EGR: 226 Microcontroller Programming and Applications

Winter 2021

Instructor Prof. Trevor Ekin

**Lab 8: Creating a Pulse Width Modulated (PWM) Pseudo Analog Voltage Using**

**Timer Peripherals of a Microcontroller**

Gabriel Gasbarre

March 16th, 2021

Table of Contents

[1. Objectives 3](#_Toc66887043)

[Equipment 3](#_Toc66887044)

[2. Introduction 4](#_Toc66887045)

[3.1: Pre-Lab: 4](#_Toc66887046)

[3.2: Part 1- Controlling the DC motor using the SysTick timer 4](#_Toc66887047)

[3.3 Part 2 - Controlling the Speed of a DC Motor using TimerA in PWM Mode 5](#_Toc66887048)

[3.4 Part 3 – Generating a PWM signal by reading the Duty Cycle from a keypad 5](#_Toc66887049)

[4. Procedure: 5](#_Toc66887050)

[4.1: Part 1 - Controlling the DC motor using the SysTick timer 5](#_Toc66887051)

[4.2: Part 2- Controlling the Speed of a DC Motor using TimerA in PWM Mode 6](#_Toc66887052)

[4.3 Part 3 - Generating a PWM signal by reading the Duty Cycle from a keypad 7](#_Toc66887053)

[5. Results: 8](#_Toc66887054)

[5.1 Part 1 Results 8](#_Toc66887055)

[5.2: Part 2 Results 8](#_Toc66887056)

[5.3: Part 3 Results 9](#_Toc66887057)

[6. Conclusions. 9](#_Toc66887058)

1. Objectives

To develop a program for the MSP432 microcontroller that interfaces to a DC motor and controls the speed using PWM

To use the SysTick timer to develop a PWM signal

To use the Timer A configured to provide a PWM signal as an output with connection to an I/O pin

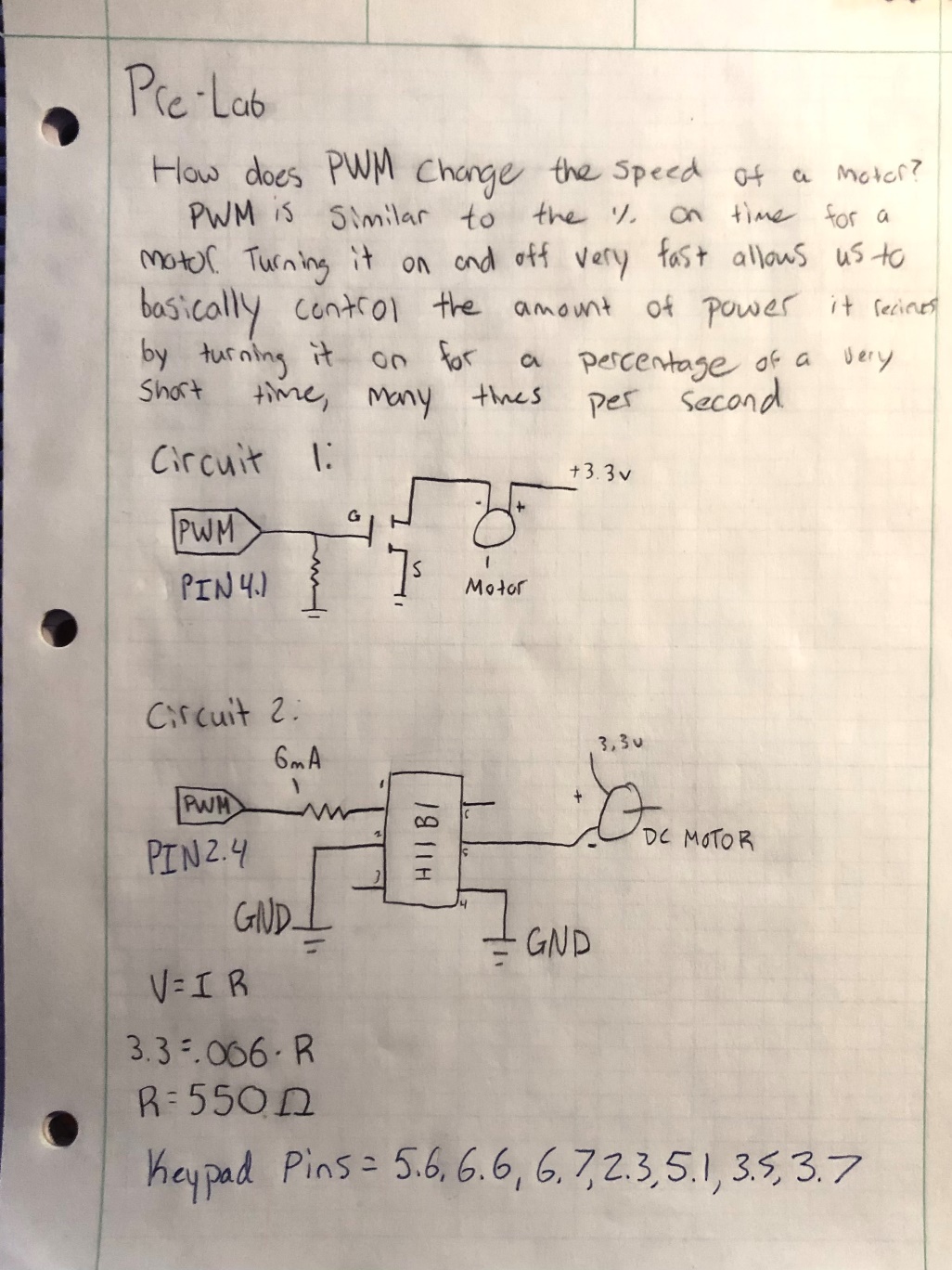
To use a keypad to enter a value for setting the speed level

Equipment

|  |  |  |
| --- | --- | --- |
| **Part** | **Description** | **Model** |
| CCS (Code Composer Studio) | Integrated development environment to develop applications for Texas Instruments embedded processors. | 10.0.00010 |
| MSP432 | Mixed-signal microcontroller family from Texas Instruments. | MSP432P401x |
| EGR:226 Lab 6 Exercise | Interfacing a keypad with the MSP432 | N/A |
| Optocoupler | electronic component that transfers electrical signals between two isolated circuits by using light. | H11B1 |
| 300 DC Motor | rotary electrical motor that converts direct current electrical energy into mechanical energy | JQ24-35H390F |
| Keypad | EGR 226 Lab Kit component | N/A |
| MOSFET Transistor | Used to amplify or switch electronic signals and electrical power | 2N7000 |

1. Introduction

3.1: Pre-Lab:



3.2: Part 1- Controlling the DC motor using the SysTick timer

For part one, students are to use the SysTick timer to generate the PWM mode in order to create an analog signal for controlling the speed of the DC motor. This motor works best between 30 to 50 hz, so an operation frequency of 40 hz will be ideal. This motor can also be run from the +3.3v power supply directly from the MSP432. To create an operational circuit, a transistor will be used to wire the circuit between the MSP432 GPIO pins, +3.3V, and the DC motor. The duty cycle for the PWM should be a declared variable in the CCS code, and will be passed to the motor control function.

## 3.3 Part 2 - Controlling the Speed of a DC Motor using TimerA in PWM Mode

For part 2, students are to configure TimerA to create a PWM signal that is connected directly to an I/O pin for controlling the DC motor. Students will also modify their circuit in order to incorporate the Optocoupler H11B1 instead of the transistor. This combination of electrical components also works best at 40Hz. The TimerA peripheral selected by the student should be configured so that the duty cycle can be modified by changing the value of a declared variable.

## 3.4 Part 3 – Generating a PWM signal by reading the Duty Cycle from a keypad

For part 3 students are to incorporate their keypad into the MSP432 program and circuit from part 2. Ideally, students will use their program developed in the pre-lab assignment and combine it with their code from part 2 in order to enter a value from the keypad for the PWM duty cycle (0 to 100) that will change the speed of the motor.

# 4. Procedure:

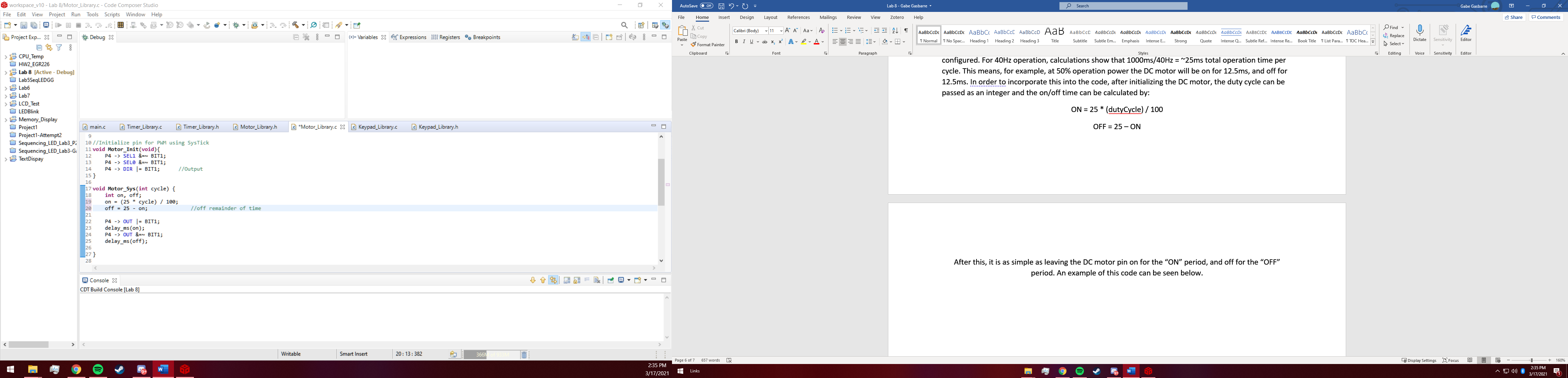
4.1: Part 1 - Controlling the DC motor using the SysTick timer

To create a pseudo-PWM using the SysTick timer, first the duty cycle timing must be configured. For 40Hz operation, calculations show that 1000ms/40Hz = ~25ms total operation time per cycle. This means, for example, at 50% operation power the DC motor will be on for 12.5ms, and off for 12.5ms. In order to incorporate this into the code, after initializing the DC motor, the duty cycle can be passed as an integer and the on/off time can be calculated by:

ON = 25 \* (dutyCycle) / 100

OFF = 25 – ON

After this, it is as simple as leaving the DC motor pin on for the “ON” period, and off for the “OFF” period. An example of this code can be seen below. This function must be continuously looped in order to create the desired effect of a DC motor running at a variable power.



4.2: Part 2- Controlling the Speed of a DC Motor using TimerA in PWM Mode

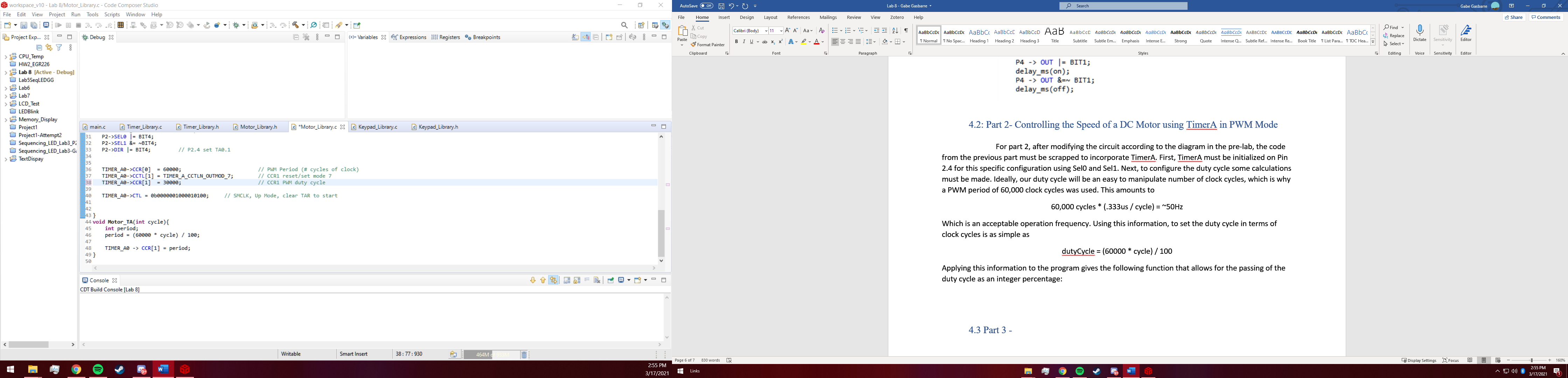
For part 2, after modifying the circuit according to the diagram in the pre-lab, the code from the previous part must be scrapped to incorporate TimerA. First, TimerA must be initialized on Pin 2.4 for this specific configuration using Sel0 and Sel1. Next, to configure the duty cycle some calculations must be made. Ideally, our duty cycle will be an easy to manipulate number of clock cycles, which is why a PWM period of 60,000 clock cycles was used. This amounts to

60,000 cycles \* (.333us / cycle) = ~50Hz

Which is an acceptable operation frequency. Using this information, to set the duty cycle in terms of clock cycles is as simple as

dutyCycle = (60000 \* cycle) / 100

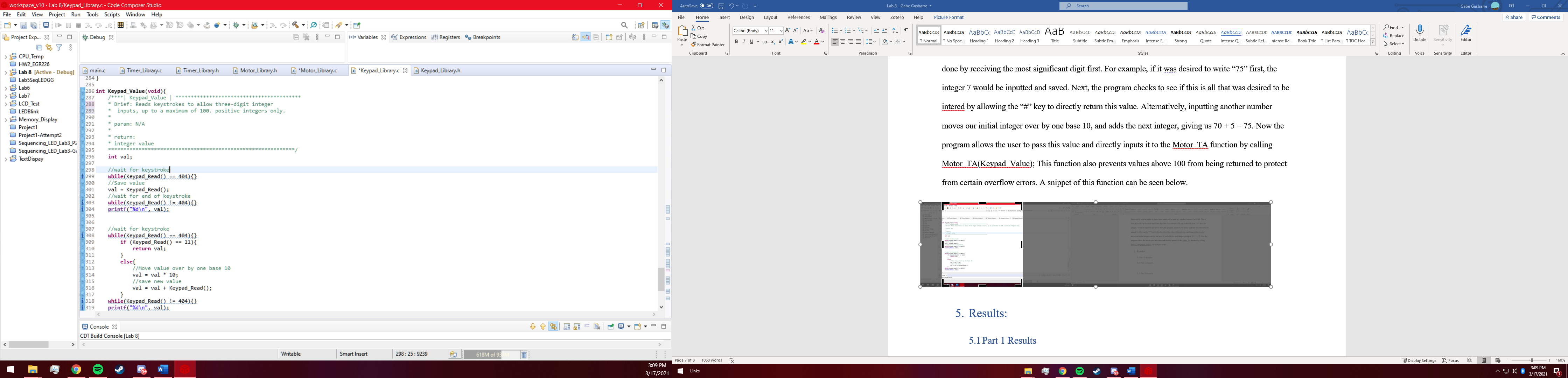
Applying this information to the program gives the following function that allows for the passing of the duty cycle as an integer percentage:



All that is left to make this operational is to call Motor\_TA from main with the desired operation power percentage passed to the function.

## 4.3 Part 3 - Generating a PWM signal by reading the Duty Cycle from a keypad

To incorporate a Keypad into our program from part 2, it is necessary to read and pass a variable from 0 – 100 to the Mortor\_TA function. This is done by creating a custom keypad read function. Using the keypad Library from lab 6, it is already possible to receive a single integer at a time, now functionality must be added to make these inputs add to equal any number between 0 and 100. This is done by receiving the most significant digit first. For example, if it was desired to write “75” first, the integer 7 would be inputted and saved. Next, the program checks to see if this is all that was desired to be intered by allowing the “#” key to directly return this value. Alternatively, inputting another number moves our initial integer over by one base 10, and adds the next integer, giving us 70 + 5 = 75. Now the program allows the user to pass this value and directly inputs it to the Motor\_TA function by calling Motor\_TA(Keypad\_Value); This function also prevents values above 100 from being returned to protect from certain overflow errors. A snippet of this function can be seen below.



# Results:

* 1. Part 1 Results

Part one was unusually by far the easiest of the three parts as almost all functionalities had already been completed at one point or another during other labs. Transistor usage was already very well understood, and the timer library for the SysTick timer had already been made, so it was as simple as making some simple calculations to find the duty cycle. This in turn resulted in what was arguably a flawless use of the DC motor using the SysTick timer.

5.2: Part 2 Results

Part two of this lab was significantly more difficult as TimerA had never been used before. Though there were very good code examples offered by Dr. Kandalaft, These initialization functions had not been optimized for the DC motor, requiring some calculations as seen above. There was also significant failures in figuring out what value exactly should be used for TIMER\_A0->CCR[1]. However, after these setbacks it was found that TimerA was most certainly more userful, powerful, and efficient for PWM than SysTick timer.

## 5.3: Part 3 Results

Part 3 was another pretty simple part, as many functionalities had already been made. Hooking up the kepad to the same pins as used in lab 6 resulted in no real input issues, meaning all that was to be done was to save a value and pass it to the Motor\_TA function. After some debugging, a nearly flawless function with significant versatility was made operational and satisfactory.

# Conclusions.

Definitely a very informative lab in terms of PWM education, especially with combining PWM functionality with TimerA. With almost no knowledge of timerA going into the lab, it is rewarding now having a decent understanding. Though the keypad functionality of the lab was quite repetitive when compared to other keypad labs, it was interesting to combine multiple components together to create an externally controllable motor.