UC Berkeley, ME100, Spring 2021
Lab 4: Using MQTT

Plan to complete these tasks during the week of March 1
Checkoff due on Gradescope by Monday March 9, 11:59pm

Introduction

In Lab 2, you developed an application to log electrical output data from your solar cell. It would be helpful to be able to analyze these data with graphing software on a personal computer, or to transmit the data to other people who could analyze it. Transmitting the data over I2C, as in Lab 2, is fine for very local communication, but not suitable if the sensing device is in a different location from the person analyzing the data. Instead, we would like to use the Internet for data transmission. The objectives of this lab are therefore to:

- Become familiar with and learn to use the MQTT protocol to publish data-containing messages from your ESP32 over the Internet.
- Learn to publish MQTT messages from your computer which can be subscribed to (received) by your ESP32.
- Use the `matplotlib` library in Python to produce graphs of your measurement data.
- Analyze the data to establish the maximum power point of your solar cell.
- Become familiar with the free MQTT broker `broker mqttdashboard.com` and also the Thingspeak MQTTbroker from The Mathworks.

Step 1: Checking out MQTT

We will use the free broker available at `broker mqttdashboard.com`. from HiveMQ. It is not secure: no username or password are needed. We will use a simple MQTT Dashboard from HiveMQ that will help to explore MQTT topics and debug whether our messages are getting sent or received.

1. Using your favorite browser, navigate to https://www.hivemq.com/public-mqtt-broker/ and select the try MQTT Browser Client. You will be brought to a new page which is your MQTT browser, select connect. You should see a green LED with a message connected.
2. Edit the MicroPython program called `mqtt_hello_world.py` (available on bCourses under Files > Labs > Lab 4). This program was demonstrated in Lecture 17; it publishes MQTT test messages periodically.
    - Edit the `session` string to `<yourname>/esp32/helloworld`, where `<yourname>` is your name or a unique identifier (**do not include the <> symbols**).
    - Edit the `broker` string to match the MQTT broker we are using. (broker.mqttdashboard.com)
3. Run `mqtt_hello_world.py` from the `shell49` prompt with your ESP32 connected.

- Confirm that the ESP32 is connected to the Internet. The program will return an error message if it is not connected, or report the IP address if it is. For this to work, your ESP32 will need to have on its flash memory the `boot.py` script that you developed in Lab 1, which connects to your WiFi network.

4. Back to the MQTT Browser Client page from step one. Copy your subscription topic from mqtt_hello_world.py (<yourname>r/ESP32/helloworld/**host**/hello) to the publish topic on the MQTT Browser Client page. Select add Subscription. Under topic copy your publish topic from mqtt_hello_world.py (ganwar/ESP32/helloworld/**mcu**/hello). Verify that the Browser Client receives the messages sent by the ESP32.
5. Say hello back to your microcontroller by publishing in MQTT Bowser Client. You should receive your message on your command prompt.

Step 2: Sending data from ESP32 to remote host and writing a plot client

Frequently microcontrollers are used to perform measurements and the results sent to the cloud for processing by one or more clients. This part of the lab shows how to send data using MQTT from a microcontroller to a host computer (e.g. your PC or laptop) for plotting.

The application consists of two separate programs:

- `mqtt_plot_mcu.py` which is executed on the microcontroller.
- `mqtt_plot_host.py` which is executed on the host (your computer).

First, download these sample programs from bCourses and study them. Change the value of the `session` variable and broker variable in both programs to something unique like your name or UID. Change the port parameter to 1883. Then, follow the following steps to run these programs:

1. Run the following commands in a terminal window (this step is only needed once):

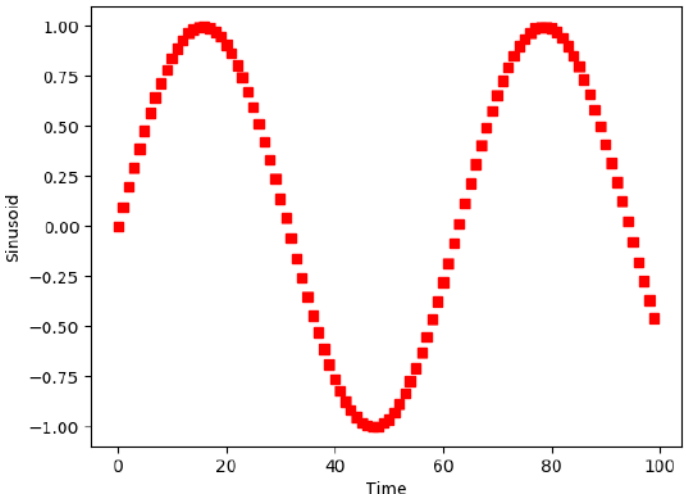
```
pip install paho-mqtt
pip install matplotlib
```

2. Start `mqtt_plot_host.py` on the host computer from a terminal window:

```
python mqtt_plot_host.py
```

3. Open a separate terminal window, start `shell49`, and run `mqtt_plot_mcu.py` on the microcontroller.

Check the messages in both terminal windows for errors. If everything goes well, a plot like the one below will appear:



Step 3: Complete solar characterization app

Now you have all the components to build the solar cell characterization IoT app! These include:

- Hardware components: circuit with solar cell, potentiometer, INA219, and ESP32.
- Software components to:
    - Connect the ESP32 to the internet: `boot.py`, stored on the ESP32, establishes a connection whenever the ESP32 boots (is turned on or reset) and the WiFi network is reachable.
    - Take I–V measurements from the solar cell using the INA219 and send data to ESP32 using I2C.
    - Send data from the ESP32 to a host computer using MQTT and plot the results.

Now you will put everything together, testing after each step. Note: Test your code using the `run` command in `shell49`. That way any `print` statements in your code can be output on your computer to help you debug. After you debug your code, you can remove or "comment out" the `print` statements and upload to `/flash/main.py` to run automatically.

1. Make a copy of `mqtt_plot_mcu.py` and rename it to something like `mqtt_plot_mcu_solar.py`.
2. Modify `mqtt_plot_mcu_solar.py` to publish MQTT messages containing voltage, current, resistance and power measurements from your solar cell circuit, instead of the sample sine wave you published in Step 2 above. You can accomplish this goal by incorporating code from the script `measure_solar.py` that you modified in Lab 2. You will need to incorporate:
    - The relevant `import` commands,
    - The code that sets up and configures the INA219, and
    - Code that takes measurements from the INA219. Replace the code in `mqtt_plot_mcu_solar.py` that generates and publishes the test sine wave with code that publishes a message containing V, I, P, and the load resistance, say, every 0.3 seconds.

- Your code should publish messages containing the data (on the …/data topic) while the load resistance is below 800 ohms. When the load resistance exceeds 800 ohms, the code should send a message on the (…/plot) topic to instruct the host to plot the data.

3. Make a copy of mqtt_plot_host.py and rename it to something like mqtt_plot_host_solar.py.
4. Modify mqtt_plot_host_solar.py as follows:
   - The code should receive the MQTT messages containing V, I, P and load resistance, and append the data to corresponding variables as long as messages are being received. You can model this code on how the example sine wave messages were processed in mqtt_plot_host.py. Also print the data to the screen as they are received.
   - Once any message is received on the …/plot topic (signaling that the load resistance has exceeded 800 ohms), your code should generate a plot of power against load resistance, from which you will be able to determine the maximum power point of the solar cell. You may wish to add other plots as well (e.g. using plt.subplots()), for debugging purposes.
5. Run python mqtt_plot_host_solar.py in one terminal window, and run mqtt_plot_mcu_solar.py from shell49 in another terminal. Ensure that the potentiometer is set to its minimum resistance value before you run mqtt_plot_mcu_solar.py. Then, with bright illumination on the solar panel, very slowly turn the potentiometer, watching the terminal window in which you are running mqtt_plot_host_solar.py to confirm that data are being received. When the resistance first exceeds 800 ohms, the plot of power versus resistance should be generated.
6. Stop a moment in awe, appreciating your accomplishment! Congratulations on your very first IoT app.

Step 4: Explore Thingspeak as an alternative way of working with MQTT

The Eclipse MQTT broker is just one of many brokers available. Another is *Thingspeak*, which is run by The Mathworks, the makers of Matlab, and provides a convenient web interface with which to analyze and plot data received in the form of MQTT messages. It is a commercial product but the free version to which you have access can log data at intervals of 15 seconds or longer. Here, you will use Thingspeak as alternative way of logging data from your solar cell circuit.

1. Either create an account at thingspeak.com, or, if you already have a Mathworks account (e.g. for downloading Matlab), you can use that.
2. From the Thingspeak screen, click New Channel, and give it a name related to your solar cell lab. Create two fields, Voltage and Current, and click Save Channel.
3. The new channel's dashboard will appear. The channel will be given a numeric ID (a 7-digit number). Click on the API Keys tab and make a note of the Write API Key.
4. Make a copy of mqtt_plot_mcu_solar.py and rename it something like mqtt_plot_mcu_solar_thingspeak.py. Modify this program as follows:
   - Amend the broker to mqtt.thingspeak.com
   - Amend the topic to channels/<channel ID>/publish/<Write API Key>. (Omit the <> symbols).
   - Amend the message format to:
     "field1=<Voltage numerical value>&field2=<Current numerical value>"

- Add a `sleep(15)` command so messages are sent no more frequently than the free version of Thingspeak will accept.

5. Run your modified `mqtt_plot_mcu_solar_thingspeak.py` and watch the data being logged in the web interface. Then explore the plotting functions within the Thingspeak interface. Click “Matlab Visualization”, “Create a 2-D Line Plot”, and write some simple Matlab code to create a live plot of power against load resistance. Because of the 15 second limit on the period of data acquisition, we do *not* expect you here to create a detailed power–resistance plot showing the peak power very clearly, as that could take a long time to acquire.

Gradescope signoff

We ask you to upload/answer the following:

1. A video showing your completed Eclipse MQTT solar cell app operating, with the terminal output on your computer screen clearly visible.
2. An image of the output power vs load resistance graph that is plotted by your python code
3. What is the maximum power you were able to obtain from your cell and at what load resistance did this occur?
4. Upload your code:
   - `mqtt_plot_mcu_solar.py`
   - `mqtt_plot_host_solar.py`
   - `mqtt_plot_mcu_solar_thingspeak.py`
5. Screenshot of your Thingspeak channel with the 2D line plot of power vs resistance visible (note that because of the slow sampling rate we do not expect many data points or a clear peak to be visible; we just want to see proof of principle).