

# Reading from Analog and Encoder Sensors. Outputting to PWM.

---

## Part 1: Reading an Analog Signal

### Objective:

Control the brightness of a LED using pulse width modulation (PWM) based on the position of the potentiometer.

### Background Information:

#### Analog to Digital Converter (ADC):

Arduino's onboard *analog to digital converter (ADC)* converts an analog signal to a digital value. The following equation relates the input voltage to the converted digital value:

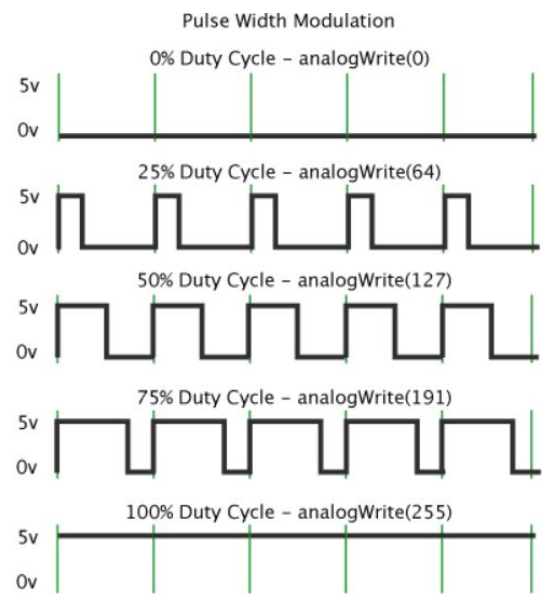
$$\frac{A/D_{result}}{ADC_{resolution}} = \frac{V_{in}}{V_{ref}}$$

- $V_{in}$  is the input voltage – typically from a sensor or in this case a potentiometer
- $V_{ref}$  is the reference voltage (usually 5v or 3.3v)
- $ADC_{resolution}$  is the resolution of the ADC conversion, in this case it is 10 bits (1023)

**Question:** If the value of the ADC on A0 of the Arduino is 512, what is the input voltage on A0?

## Pulse Width Modulation (PWM)

If the microcontroller does not have Digital to Analog Converter (DAC) then it can only output a binary high or low voltage (5v or 0v). *Pulse width modulation (PWM)* is used to create an *average* voltage by varying the duty cycle, or the amount of time the signal is on/off. A 100% duty cycle corresponds to fully powering the LED with 5v and a 0% duty cycle means the LED will receive 0v. A 50% duty cycle means *on average* the voltage seen by the LED is 2.5v. The value assigned to the PWM block can be from 0 to 255: 100% duty cycle corresponding to 255 and 0% duty cycle corresponding to 0.



*PWM duty cycle*  
(<https://www.arduino.cc/en/Tutorial/PWM>)

## Simulink Model

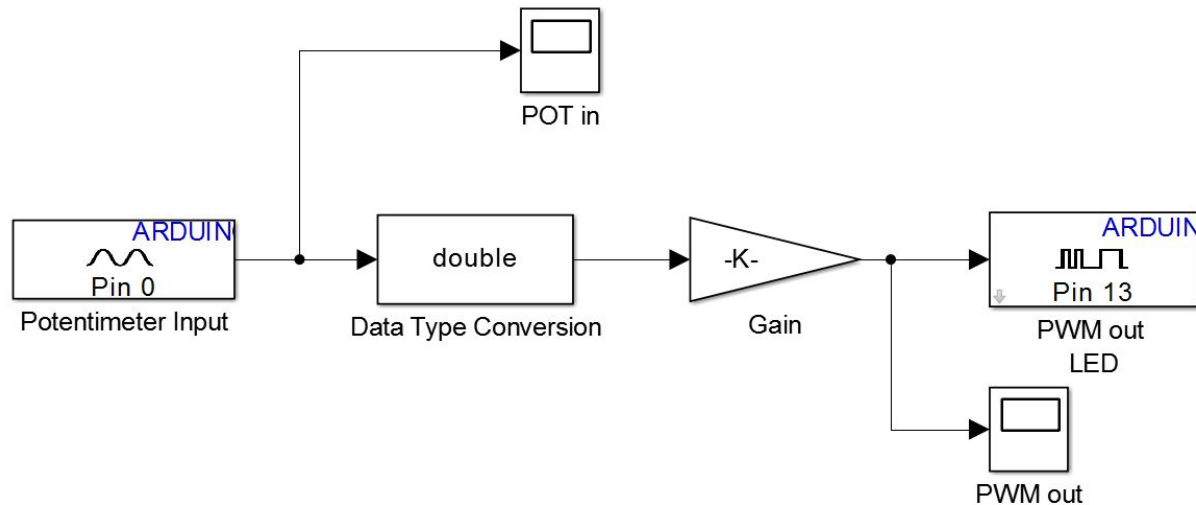
It is easiest to start with a Simulink model that is already set up for hardware implementation. The Demo files for your hardware should be used for this purpose. You can always run the demo file if you think you might have a hardware problem.

After installing RASPLib, right click on the Rensselaer Arduino Support Package to open the library, under demos open the demo for MinSegShield M1V4 (or MinSegMega depending on your hardware), save it to another location under a useful name.

Delete everything in this diagram except the blocks you want to use. (these blocks have different names depending on your hardware system).

Build and run the following Simulink diagram with the following settings.


(Note: The "POT in" and "PWM out" blocks both use the 'Scope' block found under 'Sinks')



- **Configuration Parameters:** The “Fixed-step size” should be .03
- **Analog Input:** Set the Pin number to 0 (Pin 13 for MinSegMega) and the sampling time to -1. When the sample time is -1 the block will inherit the sample time used in the simulation settings.(in this case .03). You can find the pin number for your potentiometer written on the shield near the potentiometer.
- **PWM:** Set the Pin to 13 (the onboard LED).
  - **Note:** If pin 13 is used for an output you might have to go the configuration settings and unselect “detect on overrun”
- **Gain:** Set the gain to the correct value to so the analog input’s full range will utilize the full PWM output
  - Hint: what is the maximum digital value for the PWM block and the maximum value for the ADC?
- **Data Type Conversion:** Go to Simulink-> Signal Attributes. Double click this block and set output data type to double. This is done so that floating point math is performed to avoid truncation and division errors associated with fixed point numbers.

**Question:** What is the correct value for the gain such that the full range of the potentiometer will utilize the full range of the PWM?

### Checkpoint 1:

Observe the scopes while changing the position of the potentiometer wiper. Click the ‘Autoscale’ button  to see the full range of the graphs.

**Question:** At what approximate potentiometer wiper position is the LED brightest? At what position of the potentiometer wiper is the LED completely dim?

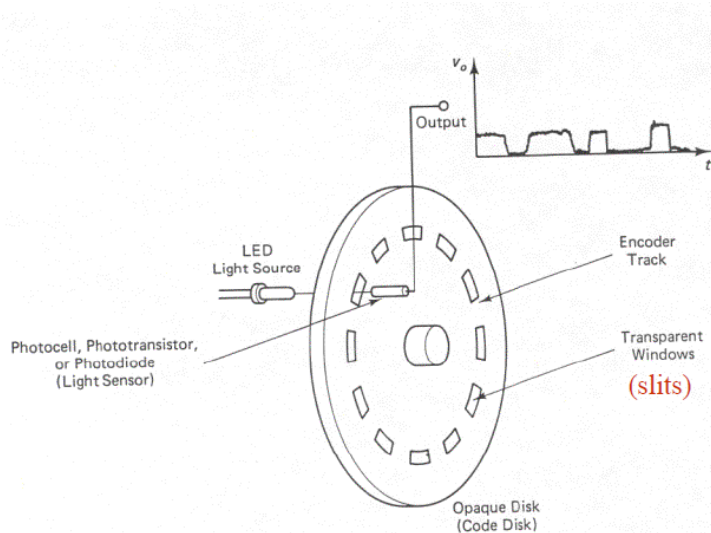
## Part 2: Reading Encoders

### Objective:

For Part 2, an encoder will be used to determine the position of the output motor shaft. The brightness of the LED will be controlled based on the position.

### Background Information:

An encoder is a device that can be used to determine position. Most encoders use optical sensors to output a pulse train which can be decoded to determine position and direction. In this lab, the NXT's internal encoder will be used to track angular position.



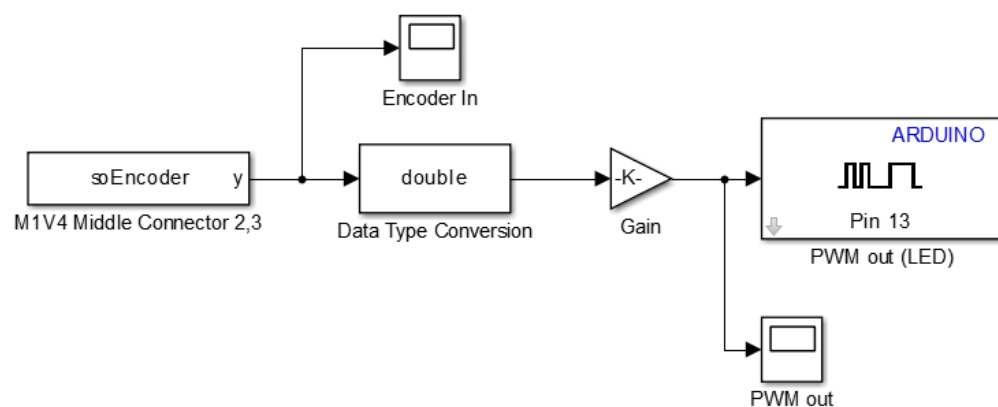
- The NXT encoder has a raw resolution of 12 counts. The figure on the left below show the actual encoder wheel in the NXT motor. It has 12 holes. The optical sensor will output a high or low (1 or 0) depending on the position of this encoder. Each revolution will produce 12 pulses.



- If the output motor shaft moves 1 revolution the encoder wheel turns 15 revolutions – this means for **one revolution of the motor output shaft there will be  $15 \times 12 = 180$  pulses from the encoder.**
- The encoder device provides two channels A & B. These channels are the encoder pulses with one channel shifted by 90 degrees. By monitoring these two channels both the position and direction of the motor shaft can be determined. **If quadrature decoding is used the resolution can be increased by 4 times.**
- The Encoder block included with this lab performs quadrature decoding so the total resolution is:

$$\text{Angular Resolution} = 15 * 12 * 4 = 720 \text{ pulses per revolution of output shaft}$$

### Simulink Model



**Question:** What value does the gain need to be so that 1 revolution of the motor output shaft will obtain the maximum value of the PWM? What happens if the motor is turned more than 720? What happens when the encoder position is negative?

## Checkpoint 2:

Run the simulation and observe the scope and the LED when you rotate the encoder's wheel clockwise and counter-clockwise.

Verify that one revolution of the output motor shaft results in a position of 720. You can also add a 'Sinks -> Display' block to see exact values.