EE49 Fall 2018

Lab 4 Battery Power & Deepsleep

Name	SID	Checkoff

Objectives: Build power management system for ESP32 that can use a combination of a rechargeable battery and energy from a solar cell. Explore methods to reduce the ESP32 power consumption that enable extremely long autonomous operation using harvested power. from a solar cell.

Parts and Tools

For this laboratory you need the Lithium Polymer battery and the components from the last lab. You will also use a programmable DC power supply. This is described below.

Figure 1: NI PXI-4110: Programmable DC Power Supply

In Hesse 122, the lab computers are connected to NI PXI-4110 made by National Instruments (see Figure 1). This device offers a DC voltage source. It can be programmed to set the supply voltage and to limit the current draw. Use Channel 0 (red wire) for the positive terminal of the voltage source and GND (black wire) as the negative terminal. You can get up to 6 Volts and 1 Amp. A header is provided for you to connect the supply to your breadboard (see Figure 2).

Note: Make sure you do not insert your header the wrong way which will short all 4 of its pins. This will result in high currents! Also be mindful about where you are connecting the voltage supply on your breadboard.

To use the power supply, log into the lab computers and launch the program from the task bar as shown in Figure 3. The computer should connect to the PXI and give you a front panel from which to control the power supply (see Fig 4). We will only be using Channel 0. From here you can adjust Voltage Level, Current Limits, and enable the supply.

WARNING: Applying higher than rated voltage or reversing polarity of the connection can potentially fry your board. You will be responsible for sourcing a new one if this happens. Do not enable a power supply until you are sure you have connected it properly and configured the output correctly.

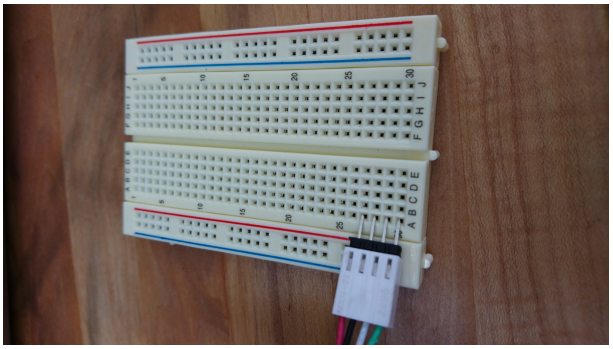

Figure 2: Pin header to connect to plug into breadboard.

Figure 3: Taskbar to launch program to manage the PXI.

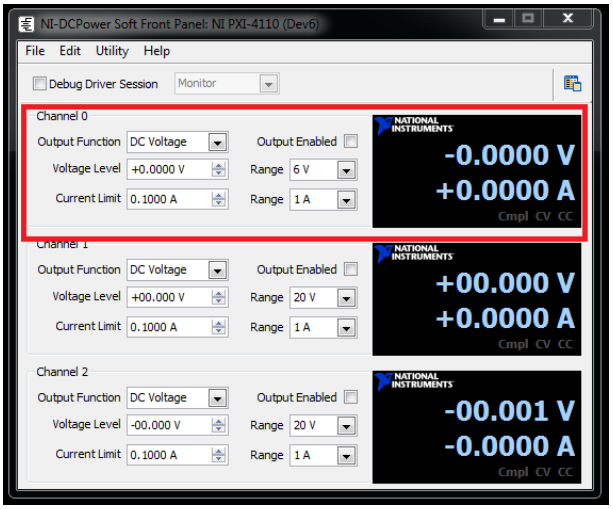

Figure 4: NI Front Panel

1 Prelab

For the prelab, you will need to run code on the ESP32 and a host computer (lab computer or your laptop).

1 Make sure you have completed Lab 3.

2 Read and understand the entire lab guide.

3 Study the guide above for the NI PXI-4410 Power Supplies.

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

5 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:

```
python mqtt_plot_host.py
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.

1. 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
2. Printout of your data displayed on the `thingspeak` website
3. Familiarity with lab guide

Checkoff: ☐

2 Lab

Part A: 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: ☐

Part B: 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 can be programmed to set a precise output voltage and also to set 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 deliberately set the limit too low and verify that the supply indicates overload.

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

2 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.

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

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

5 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.

6 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!):

```
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`.

7 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 Amp 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.

8 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).

Checkoff: ☐

Part C: 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.

1 Write a function `/flash/main.py` that blinks the LED for one minute and then terminates. Disconnect the USB cable, and power the HUZZAH32 from the battery. Hard reset the ESP32 and verify that the LED blinks.
Checkoff: ☐

2 Delete `/flash/main.py`.

3 Modify your program using 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).
Flash the program to `/flash/main.py`.

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

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

6 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: ☐