Name	SID	Checkoff

Objectives: Build system that can run off a rechargeable battery and harvest energy from the solar cell. Explore ways to reduce power consumption to enable essentially "forever" on harvested power.

Parts and Tools

For this laboratory you need the Lithium Polymer battery and the components from the last lab.

Prelab

To complete the prelab you will need to run code on the ESP32 and a host computer. You may do this in 199 Cory Hall, the laboratory (if available and not in use), or on your own computer.

a) Read and understand the entire lab guide.

b) Complete Lab 4. If you encountered problems with the `plotserver` follow the instructions posted on bcourses in `Files/Guides/MQTT.pdf`. Perform the measurement results with available light and save the created plot for presentation during the prelab checkoff at the start of the lab session.

c) Study the manual for the E3631A laboratory power supply posted on bcourses (under Files/Guides).

d) Create an account on https://thingspeak.com and follow the instructions to set up a channel with fields for `voltage` and `current`.

e) Write a micropython program to send `voltage` and `current` measurements from the INA219 to `thingspeak`. Go to the `thingspeak` website to verify that the data has been correctly received. Print a screen shot of your data displayed on the `thingspeak` website.
Use the following values for MQTT:

```
broker = "mqtt.thingspeak.com"
topic = "channels/" + TS_CHANNEL_ID + "/publish/" + TS_WRITE_KEY
```

`TS_CHANNEL_ID` and `TS_WRITE_KEY` are values you obtain from `thingspeak` when setting up the channel. Format the message as follows:

```
message = "field1={}&field2={}".format(v, i)
```

where v and i are the values of the measured voltage and current.

Get started early on the prelab to have sufficient time for resolving potential problems. Use Piazza, office hours, and discussions to get help and your questions answered so that you will come completely prepared.

Prelab Checkoff

At the start of the lab session, show the following results from the prelab to the instructor to get credit. Note: you need to complete all parts of the laboratory to pass the course (regardless of points earned).

a) Printout of plot of voltage, current, and power versus resistance of the solar cell. Be ready to show and explain the code used to produce this plot. Checkoff: ☐ (25 points)

b) How many separate voltages can the E3631A simultaneously generate? Checkoff: ☐ (5 points)

c) Printout of your data displayed on the `thingspeak` website Checkoff: ☐ (20 points)

d) Familiarity with lab guide Checkoff: ☐ (5 points)

Lab

Characterize Solar Cell

If you did not have appropriate lighting to characterize the solar cell in the prelab, rerun this test in the lab with artificial light. Checkoff: ☐ (5 points)

Measure ESP32 Current Draw

To measure the current draw of the ESP32, we need to power the HUZZAH32 board from the laboratory power supply, rather than the USB port.

Lab supplies cannot only be programmed to a precise output voltage, but also be set to limit the maximum output current. This feature is extremely useful in experimental setups as it may protect a circuit from damage in the case of faulty wiring or other errors. It is good practice to always set a current limit only slightly higher than the current required by the experiment under normal operation. A simple way to check that the current limiting works as expected is to initially set the limit deliberately too low and verify that the supply indicates overload.

a) Program one of the outputs of the lab supply to 3.3 V. Verify with the DMM.

b) Set the supply current limit to 50 mA. Verify by shorting the supply output with the DMM set to measure current. The DMM should show no more than 50 mA current.

c) Set the supply current limit to the expected current drawn by the HUZZAH32 board under normal operation.

d) Verify that the HUZZAH32 connects to the internet after reset (based on code in `boot.py`).

e) Disconnect the USB cable from the HUZZAH32 (and the battery, if connected) and power it instead from the lab supply programmed to 3.3 V. Connect the supply between the `VBAT` (3.3 V) and `GND` (0 V) pins of the HUZZAH32 board with the DMM configured as ammeter in series. Peruse the pin diagram posted at https://github.com/bboser/IoT49. Record the current drawn.

f) Disconnect the lab supply, and reconnect via USB. Test the following program on the ESP32 (be sure to be able to explain the operation of the program during the prelab checkoff!):

```python
from time import sleep
from machine import deepsleep, Pin
from board import LED

led = Pin(LED, mode=Pin.OUT)
led(1)
print("awake")
sleep(60)
print("deepsleep")
led(0)
deepsleep(60000)
```

The LED should be on for approximately 60 seconds, followed by off (during deepsleep) for another 60 seconds.
Once verified, flash the program to the ESP32 as `/flash/main.py`.

g) Disconnect the USB cable from the HUZZAH32 (and the battery, if connected) and power it instead from the lab supply programmed to 3.3 V. Connect the supply between the `VBAT` (3.3 V) and `GND` (0 V) pins of the HUZZAH32 board with the DMM configured as ammeter in series. **Important:** Set the current range to 1 A fixed (rather than autoranging) for this test. Otherwise the DMM interrupts the circuit when changing the measurement range, causing the ESP32 to reboot. You should see the current alternating between normal operation (LED on) and deepsleep (LED off). Record the two measured currents and demonstrate the setup to the instructor. Checkoff: ☐ (15 points)

h) Disconnect the lab supply and reconnect by USB. Reflash micropython to the ESP32 to remove `/flash/main.py` and get back a REPL prompt. Reflash `/flash/boot.py`.
Suggestion: A better solution is to configure a pin of the HUZZAH32 as input with the pullup enabled. Modify `/flash/main.py` to enter deepsleep only if that pin is high (1). To disable deepsleep, simply connect a wire between that pin and GND. An alternative would be to terminate the program after, e.g., 10 runs using a counter value stored in RTC memory (which is retained during deepsleep).

Operation from Battery and Solar Power

In this part we will power the ESP32 from the battery and then connect the solar cell for recharging.

a) Flash a `/flash/main.py` to the ESP32 that blinks the LED for one minute and then terminates. Disconnect the USB cable and power the HUZZAH32 from the battery. Verify that the LED blinks.
. Checkoff: ☐ (5 points)

b) Delete `/flash/main.py`.

c) Modify your program according to this template:

```
# turn on LED
...

# measure solar cell voltage and current with the INA219
# and send result to thingspeak
...

# turn off LED and
# enter deepsleep for 10 seconds
# (for testing, increase to 5 minutes when code is verified)
...
```

Verify the program (check the `thingspeak` website that values are uploaded) and flash to `/flash/main.py`.

d) Disconnect the USB cable and connect the solar cell between `VUSB` and `GND` on the HUZZAH32 board.

e) Make sure the solar cell is well illuminated, then reset the ESP32.

f) If everything works correctly, the microcontroller runs `boot.py`, connects to the internet, and turns on the red LED. The yellow LED (battery charging indicator) is off since all the current from the solar cell is needed to power the ESP32, with any balance (if the solar cell does not generate sufficient power) coming from the battery. Now `main.py` runs, takes a measurement with the INA219 and sends the result to `thingspeak`. The red LED turns off, and your program executes `deepsleep`, the ESP32 turns off. Since now the solar cell generates more power than is needed, it charges the battery, evidenced by the yellow LED which now turns on. After the deepsleep period expires, the process repeats.

Check the recorded voltage and current values on the `thingspeak` website.
. Checkoff: ☐ (20 points)