ECE271: Microcomputer Architecture and Applications — University of Maine

Lab #10: Analog to Digital Converter (ADC)

Week of 11 April 2022

Goals

1. Understand basic ADC concepts (successive approximation, sampling error, resolution, and data alignment).
2. Map a GPIO pin to an ADC input.
3. Understand the tradeoff between conversion accuracy and conversion speed.

Pre-lab

1. Complete the pre-lab before attending lab. The pre-lab is in a separate pdf file, found on the website.

Lab Procedure

The first part of this lab is using the Analog/Digital Converter (ADC) to measure the voltage generated by a potentiometer operating as a voltage divider. The second part of the lab is using the ADC to measure the output of an Infrared (IR) photodiode.

Part A – Initial Setup

1. You can use a previous lab as a template, but in general we aren't really re-using any code from previous labs.

2. If you're using Linux, I've posted an updated template to the website that has some more definitions in the header file that will be helpful with this lab.

Part B – Set up the ADC

1. We will be following the flowchart in Figure 20-12 in the textbook.

2. Make sure the HSI clock is turned on so the ADC can use it.

   (a) You can follow the directions in the flowchart, or else just reuse the code from the last lab and have the whole board use the HSI clock.

3. Set up Pin A1 for analog input.

   (a) You should have calculated these values in the prelab.

   (b) Also be sure you enable the `RCC_AHB2ENR_GPIOAEN` clock.

4. Set up the ADC. This is a really long and complicated process. Use the flowchart, and also the manual (around roughly page 600) should help.

   (a) Enable the ADC clock bit `RCC_AHB2ENR_ADCEN` in register `RCC->AHB2ENR`.

   (b) Disable ADC1 (as we can only modify the settings if it is disabled) by clearing `ADC_CR_ADEN` in `ADC1->CR`.

   (c) Enable the I/O analog switches voltage booster, `SYSCFG_CFGR1_BOSTEN` in `SYSCFG->CFGR1`.

   (d) Enable conversion of internal channels by setting `ADC_CCR_VREFEN` in `ADC123_COMMON->CCR`.

   (e) Configure the ADC prescaler to have the clock not divided (which involves setting all bits to zero) `ADC_CCR_PRESC` in `ADC123_COMMON->CCR`.

   (f) Select synchronous clock mode (HCLK/1) (0b01) via `ADC_CCR_CKMODE` in `ADC123_COMMON->CCR`.

   (g) Configure all ADCs as independent by clearing `ADC_CCR_DUAL` in `ADC123_COMMON->CCR`.

   (h) The ADC comes up in deep sleep mode. To wake it up you'll want to call a `ADC1_Wakeup()` function. The code for this is provided in Example 20-4 in the textbook.

   (i) Configure the ADC to have 12-bit resolution (0b00) in `ADC_CFGR_RES` in `ADC1->CFGR`.

   (j) Set right-alignment of the 12-bit result inside the results register by clearing `ADC_CFGR_ALIGN` in `ADC1->CFGR`.

   (k) Select 1 conversion in the regular channel conversion sequence with by clearing. `ADC_SQR1_L` in `ADC1->SQR1`. (Example 20-6 in the textbook has code for this and the next two steps).

   (l) Specify channel #6 as the 1st conversion by setting "6" in the proper place in the `ADC1->SQR1` register.

   (m) Set channel #6 to be single ended by clearing `ADC_DIFSEL_DIFSEL_6` in `ADC1->DIFSL`.

   (n) Select the ADC sampling time. Set this to 1. `ADC_SMPR1_SMP6` in `ADC1->SMPR1`.

   (o) Select ADC to discontinuous mode by clearing `ADC_CFGR_CONT` in `ADC1->CFGR`.

   (p) Select software trigger by clearing `ADC_CFGR_EXTEN` in `ADC1->CFGR`.

   (q) Enable ADC1 by setting `ADC_CR_ADEN` in `ADC1->CR`.

   (r) Wait until ADC1 is ready by waiting for the `ADC_ISR_ADRDY` bit in `ADC1->ISR` to go high.

Part C – Set up the Potentiometer

1. Get a 10k potentiometer from the parts closet.

2. Connect it to the pin PA1 following the diagram in Figure 1.

3. The potentiometer acts as a voltage divider, the output to PA1 should vary from ground to 3V as you turn the dial.

4. If you still have the keypad hooked up, you probably want to disconnect it from pin PA1 so it won't interfere with the measurements.

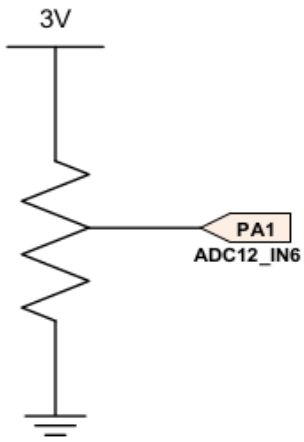

Figure 1: Potentiometer based voltage divider.

Part D – Measuring the Value

1. In `main()` have a while loop that does a manual software trigger of the ADC. See the last part of the textbook Figure 20-12 flowchart.
2. Start a conversion by setting `ADC_CR_ADSTART` bit in `ADC1->CR` register.
3. Wait until the `ADC_CSR_EOC_MST` bit goes high in `ADC123_COMMON->CSR`, which means a successful conversion has completed.
4. Read the value `ADC1->DR` for the converted value. Store this value to a global variable. To view it, set up a debugger watch on the variable as you did in the previous lab.
5. If all went well, turning the potentiometer should change the value from 0 to roughly 4095 (0 to `0xfff` in hex).

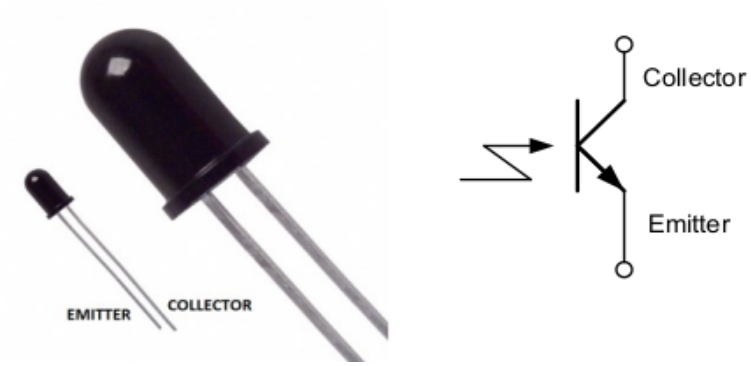

Figure 2: QSD124 Infrared Phototransitor (receiver).

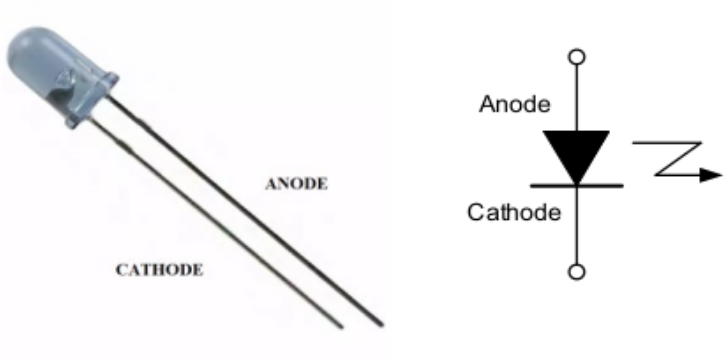

Figure 3: IR333-A Infrared LED (transmitter).

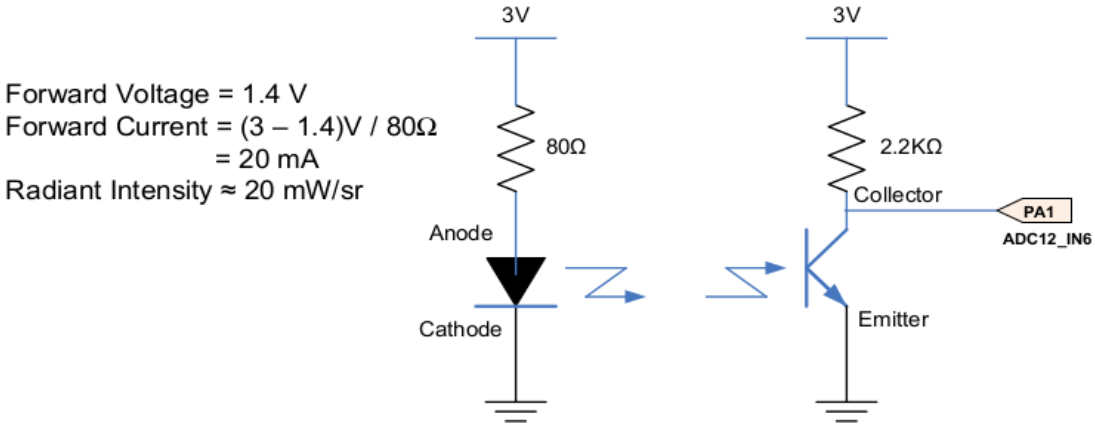

Figure 4: Connection diagram.

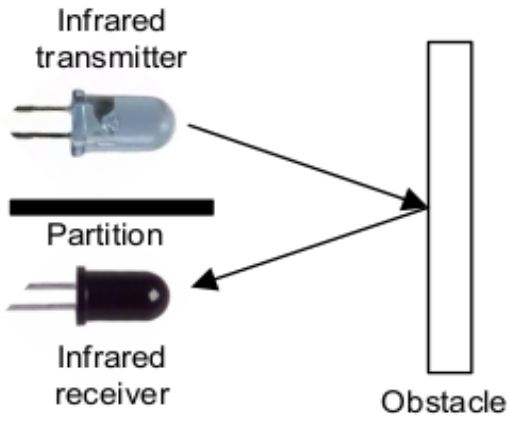

Figure 5: Proximity sensor.

Part E – Set up the IR Proximity Sensor

1. We will implement a simple proximity sensor using an infrared LED and receiver. The infrared light will leave the transmitter, and reflect off a nearby object and cause current to flow in the receiver. This will lead to a voltage proportional to the intensity of the light.

   You might want to put some sort of partition between the sender and receiver so stray light doesn't go between them.

   You can often verify if your IR LED is glowing by looking at it through your cellphone camera. Modern CCD cameras can see infrared, but a filter is usually used to block it. Depending on your cell phone manufacturer and the quality of the filter they provided you can possibly see if the IR LED is on.

   We will provide the transmitter/receiver, you can get the 82 Ohm resistor (the diagram shows 80 Ohm but 82 is close enough) and 2.2k resistor from the parts cabinet.

   Some more info on this can be found in Figures 2 through 5.

2. Hook PA1 to the proper point on the phototransistor and watch the ADC value in the debugger like you did with the potentiometer.

Part F – Something Cool

Do something cool! You can come up with something on your own, but here is a list of ideas you can use.

1. Have the red LED turn on if the ADC crosses a threshold (the potentiometer is turned half way or similar)
2. Measure distance with the IR LED.
3. Show the ADC results on the LCD
4. Use the potentiometer to control the stepper motor or servo motor.
5. Count the times your hand goes near the IR LED and display the count on the LCD.
6. Use a timer to periodically trigger the ADC.
7. Use the analog watchdog in the ADC to trigger an event (an LED turns on?) when something gets too close to the sensor.

Lab Demo

1. Submit your code
    - Complete a README with the post-lab answers.
    - Make sure the code is properly commented.
      This includes a header at the top of your main.c with your name and a brief summary of the lab.
    - Check your code and README into your gitlab tree.
2. Demo your implementation to your lab TA.
    (a) Demonstrate that the value returned by the ADC varies when you use the potentiometer.
    (b) Demonstrate that the value returned by the ADC varies when using the Infrared sensor.

Post-Lab

- Place your answers to the question in a file Readme.md
- Submit with your code via the gitlab server.
- Questions:
    1. Potentiometer
        (a) Set the potentiometer to near the middle and measure the voltage on the PA1 pin with a voltmeter:
        _____
        (b) In the debug environment, find the value of the ADC DR register:
        _____
        (c) Calculate what the reference voltage must be in order to fit the previous two values you measured:
        _____