# Experiment 6: DAC and ADC

## 1.0 Introduction

*Interfacing a microcontroller with real-world devices often means working with analog voltages that are not constrained by digital logic's high/low dichotomy. In this lab, you will gain experience using the digital-to-analog converter and the analog-to-digital converter built in to your microcontroller. You will also practice writing code to manipulate the converted values as well as display them on an output device.*

| Step | | Points |
|---|---|---|
| 4.0 | Prelab | 10 |
| 6.1 | Wiring | 5 |
| 6.2 | Sine wave synthesis | 10 |
| 6.3 | Sine wave mixing | 10 |
| 6.4 | Modulating harmonic amplitudes | 10 |
| 6.5 | Serial communication and frequency counting | 10 |
| 7.0 | Post lab submission | 5* |
| **Total** | | **60*** |

***All lab points are contingent on submission of code completed in lab.**

## 2.0 Objectives

1. To understand the concept of analog-to-digital conversion.
2. To understand the concept of digital-to-analog conversion.
3. To learn how to use STM32F0 ADC peripherals to measure and process analog.

## 3.0 Equipment and Software

1. Standard ECE362 Software Toolchain (Installation instructions available in Experiment 0: Getting Started)
2. STM32F0-DISCOVERY development board (For procurement instructions, see Experiment 0: Getting Started)

## 4.0 Prelab

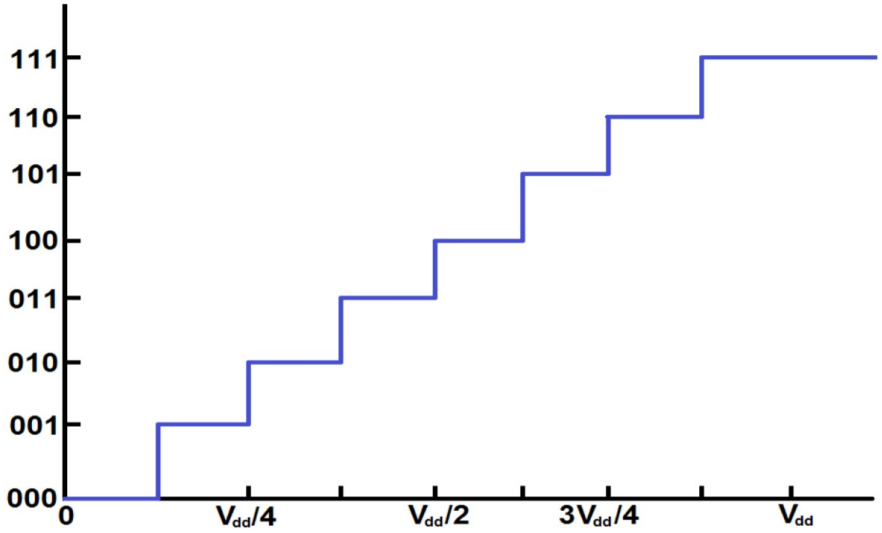
Read over section 5 "Background," and complete the prelab on the course website.

# 5.0 Background

## 5.1 Analog-to-Digital Conversion

Audio signals, analog sensor input waveforms, or input waveforms from certain types of input devices (sliders, rotary encoders, etc.) are all examples of analog signals which a microcontroller may need to process. In order to operate on these signals, a mechanism is needed to convert these analog signals into the digital domain; this process is known as analog-to-digital conversion.

In analog-to-digital conversion, an analog signal is read by a circuit known as an analog-to-digital converter, or ADC. The ADC takes an analog signal as input and outputs a quantized digital signal which is directly proportional to the analog input. In an n-bit analog-to-digital converter, the voltage domain of the digital logic family is divided into 2n equal levels. A simple 3-bit (8-level) analog-to-digital quantization scheme is shown in figure 1, below:




## 5.2 Configuring the STM32F0 ADC

The STM32F0 microcontroller has an integrated 12-bit analog-to-digital converter. The converter has 16 channels, allowing for readings from up to 16 different independent sources. The ADC is capable of converting signals in the 0-3.6V range, and is able to take samples every 1µs. Many other ADC features are described in greater detail in the microcontroller documentation.

Before any ADC configurations can be made, a few initializations must be done. First, the ADC clock must be enabled using reset clock control (RCC), as was done with other peripherals in previous experiments. Additionally, those inputs which are to be configured as analog I/O must be configured for analog mode in the corresponding GPIOx_MODER register. This is described in further detail in section 9.3.2, “I/O pin alternate function multiplexer and mapping”, of the STM32F0 family reference manual.

Prior to performing any configuration of the ADC, the peripheral must first be turned on. ADC activation must take place before any other ADC configurations are performed; failure to follow this order may result in the ADC entering an unknown state, at which point the ADC must be disabled and restarted. Enabling the ADC requires writing ‘1’ to the ADC enable bit, ADEN, in the ADC control register, ADC_CR. Once this is done, software must wait until the ADC is in a ready state, indicated by the bit ADRDY (located in the ADC status register, ADC_ISR) being read as ‘1’ to software.

For the purposes of this experiment, we are interested in configuring the microcontroller ADC for single conversion, software-driven mode. This means that the ADC will perform a single conversion, initiated in software. The ADC will then await the next software trigger before another conversion is initiated. This is the default operating mode of the ADC, and no special configurations will be needed within the ADC peripheral. In the event that special operating conditions (DMA, continuous conversion mode, etc.) need to be performed by the ADC, operating modes can be specified via the ADC configuration registers: ADC_CFGR1 and ADC_CFGR2.

When performing a conversion, the ADC will read and convert values on any active channels. By default, all ADC channels are disabled. Prior to use, it is important to enable those channels on which conversions will be performed. This is done through the ADC channel selection register, ADC_CHSELR.

### 5.3 Using the STM32F0 ADC

Once the ADC has been properly enabled and configured, initiating an analog-to-digital conversion is as simple as writing a '1' to the ADSTART bit in the ADC_CR register. Writing a '1' to this bit instructs the converter to automatically begin conversion of all ADC channels. This bit will automatically be cleared by hardware once the conversion has completed. Converted ADC data is ready to be processed by the core once the end of conversion bit, EOC, is activated in the ADC_ISR register.

In some situations, particularly when disabling the ADC, it may be desirable to stop a conversion which is already in process. This can be done by writing a '1' to bit ADSTP of the ADC_CR register. Doing so will stop the ongoing conversion process, discarding any data, and will automatically clear the ADSTART bit.

Once an analog-to-digital conversion has been completed, the converted data can be read from the ADC data register, ADC_DR. Data will be right-aligned or left-aligned, depending on configuration settings (right-aligned by default).

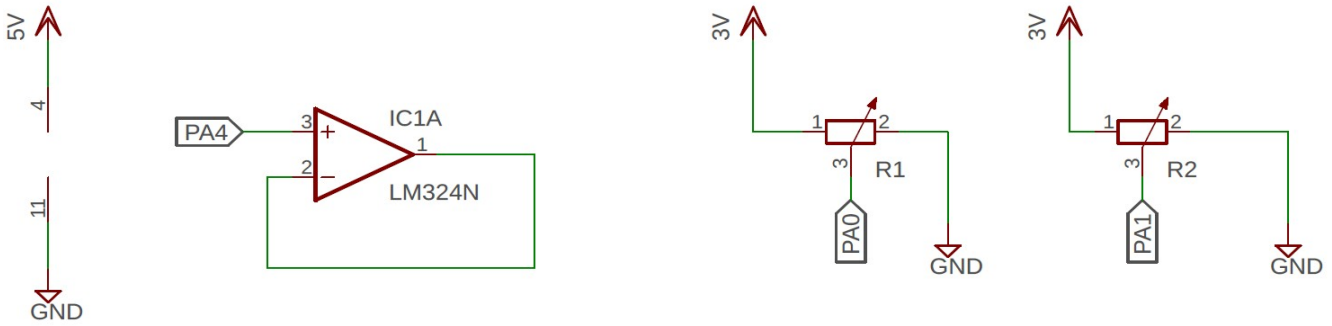
## 6.0 Experiment

For this lab, you will need to complete functions defined in the main.c file provided to you. Each section will describe what functions need to be completed.

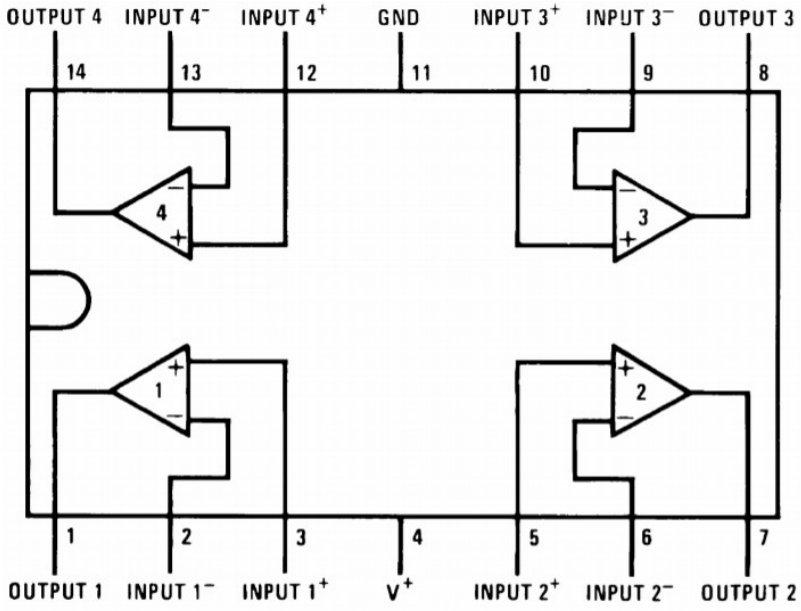
### 6.1 Wiring

In the next section we will use the DAC to output a tone to the speaker. To electrically isolate the speaker from the DAC output we will channel the DAC output though a voltage follower using the LM324 quad op-amp provided in your 362 development kit. Note: the op-amp does not have sufficient power to fully drive the speaker without distortion, but it will be enough to be audible. (Those of you doing audio output for a mini-project should consider using a better amplifier.) **Certain sections of this lab will ask you to connect the speaker to the output of the LM324 IC. To do that, you must connect one terminal of the speaker to ground and the other to pin 1 of LM324.**

Later sections of this lab will use the two 10K potentiometers (from your ECE master lab kit) as inputs to the built in ADC to control the amplitudes of a generated sine wave. Wire your circuit as shown in the diagram.




Note the 4 and 11 correspond to the pins of LM324, for reference the pinout of LM324 is shown below. Make sure to connect the LM324 to 5V and the potentiometers to 3V.




Finally, you will use the serial port for reporting information. Configure that exactly as you did for homework 4. In your **`main()`** function, call **`serial_init()`** to configure the serial port.

### 6.2 Sine wave synthesis

The goal of this section is to output a sine wave from the onboard DAC. For this purpose, you have been provided with a file "wavetable.h" which contains the values you will need to use for sine wave generation, make sure to include this in your project.

To complete this section you will need to complete the following functions:

a) **`setup_gpio()`** : This function enables clock to port A, sets PA0, PA1, PA2 and PA4 to analog mode.

b) **`setup_dac()`**: This function should enable the clock to the onboard DAC, enable trigger, setup software trigger and finally enable the DAC.

c) **`setup_timer2()`** : This function should, enable clock to timer2, setup the prescaler and auto-reload register so that the interrupt is triggered every 100μs, enable the timer 2 interrupt, and start the timer.

d) **`TIM2_IRQHandler()`**: The interrupt handler should start the DAC conversion using the software trigger, and should use the wavetable.h to read from the array and write it into the DAC. Every time the interrupt is called, you will read a new element from the "wavetable" array. So you might need to use a global variable as an index to the array. Note that the array has 100 elements – make sure you do not read wavetable[100].

Refer to the lecture notes on how to setup DAC and trigger it using a software trigger. Once you have completed this section, connect the speaker to the output of the LM324. You should an audible tone.

Demonstrate this to your TA.

#### 6.2.1 Disconnect the speaker from the output of the op-amp, and connect the LM324/op-amp output to the oscilloscope. What is the frequency of the sine wave?

#### 6.2.2 Is waveform a true sine wave?

#### 6.2.3 What filter would you use to improve the shape of the wave?

**6.3 Sine wave mixing**

This section aims to demonstrate tone mixing. Tone mixing will add two different frequencies of sine waves to get a resultant wave composed of the two harmonics.

To keep this lab experiment as simple as possible, we will use a single wave table. To generate two different frequencies from a single wave table, we will sample it at different rates. This is performed by sampling (reading the array) at twice the rate (jump 2 indices instead of 1 index) and adding it to the base sampling.

When adding two waves of the same amplitude and different frequencies, sometimes they will constructively interfere to double the amplitude, and other times they will destructively interfere to have a zero amplitude (a null). To keep the amplitude within the bounds for our DAC, we will need to divide the amplitude of each component by two (implement this using a right shift)

As an example:

```
DAC_out = (wavetable[s1] + wavetable[s2]) >> 1
s1++;
s2 += 2;
```

Note the above code is guide on how to perform the logic described above, you will need to adapt this into syntactic C to get it to work in your code. You should modify the code in the timer2 interrupt handler to make this change. Hook up the output of the LM324 to the scope, and observe the output.

Use the FFT functionality of the scope to see the amplitudes of the two sine components.

Demonstrate this to your TA.

### 6.4 Modulating harmonic amplitudes

For this section, we will use the two potentiometers as volume controls which control the amplitude of the two harmonics we generated in the previous section.

For this you will need to complete the following functions:

1) **`setup_adc()`**: This function should enable the clock to ADC, turn on the clocks, wait for ADC to be ready.

2) **`read_adc_channel(unsigned int channel)`**: This function should return the value from the ADC conversion of the channel specified by the "channel" variable. Make sure to set the right bit in channel selection register and do not forget to start ADC conversion.

3) **`main()`**: In your main function, within the infinite loop, call **`read_adc_channel()`**, on channels corresponding to PA0 and PA1, and store the result in the global variables "a1" and "a2". Make sure to divide the value returned from "read_adc_channel" by 4095.0 to normalize the "a1" and "a2" between 0 and 1.

4) **`TIM2_IRQHandler()`**: Update your timer2 interrupt handler to use the values of a1 and a2 when performing the wave synthesis.

Your code should look something like:

```
(a1 * wavetable[s1] + a2 * wavetable[s2]) >> 1
```

Connect the output of the LM324 op-amp to the scope, vary the potentiometers and observe sine wave mixing.

Demonstrate this to your TA.

**6.4.1 Disconnect the LM324 output from the scope and connect the output to the speaker. Is the tone the same as a pure sine wave?**

**6.4.2 Use the FFT on the scope to determine the two frequency components in the generated waveform. Type your answer in Hz in the post lab writeup.**

### 6.5 Serial communication and frequency counting

This section will sample the generated sine wave, calculate its frequency and output the frequency and the harmonic's amplitudes (a1 and a2) into the serial terminal. For this section you will need to modify your main function, and implement setup_timer3() and TIM3_IRQHandler().

Connect the LM324 output to PA2 (channel "x" of the ADC. You will need to figure out x).

a) **`setup_timer3()`**: This function should, enable clock to timer3, setup prescaler and auto-reload register so that the interrupt is triggered 200μs, enable the timer 3 interrupt and start the timer.

b) **`TIM3_IRQHandler()`**: The interrupt handler should read the value from the ADC's channel 2 input. Use the **`insert_circ_buffer()`** function to insert the read value into the circular buffer.

c) **`main()`**: In the infinite loop in you main function add the following logic:

- Call **`serial_init()`**.
- Loop through the circular buffer, to calculate min and max.
- Call **`get_time_period(min, max)`** to estimate the period of the sampled signal.
- Calculate the frequency.
- Use printf to display, "a1", "a2" and "frequency" in the serial terminal.

Demonstrate this to your TA.

## 7.0 Complete post lab and submit code

Submit your code and complete the post lab on the website. The portal for post-lab will close 10 minutes after your scheduled lab.