EECE 321 Assignment #4 – Pulse Width Modulation - Arduino

Work either individually or in groups of **two or three** to complete these exercises. Students are required to submit suitable format electronic documents through Moodle for their solutions (one per group); **paper submissions will not be graded**.

Be certain to provide references for your sources and the names of your collaborators. **Work that is claimed to be an individual effort, but is decidedly the work of others, will receive a grade of zero (0)**.

Pulse width modulation is an important technique to convert digital values to analog ones using simple circuitry.

This lab is broken into several phases and will take several weeks to complete. By the end, you will have learned, refined, and mastered several skills. It starts with programming an Arduino board to output a PWM signal that controls the intensity of a light emitting diode (LED). As you have learned in your electronics class, an LED produces luminosity as a function of the current. The current in turn, is a non-linear function of the applied voltage. You typically use a current limiting resistor to set the current and intensity. The effective range of control of intensity is fairly limited by adjusting the voltage. Instead, we rely on the human vision system’s low pass filtering characteristic. The eye (and the processing in the brain) can be remarkably fast as evidenced by a major league batter being able to hit a fast ball travelling at ~90 mph. Nevertheless, there is a persistence effect to the eye where it continues to “see” a light even after it has been turned off. If we switch an LED off and on fast enough, the eye will not perceive any flicker. Instead, the eye and brain will time-average it. By adjusting the amount of time the LED is on compared to the time it is off, we can effectively adjust its perceived intensity. It is convenient to keep the total switching period constant. The percentage of time it is on to the total period is called the Duty Factor (DF). For example, 50% DF means that the light is on for the same amount of time it is off. The reciprocal of the switching period is the switching frequency, called refresh rate in some contexts.

Many PWM applications are static DF or set points. A few applications change the set point as fast as each switching period. Since the setting cannot be changed faster than the switching frequency, this limits the highest frequency of change. For this reason, PWM is rarely used for high fidelity audio.

A binary digital value is expressed by a certain number of bits. Let us assume that we have a monopolar signal. In other words, all values are greater than or equal to zero. A binary value of all ones corresponds to the full scale (FS) analog value. Assuming a constant FS value, using more bits divides it into more analog values with a power of two relationship. In other words, 11 bits has twice as good of resolution as 10 bits. Since the analog value is controlled by the amount of time the switch is on, more bits of resolution mean that the switching period is divided into more time slices. These relationships imply implementation tradeoffs with hardware switching speeds.

1. Arduino PWM to control an LED
   1. Start the Arduino IDE
   2. Declare a constant for a maximum 8 bit integer: **const** **int MAX\_8\_INT = 255;**
   3. Declare a variable *led* and set it equal to 10. There is an LED built onto the DUE board but it is not connected to a PWM pin. We will connect an LED to pin 10 : **const** **int led = 10;**
   4. Declare a variable *brightness*: **int brightness;**
   5. Declare a variable *fade\_amount*: **int fade\_amount;**
   6. Initialize *brightness* to 0 and *fade\_amount* to 1 in the *setup* routine.
   7. In the *loop* routine, output the current brightness value to the *led* pin: **analogWrite(led, brightness);**
   8. Then update the brightness level: **brightness = brightness + fade\_amount;**
   9. Check to see if it is at the maximum or minimum value: **if (brightness == 0 or brightness == MAX\_8\_INT)**. If so, update the fade amount: **fadeAmount = -fadeAmount;**
   10. Add a little delay in the loop to set the switching period: **delay(30);**
   11. Compile, download, and run the program. Describe what you observe.

Starts with nothing and gradually brightens to a dim LED

* 1. Connect an oscilloscope and observe the output waveform. Describe what you observe.

Pulse wave where the width varied from small to large.

* 1. Connect a potentiometer between 3.3V and ground with the wiper connected to an analog input pin, e.g. A0.
  2. Rewrite your code to input a value from the ADC, scale it appropriately (the ADC value ranges from 0 to 1023 for default 10 bit resolution while the PWM is always 8 bits), and output the value to the PWM.
  3. Vary the pot and describe what you observe
  4. Capture oscilloscope plots where the pot is near the low end of its range and another where the pot is near the high end of its range.

The Arduino DUE uses an ATMEL microcontroller as do other Arduino boards. ATMEL engineers have developed circuitry in their microcontroller chips that automatically implements a PWM and control it via registers. A software engineer wrote a driver in C++ that makes the appropriate settings in these registers when you call analogWrite. In the next lab, you will develop a custom PWM component in NIOS-II for QSYS. Write a short report describing your results and what you learned. Include…

* 1. Include your source code.
  2. include the plots from the oscilloscope showing at least 2 periods