Software Technology of Internet of Things
Sampling Physical Phenomena Exercises:
Measurements Ahead!

Aslak Johansen <asjo@mmmi.sdu.dk>

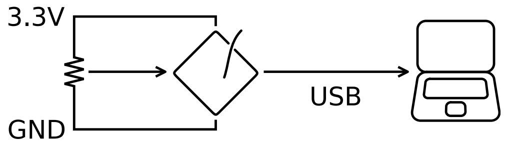
Mar 7, 2022

You have now reached level 3, and the quests are about to follow suit. Digital measurements are for noobs; analog is where the real action is! Be it in a serially connected component or through the AD converter built into the microcontroller.

Quest 1: Seeing Potential (15XP)

Let us start with the simplest possible setup, and see if you can manage to read the value produced by a potentiometer. While a big step up from the exercises of levels past, analog won't get any easier than this!

1. ***Locate Documentation*** more precisely, the part about the ADC[1].
2. ***Establish Physical Setup*** Use a breadboard and a potentiometer to establish the following setup:



   Note: The "output" of the potentiometer[2] may be fed into any ADC-capable pin on the device. Note down which pin you are connecting it to though. You are going to need it later.

3. ***Implement logic*** Implement a loop that continuously does an AD conversion and prints out the raw result over the serial line.

[1]`https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html`

[2]A potentiometer is mechanical component that allows a user to adjust the resistance of some resistor.

Hint: Which type of variable should be used to hold the result of an AD conversion?

4. ***First Test*** Build and flash the program to your device. Open a serial terminal to the device and play around with adjusting the potentiometer. Does it behave as you expect?

5. ***Refactor*** Move logic for performing a single sample into a separate function called `sample`. You decide its prototype.

6. ***Second Test*** Repeat step 4.

Quest 2: Guessing Game (10^{+1}XP)

Oh good, I see you made it! I never had any doubt of it . . . It's time for a treat; in the next quest you will implement a game. It's a classic, and a simple one at that.

1. ***Clone*** Make a copy of the program from Quest 1. You should keep `sample` and clear all but the ADC initialization part of `app_main`. The remainder of this quest will take place inside of `app_main`.

2. ***Main Loop*** Add a do-forever loop.

3. ***Greeting*** In the beginning of this loop, write out (over the serial link) *"Shall we play a game?"*. React to a y/n answer. In case of a "n", call `esp_restart` from `esp_system.h`. In case of a "y" you do the following:

4. ***Produce a Number*** Call `sample` to get a value representing the current state of the potentiometer. Keep this value safe, but secret.

5. ***Welcome Message*** Print out *"I am thinking about a number between 0 and 17 (both included). I bet you can't guess it in 8 tries!"*. Here you should replace 0, 17 and 8 with appropriate values. How do you figure out what 17 should be replaced by?

6. ***Game Loop*** Add a boolean variable called `done` and initialize it to `false`. Also, add a `uint8_t` variable called `count` and initialize it to zero. Next, add a while-not-done loop that is nested inside the loop from step 2. The rest of the steps take place inside this loop.

7. ***Query User*** Print out the text *"You have guessed 7 times. Please try again."* (in case of a `count` value of seven).
   Bonus: Use an inline if-then-else to choose between *"time"* and *"times"* depending on the value of `count`.

8. ***Fetch Answer*** Read a line from the serial connection, and parse it to an integer. Give an appropriate error message if the parse operation fails.

9. ***Evaluate Response*** Determine whether the response is below, equal to, or higher than the number from step 4.

10. ***Produce Feedback*** If there's a match you should print out *"You got it!"* (and add a text stating whether it was managed within the number of tries from step 5) and break out of the inner loop. Otherwise, you should print out whether the guess was too high or too low.

11. ***Test*** Never forget the importance of testing!

12. ***Bonus*** For an extra 1XP you can add a cheeky "the only winning move is not to play" remark and break out of the inner loop should the player fail to guess the number in the number of tries from step 5.

Quest 3: Getting Real (10XP)

So far you have worked with a potentiometer. That's nice, but real analog sensors require a curve to be applied for us to see human-recognizable values. In this quest you will construct such a curve and apply it the raw values from a temperature sensor in order to produce temperature measurements in $^{\circ}C$ rather than raw AD converter values.

1. ***Physical Setup*** Replace the potentiometer with an analog temperature sensor.
   - We are using the Texas Instruments LMT86. The datasheet will tell you which pins need to go where.

2. ***Verify Setup*** Run your application from Quest 1 and verify that the raw ADC values go one way when you apply heat (e.g., a live finger) and another when you cool it (e.g., by removing the finger again).

3. ***Unit Conversion*** Make a function called `convert` that takes one parameter (of the same type as the return value of `sample`) and returns an `int8_t`. This function goes through the following steps:
   (a) The input value is a raw AD conversion. That means that it is somewhere between 0 (included) and 2^k (not included), where k is the number of bits in the AD conversion.
   (b) First, you should find the formula for converting this to a voltage. For this you need the *reference* for the AD conversion.
   (c) Second, you should find a formula for converting a voltage to temperature. Since this is sensor dependent, you should look for the elements in the datasheet for the sensor. Section 8.3.1 of the LMT86 datasheet contains all the data that you will need.
   (d) You need to produce an integral value (in $^{\circ}C$).
   (e) Combine these formulas into one and do the calculation is such a way that you loose as little precision as possible.

4. ***Verification*** Verify that the function gives a reasonable output using a variant of the code from Quest 1.

5. ***Test*** Test the function using a variant of the code from Quest 2.

Quest 4: Cracks are Forming (20XP)

This is very nice, but inside that signal lies a hidden truth. One that is not easily gleaned through a few hundreds of readings. In order to see it you must capture enough data to observe how the sensor setup responds to a range of environmental conditions.

If your Python is a bit rusty you might want to check up on it[3].

1. ***Prepare Logging*** You will need a few things for this:
   - A cup with hot or cold water.
   - A program that can timestamp individual lines received over a serial link[4].
2. ***Experimental Setup*** Use the setup from Quest 1 where:
   - The temperature sensor is in direct contact with the cup.
   - The device is running the application developed in Quest 1.
   - The laptop is running a program designed to timestamp each reading and store the result in a file.
3. ***Conduct Experiment*** Conducting the experiment is going to take a while. Ideally, you should wait for the temperature of the cup to reach equilibrium with the surrounding air.
4. ***Monitor Progress*** As data is continuously being appended to the logfile you can work on the file (in read mode, that is) without affecting the collection process[5]. Lets do that!
5. ***Plot Results*** Use the following python code to plot the contents of the logfile[6]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('readings.csv', names=['time', 'value'])

plt.figure(figsize=[5,10])
plt.xlabel('Time/[epoch]')
plt.ylabel('raw adc value/[]')
plt.scatter(df['time'], df['value'],label='measurements')
plt.legend(loc=2)
plt.savefig('plot.pdf')
```

[3]`https://github.com/aslakjohansen/enigma-python-intro/blob/master/doc/Enigma%20-%20Python%20Introduction%20Webinar%20Handouts.pdf`

[4]`https://github.com/aslakjohansen/serial-logger`
Binaries for popular platforms are available on itslearning.

[5]Technically, anything your computer does will affect the timestamping, but since we are not using this to draw hard conclusions, we can get away with it

[6]You will likely have to install the following python packages using `pip` or similar: `pandas`, `numpy`, and `matplotlib`.

6. ***First Evaluation*** Can you explain what you see?

7. ***Calculate Heatmap*** A heatmap is a 2d histogram, that is a 2d array of buckets (aka integers). For this particular purpose we want to split the time-dimension into reasonable steps and map the raw conversion value straight to a bucket. If we go for $t_{\text{count}} = 600$ steps on the time axis what we get is an array with each bucket spanning $(t_1 - t_0)/t_{\text{count}}$ seconds on the x-axis and 1 on the y-axis. Create nine functions:

   (a) `index` which, given a timestamp, returns the x-coordinate of the bucket.

   (b) `west` which, given the x-coordinate of a bucket gives the timestamp of the left edge of the bucket.

   (c) `east` which, given the x-coordinate of a bucket gives the timestamp of the right edge of the bucket.

   (d) `south` which, given the y-coordinate of a bucket gives the value of the bottom edge of the bucket.

   (e) `north` which, given the y-coordinate of a bucket gives the timestamp of the top edge of the bucket.

   (f) `filename2ts` which, given a filename loads in the contents of this file, parses each line and returns a list of timestamp$\times$value pairs.

   (g) `ts2heatmap` which, given such a list of timestamp$\times$value pairs, calculates and returns a heatmap in the form of a 2d list.

   (h) `max` which, given a heatmap in the form of a 2d list, returns a the highest bucket value.

   (i) `gen_colormap` which, given a max bucket value returns a function (e.g., a lambda) that maps a bucket value to a color of the form `"#rrggbb"` where `rr`, `gg`, and `bb` are hex values. To make zero-values stand out make sure to map them to some color that clearly does not fit on your chosen gradient.

   `south` and `north` will come in handy should you decide to experiment with the bucket size. Make sure that the edge of a bucket aligns with the closest edges of the closest four buckets.

8. ***Plot Heatmap*** In order to produce the visual heatmap you need to iterate through the 2d list. For each position you need to map the bucket value to a color and draw a rectangle. The following code shows how you can plot rectangles:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

df = pd.read_csv('readings.csv', names=['time', 'value'])

plt.figure(figsize=[16,10])
fig, ax = plt.subplots()
```

```
ax.add_patch(Rectangle((50, 120), 50, 10, color="#1f77b4"))
ax.add_patch(Rectangle((100, 120), 50, 10, color="#ff7f0e"))

plt.xlabel('Time/[epoch]')
plt.ylabel('raw adc value/[]')
plt.scatter(df['time'], df['value'],label='measurements')
plt.legend(loc=2)

plt.savefig('plot.pdf')
```

9. ***Second Evaluation*** Open the resulting PDF and inspect the result of your masterpiece!
   - Can you explain what you see?
   - How does this compare to the plot from step 5 in terms of gleanable insights?
10. ***Reflection*** How does this approach scale with the number of sensor readings?

Quest 5: Improving Precision (5XP)

As you have probably noticed by now, the readings do not appear to be particularly stable. You must observe how the signals from three physical variations of the setup differ. What you find is the results of the works of Electrical Engineers. Do not underestimate them. They are capable of doing both good and great evil.

1. The temperature sensor setup connected to a laptop connected to wall-power.
2. The temperature sensor setup connected to a laptop running off its battery.
3. The temperature sensor setup connected to a laptop connected to wall-power, but with a $0.1\mu F$ capacitor between ground and the sensor output.