Name: ***Roll Number:***

Laboratory 2 – Analog output from the Arduino

Introduction:

The Arduino has 6 analog input pins (A0-A5). But it does not have a real method of producing an analog output voltage. To control most physical systems, it is necessary to produce an analog output voltage (for example, to vary the base voltage on a transistor. In this lab, we will look into techniques of producing an analog output VOLTAGE from the Arduino. Note that we will focus on analog voltage control (analog current control will be done in a later lab).
You will have to recall some of the analog electronics techniques learnt in earlier labs to optimize the solution for this lab.
SOLUTIONS MUST INCLUDE UNIQUELY IDENTIFIABLE SCRENSHOTS of your code and PICTURES/VIDEOS of your experimental demonstrations.

Part A: Background and buildup: `analogWrite()` function

Looking at the Arduino language reference, you will find, surprisingly, that there does exist a function `analogWrite(pin,value)` that should allow you to write an *effective* analog value to a few selected pins (that normally function as digital I/O pins). Look through the Arduino language reference and answer the following questions (the reference is available from the 'Help' menu of the IDE)

A.1) [Datasheet investigation] **1**

Which pins can be used by the `analogWrite(pin,value)` function?
What is the maximum current that any of the `pin's` can source?
The LED's in your parts kit can tolerate a maximum forward current of 20mA. What is the appropriate circuit you must setup to test `analogWrite(pin,value)`?

A.2) [Further datasheet and source code reference investigation] **1**

What is the range of allowed values for the variable `value`?
Why does `value` have to be in this range?

The language reference mentions that 'PWM' – pulse width modulation is used to create a digital square wave of adjustable duty cycle corresponding to `value` on the set `pin`.
Look up the definition of 'Pulse Width Modulation' – essentially it sets the duty cycle of a square wave. By averaging (i.e. *filtering*) such a square wave, you would get an analog voltage value ranging from 0% to 100% of full scale with some ripple left over because of the underlying square wave.

A.3) Write a program using `analogWrite(value,pin)` with `value` and `pin` set within the allowed parameter range.
Use an LED connected to `pin` to demonstrate your programs operation.
Solving the following sub-questions requires thinking through and explaining two gotchas:
A.3.1) The digital output voltage at `pin` is *always* swinging between 0V and 5V. The LED turn-on threshold voltage is less than 5V. So why does the apparent brightness change as you vary `value` in the above function call? **1**

A.3.2) Using only the parts available in your kit, (i.e. no DMM!) make an analog circuit in combination with your Arduino program to measure the threshold turn on voltage of the LED as accurately as possible. You can use both the calculated voltages based on the PWM `value` settings, and other pins available on the Arduino board to accurately measure the threshold turn ON voltage of the LED. **4**
Furthermore, solving part A.4 below may help you make a clear experimental demonstration with picture/video and screenshots of this question A.3.2)

Note: For Parts A.4 and A.5 below, continue working in the digital domain. We will look at 'true' analog output in the next lab assignment

A.4) **[OPEN, HARD Question]** Without access to a DSO, unfortunately you cannot measure the actual time domain voltage waveform being produced at `pin`. The frequency of the PWM square wave is high enough that due to persistence of vision, you will not see any perceptible variation in the LED brightness once the analog voltage driving it has exceeded the threshold to turn it ON.
Question: Devise and demonstrate a clever method of revealing the fact that the voltage driving the LED is, in fact, a digital square wave between 0V and 5V and not a truly analog voltage whose value can range continuously between 0V and 5V. **6**
Hints to approach this question:-
1) Look into the underlying source code of `analogWrite(value,pin)`
2) You typically make the function call in your `loop()` program only once. What do you think happens in the short time span between when `loop()` ends, and the program code pointer jumps back to the start of `loop()` and repeats?
3) Can you write a cleverer version of `analogWrite` (still using the digital I/O pins!) and add some associated analog circuitry to reveal the PWM nature of the output.

A.5) The Arduino is driving a digital (0-5V) signal during the `analogWrite`(..) function. Hence after filtering etc, your final analog output voltage range is restricted to slightly less than the 0-5V span.

Suppose an application requires an output voltage of a larger range. **7**
WITHOUT using any extra power supply, design a digital-to-analog converter that can produce an output voltage spanning 0V to 10V (i.e. double the value normally allowed by the 5V digital supply on the Arduino pins)
NOTE: this is a design exercise. It would be difficult to test your output in practice without access to a DMM, and you risk burning the limited number of LED's provided in your parts kit.
Hence, provide an LTSpice simulation of your circuit design. Use a time-dependent variable duty cycle 5V voltage source in your LTSpice simulation to simulate output from one of the Arduino `pin`'s