# SMART ENERGY METER using an ESP32

*A project report submitted in partial fulfilment of the requirements forthe 6TH semester of the degree of*

**BACHELOR OF TECHNOLOGYIN**
**ELECTRONICS ENGINEERING**

*Submitted by*


| | | |
|---|---|---|
| **ANISH RAJ** | **-** | **21001017008** |
| **MANOJ KUMAR SINGROHA** | **-** | **21001017035** |
| **SHUBHAM KUMAR** | **-** | **21001017062** |


*Under the Supervision of*

**Dr. ROHIT TRIPATHI**


**Department of Electronics Engineering**

Faculty of Engineering and Technology

**J.C. Bose University of Science and Technology, YMCA, Faridabad, Haryana-121006**

# TABLE OF CONTENTS:

# 1. INTRODUCTION:

Electrical appliances are all around us, starting from the indispensable charger of your smartphone to heavier appliances like room heaters, air conditioners, washing machines, and what not. While our monthly electricity bill gives us a vague idea of our culminating electricity consumption, it provides little to no detail about the current being consumed by particular devices. To determine the exact and instantaneous consumption of power we must have a way to detect the voltage and the current draw of our appliance.

In this article, we will see how you can make a simple **SMART ENERGY METER using an ESP32** and some commonly available sensors.

# 2. PROJECT OVERVIEV:

In essence, a smart power meter works exactly like a traditional meter, in that it measures, monitors, and records your energy consumption. However, there's one big difference. Rather than requiring a meter reader to check the usage, this electric meter relays your consumption information directly to the utility operator, as it's a digital device with a communication module. On average, these smart meters send the usage information every 15 minutes or so, which eliminates the need for a dedicated meter reader. A smart electric meter, also known as a smart meter or digital meter, is an advanced electrical metering device that offers several advantages over traditional analog meters. Smart meters are being deployed worldwide as part of efforts to modernize the electrical grid and provide more accurate and efficient energy management. Here are some key features and benefits of smart electric meters:

## 3. Components:

- ESP32 WIFI Module
- HI LINK 5V 3W SMPS
- 0.96” 128X64 I2C LED
- ACS712 Current Sensor
- 220V AC 3 Pin Socket MALE
- 220V AC 3 Pin Socket FEMALE
- 3D Printed Casing
- Jumping wire and Breadboard

## 4. Components Required with Cost:

| S No. | COMPONENTS USED | COST |
|---|---|---|
| 1. | ESP32 WIFI Module | 500 |
| 2. | HI LINK 5V 3W SMPS | 100 |
| 3. | 0.96” 128X64 I2C LED | 200 |
| 4. | ACS712 Current Sensor | 250 |
| 5. | 220V AC 3 Pin Socket MALE | 70 |
| 6. | 220V AC 3 Pin Socket FEMALE | 50 |
| 7. | 3D Printed Casing | 100 |
| 8. | Jumping wire and Breadboard | 150 |
| **TOTAL** | | **1420** |

# 5. Working Principal:

- The working principle of a smart meter involves several components and processes designed to accurately measure and communicate electricity usage. Here is an overview of the key working principles of a smart meter:
- Electricity Measurement:
- Current Measurement: Smart meters are equipped with current sensors that measure the flow of electricity through a circuit.
- Voltage Measurement: Voltage sensors measure the electrical potential difference in the circuit.
- Power Calculation: By multiplying current and voltage measurements, the smart meter calculates the real-time power consumption in watts (W).
- Data Sampling:
- Smart meters continuously sample and record voltage and current measurements, typically at a high frequency (e.g., every few seconds). This results in a stream of data that represents power consumption over time.
- Data Processing:
- The sampled data is processed within the smart meter to calculate various metrics, including:
- Real-time energy consumption (in kilowatt-hours, kWh).
- Voltage quality (voltage dips, spikes, and harmonics).
- Load profiling to understand when and how electricity is being used.
- Smart meters may store data locally for short-term analysis, typically covering a few months of consumption data.
- One of the most significant features of a smart meter is its ability to communicate data to external systems. This communication can be achieved through various means:
- Wireless Communication: Smart meters may use wireless protocols like Zigbee, Wi-Fi, or cellular networks to transmit data to a central data collection point.
- Power Line Communication (PLC): Some smart meters use existing power lines for communication, sending data over the electrical grid.
- Remote Reading:
- Utility companies can remotely read the data from smart meters, eliminating the need for manual meter reading. This data is collected for billing and grid management.
- Time-of-Use (TOU) Rates:
- Smart meters support Time-of-Use pricing, where electricity rates vary throughout the day. Consumers can be charged more during peak hours and less during off-peak hours, encouraging energy conservation.
- Outage Detection and Reporting:
- Smart meters can detect power outages in real-time. When an outage occurs, they send notifications to the utility company, enabling quicker response and restoration of power.
- Most smart meters also have local user interfaces, such as digital displays or LEDs, allowing consumers to monitor their own energy usage and view real-time information.

## 6. CIRCUIT DIAGRAM:

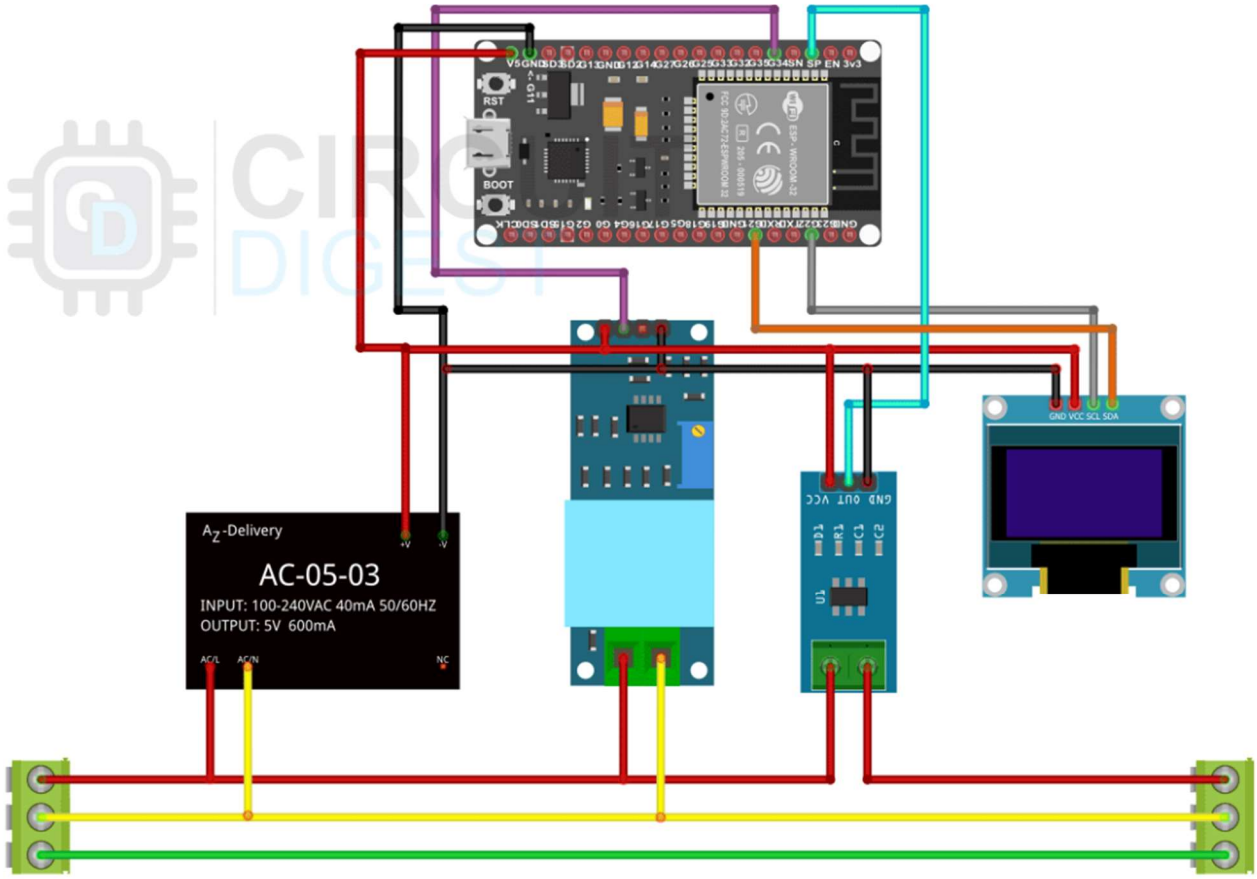



FIG. 1

The diagram shows how the Hi-link SMPS and input terminal of the voltage sensor (ZMPT101B) are connected in parallel to the AC Live and AC Neutral, while the current sensor (ACS712) forms a series connection with the live AC Wire.

The OLED is connected via the I2C pins on the ESP32 and the Voltage and Current sensor are connected to Pin 34 and Pin 36(VP) respectively. These pins are capable of analog to digital conversion using the 12-BIT ADC inside the ESP32.

All the components are powered by the output 5v of the Hi-Link module. (Vo+ and Vo- terminals)

# 7. SOURCE CODE:

```
#define BLYNK_TEMPLATE_ID "TMPL310ItBqm1"
#define BLYNK_TEMPLATE_NAME "SMART ENERGY METER"
#define BLYNK_AUTH_TOKEN "6YEI2SW9EkFwMhRWFqQ-d0_Nl0vyjrXa"

#include <BlynkSimpleEsp32.h>
#include <Wire.h>
#include <Wifi.h>
#include <WifiClient.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "ZMPT101B.h"
#include "ACS712.h"


#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C


Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

char auth[] = "6YEI2SW9EkFwMhRWFqQ-d0_Nl0vyjrXa";
BlynkTimer timer;

ZMPT101B voltageSensor(34);
ACS712 currentSensor(ACS712_20A, 36);

float P = 0;
float U = 0;
float I = 0;
long dt = 0;
float CulmPwh = 0;
float units = 0;
long changeScreen = 0;
float lastSample = 0;

unsigned long lasttime = 0;
long ScreenSelect = 0;

void setup() {
```

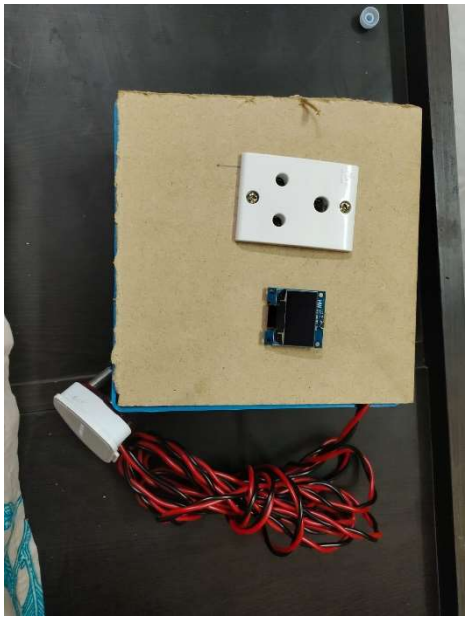
```
  Serial.begin(9600);

  Blynk.begin(auth, "Anish", "anish098");

  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;) {}
  }

  // Calibration commands need to be run on first upload
  // CalibCurrent();
  // CalibVoltage();

  timer.setInterval(1000L, sendSensorDataToBlynk); // Update Blynk every second
}

void loop() {
  Blynk.run();
  timer.run();

  U = voltageSensor.getRmsVoltage();
  if (U < 55) {
    U = 0;
    CulmPwh = 0;
  }

  I = currentSensor.getCurrentAC();
  dt = micros() - lastSample;

  if (I < 0.15) {
    I = 0;
    CulmPwh = 0;
  }

  P = U * I;
  CulmPwh = CulmPwh + P * (dt / 3600); // Accumulate power consumption
  units = CulmPwh / 1000;

  if (millis() - changeScreen > 5000) {
    ScreenSelect += 1;
    changeScreen = millis();
  }

  if (millis() - lasttime > 500) {
```

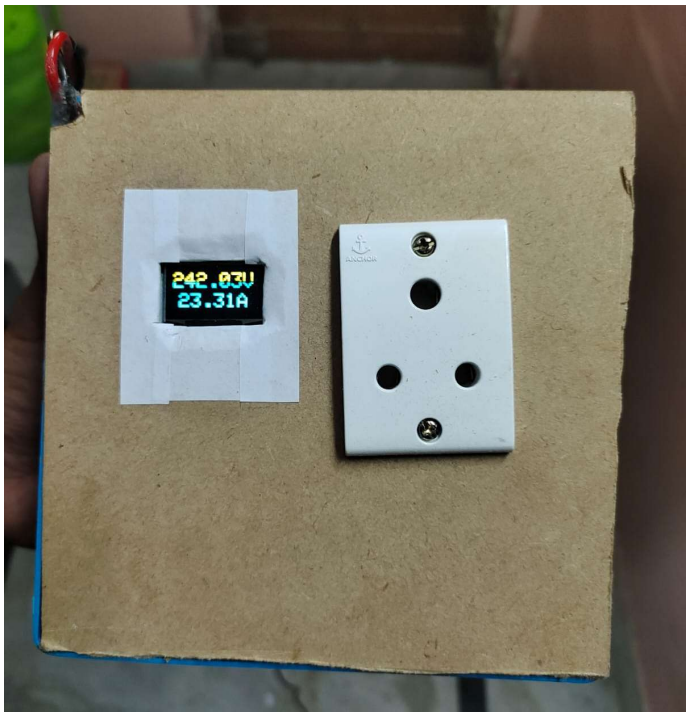
```
    switch (ScreenSelect % 4) {
      case 0:
        displayVoltCurrent();
        break;
      case 1:
        displayInstPower();
        break;
      case 2:
        displayEnergy();
        break;
      case 3:
        displayUnits();
        break;
    }
  }
  lastSample = micros();
}

void displayVoltCurrent() {
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(3);
  displayCenter(String(U) + "V", 3);
  display.setTextSize(3);
  displayCenter(String(I) + "A", 33);
  display.display();
  lasttime = millis();
}


void displayInstPower() {
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(2);
  display.setCursor(0, 0);
  displayCenter("Power", 3);
  display.setTextSize(3);
  if (P > 1000) {
    displayCenter(String(P / 1000) + "kW", 30);
  } else {
    displayCenter(String(P) + "W", 30);
  }
  display.display();
  lasttime = millis();
```

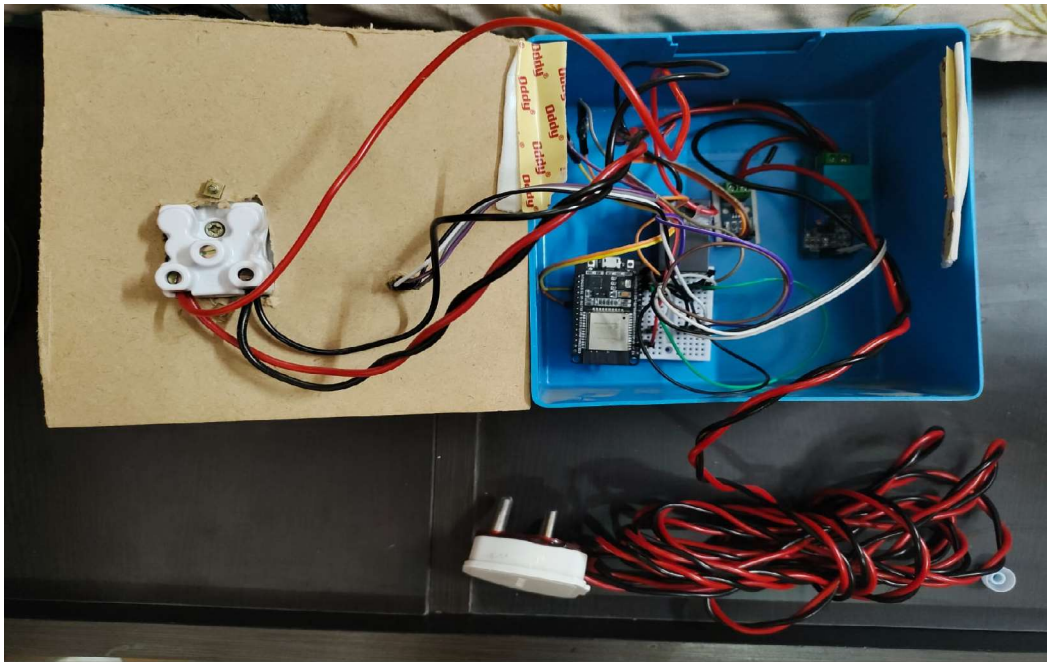
```
}

void displayEnergy() {
  display.clearDisplay();
  display.setTextColor(WHITE);
  if (CulmPwh > 1000000000) {
    display.setTextSize(2);
    displayCenter("Energy kWh", 3);
    display.setTextSize(3);
    displayCenter(String(CulmPwh / 1000000000), 30);
  } else if (CulmPwh < 1000000000 && CulmPwh > 1000000) {
    display.setTextSize(2);
    displayCenter("Energy Wh", 3);
    display.setTextSize(3);
    displayCenter(String(CulmPwh / 1000000), 30);
  } else if (CulmPwh < 1000000 && CulmPwh > 1000) {
    display.setTextSize(2);
    displayCenter("Energy mWh", 3);
    display.setTextSize(3);
    displayCenter(String(CulmPwh / 1000), 30);
  } else {
    display.setTextSize(2);
    displayCenter("Energy uWh", 3);
    display.setTextSize(3);
    displayCenter(String(CulmPwh), 30);
  }
  display.display();
  lasttime = millis();
}

void displayUnits() {
  display.clearDisplay();
  display.setTextColor(WHITE);
  if (units > 1000000) {
    display.setTextSize(2);
    displayCenter("Units", 3);
    display.setTextSize(3);
    displayCenter(String(units / 1000000), 30);
  } else if (units < 1000000 && units > 1000) {
    display.setTextSize(2);
    displayCenter("MilliUnits", 3);
    display.setTextSize(3);
    displayCenter(String(units / 1000), 30);
  } else {
```

```
    display.setTextSize(2);
    displayCenter("MicroUnits", 3);
    display.setTextSize(3);
    displayCenter(String(units), 30);
  }
  display.display();
  lasttime = millis();
}


void sendSensorDataToBlynk() {
  Blynk.virtualWrite(V1, U);      // Send voltage to V1 in Blynk app
  Blynk.virtualWrite(V2, I);      // Send current to V2 in Blynk app
  Blynk.virtualWrite(V3, P);      // Send power to V3 in Blynk app
  Blynk.virtualWrite(V4, CulmPwh); // Send cumulative power to V4 in Blynk app
  Blynk.virtualWrite(V5, units);   // Send units to V5 in Blynk app
}


void displayCenter(String text, int line) {
  int16_t x1, y1;
  uint16_t width, height;
  display.getTextBounds(text, 0, 0, &x1, &y1, &width, &height);
  display.setCursor((SCREEN_WIDTH - width) / 2, line);
  display.println(text);
  display.display();
}
```

# 8. PROJECT PICTURES:





## 9. APPLICATION:

- Two-way Communication: Smart meters can communicate data both ways – from the meter to the utility company and from the utility company to the meter. This enables real-time data exchange and control.
- Remote Reading: Utility companies can remotely read the meter data, eliminating the need for manual meter reading and improving billing accuracy.
- Real-Time Data: Smart meters provide real-time information on energy consumption, allowing consumers to monitor their usage and adjust their behavior to save energy.
- Time-of-Use (TOU) Rates: Smart meters enable time-based pricing, where electricity rates can vary throughout the day. Consumers can use electricity during off-peak hours to save money.
- Outage Detection: Smart meters can detect power outages and notify utility companies immediately, allowing for faster response times.
- Load Profiling: Smart meters record detailed data on energy usage patterns, helping utilities understand peak demand periods and plan accordingly.
- Integration with Renewable Energy: Smart meters can be integrated into a smart grid, allowing for better integration of renewable energy sources like solar and win.

## 10.CONCLUSION:

In summary, simple **SMART ENERGY METER using an ESP32** work by accurately measuring electricity consumption through current and voltage measurements. They then process and store this data, communicate it to utility companies and consumers, and enable features like Time-of-Use pricing, outage detection, and real-time monitoring. The primary goal is to improve the accuracy of billing, promote energy efficiency, and enable more efficient management of the electrical grid.