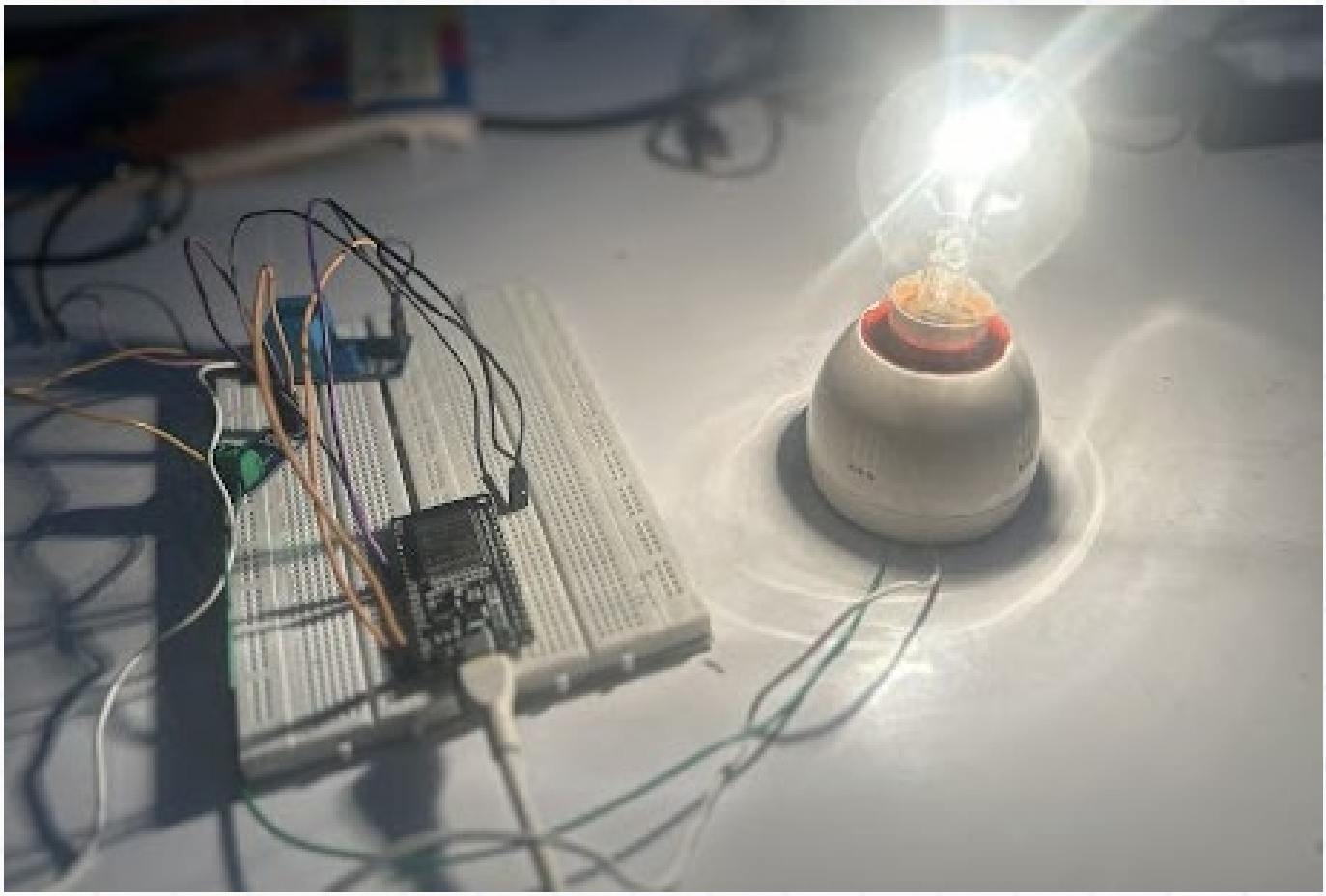
Build an IoT Energy Meter



Index

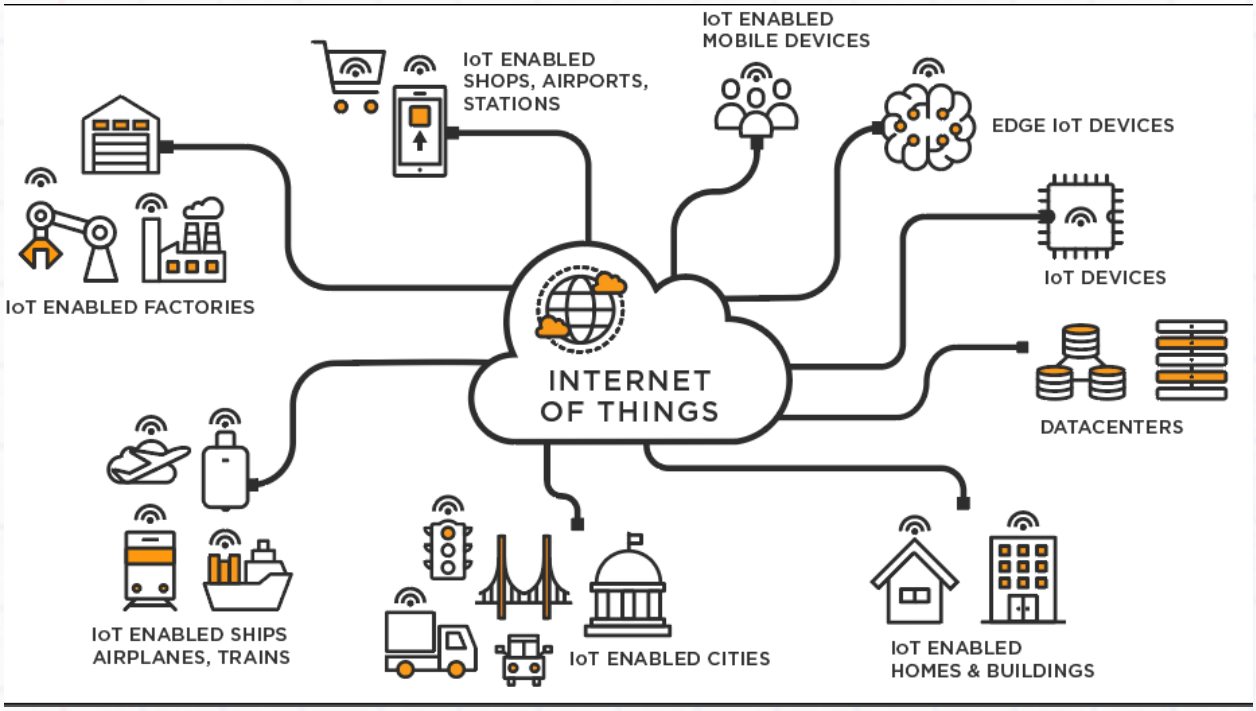
Aim

Build an IoT (Internet of Things) Energy Meter that measures electrical energy consumption and other parameters in real-time and tracks this data remotely on a web IoT platform.

Concept

What is IoT?

IoT is a system of interconnected computing devices that use internet protocols to communicate and transfer data. This allows remote devices to communicate amongst each other without the need for human involvement,



How this project works

- In this project, we use voltage and current sensors and using their values calculate the power and energy consumption from everyday electrical circuits. Since the sensors read the current, voltage values as analog wave signals, which are harder to deal with, we use an Analog to Digital Converter (ADC) to convert them to digital values - i.e. numerical values.
  These values are received by the ESP 32 board using the External Interrupt mechanism as we have done for the flow sensor in the IoT Water Meter project.

- We then use these to compute parameters like power factor, power and energy consumed.
- The ESP32 Wifi module sends the data to a Ubidots cloud server using an API. The energy consumption data can then be viewed in real-time on the Ubidots web dashboard.

Components

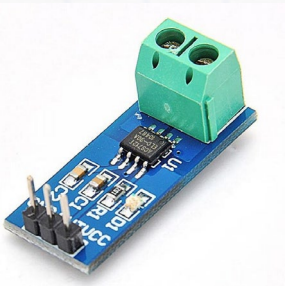
1. ESP32 module
2. Breadboard
3. ZMPT101B 250V Voltage sensor
4. ACS712 20A Current sensor
5. 40W Bulb + 60W Bulb
6. holder
7. 2-pin plug with wiring
8. Jumper wires
9. 1 USB cable

Esp 32 Module

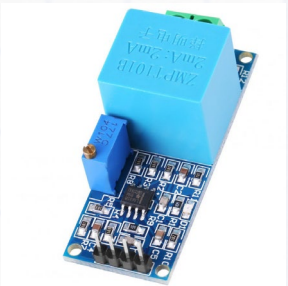
Bulb

2-pin plug with wiring

Holder



ACS712 20A Current sensor



ZMPT101B 250V Voltage sensor

Jumper Wires

USB

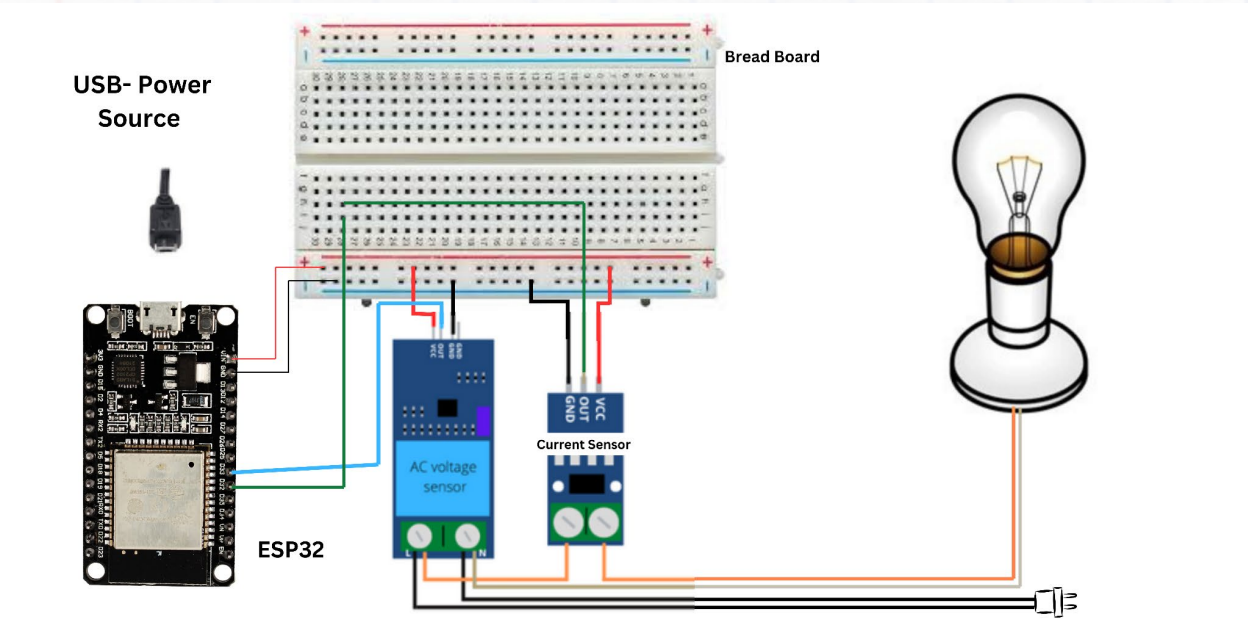
Breadboard

Connections

Caution: High Voltage Electrical Equipment. Do not touch the circuit once it's connected to the power supply. For any changes make sure the power supply is turned off.

Circuit Diagram

NOTE: Before starting the connections, verify that all the jumper wires are working using a multimeter. Also, ensure that the connections are strong, or the setup may not work.



- VCC to Red line of breadboard
- GND to Blue line of breadboard

AC Voltage Sensor:

- VCC Pin: Connected to the red power line on the breadboard.
- GND Pin: Connected to the blue ground line on the breadboard.
- OUT Pin: Connected to an analog input pin on the ESP32 (GPIO Pin 33).

Current Sensor:

- VCC Pin: Connected to the red power line on the breadboard.
- GND Pin: Connected to the blue ground line on the breadboard.
- OUT Pin: Connected to another analog input pin on the ESP32 (GPIO Pin 32).

- Connect the AC live and neutral wires to the AC voltage sensor.
- Pass the AC live wire through the current sensor.
- Connect the bulb's live wire through the current sensor and the neutral wire directly to the AC power source.

Software

Launching the IDE for our project

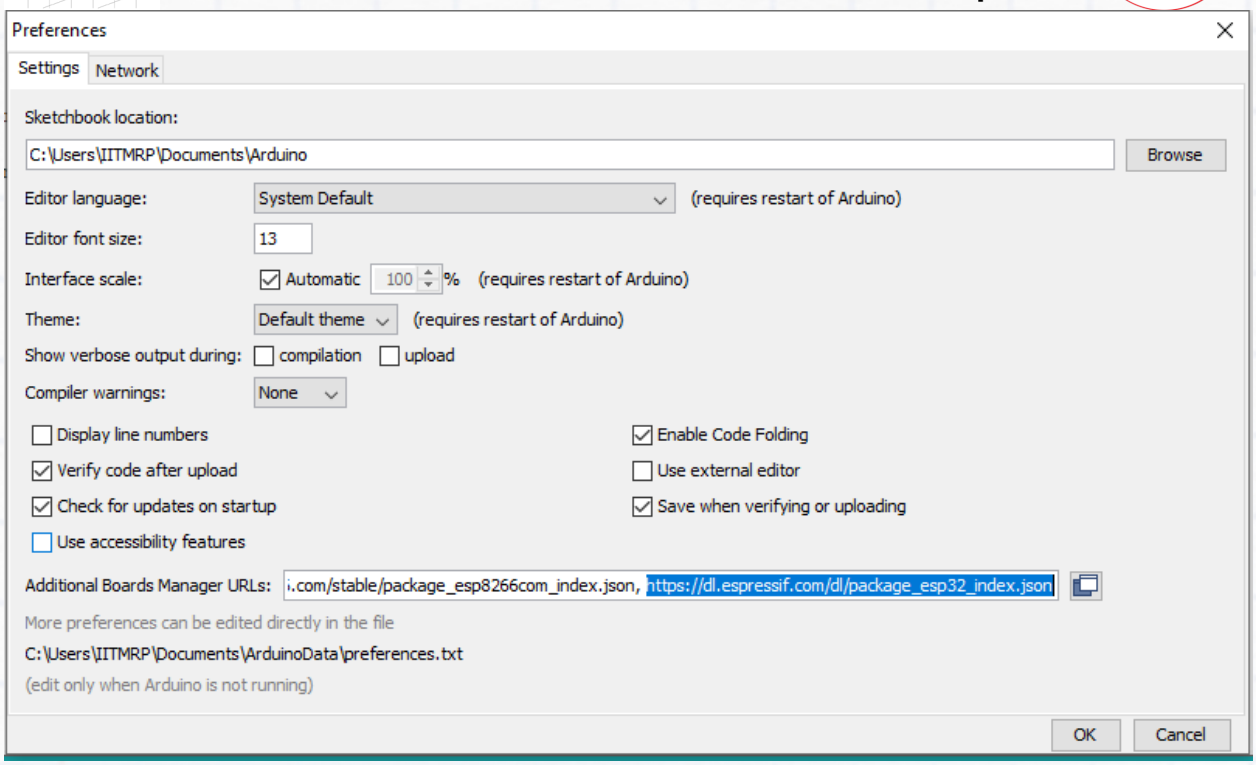
1. Install the Arduino IDE

If you haven't already installed the Arduino IDE, download and install it from the official Arduino website.

2. Install the ESP32 Module in Arduino IDE

- Open the Arduino IDE.
- Go to File > Preferences.
- In the "Additional Board Manager URLs" field, add the following URL:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json



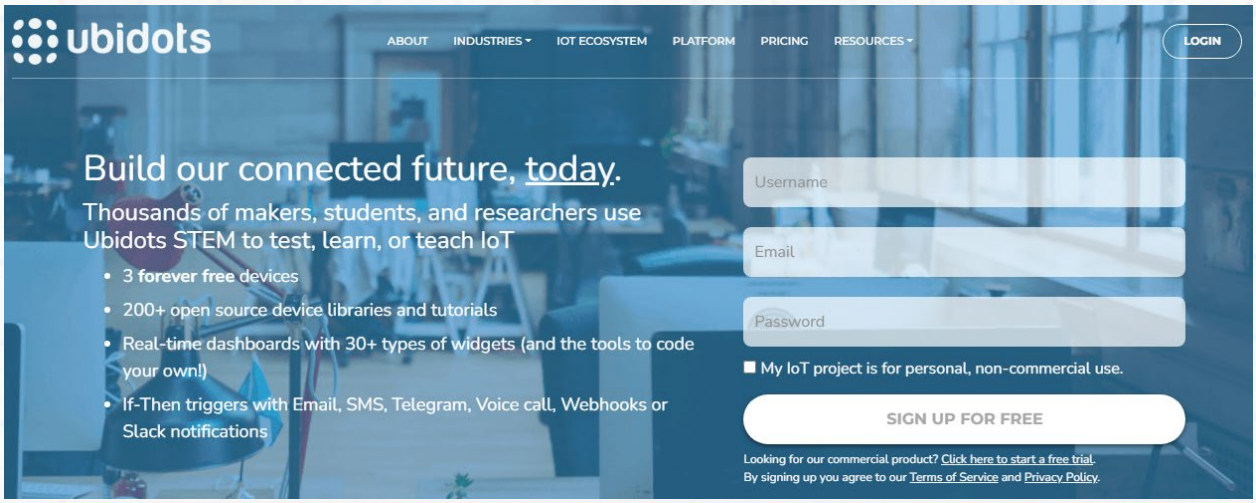
- Go to Tools > Board > Boards Manager.
- Search for ESP32 and install the ESP32 package. This will take some time to install.

3. Install Required Libraries

- Go to Sketch > Include Library > Manage Libraries.
- Connect Your ESP32 Module
- Connect your ESP32 Module to your computer using a USB cable.
- Go to Tools > Board > ESP32 Dev Module.
- Go to Tools > Port and select the COM port to which your ESP32 is connected.
- Go to sketch > Include Library > Manage Libraries.
- Search EmonLib and install it.
- Inside the extracted **ESP32_Flow_Meter** folder, you will find 2 folders named **esp32-mqtt-main** & **pubsubclient-master**. Copy them and paste them inside the 'Libraries' folder located in Documents > Arduino > Libraries.

4. Ubidots Setup

1. For monitoring the values of the electrical parameters, we will be using the Ubidots IoT Platform, as done for the Water Meter project. If you haven't, go to Ubidots.com/stem/ and sign up for free.
2. On the Ubidots website, click on your profile icon and select API credentials.
3. Copy the 'Default token' on the top right part of the page and paste it into the UBIDOTS_TOKEN variable, which is immediately below the Wifi variables in the Arduino code.



5.Code

"Open the .ino file which was attached in the folder and write a code by the below instruction."

Explanation:

Include Libraries:

- #include "UbidotsEsp32Mqtt.h": Includes the Ubidots MQTT library for ESP32.
- #include "EmonLib.h": Includes the Emon Library for energy monitoring.

Defining Constants:

- WIFI_SSID, WIFI_PASS: Wi-Fi credentials.
- UBIDOTS_TOKEN: Ubidots token for authentication.
- DEVICE_LABEL: Label for the device publishing data to Ubidots.
- VRMS_VARIABLE_LABEL, IRMS_VARIABLE_LABEL, POWER_VARIABLE_LABEL, POWER_FACTOR_VARIABLE_LABEL, ENERGY_VARIABLE_LABEL: Variable labels for different measurements.
- PUBLISH_FREQUENCY: Frequency in milliseconds for publishing data.
- timer, analogPin: Variables to manage time and define pins for reading data.
- Ubidots ubidots(UBIDOTS_TOKEN): Creates an instance of the Ubidots class with the provided token.

```
/****************************************
 * Include Libraries
 ****************************************/
#include "UbidotsEsp32Mqtt.h"
#include "EmonLib.h"  // Include Emon Library

EnergyMonitor emon1;  // Create an instance

float energyConsumed = 0;  // Variable to store total energy
unsigned long lastTime = 0;

/****************************************
 * Define Constants
 ****************************************/
const char *WIFI_SSID = "";  // Put your Wi-Fi SSID here
const char *WIFI_PASS = "";  // Put your Wi-Fi password here
const char *UBIDOTS_TOKEN = "";  // Put your Ubidots TOKEN here

const char *DEVICE_LABEL = "ENERGY_METER";  // Device label for publishing data

const char *VRMS_VARIABLE_LABEL = "VRMS";  // Variable label for VRMS
const char *IRMS_VARIABLE_LABEL = "IRMS";  // Variable label for IRMS
const char *POWER_VARIABLE_LABEL = "POWER";  // Variable label for POWER
const char *POWER_FACTOR_VARIABLE_LABEL = "POWER_FACTOR";  // Variable label
for POWER FACTOR
```

```
const char *ENERGY_VARIABLE_LABEL = "ENERGY";  // Variable label for ENERGY

const int PUBLISH_FREQUENCY = 2000;  // Update rate in milliseconds

unsigned long timer;


Ubidots ubidots(UBIDOTS_TOKEN);
```

Callback Function:

- void callback(char *topic, byte *payload, unsigned int length): Handles incoming MQTT messages and prints them to the serial monitor.

Setup Function:

- void setup(): Initializes the serial communication, connects to Wi-Fi, sets up Ubidots, and initializes the energy monitor.
- Serial.begin(9600): Starts serial communication at a baud rate of 9600.
- ubidots.connectToWifi(WIFI_SSID, WIFI_PASS): Connects to the specified Wi-Fi network.
- ubidots.setCallback(callback): Sets the callback function for MQTT messages.
- ubidots.setup(): Sets up the Ubidots client.
- ubidots.reconnect(): Reconnects to the Ubidots server.
- emon1.voltage(33, 45, 1.7), emon1.current(32, 3): Initializes voltage and current sensors with specified pins and calibration values.
- timer = millis(): Initializes the timer.

Copy the code that is inside in the callback function and void setup and paste it on your code.

```
/****************************************
 * Auxiliar Functions
 ****************************************/
void callback(char *topic, byte *payload, unsigned int length)
```

```
{
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

/*****************************************
 * Main Functions
 *****************************************/
void setup()
{
  Serial.begin(9600);
  // ubidots.setDebug(true);  // uncomment this to make debug messages available
  ubidots.connectToWifi(WIFI_SSID, WIFI_PASS);
  ubidots.setCallback(callback);
  ubidots.setup();
  ubidots.reconnect();

  emon1.voltage(33, 220, 1.7);  // Voltage: input pin, calibration, phase_shift
  emon1.current(32, 10);  // Current: input pin, calibration.

  timer = millis();
}
```

Global Variables and Arrays:

- Various strings (rxed, rtc_string, vrms_string, irms_string, power_string, powerfactor_string, energy_string) for handling received data.
- Arrays (vol[400], cur[400]) for storing voltage and current values.
- Float variables (power, powerfactor, irms, vrms, energy) for storing computed values.
- long int rtc for handling real-time clock values.

```
String rxed = "";
String rtc_string = "";
String vrms_string = "";
String irms_string = "";
String power_string = "";
String powerfactor_string = "";
String energy_string = "";

float vol[400] = {0.0};
float cur[400] = {0.0};
float power = 0.0;
float powerfactor = 0.0;
float irms = 0.0;
float vrms = 0.0;
long int rtc = 0;
float energy = 0.0;
```

Main Loop:

- Checks if the Ubidots client is connected and reconnects if necessary.
- Reads and calculates voltage, current, real power, apparent power, and power factor using emon1.calcVI(20, 2000).
- Computes energy consumed in watt-hours.
- Updates the last measurement time.
- Prints all sensor values to the serial monitor.
- Delays for 1 second before the next loop iteration.

Copy the code that is inside in the void loop and paste it on your code.

```
void loop()
{
  if (!ubidots.connected())
  {
    ubidots.reconnect();
  }

  emon1.calcVI(20, 2000);  // Calculate all. No.of half wavelengths (crossings), time-out
```

```
  float realPower = emon1.realPower;  // Extract Real Power into variable
  float apparentPower = emon1.apparentPower;  // Extract Apparent Power into
variable
  float powerFactor = emon1.powerFactor;  // Extract Power Factor into Variable
  float supplyVoltage = emon1.Vrms;  // Extract Vrms into Variable
  float Irms = emon1.Irms;  // Extract Irms into Variable

  // Calculate energy in kWh
  unsigned long currentTime = millis();
  double elapsedTimeHours = (currentTime - lastTime) / 3600000.0;  // Time in hours

  // Integrate power over time to calculate energy (E = P * t)
  energyConsumed += realPower * elapsedTimeHours;  // Energy in watt-hours (Wh)

  // Update the last measurement time
  lastTime = currentTime;

  // Print all values to serial monitor
  Serial.print("Real Power: "); Serial.print(realPower); Serial.println(" W");
  Serial.print("Apparent Power: "); Serial.print(apparentPower); Serial.println(" VA");
  Serial.print("Power Factor: "); Serial.print(powerFactor); Serial.println();
  Serial.print("Supply Voltage: "); Serial.print(supplyVoltage); Serial.println(" V");
  Serial.print("Irms: "); Serial.print(Irms); Serial.println(" A");
  Serial.print("Total Energy: "); Serial.print(energyConsumed); Serial.println(" kWh");

  delay(1000);  // Delay for 1 second before next loop
}
```

Publishing Data to Ubidots:

- Publishes sensor values to Ubidots at the specified frequency.
- Calls ubidots.add(VRMS_VARIABLE_LABEL, supplyVoltage) to add voltage data and publishes it.
- Repeats the process for IRMS, power, power factor, and energy consumed.
- Calls ubidots.loop() to handle Ubidots client tasks.
- Updates the timer after publishing data.
- Flushes the serial buffer and delays for 10 milliseconds before the next loop iteration.

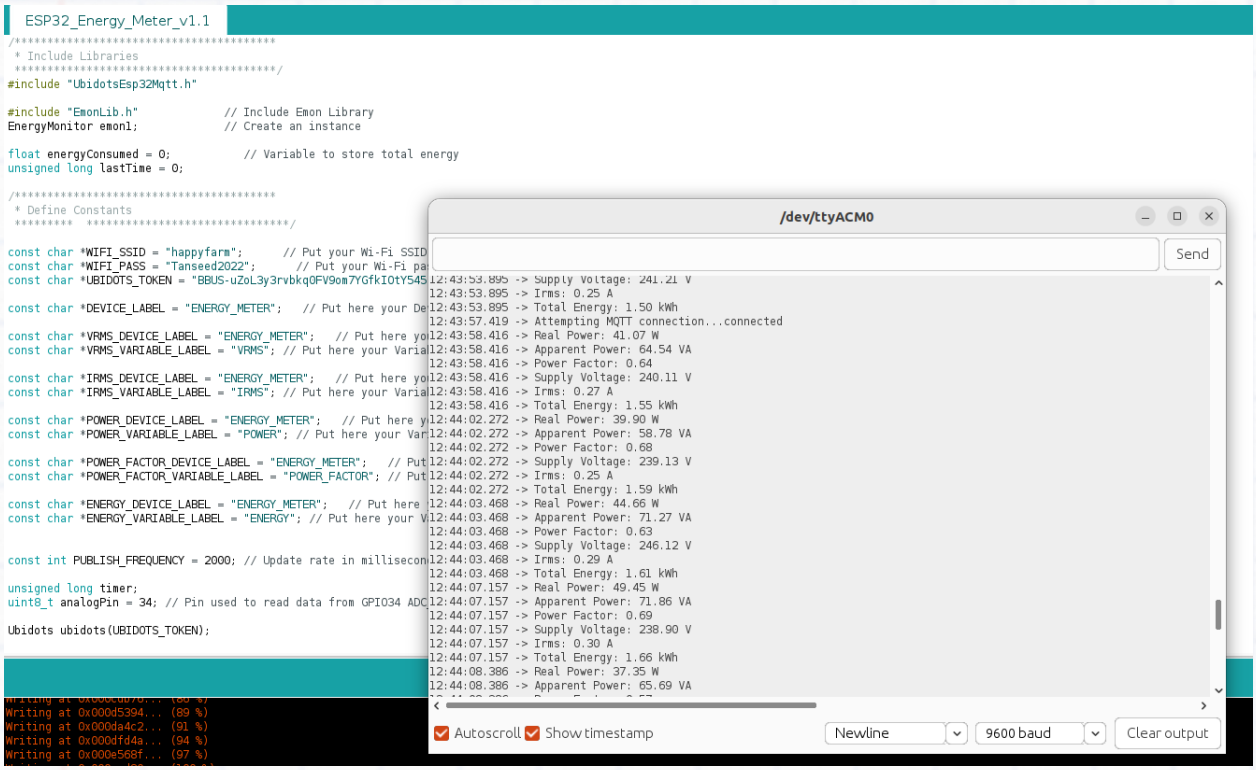
After delay(1000) in the above void loop and code for publishing Data to ubidots

```
 if ((millis() - timer) > (unsigned long)PUBLISH_FREQUENCY)  // triggers the routine every 5 seconds
 {
   ubidots.add(VRMS_VARIABLE_LABEL, supplyVoltage);  // Insert your variable Labels and the value to be sent
   ubidots.publish(DEVICE_LABEL);
   delay(500);
   ubidots.add(IRMS_VARIABLE_LABEL, Irms);  // Insert your variable Labels and the value to be sent
   ubidots.publish(DEVICE_LABEL);
   delay(500);
   ubidots.add(POWER_VARIABLE_LABEL, realPower);  // Insert your variable Labels and the value to be sent
   ubidots.publish(DEVICE_LABEL);
   delay(500);
   ubidots.add(POWER_FACTOR_VARIABLE_LABEL, powerFactor);  // Insert your variable Labels and the value to be sent
   ubidots.publish(DEVICE_LABEL);
   delay(500);
   ubidots.add(ENERGY_VARIABLE_LABEL, energyConsumed);  // Insert your variable Labels and the value to be sent
   ubidots.publish(DEVICE_LABEL);
   delay(500);

   ubidots.loop();
   timer = millis();
 }

 Serial.flush();
 delay(10);
}
```

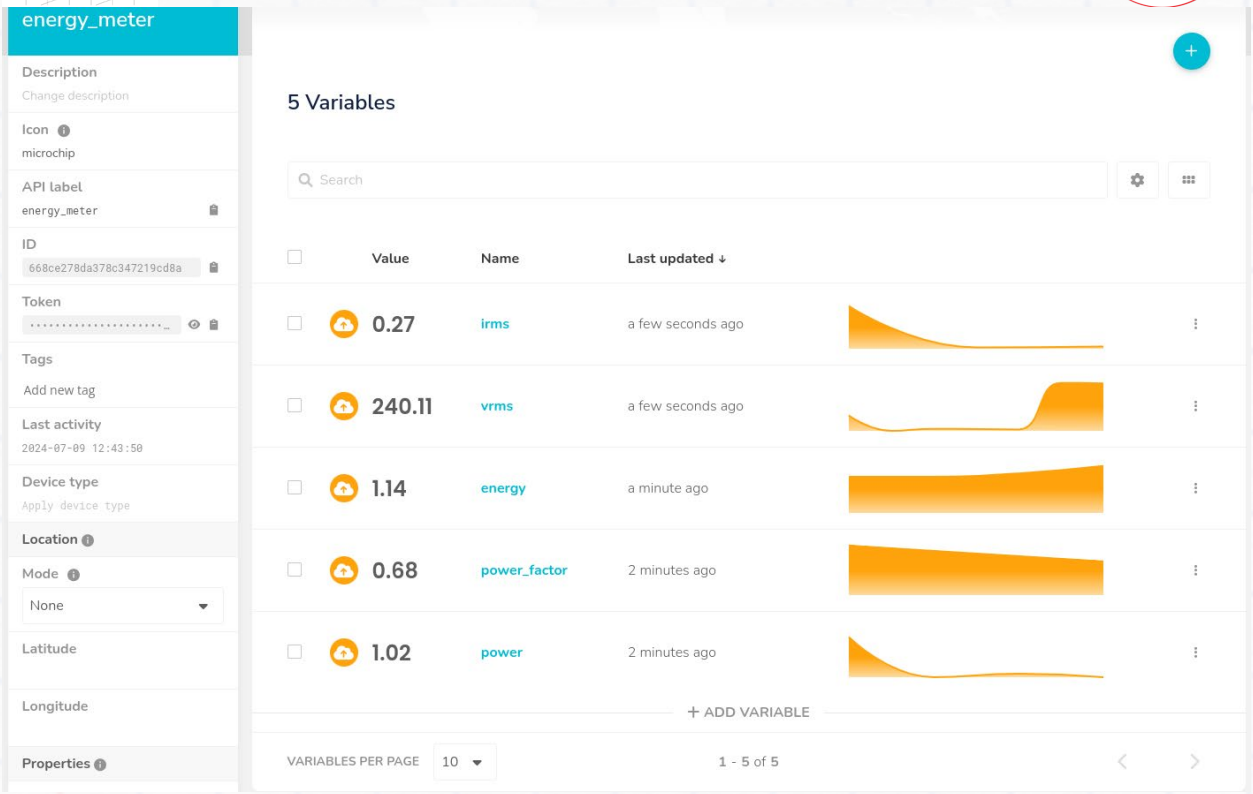
After completing all the steps verify the code and upload a code to the ESP32.

6. Upload the Code

- Copy the provided code into a new sketch in the Arduino IDE.
- Fill in your wifi connection's name and password within the empty quotation marks next to the WIFI_SSID and WIFI_PASS variables. Ensure the Wifi network has no firewalls, else the ESP32 won't connect to it.
- Click the 'Verify' button on the Arduino IDE - shown by a tick symbol. Once you receive the "Done compiling" message, click the 'Upload' button, shown as a right arrow near the 'Verify' button. Wait until the "Done uploading" message appears.
- After uploading the code, Click the magnifying glass icon on the top-right of the Arduino IDE. It will open the Serial Monitor. If the connection was successful.



- You can see a vrms, irms, power, power factor and energy values.
- Now go to the Devices page of the Ubidots website. You will find a device named energy_meter containing variables vrms, irms, power, power factor and energy.



Challenges

1) Using a multimeter, calculate the voltage reading of a regular “220V” power socket that you will be using to test the project. Note the reading value.

2) Initially, all the values in the Serial Monitor should read zero. Now put the 40W bulb into the bulb holder. Connect the 2-pin plug into the 220V power socket and turn it on.

CAUTION: From now onwards, DO NOT touch any part of the bulb circuit or the sensors while the power is on.

After turning on the bulb circuit, the variables should update in the Serial Monitor. Go to the Ubidots Dashboard page - the meter gauges should reflect the change in values.

3) Calibration: Is the reading for vrms of the circuit the same as the reading taken using the multimeter before? If not, go to code in arduino IDE and adjust the value of the V_calibration variable until the value of vrms matches the multimeter reading. Since we are using a 40W bulb, the value of power in the Serial Monitor should lie between 39-41

watts. Now that our vrms reading is accurate, inaccuracy in the power value will depend on the irms
value's accuracy. This can be fixed by adjusting the I_calibration variable in code Now that calibration is done, we can test our IoT Energy Meter. Turn off the bulb circuit and replace the 40W bulb with the 60W bulb provided in the kit. The value of power should now be between 58-62 watts and the irms value should be greater than before.

Real-world Application

This IoT Energy Meter setup can be used for any 220V single-phase AC appliance (AC here means 'Alternating Current', not 'Air Conditioner'!) that is rated under 5A. This can be done by replacing the bulb with the appliance.

For example, you can replace the bulb with a 220V induction motor and connect a fan to it. For trying out a wider range of appliances, you can replace the bulb holder with a 2-pin female socket. Now, you can directly plug in any permitted appliance (should be rated 220V single-phase AC and under 5A) and monitor its energy consumption.

Make a video presenting how you built and installed the IoT Energy Meter in your college and the various appliances you connected and monitored. Share it with the Build Club Community on the Discord Server!