# ECE 153B – Winter 2024
# Sensor and Peripheral Interface Design

## Lab 5 – ADC and DAC

**Deadline:** Mar 1, 2024, 10:00 PM

## Objectives

1. Understand basic ADC and DAC concepts.
2. Map a GPIO pin to be used as an ADC input.
3. Map a GPIO pin to be used as a DAC output.
4. Learn how to utilize ADC output.
5. Learn how to adjust analog output with DAC.

## Grading

| Part | Weight |
|---|---|
| Part A | 50 % |
| Part B | 50 % |

Submit your code and answers to the questions to the submission link on Gradescope by the specified deadline. In the week following the submission on Gradescope, you will demo/checkoff your lab to the TA.

*Please note that we take the Honor Code very seriously, do not copy the code from others.*

## Necessary Supplies

- STM32L4 Nucleo Board
- Type A Male to Mini B USB Cable
- Breadboard & Jumper Wires
- 1 kΩ Potentiometer

# 1 Lab Overview

A. You will use the ADC to measure input voltage that is adjusted by a potentiometer. Then, using your measurement, you will change the behavior of the LED.

B. You will use the ADC to measure input voltage that is adjusted by the DAC. Then, using your measurement, you will change the behavior of the LED. The DAC output will be adjusted using the button interrupt.

# 2 Part A – ADC

For this portion of the implementation, you will be writing code in `ADC.c`.

## 2.1 ADC Setup

We will be using **PA1** for analog input. In the `ADC_Pin_Init()` function, set up **PA1** to have the following configuration. In addition, connect **PA1** to the ADC input (see `GPIO_ASCR`).

| Pin | Mode | Pull-Up, Pull-Down |
|---|---|---|
| **PA1** | Analog | No Pull-Up, No Pull-Down |

Next, we will set up the common configuration for ADC. In the `ADC_Common_Configuration()` function, write code to set the ADC common configuration.

1. In the configuration register of the system configuration controller (SYSCFG − >CFGR1), enable the I/O analog switch voltage booster.
2. Next, modify the ADC common control register. (ADC123_Common-− >CCR)
   (a) Enable $V_{\text{REFINT}}$.
   (b) Ensure that the clock is not divided.
   (c) Ensure that the ADC clock scheme is set to HCLK/1 synchronous clock mode.
   (d) Ensure that all ADCs are operating in independent mode.

By default, ADC is in deep sleep mode to ensure that its supply is internally switched off to reduce leakage currents. Thus, it is necessary to "wake up" the ADC. We have provided this code for you.
Now, we will set up the specific ADC channel that we are going to use. The board has three available ADCs – in this lab, we will be using `ADC1`. Table 1 shows that **PA1** is connected to channel 6 of `ADC1`.

1. Enable the ADC clock.
2. Using one of the RCC reset registers, reset the ADC. Then, clear the reset bit.
3. Before modifying any of the ADC settings, ensure that it is disabled (just like how we disabled UART before modifying its settings in Lab 4).
4. In the ADC configuration register, configure the ADC to have 12-bit resolution and right-alignment.
5. In the ADC regular sequence register, set the sequence length to 1. That is, there will be only 1 conversion in the conversion sequence. Then, ensure that channel 6 is used for the first conversion.

| Analog Input Channel | Pin | Analog Input Channel | Pin |
|---|---|---|---|
| ADC123_IN 1 | **PC 0** | ADC12_IN 13 | **PC 4** |
| ADC123_IN 2 | **PC 1** | ADC12_IN 14 | **PC 5** |
| ADC123_IN 3 | **PC 2** | ADC12_IN 15 | **PB 0** |
| ADC123_IN 4 | **PC 3** | ADC12_IN 16 | **PB 1** |
| ADC12_IN 5 | **PA 0** | ADC3_IN 6 | **PF 3** |
| ADC12_IN 6 | **PA 1** | ADC3_IN 7 | **PF 4** |
| ADC12_IN 7 | **PA 2** | ADC3_IN 8 | **PF 5** |
| ADC12_IN 8 | **PA 3** | ADC3_IN 9 | **PF 6** |
| ADC12_IN 9 | **PA 4** | ADC3_IN 10 | **PF 7** |
| ADC12_IN 10 | **PA 5** | ADC3_IN 11 | **PF 8** |
| ADC12_IN 11 | **PA 6** | ADC3_IN 12 | **PF 9** |
| ADC12_IN 12 | **PA 7** | ADC3_IN 13 | **PF 10** |

Table 1: Pin Definition for `ADC` Input Signals on STM32L4 Processor

6. In the ADC differential mode selection register, ensure that channel 6 is set to single-ended mode.
7. In the ADC sample time register, set the sampling time for channel 6 to be 24.5 ADC clock cycles.
8. In the ADC configuration register, ensure that the ADC is in single conversion mode and that hardware trigger detection is disabled.
9. Now that we are done with the configuration, enable the ADC. Before moving on, wait until the ADC is ready (this information is available in the interrupt and status register).

## 2.2 Measuring Potentiometer Output

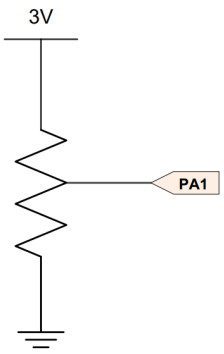


Figure 1

Wire the potentiometer to output a voltage between 0 V and 3.3 V (the board has a `3.3V` pin). The output of the potentiometer should be connected to **PA1** since we have configured it as analog input to the ADC.

Recall that we set up the ADC to operate in single conversion mode. This means that when an ADC is triggered, it will only measure once. Furthermore, we disabled hardware triggering, which means that we are able to trigger the ADC using code. The following steps outline how to trigger an ADC to get a single measurement.

1. Through the ADC control register, start a regular conversion.

2. Wait until the ADC conversion (i.e. regular conversion of the master ADC) is complete. This information can be found in the ADC common status register.

3. Read the ADC data register to retrieve the measurement. The ADC should return a value between 0 and roughly 4096.

Now, using 4096 as the maximum possible value that the ADC returns, compute the voltage that was measured. Note that

$$\frac{\text{measurement}}{4096} = \frac{V_{\text{in}}}{V_{\text{ref}}}$$

### 2.3 Monitoring the ADC

To monitor the waveform of the ADC output, we will use the Logic Analyzer in the debugger. But, first we need set up the MSI clock and enable trace in the debug settings.

1. Modify the SysClock.c to be a 8MHz MSI clock. If there is a buffer issue during trace collection, tune down the MSI clock and adjust the PWM prescaler to keep LED running smoothly.

2. Go to "Options for Target 1 'target 1'... " > Debug Tab > Settings next to "ST-Link Debugger" > Trace Tab.

3. Check "Trace Enable" and "Use Core Clock". Change Core Clock to the frequency used by the MSI clock.

4. Check "Enable" in Timestamps and set Timestamps Prescaler to 64.

Now, we will setup the Logic Analyzer within the debug mode. (See Appendix for more details)

1. Select "View" > "Analysis Windows" > "Logic Analyzer".

2. In Logic Analyzer window, click setup to add the ADC result variable we want to monitor.

3. Click the "New (Insert)" icon next to the red cross. Enter the variable you want to monitor and set the max value of that variable. Close the setup window.

4. Run your program and observe the resulting ADC reading in the Logic Analyzer window.

### 2.4 Applying ADC Measurement

Now that we have a method of measuring the input voltage, we can implement different functionalities that depend on the measured voltage. In this lab, we will change the LED brightness according to the measured voltage. When the measured voltage is 0 V, the LED should be off and when the measured voltage is 3.3 V, the LED should be at its brightest. Please write your own code in `main.c`, `PWM.c`, and `PWM.h` to accomplish these two functionalities.

## 3 Part B – DAC

For this portion of the implementation, you will be connecting the output of the DAC to the ADC input. Remove the potentiometer, and connect a jumper wire from **PA1** to **PA4**. You will be writing code in `DAC.c`. You may reuse the ADC, PWM, and main code behavior from Part A. The DAC will output an analog voltage between 0V and 3.3V depending on the digital conversion value supplied.

### 3.1 DAC Setup

We will be using **PA4** for analog output. In the `DAC_Pin_Init()` function, set up **PA4** to have the following configuration.

| Pin | Mode | Pull-Up, Pull-Down |
|---|---|---|
| **PA4** | Analog | No Pull-Up, No Pull-Down |

Next, we will set up the specific DAC channel that we are going to use. The board has one DAC – `DAC1`. Table 2 shows that **PA4** is connected to channel 1 of `DAC1`.

| Analog Output Channel | Pin |
|---|---|
| 1 | **PA4** |
| 2 | **PA5** |

Table 2: Pin Definition for `DAC1` Output Signal on STM32L4 Processor

Implement the following in `DAC_Init()`:

1. Enable the DAC clock.
2. Before modifying any of the DAC settings, ensure that it is disabled
3. In the DAC control register, configure the DAC to be triggered by software.
4. In the DAC control register, disable the trigger.
5. In the DAC mode control register, set the mode to normal mode to external pin only with buffer enabled.
6. In the DAC control register, enable DAC channel 1.

Once the DAC is set up, implement `DAC_Write_Value()` by writing to the right-aligned 12-bit conversion register for channel 1.

### 3.2 Interrupt

Next, we will implement a button interrupt which will control the DAC output. You will be writing code in `EXTI.c`. The button press will increase the DAC output until the maximum value is reached. Once the maximum value is reached, pushing the button will decrease the DAC output until the minimum value is reached. Once the minimum value is reached, the DAC output will increase again, etc.

1. Setup an interrupt for the button in `EXTI_Init()` as was done in Lab 2.
2. Write the interrupt handler for the button.
   - When the button press is increasing the DAC value, increase the DAC output by `DAC_INCREMENT`. Be careful to ensure the value does not exceed the maximum 12-bit value `DAC_MAX` or the DAC value will overflow.
   - When the button press is decreasing the DAC value, decrease the DAC output by `DAC_INCREMENT`. Be careful to ensure the value does not drop below the minimum value `DAC_MIN` or the DAC value will underflow.

# Appendix

### Getting Started Guide: Logic Analyzer

From the menu bar, selecting **View** → **Analysis Windows** → **Logic Analyzer** opens up a separate window for the Logic Analyzer. The Logic Analyzer can display a trace of static/global variables over the time that the program is run for. Note that local variables cannot be displayed and that register values cannot be analyzed.

Let's say that we want to use the Logic Analyzer to monitor the variable `output` that is defined in the data area. Note that the `EXPORT output` instruction makes the variable `output` a global variable, which means that it can be analyzed using Logic Analyzer. First, we need to set up the Logic Analyzer – to do this, click **Setup**. Add the signal `(signed int) output` that should be observed and ensure that the data display range is adjusted such that the entire curve can be seen (see Figure 2). Figure 3 shows the signal (in this example, a sine wave) that the Logic Analyzer displays.

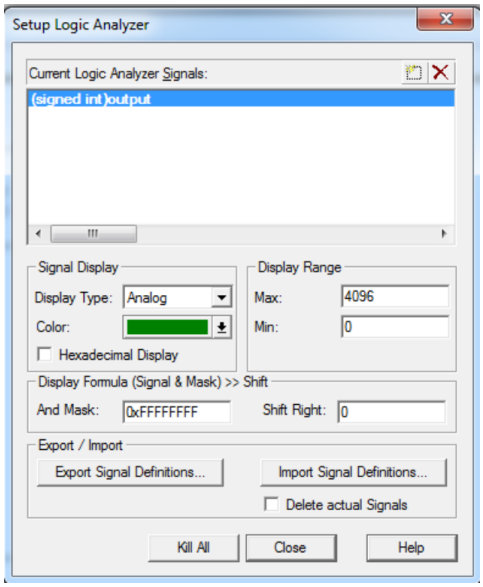


Figure 2: Logic Analyzer Setup

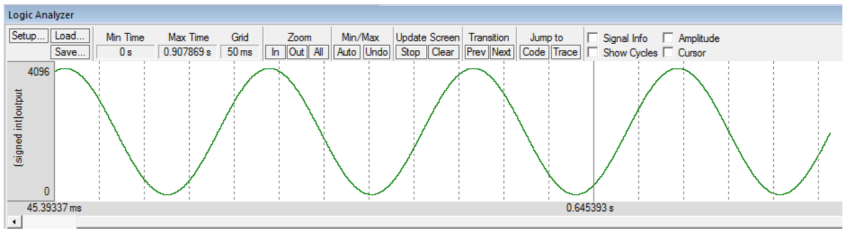


Figure 3: Logic Analyzer Display

## 4 References


[1] STM32L4x6 Advanced ARM-based 32-bit MCUs Reference Manual

[2] Yifeng Zhu, "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C", ISBN: 0982692633