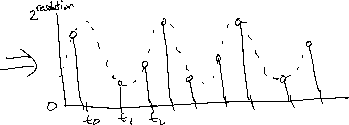
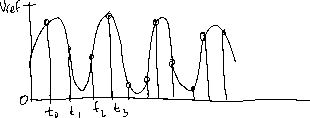
# ADC Tutorial

This tutorial will teach you how to initialize and use an ADC peripheral to convert an analog input signal to a digital value inside the microcontroller (STM32F4). This tutorial you will use a potentiometer to turn on a series of LED’s depending on what voltage signal the potentiometer is sending. We will be using a STM32F4 Discovery board in this example since there are four preconfigured LED’s that we can use.

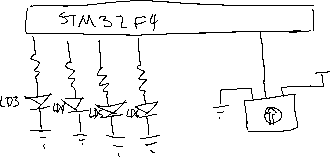
# Background

An Analog to Digital Converter takes an input signal that is analog and converts it to a digital signal. Analog meaning a continuous voltage signal that ranges from two potentials (commonly 0 – 5 Volts). This peripheral is used for several different types of applications such as position sensing, signal conversion, volume control, etc. It works by sampling an analog signal at discrete points of times and then returning a value that represents what percentage the analog signal is between its two reference voltages. Take a look at the below example.



As you can see the ADC takes the value of the analog signal at discrete times and returns a value between 0 and 2^resolution. Resolution being the number of bits to describe the signal. For example, an eight-bit resolution will have 256 possible positions to represent all voltages between 0 and Vref. This means that a value of 128 is 0.5\*Vref. Other resolutions are possible such as 12 bit which has 4096 values to represent the same range making it much more precise. An ADC can be configured to either convert at a periodic rate called continuous conversion or be done as requested by software. When a conversion for an ADC finishes a conversion\_complete flag is raised alerting the user that the ADC value can be read.

# Deliverables



<https://1drv.ms/v/s!As0nXV-wtCHfiPJJvKkkYqOHw5NEnQ>

Video of operation

Above is the sketch of the circuit that is desired. The LED’s are already on the board so only the potentiometer needs to be fitted to the circuit. Notice how the pot is between 3 (3 Volts) and Gnd that is because the STM microcontrollers use 3 as the reference voltage for all ADC calculations. This means that a 3-volt signal will be recorded as 100% and a 5-volt signal will also.

In this the goal is to light LED’s as the potentiometer is moved from 0 to 3-volts. Between 0 and 0.5-volts no LED’s will be lit. At .5 to 1.25-volts LED3 will be on. 1.25-2-volts LED4 will be lit. 2-2.75-volts LED5 will be lit. Finally, 2.75-Max LED6 will be lit. So, at 3 volts all LED’s are lit up.

<Insert Video of Op Here>

# Materials

## Physical Materials

1. STM32 F4 Discovery Board
2. Potentiometer (preferably one that can go into a bread board)
3. Breadboard
4. Jumper Wires
5. USB Mini Connector

## Software

1. Have the most recent PER VM installed and you should have all that is necessary.

Or

1. STMCubeMx
2. System Workbench

# Procedure

## Circuit Setup

To set up the circuit you need to connect the potentiometer to the breadboard. Use a male to female jumper wire to connect the ground pin of the Pot to GND on the STM Board. Similarly, do the same for the power pin and connect it to 3 volts. The last connection is the signal pin of the potentiometer. There are many choices that could be made however in this case we are going to use PA1. So again connect to PD11 with a jumper wire.

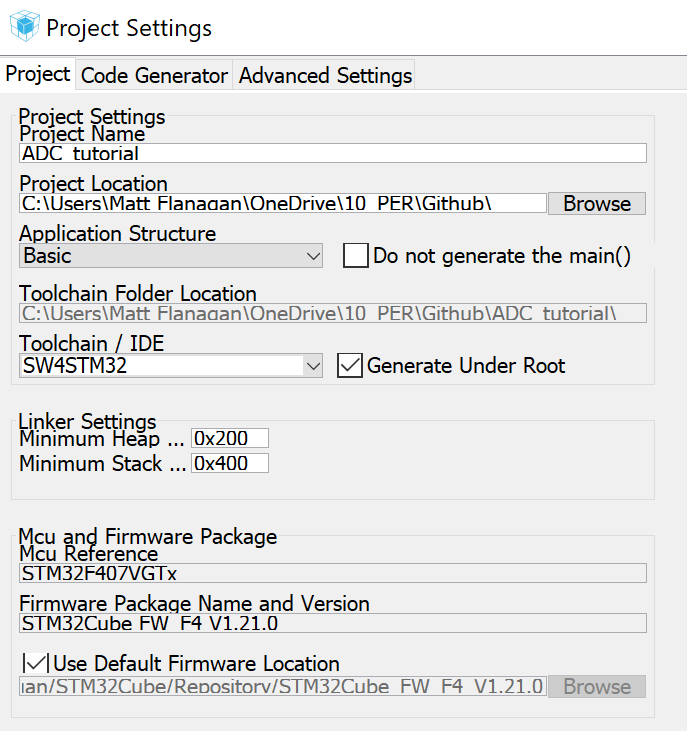
## STM CUBE

We use stmcubemx to do all pin, middleware, and clock configuration for our micros. This saves us countless hours of coding and debugging on stuff that just gets in the way of the real meat of the coding.

1. Open STM Cube and select create new project
2. Select Board Selector
   1. Click on STM32F4
   2. Scroll to the bottom and click on STM32F4Discovery
   3. Click Start Project
   4. Click ok to initialize all peripherals to default

Now you will see a graphical view of the STM32F4 microcontroller on your screen. I encourage you to play around with this. Clicking on pins will show you all configurations that are possible for that specific pin. Clock Configuration tab allows you to customize the clock for all your peripherals (typically don’t need to do anything in this section). Configuration is where you will go to set up the configuration of all your peripherals for example setting the resolution of the ADC.

### Configure ADC on Cube

1. On the pinout tab select PA1 (should be on the middle right)
   1. Choose ADC1\_EXT1
2. On the left choose ADC1 and select IN1. This tells the program which channel to sample on for the ADC. This should make ADC1 turn green.
3. Go to the configuration tab and click on ADC.
   1. Keep Clock Prescalar at 2. The prescalar determines how fast the ADC should actually sample the incoming analog signal.
   2. Leave resolution as 12 bits. This is how precise the ADC is. For 12 bits you have 2^12 values to represent 0 to 100%. For 8 bits you have 2^8 or 16 times less resolution.
   3. Use right alignment. It determines how the result will be placed in the register. If left aligned bit shifting will be required to calculate the correct value.
   4. Leave the next 4 as default they deal with hardware activated conversions. For this tutorial it is not required. We will initialize the conversion in software.
4. Click ok. All ADC configurations should now be set correctly.
5. Now generate code by going to project->generate code (ctrl + shift + g).
   1. 

Follow the above settings. Name and store the project where you would like.

After code is completed open it in system workbench. It should import properly, if not look at the above steps.

## Eclipse (System Workbench)

You have now generated all configuration code for the board and now you only must implement the software algorithms for ADC and the LED’s. With the generated code you will notice in the comments USER CODE Begin and USER CODE End. All of your code must be between those two headers or when you regenerate code from cube your code will be overwritten.

We use the HAL library that is provided by STM. HAL stands for hardware abstraction layer and it gives several useful functions to interface with the hardware peripherals instead of having to directly work with the registers.

HAL\_ADC\_Start – Used to start the ADC conversion. Must be done before getting the value.

HAL\_ADC\_Stop – Used to stop the ADC conversion. Must be done after ADC conversion is finished.

HAL\_ADC\_PollForConversion – A HAL function that waits for the conversion\_complete flag to be set indicating a value can be extracted from the ADC1 register.

HAL\_ADC\_GetValue – Actually returns the value of the ADC conversion.

Look at the above two functions and see if you can’t figure out how to use them to solve the tutorial problem. If you get stuck the code is on the Purdue Electric Racing GitHub. Again, remember to write your code in the while loop so it will run forever and that it is in the USER CODE sections.

# Building Your Project

In eclipse you can build your project by going to the project tab and select build project. If your code has any issues errors will show up in the console towards the bottom of eclipse. If you can build your project with no issues it’s time to test your code!

# Running Your Project

To run your project, make sure you have the STM board plugged into your computer with the USB mini. Then go to project->debug as->ac6 STM32 Application. If all goes well, you should have a new debugging perspective open up that will be stopped at the beginning of your main function. You have all the functionality of a normal IDE with regards to stepping and debugging tools. If you hit continue then your code will be running. Try it out and see if it works as you intended. Once you flash your board the program is forever loaded on the micro until a new program is loaded so once you power it up again it will always run your ADC code until it is re-flashed.

# Finished

Congrats if you made it here it means that you finished the tutorial. Congratulations you will be a great programmer for PER sometime soon. ADC is one of the simpler peripherals to use however it is extremely common and useful. All of the skills you used here will be used in all other peripherals just some might be more involved then the others.