EE 14 Lab 7: Analog-to-Digital Converter

Lab report due the week following break (25-28 March 2025)

1 Introduction

The GPIO ports on our microcontroller are excellent for reading digital signals (0 or 3.3V), but lots of useful sensors and other devices work by producing a varying *analog* voltage rather than a digital low or high. We can read these analog voltages using the built-in analog to digital converter (ADC), which converts an analog voltage into a binary number the processor can use.

In this lab, you'll write code to read the position of a thumbstick by sensing the voltages produced by potentiometers on the X and Y axes.

In the second part of the lab, you'll measure the performance of the ADC code and estimate how quickly it can sample. The datasheet says that the ADC is capable of 5 megasamples/second (i.e., 5 million measurements per second) — we'll see how close we are!

After successfully completing this lab, you should be able to:

- Wire a potentiometer to generate an analog voltage
- Write code to read an analog voltage using an ADC
- Explain the limits of this method of using the ADC

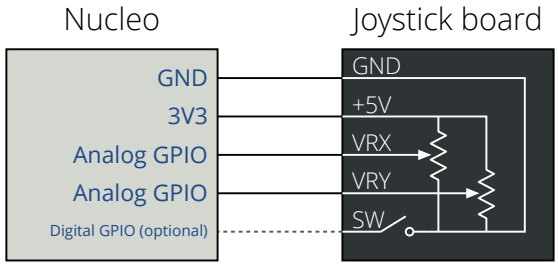
2 In lab

Joystick

L1: Wire up your joystick as shown below. Note that the pin labeled "+5V" should be connected to 3.3 V, **not** 5 V.

Pick your favorite two analog-capable GPIO pins (labeled ADC1_X on the pin diagram). Note that A7 (ADC1_7) is also the UART TX pin for serial communication with your computer, so don't use that one!

The joystick SW pin is just a switch to ground. If you want to, you can wire it up to a GPIO pin and read it just like the other pushbutton switches you've used.



L2: Create an empty PlatformIO project with the same settings as usual:

- **Board**: ST Nucleo L432KC
- **Framework**: CMSIS

L3: Download the framework starter files from the course website and add them to your project. Read the descriptions of the functions in `adc.c` to understand how to use them.

L4: Write code to perform an ADC conversion on one of the inputs, and print the result on the serial terminal. Move the joystick and confirm that it works — you should be able to read numbers across the full range from 0 to 4095.

L5: Write code to read the other joystick axis and make it do something fun. Some ideas:

- Use the axes to pick a color to show on your RGB LED. (For example, use the x-axis to control the brightness of the red channel, and the y-axis for blue.)
- Use some of the LEDs in the 7-segment display to show which quadrant/section the joystick is in.
- Set up a row of LEDs, and use the joystick to select which one is lit.
- Make a little game where something indicates a target position for the joystick, and the user tries to get to that spot as quickly as possible (think "snake" or "whack-a-mole").

Pushing the limits

L6: Design an experiment to figure out how long it takes to run each of the functions `adc_configure_single()` and `adc_read_single()`. There are several different ways to do this; talk with your labmates and TAs if you need ideas.

How many times per second can you sample a single channel (i.e., just calling `adc_read_single()` repeatedly)?

How many times per second can you sample two channels (as with your joystick), given that you have to reconfigure the ADC to switch channels before each read?

L7: Due to how the SAR ADC works, higher-precision ADC conversions take longer. Modify the `adc_read_single` function to do a 6-bit conversion and then measure how long it takes.

What is your maximum sample rate for a single channel now?

Optional: configure multiple reads

The previous section illustrates why the ADC can be configured to take up to 16 measurements in sequence without intervention. A couple of small changes will let you do this.

L8: Make a copy of the `adc_configure_single()` function (maybe it could be `adc_configure_multiple()`?) which accepts an array of pins, instead of a single pin. You'll need to modify how the ADC "sequence registers" are configured (SQR1 – SQR4), but that should be it.

L9: Make a copy of the `adc_read_single()` function that accepts a pointer to an array where it will store the measurements. The ADC will initiate the next reading as soon as the data register (DR) is read, so you can simply copy the values from DR into the array one at a time until the sequence is complete.

Once you've done this, modify your joystick code to capture both axes with a single call to your new function. How much faster can you sample now?

What to turn in

Your lab report should contain the following:

- A cover page with your name, date, lab section, and lab TA names, and a 1-2 sentence description of the lab.
- A description of your timing experiment:
  - Explain what you are trying to measure
  - What measurements you took, and how they relate to the thing you want to know
  - Any compensation you made to your measurements. For example setting GPIO pins and reading timers both take time, so how are you compensating for this?
  - A table of your measurements
  - Your calculations and final answers to lab questions 6 and 7.

  This should probably be between 1-2 pages. We don't need a detailed history of everything you tried, just an explanation of how you got to your final answer. There should be enough detail that someone else could follow your logic, trust your answers, and reproduce your experiment if they wanted to verify.
- Answers to the following questions:
  - What do you understand now, that you didn't before the lab?
  - What are you confused or curious about? (Yes, ADCs can be confusing... but get more specific than that!)
  - How long did it take you to complete this lab? (This will help us calibrate the workload for future semesters.)
- Your completed code, as text in a monospace font at the end of the report (not a screenshot!). You should be able to copy and paste straight out of VSCode and keep the syntax highlighting. The LaTeX listings package is also a great way to typeset code.