Name	SID	Checkoff

Objectives: Complete IoT app started in last lab.

Attention!

The maximum voltage on any ESP32 pin is 3.3 V. Applying a higher voltage will damage the ESP32 (no smoke signal, the device will quietly die).

Before connecting anything to one fo the pins of the Huzzah32 board, verify (with a multimeter) that the voltage is no more than 3.3 V.

Parts and Tools

For this laboratory you reuse the components from the last lab.

Prelab

Complete the firmware for the solar-cell characterization hardware built in the last lab.

a) Check out MQTT.
Note: use broker `iot.eclipse.org` (no username or password) rather than iot49.eecs.berkeley.edu to avoid undiagnosed errors. Check https://iot.eclipse.org/getting-started for the correct values to use for the `hobbyquaker` MQTT web client (follow the instructions for websocket).

- On the host (EECS computer or your own), start the `hobbyquaker` MQTT web client (see lecture for an example) and subscribe to the above topic (e.g. `cynthia/esp32/hi`).
- Write a MicroPython program that publishes an MQTT message each second. Use one of the free brokers suggested in the lecture.
  - Topic: `yourname/esp32/hi` (replace 'yourname' with your name, without blanks. E.g. `cynthia/esp32/hi`).
  - Message: `Hello i` (replace 'i' with a random number so that each message sent is different to help testing the setup).
- Make sure the ESP32 is connected to the internet and run the MicroPython program. To execute a MicroPython program on the ESP32, you may either
  - Install it on the ESP32 as `flash/main.py`. The program will be executed each time you press the reset button (after `boot.py`). Alternatively,
  - Use `shell49`'s run command.

    ```
    run program.py
    ```

    sends `program.py` stored on the host computer to the ESP32, executes it, and prints output from `print` statements on the console. Make sure to start `shell49` in the directory where `program.py` is stored.
    The `ls` command in `shell49` lists all files in the current directory. `cd` changes the working directory. Use the `help` command in `shell49` to learn about available commands.

- Verify that the MQTT web client receives the messages sent by the ESP32.

b) Install the plotting software from https://github.com/bboser/iot-plot.

c) Write a simple program to check out remote plotting from the ESP32. E.g. the example presented in class. Use the `iot.eclipse.org` broker and a unique string for the `session` (e.g. `cynthiassession`) to avoid confusion when several people use the same MQTT broker at the same time.

d) Start the plotserver on the host computer with

```
plotserver -s session
```

Replace `session` with the same string as above (e.g. `cynthiassession`). Now run the sample program on the ESP32. The `plotserver` announces when it receives a new plot (from the esp32) and saves it on the host computer as a pdf file. Check it out!

Use Piazza, office hours, and discussions to get help and your questions answered so that you will come completely prepared for completing your first IoT app in the lab!

Lab

Complete Solar Characterization App

Now you have all the components, individually tested, to put together the solar cell characterization IoT app! They include:

- Circuit with solar cell, potentiometer, INA219, and ESP32.
- Software components to:
  a) 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.
  b) Take current and voltage measurements from the solar cell with the INA219 connected to the ESP32 via I2C.
  c) Send data from the ESP32 to a host computer (e.g. lab computer or laptop) using MQTT, and plot the results.

To put all the components together, combine the individual pieces one at a time, testing the outcome after each step.

a) Check that your ESP32 connects to the wifi network. An easy way to do this is to check its IP address: an value of `0.0.0.0` indicates connection failure. In the `repl` console, execute the following Python statements:

```
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
print(wlan.ifconfig()[0])
```

b) Run initializations: INA219, MQTT, PlotClient, and create new plot series. Choose a unique `session` name. Run your code and check for error messages.

c) Add the code to measure voltage and current with the INA219, compute the power and resistance (beware of division by zero!) and send the results to the host. Do this in a loop, recording a measurement every 0.3 seconds and stopping when the measured resistance exceeds 8 kΩ. To reduce the amount of redundant data, send only measurements for which the measured power changed by at least 10 percent from the previous measurement. Print out the value of each measurement sent to the MQTT broker.

d) Add the `plot_series` statement to instruct the `plotserver` to create a plot. As elsewhere, add a print statement to give feedback when testing the program. You can remove these print statements once everything has been verified to operate as intended.

e) On the host, start the `plotserver` with the same value for `session` as the one used in the program running on the ESP32.

f) Turn the potentiometer to the stop corresponding to minimum (near zero) resistance and turn on the illumination for the solar cell.

g) Start the program and slowly turn the potentiometer towards the stop corresponding to maximum (approximately 10 kΩ) resistance. Verify (by checking the output from the print statements in your program) that the ESP32 sends measurement results to the MQTT broker and issues the plot command once the resistance exceeds 8 kΩ.

h) Check the `plotserver` output for a message that it generated a plot or, error messages. Look for the plot at the location indicated by the `plotserver`.

i) Tweak the program (e.g. the 10 percent change) to make the plot look good.

j) Stop a moment in awe, appreciating your accomplishment! Congratulations to your very first IoT app. Checkoff: ☐