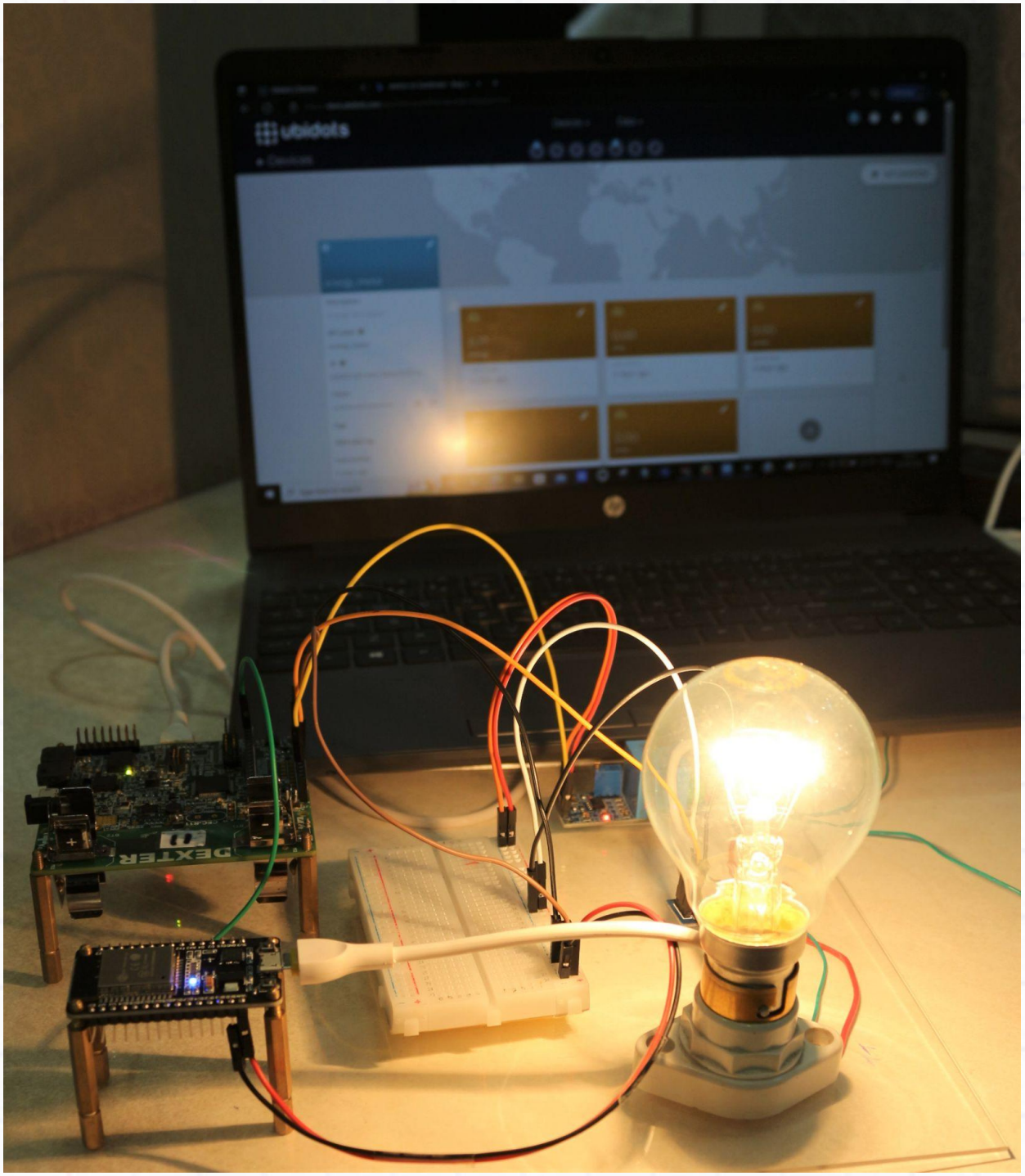
Build an IoT Energy Meter!



Index

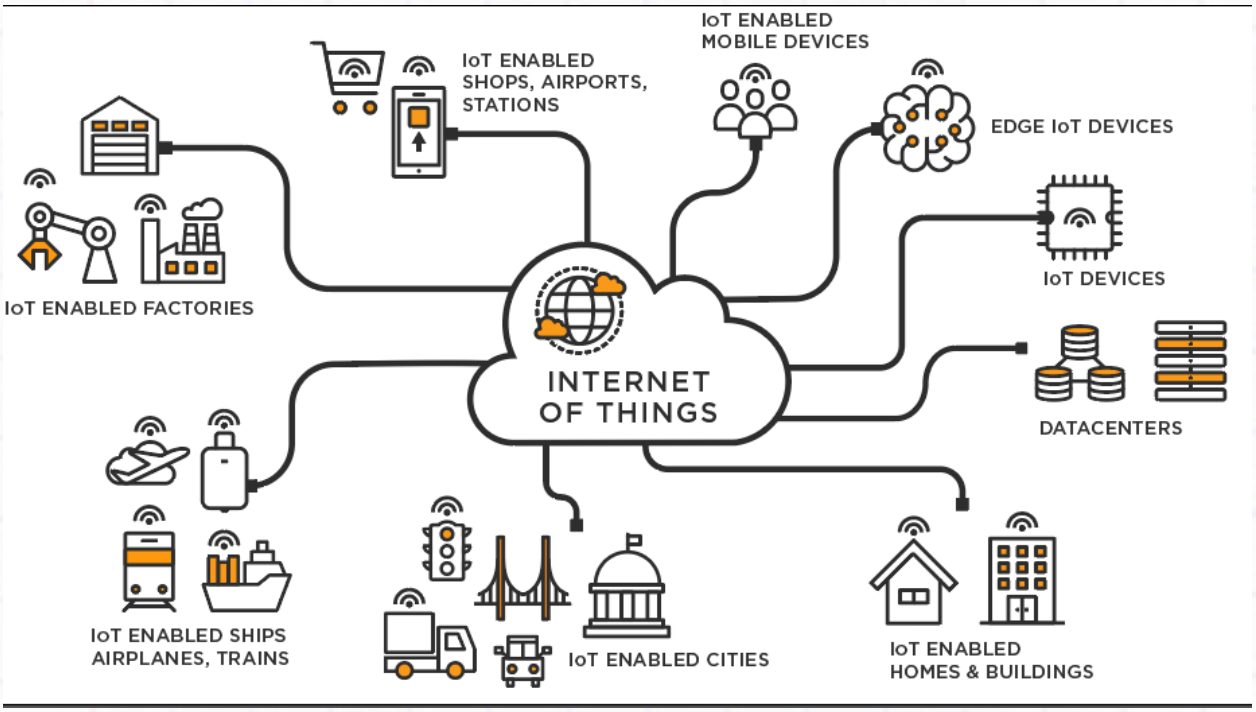
Aim

Build an IoT (Internet of Things) Energy Meter that measures electrical energy consumption and other parameters in real-time and track 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,

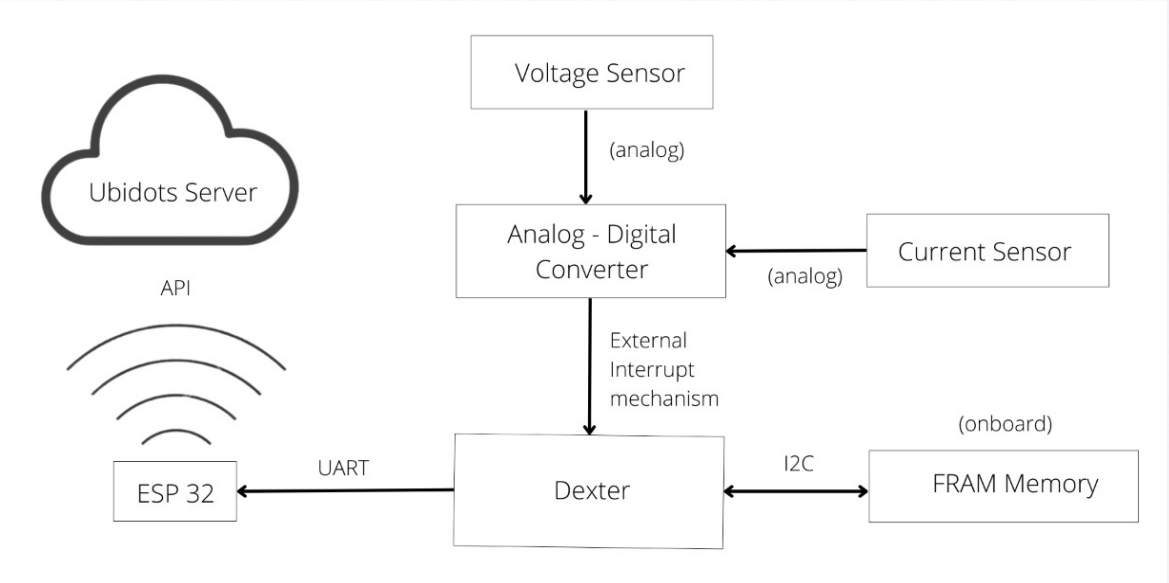


How this project works

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 - ie. numerical values. These values are received by the Dexter board using the External Interrupt mechanism, like we have done for the flow sensor in the IoT Water Meter project.

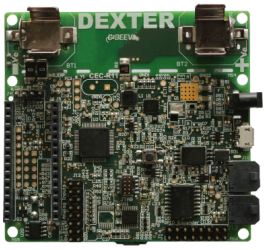
We then use these to compute parameters like power factor, power and energy consumed. This time, we store the data in the Dexter's on-board FRAM memory, using the I2C Protocol for reading and writing. .

Next, we transmit the data stored in the FRAM to the ESP32 Wifi module via the UART interface. The ESP32 Wifi module then 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.

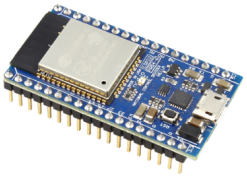


Components (*Provided in kits sent by Build Lab - IITM)

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



Dexter board

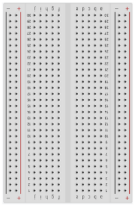

ESP32 Wifi module

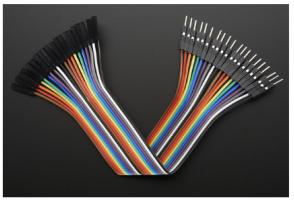
ZMPT101B
250V Voltage sensor


ACS712 20A
Current sensor


2-pin plug
with wiring


USB cable


Breadboard


Jumper wire- male to male,
male to female


Bulb

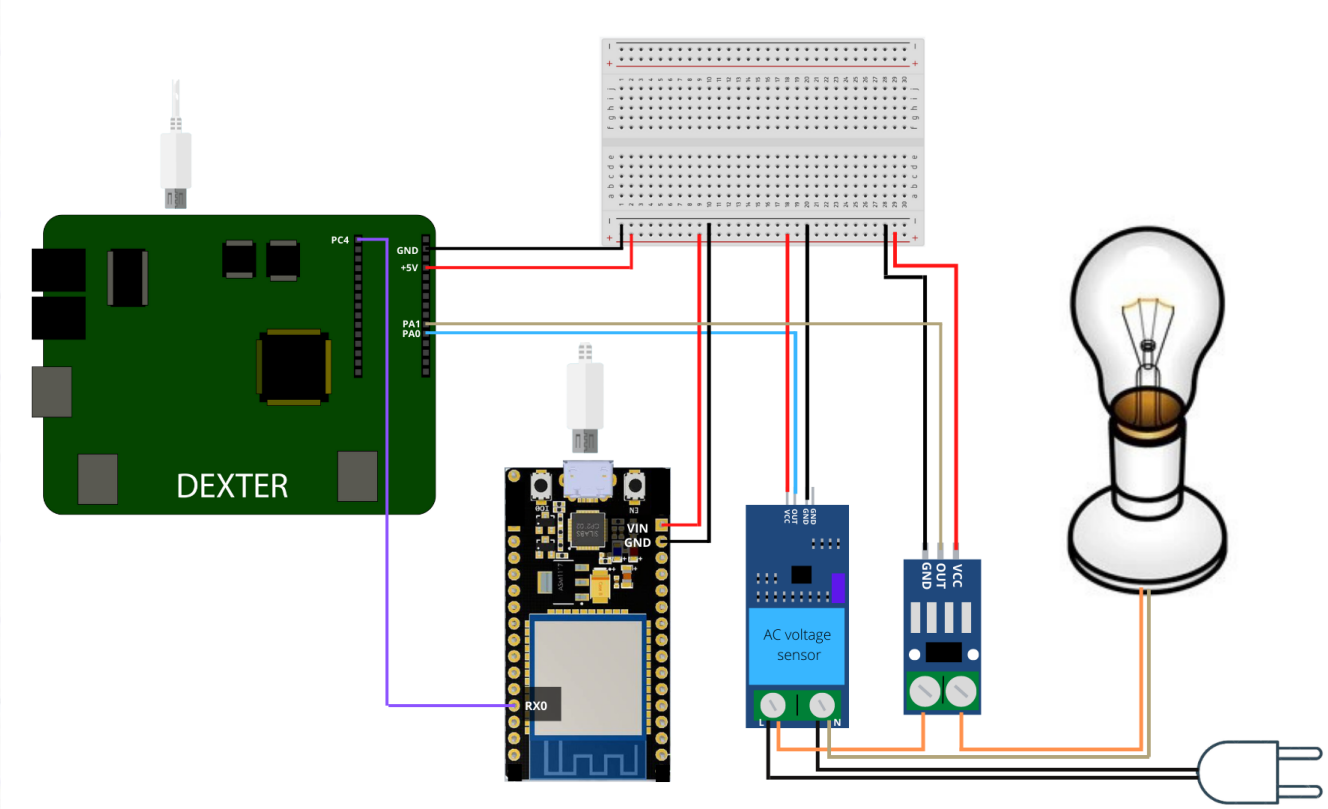

Holder

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.

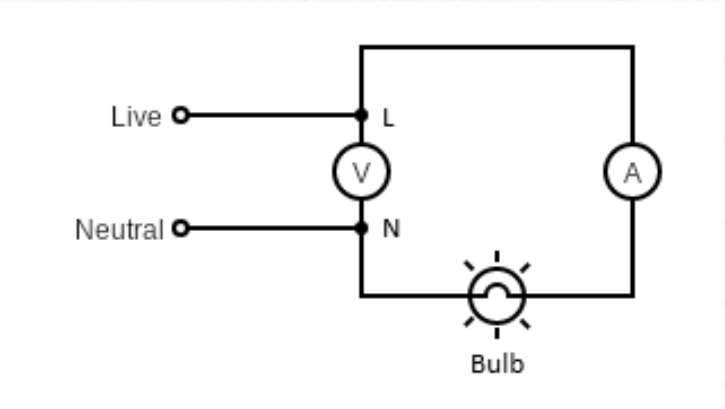
Circuit Diagram

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

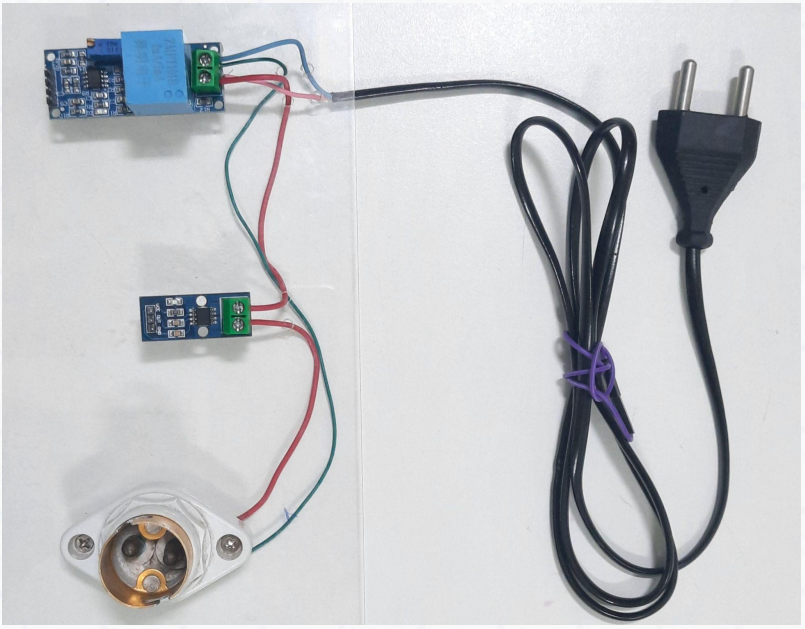


Detailed Connection Steps

Step 1



From the end of the 2-pin plug cable, cut off 3 segments of wires. With one segment, connect the Voltage sensor's terminal labeled 'L' to any one terminal of the Current sensor. Using another wire segment, connect the remaining terminal of the Current sensor to one terminal of the Bulb holder. With the last wire segment, connect the other terminal of the Bulb holder to the Voltage sensor's terminal labeled 'N'. Connect the ends of the 2-pin plug cable also to the 'L' and 'N' terminals of the Voltage sensor.



Step 2

Take 3 female-to-male jumper wires and connect the pins of the ZMPT101B voltage sensor as shown:

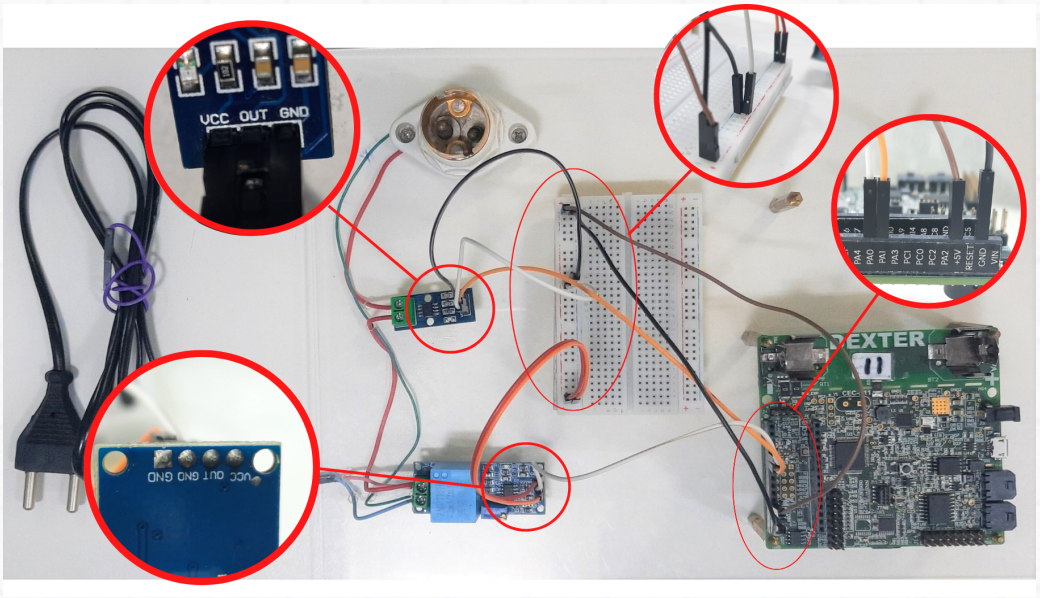
- **VCC** to **Red line** of breadboard
- **OUT** to **PA0** of the Dexter
- **GND** to **Blue line** of breadboard

Take 3 more female-to-male jumpers to connect the ACS712 current sensor:

- **VCC** to **Red line** of breadboard
- **OUT** to **PA1** of the Dexter
- **GND** to **Blue line** of breadboard

Additionally, Take 2 more male-to-male jumper wires and connect them as follows:

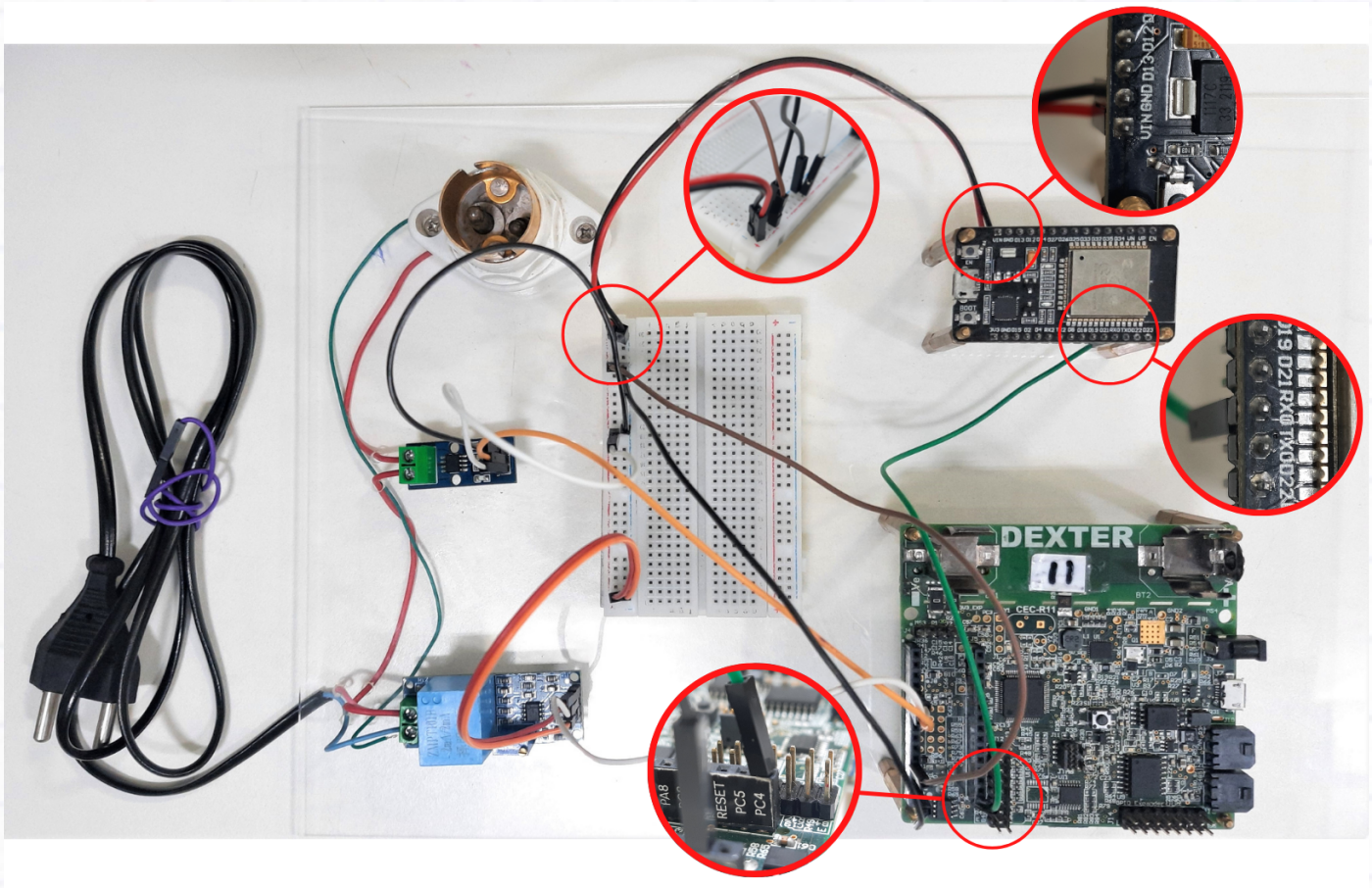
- **+5V** of Dexter to **Red line** of the breadboard
- **GND** of Dexter to **Blue line** of the breadboard



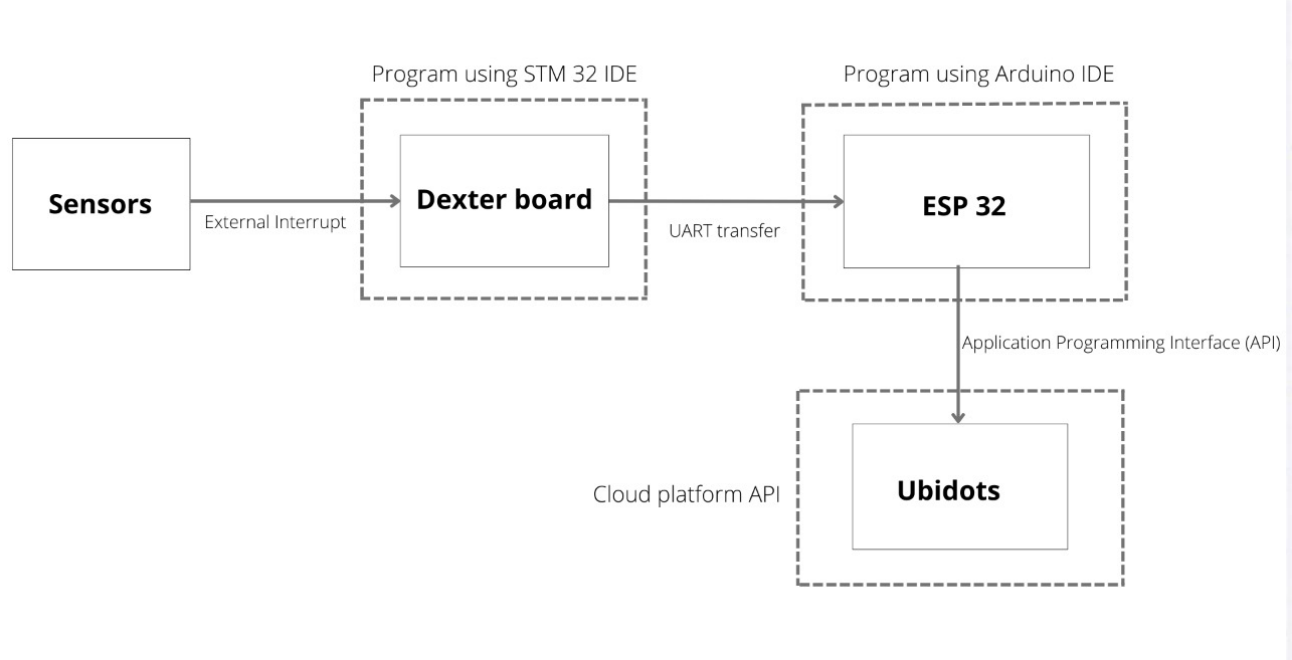
Step 3

Take 3 male-to-female jumper wires and connect the pins (**VIN**, **GND**, **RX0**) of the ESP32 Wifi module:

- **VIN** to **Red line** of the breadboard
- **GND** to **Blue line** of the breadboard
- **RX0** to **PC4** of the dexter board



Software



For Dexter

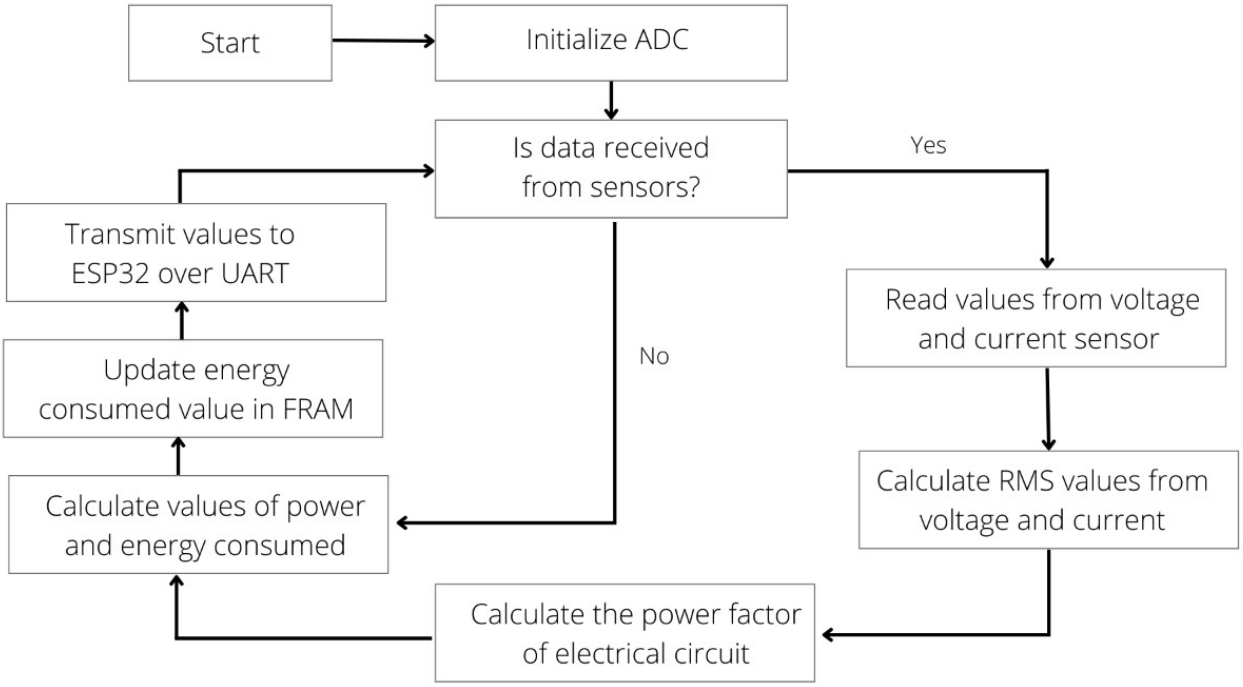
Downloads & Installation

1) Download the Project Workspace file **Dexter_Energy_Meter.zip**' given in the project page on the Build Club website.
2) In the **Workspace** folder in your C: drive, create a new folder named '**IoT_Energy_Meter**'.
3) Now, as done in every project: i) Launch the STM IDE, ii) Select the **IoT_Energy_Meter** folder as workspace, iii) Import the ZIP file **Dexter_Energy_Meter.zip**, iv) Navigate to **app.c**

NOTE: Before getting into the software for the project, it is important to understand the concept behind it and get an overview of how the project works. For this reason, we recommend revisiting the **Concept** section at the start of the manual, where we explain how this project works.

Flowchart of the Code

This flowchart diagram captures the flow of logical steps needed to implement the IoT Energy Meter project. Once this is firmly understood, code can be easily written to implement the project and make it work.



Variables & Functions

Variables:

1) **V_sensor_reading**

   Used for storing the value received from the voltage sensor.

2) **I_sensor_reading**

   Used for storing the value received from the current sensor.

3) **meter.status**

   A status flag variable that tracks the status of the energy meter. It can equal to either of 2 states - DATA_RECEIVED or V_I_COMPUTED.

4) **meter.vrms**

   Stores the Root Mean Square Voltage value (V_{RMS})

5) **meter.irms**

   Stores the Root Mean Square Current value (I_{RMS})

6) **meter.power_factor**

   Stores the value of Power Factor of the circuit.

7) **meter.power**

   Stores the value of Electrical Power flowing through the circuit.

8) **meter.Energy**

   Stores the value of total Electrical Energy consumed by the circuit.

Functions:

1) **ADC_init();**

   Initialises the Analog to Digital Converter channel for reading the current and voltage sensors..

2) **Read_voltage();**

   Reads the voltage sensor and returns the reading value.

3) **Read_current();**

   Reads the current sensor and returns the reading value.

4) **Calc_RMS_values(**V_sensor_reading, I_sensor_reading**);**

   Inputs the values read by the sensors and calculates RMS values.

5) **Measure_power_factor(**V_sensor_reading, I_sensor_reading**);**

   Inputs the values read by the sensors and calculates Power Factor.

6) **Calc_power_energy(**vrms, irms, power_factor**);**

   Calculates the Power and Energy consumed by the electrical circuit.

7) **UART_Transmit();**

   Transmits the values of the electricity parameters to the ESP32 device via UART communication interface.

8) **Read_FRAM(**data variable**);** & **Write_FRAM(**data variable**);**

   Used for reading & writing specific data from and to the FRAM memory.

Implementing the Code

Let us convert our flowchart of steps into working code for the **App** function. Use the corresponding functions and code shown below for each step.

1) Initialise ADC: **ADC_init();**

2) The code will now enter an infinite while loop

```
while(1)
{
        // Rest of code
}
```

3) Is data received from the sensors?:

```
if(meter.status == DATA_RECEIVED)
{
        // Rest of code
}
```

4) Read values from Voltage and Current sensors:

```
V_sensor_reading = Read_voltage();

I_sensor_reading = Read_current();
```

5) Calculate RMS values of Voltage and Current:

```
Calc_RMS_values(V_sensor_reading, I_sensor_reading);
```

6) Calculate the circuit's Power Factor:

meter.power_factor = Measure_power_factor(V_sensor_reading, I_sensor_reading);

7) Calculate Power, Energy and update values in FRAM:

Calc_power_energy(meter.vrms, meter.irms, meter.power_factor);

8) Transmit values to ESP32 via UART:

UART_Transmit();

After implementing these code commands inside the **App()** function in the order specified, we will write the code for the **Calc_power_energy** and **UART_Transmit** functions.

Below the **App()** function, you will find an incomplete function named **Calc_power_energy** having 3 input parameters - **vrms**, **irms** & **pf** (recall how we called the function in the App(), with 3 similar parameters). Using these parameters, you must compute the values of power & energy consumed and update them to the FRAM memory. In the space designated for User Code, implement the following steps:

```
meter.power = vrms * irms * pf;
Read_FRAM(meter.Energy);
meter.Energy += (meter.power / 3600);
Write_FRAM(meter.Energy);
```

Next, we have to write the **UART_Transmit** function. To transmit data over UART, we simply use the **printf** command. Eg.

printf("%06.2f", meter.vrms);

This command sends the value of **meter.vrms** as a 6-digit number, 2 of which come after the decimal point.

Inside the User Code section in the **UART_Transmit** function, print the data one after the other using the above **printf** commands in the order specified:

1) **meter.vrms**
2) **meter.irms**
3) **meter.power_factor**
4) **meter.power**
5) **meter.Energy**

Note that for sending **meter.Energy**, add a "**\r\n**" at the end of the "**%06.2f**". This is to tell the receiving device that one set of data values has been sent.

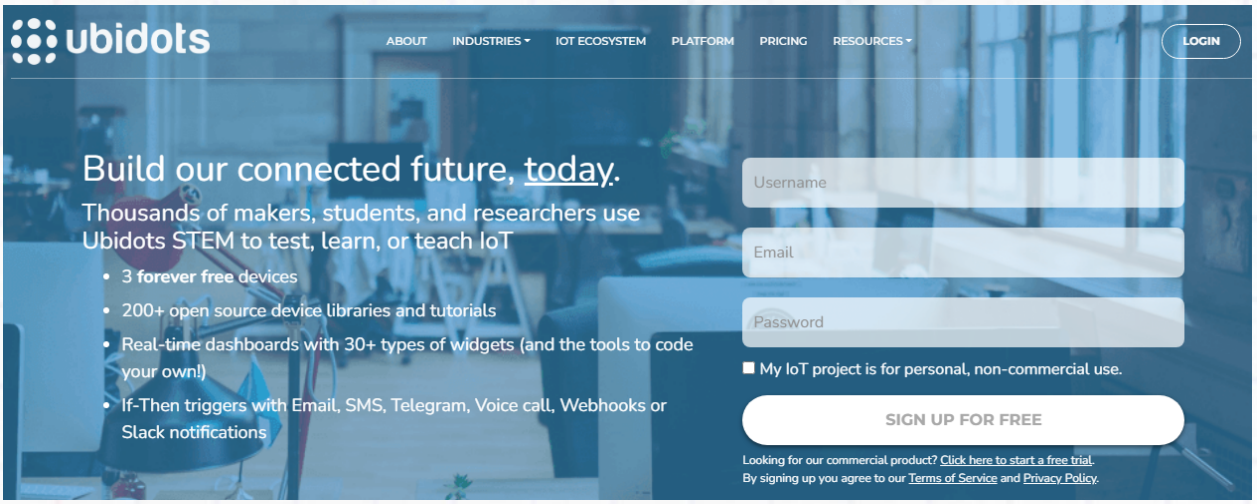
printf("%06.2f\r\n", meter.Energy);

With this, the code in the STM32 IDE for the Dexter is completed. The code can now be uploaded to the Dexter board by hitting 'Run'. Note that the project will not work yet as it is incomplete. It will only work once the code for the ESP32 is written and the Ubidots platform is set up.

ESP32 & Ubidots Setup

Downloads & Installation

1) Download and install the Arduino IDE from here, if you haven't already.
2) From the IoT Projects page on the Build Club website, download the **ESP32_Energy_Meter** ZIP file and extract it.
3) 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.



Steps*

(*Steps 1-5 are required only if not done previously in the IoT Water Meter project)

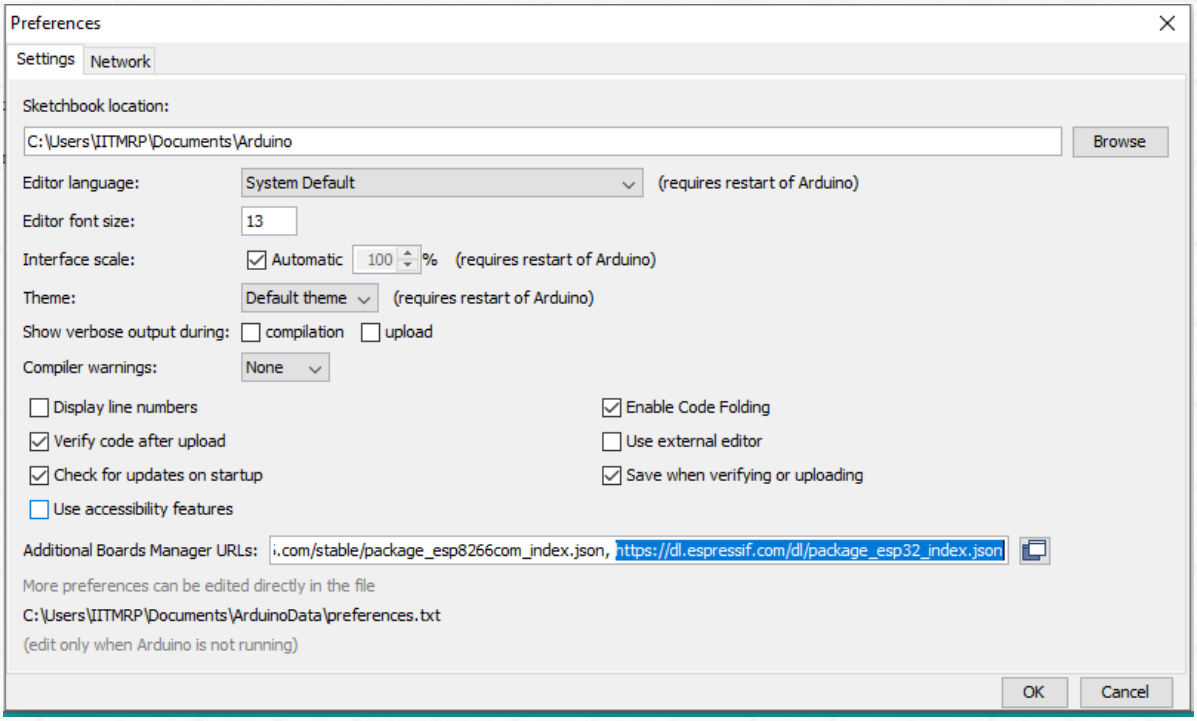
1) Inside the extracted **ESP32_Energy_Meter** folder, you will find 2 folders named **esp32-mqtt-main** & **pubsubclient-master**. Copy them and paste them inside the 'Libraries' folder located in Documents > Arduino > Libraries.

2) Launch the Arduino IDE. Go to **File** > **Open** and select the **ESP32_Energy_Meter.ino** file from the **ESP32_Energy_Meter** folder. The code will now open in the Arduino IDE.

3) Next, go to **File** > **Preferences**. In '**Additional Boards Manager URLs**', if the textbox is empty, paste the following link inside it:

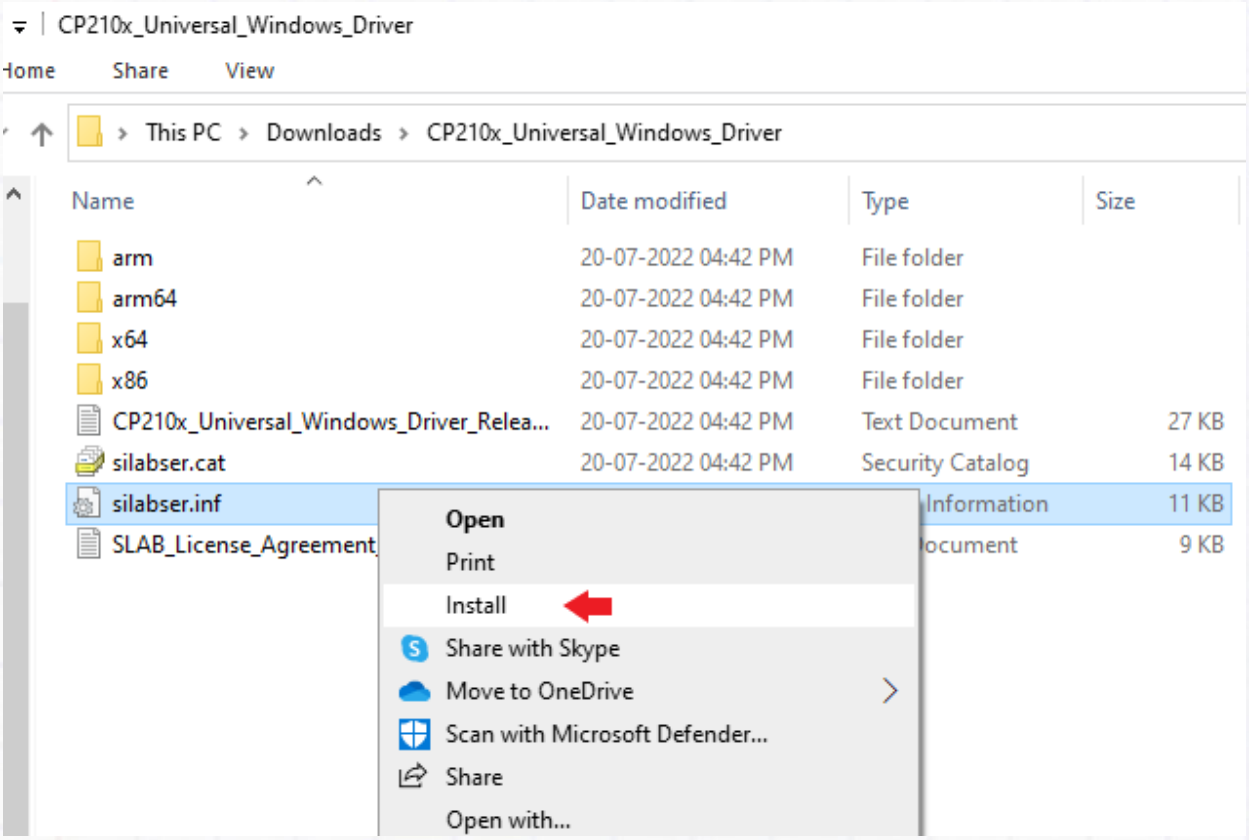
   https://dl.espressif.com/dl/package_esp32_index.json

   If the textbox already has some other links, put a comma after them and then paste the above link.



4) Next, go to **Tools** > **Board** > **Boards Manager**. In the search bar on the top, search **'esp32'** and install the package. This will take some time to install.

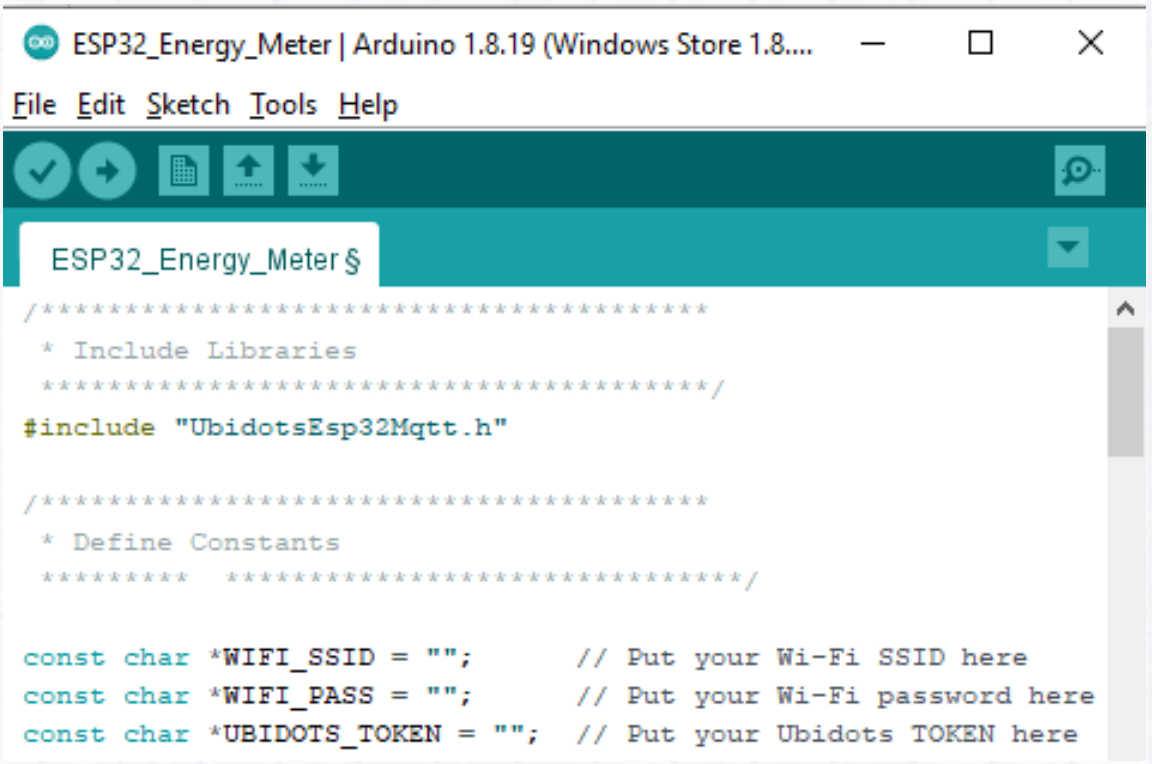
5) From this link, download the **CP210x Driver** for your OS (eg. CP210x Universal Windows Driver). Extract the ZIP folder. Right-click the **silabser.inf** file inside and select install. Follow the prompts until installation is successful. Restart your PC once installation completes.



6) Use 1 USB cable to connect the ESP32 device to your computer and the other to connect the Dexter to the computer for uploading code. Before uploading code to ESP32 from your computer, disconnect the jumper wire from the PC4 pin of the Dexter. This is necessary for the code to upload onto the ESP32.

Open the **ESP32_Energy_Meter.ino** file again in the Arduino IDE. Go to **Tools > Board > ESP32 Arduino** and select **ESP32 Dev Module**. In the same Tools menu, under **'Port'**, select the available COM port (eg. COM10).

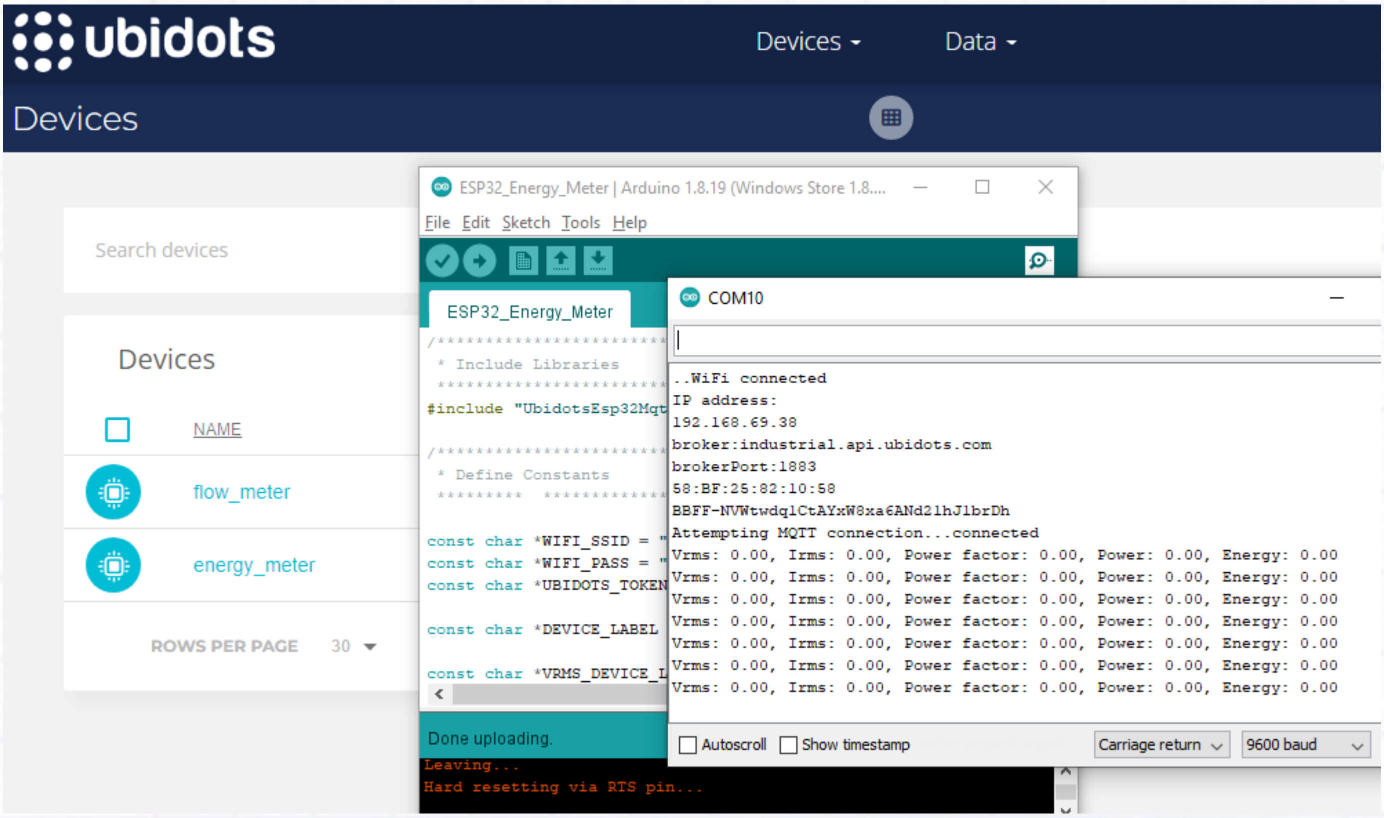
7) Now go back to the Energy Meter code in the Arduino IDE. Fill in your wifi connection's name and password within the empty quotation marks next to the **WIFI_SSID** and **WIFI_PASS** variables. Ensure the Wifi network has no firewalls, else the ESP32 won't connect to it.



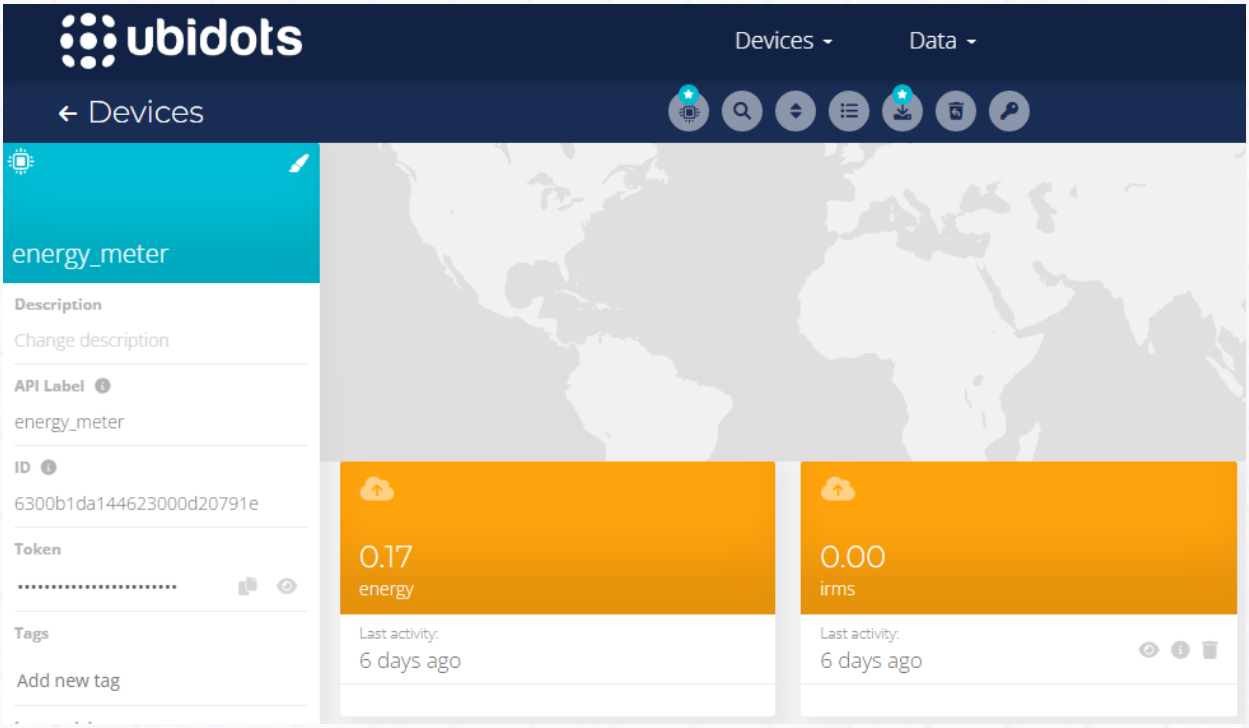
8) On the Ubidots website, click on your profile icon and select API credentials. Copy the 'Default token' on the top right part of the page and paste it into the **UBIDOTS_TOKEN** variable, which is immediately below the Wifi variables in the Arduino code.

9) 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.

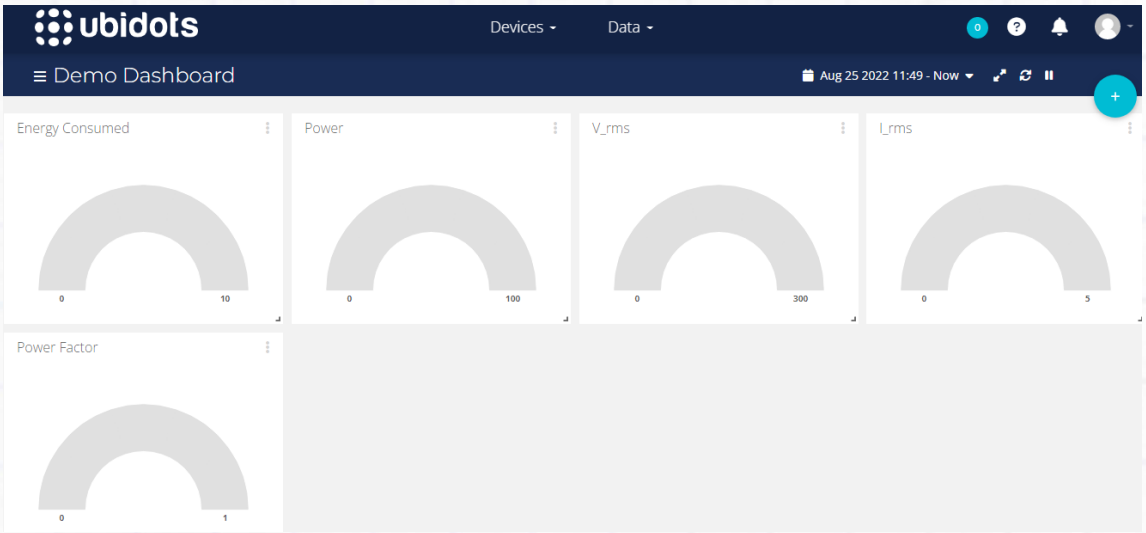
10) Now reconnect the jumper to the PC4 pin as before. Click the magnifying glass icon on the top-right of the Arduino IDE. It will open the Serial Monitor. If the connection was successful, the Serial Monitor should look like:



11) 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**.



12) Go to Data > Dashboards and click the + symbol. Select 'Gauge' and name it 'Energy consumed'. Click 'Add Variables' and select the **energy** variable under the energy_meter device. Create another gauge similarly for the **power** variable and if you wish also for **vrms**, **irms** and **power_factor**. Make sure you select a suitable value range for each gauge.



Tasks

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) Run the STM code on the Dexter and once completed run the Arduino code on the ESP32, making sure PC4 isn't connected. Reconnect the jumper to the PC4 pin.

   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 **app.c** in the STM 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 **app.c**

4) 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!