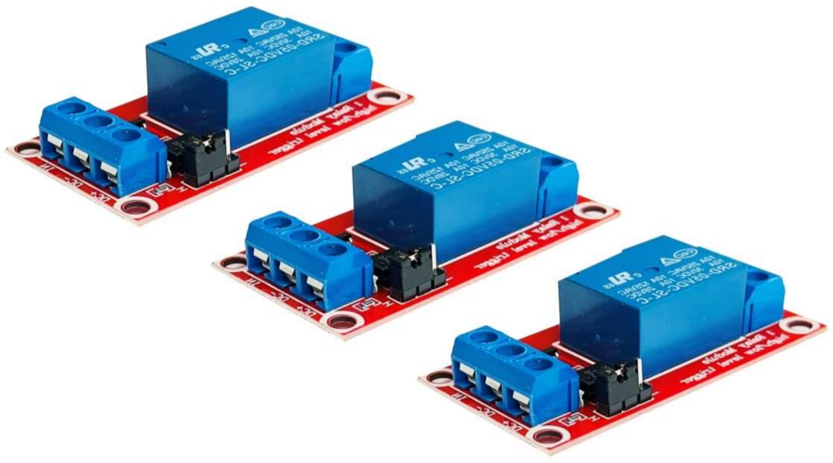

*Figure **Error! No text of specified style in document.**.9: Relay Module for Arduino, ESP32*

1.1.6 **Lithium-Ion Battery**

A lithium-ion battery (Figure 3.11) also known as a Li-ion battery, is a rechargeable energy storage device. These batteries find widespread use in portable electronics and electric vehicles.

Within a Li-ion battery, ions shift from the negative electrode to the positive electrode during discharge, and the reverse occurs during charging. The positive electrode employs an intercalated lithium compound, while graphite is used at the negative electrode. These batteries offer high energy density, exhibit no memory effect, and have low self-discharge.

Most Lithium-Ion batteries have a nominal voltage of 3.7V. When fully charged, the maximum voltage reaches 4.2±0.5V. Refer to the manufacturer's datasheet for your battery's cut-off voltage, as this value can vary depending on the battery type in use.

The Code

```
#include <ArduinoBLE.h>
#include <DHT.h>
#include <EmonLib.h>

#define DHTPIN 14
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

```
#define RELAY1_PIN 15
#define RELAY2_PIN 2
#define RELAY3_PIN 4

#define LED1_PIN 5
#define LED2_PIN 18
#define LED3_PIN 19

#define HYSTERESIS 2
EnergyMonitor emon1;

BLEByteCharacteristic relay1OffCharacteristic("19B10008-E8F2-537E-4F6C-D104768A1214",
BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic relay2OffCharacteristic("19B10009-E8F2-537E-4F6C-D104768A1214",
BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic relay3OffCharacteristic("19B1000A-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite | BLENotify);
BLEService bleService("19B10000-E8F2-537E-4F6C-D104768A1214");
BLEByteCharacteristic relay2ControlCharacteristic("19B10008-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic temperatureCharacteristic("19B10001-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic irmsCharacteristic("19B10002-E8F2-537E-4F6C-D104768A1214",
BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic vrmsCharacteristic("19B10003-E8F2-537E-4F6C-D104768A1214",
BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic powerFactorCharacteristic("19B10004-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic realPowerCharacteristic("19B10005-E8F2-537E-4F6C-D104768A1214",
BLERead | BLEWrite | BLENotify);
```

```
BLEByteCharacteristic                    reactivePowerCharacteristic("19B10006-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite | BLENotify);
BLEByteCharacteristic                      activePowerCharacteristic("19B10007-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite | BLENotify);

bool relay1State = false;
bool relay3State = false;
bool relay2State = false;
void setup() {
  Serial.begin(9600);

  pinMode(RELAY1_PIN, OUTPUT);
  pinMode(RELAY2_PIN, OUTPUT);
  pinMode(RELAY3_PIN, OUTPUT);

  pinMode(LED1_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);
  pinMode(LED3_PIN, OUTPUT);
  digitalWrite(RELAY1_PIN, HIGH);
digitalWrite(RELAY2_PIN, HIGH);
digitalWrite(RELAY3_PIN, LOW);
digitalWrite(LED1_PIN, LOW);
digitalWrite(LED2_PIN, LOW);
digitalWrite(LED3_PIN, LOW);

  if (!BLE.begin()) {
    Serial.println("starting Bluetooth® Low Energy module failed!");
    while (1);
  }
  BLE.setLocalName("RelayAndLedControl");
  BLE.setAdvertisedService(bleService);
```

```
bleService.addCharacteristic(relay1OffCharacteristic);
bleService.addCharacteristic(relay2OffCharacteristic);
bleService.addCharacteristic(relay3OffCharacteristic);
  bleService.addCharacteristic(temperatureCharacteristic);
  bleService.addCharacteristic(irmsCharacteristic);
  bleService.addCharacteristic(vrmsCharacteristic);
  bleService.addCharacteristic(powerFactorCharacteristic);
  bleService.addCharacteristic(realPowerCharacteristic);
  bleService.addCharacteristic(reactivePowerCharacteristic);
  bleService.addCharacteristic(activePowerCharacteristic);
bleService.addCharacteristic(relay2ControlCharacteristic);
  BLE.addService(bleService);

  temperatureCharacteristic.writeValue(0);
  irmsCharacteristic.writeValue(0);
  vrmsCharacteristic.writeValue(0);
  powerFactorCharacteristic.writeValue(0);
  realPowerCharacteristic.writeValue(0);
  reactivePowerCharacteristic.writeValue(0);
  activePowerCharacteristic.writeValue(0);

  BLE.advertise();
Serial.println("RelayAndLedControl");
  emon1.current(35, 111.1); // Calibration factor may need adjustment
  emon1.voltage(34, 234.26, 1.7);
  dht.begin();
}

void loop() {
  BLEDevice central = BLE.central();
  if (central) {
    while (central.connected()) {
      if (temperatureCharacteristic.written()) {
        int receivedTemperature = temperatureCharacteristic.value();
        if (receivedTemperature > 0) {
          // Control relay 1 based on temperature
          if (dht.readTemperature() > receivedTemperature) {
            digitalWrite(RELAY1_PIN, HIGH);
            relay1State = true;
          } else {
            digitalWrite(RELAY1_PIN, LOW);
            relay1State = false;
          }
        }
      }

      if (irmsCharacteristic.written()) {
        int receivedIrms = irmsCharacteristic.value();
        if (receivedIrms > 0) {
          // Control relay 3 based on IRMS
          float currentIrms = emon1.calcIrms(1480); // Sample window = 1480ms
          if (currentIrms > receivedIrms) {
            digitalWrite(RELAY3_PIN, HIGH);
            relay3State = true;
          } else {
            digitalWrite(RELAY3_PIN, LOW);
            relay3State = false;
          }
        }
      }
 if (relay2ControlCharacteristic.written()) {
  int relay2Value = relay2ControlCharacteristic.value();
  if (relay2Value == 1) {
    digitalWrite(RELAY2_PIN, HIGH); // Turn Relay 2 on
    relay2State = true;
  } else {
    digitalWrite(RELAY2_PIN, LOW); // Turn Relay 2 off
    relay2State = false;
  }
      // Update BLE characteristics with sensor readings
      updateBLECharacteristics();
    }
  }
  }
  if (relay1OffCharacteristic.written()) {
  int relay1OffValue = relay1OffCharacteristic.value();
  if (relay1OffValue == 1) {
    digitalWrite(RELAY1_PIN, LOW); // Turn Relay 1 off
    relay1State = false;
  }
}

if (relay2OffCharacteristic.written()) {
  int relay2OffValue = relay2OffCharacteristic.value();
  if (relay2OffValue == 1) {
    digitalWrite(RELAY2_PIN, LOW); // Turn Relay 2 off
  }
}

if (relay3OffCharacteristic.written()) {
  int relay3OffValue = relay3OffCharacteristic.value();
  if (relay3OffValue == 1) {
    digitalWrite(RELAY3_PIN, LOW); // Turn Relay 3 off
    relay3State = false;
  }
}
  // Update LED states based on relay states
  digitalWrite(LED1_PIN, relay1State ? HIGH : LOW);
  digitalWrite(LED2_PIN, relay2State ? HIGH : LOW);
  digitalWrite(LED3_PIN, relay3State ? HIGH : LOW);

  delay(1000); // Adjust delay as needed
}

void updateBLECharacteristics() {
  // Get sensor readings and update BLE characteristics
  float currentIrms = emon1.calcIrms(1480); // Sample window = 1480ms

  float vrms = currentIrms * 230.0; // Assuming line voltage is 230V

  float powerFactor     = emon1.powerFactor;

  temperatureCharacteristic.writeValue(dht.readTemperature());
  irmsCharacteristic.writeValue(currentIrms);
  vrmsCharacteristic.writeValue(vrms);
  powerFactorCharacteristic.writeValue(powerFactor);

  // Calculate and update real power and reactive power
  float realPower = emon1.realPower;
  float reactivePower = sqrt((vrms * vrms * currentIrms * currentIrms) - (realPower * realPower));

  realPowerCharacteristic.writeValue(realPower);
  reactivePowerCharacteristic.writeValue(reactivePower);
  activePowerCharacteristic.writeValue(emon1.apparentPower);
}
```

Mit App Inventor code

Ble using UUId’s