# Lab 2: Use of Digital to Analog Converter along with a Microcontroller

---

## 1 Learning Objective

In this lab, you are going to

1. Learn about generating basic waveforms and measuring them using an oscilloscope.
2. Explore generating low frequencies for various sounds and music.
3. Use the DAC chip to create different waveforms.

## 2 Components Needed

- Raspberry Pi 3, resistors, buzzer, and DAC chip
- We will use a DAC chip MCP4725 for the lab activities below. You are allowed to use the Adafruit open source library, whose details can be found here: https://learn.adafruit.com/mcp4725-12-bit-dac-with-raspberry-pi .
- You need to enable Raspberry Pi I2C functions via command raspi-config

## 3 Lab Activities

### 3.1 Example: Generating a Square Wave

A square wave is a waveform in which the amplitude strictly alternates between a fixed minimum and maximum. A square wave can be created by programming your Raspberry Pi to turn an output pin on and off.

Here is a simple C solution to generate a square wave with frequency 100 kHz (1/(5+5 μs)):

```
void squareWave(){
     while(1){
     setPinOn(gpioBase, pin)
     delayMicroseconds(5)
     setPinOff(gpioBase, pin)
     delayMicroseconds(5)
     }
}
```

By changing the delay period we can change the frequency of the square wave.

## 3.2 Measuring Waveforms using an Oscilloscope

Waveforms are specified by their frequency and amplitude. While a regular voltmeter can measure point values of a signal at a time, an oscilloscope can show the continuous values over a period of time of a signal. An oscilloscope can be used to display and analyze waveform of electronic signals. In other words, it draws a graph of a signal voltage as a function of time.

To view the waveform on the oscilloscope, connect the probe test cable to the signal source (red to output pin, black to ground pin). Most scopes will produce a two-dimensional graph with time on the x-axis and voltage on the y-axis.

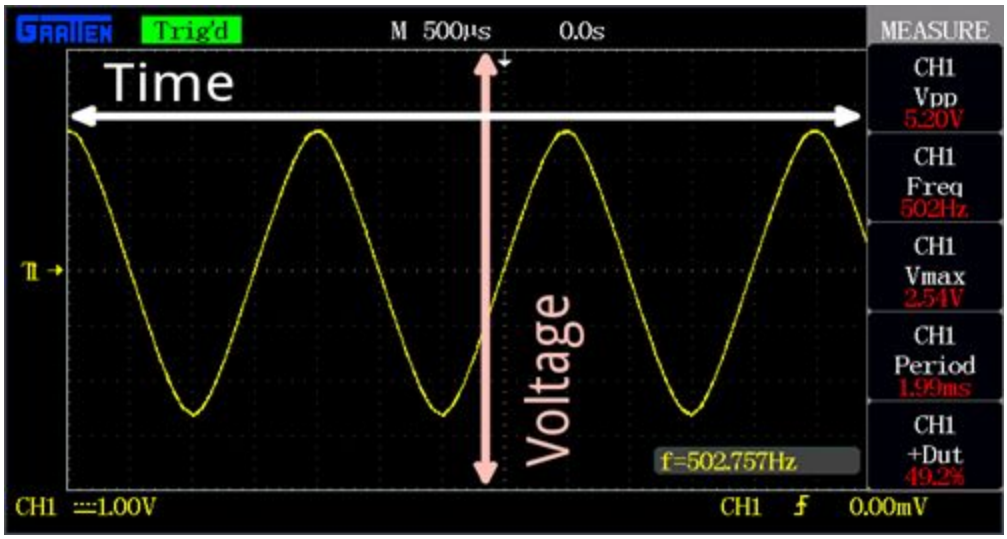



*Figure a. A sample oscilloscope output provided by Sparkfun*

On the x-axis, you can find out the timing characteristics, such as frequency, period, duty cycle, & rising/falling time, using a horizontal control to adjust time scale.
On the y-axis, you can find out the voltage characteristics, such as amplitude and maximum/minimum voltage, using a vertical control to adjust voltage scale.

In Lab 4, you will be asked to build an oscilloscope to read and visualize the waves you generated in Lab 3. We will discuss the detailed specification about oscilloscopes in the next lab.

Read the square wave you generated in part 1 with an oscilloscope to ensure that you have generated a wave with the desired frequency. Use control knobs on the oscilloscope panels to measure the highest voltage and the lowest voltage. Observe whether the highest reading is steady or if there is noise. Also, determine the fastest frequency that the digital output can create (by changing the frequency in the program). Record this frequency.

## 3.3 (Optional) Example: Generating Music using Raspberry Pi and a buzzer

We will learn how to use a buzzer to output music and write a program to adjust the frequency of a square wave based on user input.

Generally speaking, a tone is a particular frequency of sound. When we apply an audible signal (voltage, frequency) to the small buzzer, it makes continuous sound. A buzzer is an actuator that converts frequency into sound. Please note that a voltage does not make a tone but frequency is the cause of tone. Let's make a buzzer produce a C4 (middle C) tone. "Middle C" has a frequency 261.6 Hz (see 3.6 helpful hints for more detail).

Then we need to generate a square wave that has a frequency equal to 261.6 Hz. This means that every 1 second contains 261.6 cycles of the waveform (1 Hz = 1 cycle per second). The following function will produce a 261.6 Hz square wave for 1 second:

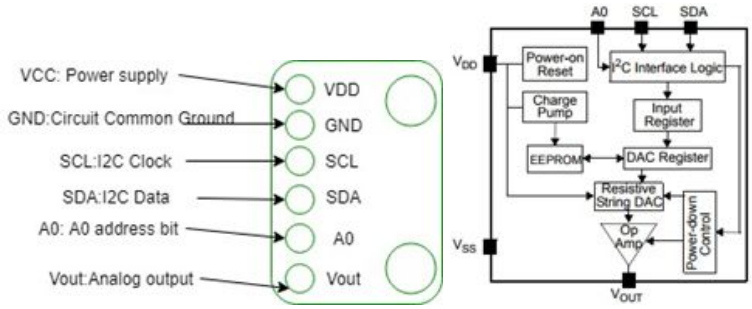
```
#include <wiringPi.h>
#include <stdio.h>

#define outputPin 0
#define C4 261.6 //Hz
#define period 1 //second
void tone(){
      long half_cycle = (long)(1000000/(2*C4))
      long numberOfLoops = (long)(freq*period);
      for(int i = 0; i < numberOfLoops; i++){
            setPinOn(gpioBase, outputPin);
            delayMicroseconds(half_cycle);
            setPinOff(gpioBase, outputPin);
            delayMicroseconds(half_cycle);
      }
}
void run(){
      tone();
       delay(20);
}
int main(){
      //set up gpio. Timer, etc. here
      while(1){
      run();
      }
}
```

Feel free to modify the code above to let the small buzzer produce a piece of music, TA can provide a small buzzer for your experiment. But this example is optional. Music note frequencies can be found here: (https://www.seventhstring.com/resources/notefrequencies.html)

## 3.4 From Digital Output to Analog Output

Now you are familiar with generating a digital output wave and how to adjust it to a frequency. In this section you will learn about how to convert a digital signal into an analog signal and how to create waves other than a square wave.




We provided you with a DAC MCP4725 12-bit chip (shown in the diagram above), which will help you convert digital signals into analog.

**You *MUST* solder the chip to the pins using soldering wires if the chip has no pins. If you don't know how to solder, your TA can show you.**

12 bit means that it will accept up to 4096 possible inputs to provide an analog output, where 0 is 0V and 4095 is the full scale voltage (which is determined by the voltage you supply to the VCC pin). According to the data sheet the VCC voltage can be in the range 2.7V to 5.5V.

There are six pins on the chip package. SDA will send data from Raspberry Pi to the chip (0-4095), and SCL (clock) will control the output rate.

You will need to enable your Raspberry Pi I2C functions in order to send signals via I2C ports. This can be done via the command "sudo raspi-config". In the menu that appears, choose option 5, "Interfacing Options". Then choose option P5, "I2C". To control the I2C output to the DAC, you can install the Adafruit Python Library found Here.


Use the code (Python) frame we provided below to generate a sine wave. Check that it is being generated as expected by displaying it on an oscilloscope.

```
sin_wave():

  t = 0.0

  tStep = 0.05

  while True:

          voltage = 2048*(1.0+0.5*math.sin(6.2832*t))

     dac.set_voltage(int(voltage))

     t += tStep

     time.sleep(0.0005)
```

### 3.5 Your Assignment: Build a Function generator using Raspberry Pi

In this section, you are going to build a function generator using what you learned from sections 1- 4. ***You should write your own functions to generate the shapes.*** The command window should display nothing until an external button is pressed. Then the system should ask for 3 input:

1. Shape of the waveform (square, triangle, or sin. Triangle is a symmetric triangle, not a ramp wave)
2. Frequency (up to 20 Hz)
3. Maximum output voltage

When the inputs are confirmed, the Raspberry Pi should output the correct wave with correct characteristic continuously until the button is pressed again. Then the system should ask for 3 inputs again (continuing the cycle). Demonstrate your finished function generator to the TA.

## 4 Question to Explore

1. What is the highest frequency that your Raspberry Pi can generate using digital output? Why is this the case?
2. Does the signal with the highest frequency stay steady or fluctuate? If not and there is noise, where does the noise come from?
3. How can you convert a digital PWM (Pulse width modulation) signal into an analog signal. (e.g. possible circuit design, software conversion)
4. What is the maximum frequency you can produce with your Raspberry Pi functional generator on different wave shapes?
5. Explain your design for the functional generator (you can use diagrams to visualize your system state machine, what function you implemented, etc.)

# 5 What to Turn In

- Answers to the questions in section 3.4 and submit it as report(**pdf/word**) on eCampus (4 points)
- Code (1 point)
- Demo the function generator: different waveform(2), different frequency(2) (e.g., 10Hz, 50Hz), Max/Min Volt(1)

# 6 Helpful Hints

1. Sample Music notes can be used in part 3: format: tone(time (second)) (all 4$^{th}$ octave): G(0.75) A(0.25) G(0.5) F(0.5) E(0.5) F(0.5) G(1) D(0.5) E(0.5) F(1) E(0.5) F(0.5) G(1) G(0.75) A(0.25) G(0.5) F(0.5) E(0.5) F(0.5) G(1) D(1) G(1) E(1) C(1)
2. Note frequencies can be found here: (https://www.seventhstring.com/resources/notefrequencies.html)