Samuel Boyle

Final Project

CPE 303

12/11/2019

*Goals:*

The purpose of this final project is to test what I have learned throughout the semester programming in assembly. The final goal of the lab was to use a keypad to type in a certain percentage. Based upon that percentage that is entered it will spin the motor at different speeds. While the motor is spinning the lcd will not only display the percent that it is spinning at, but it will display the rpms on the board and it will constantly update itself.

*Equipment Used:*

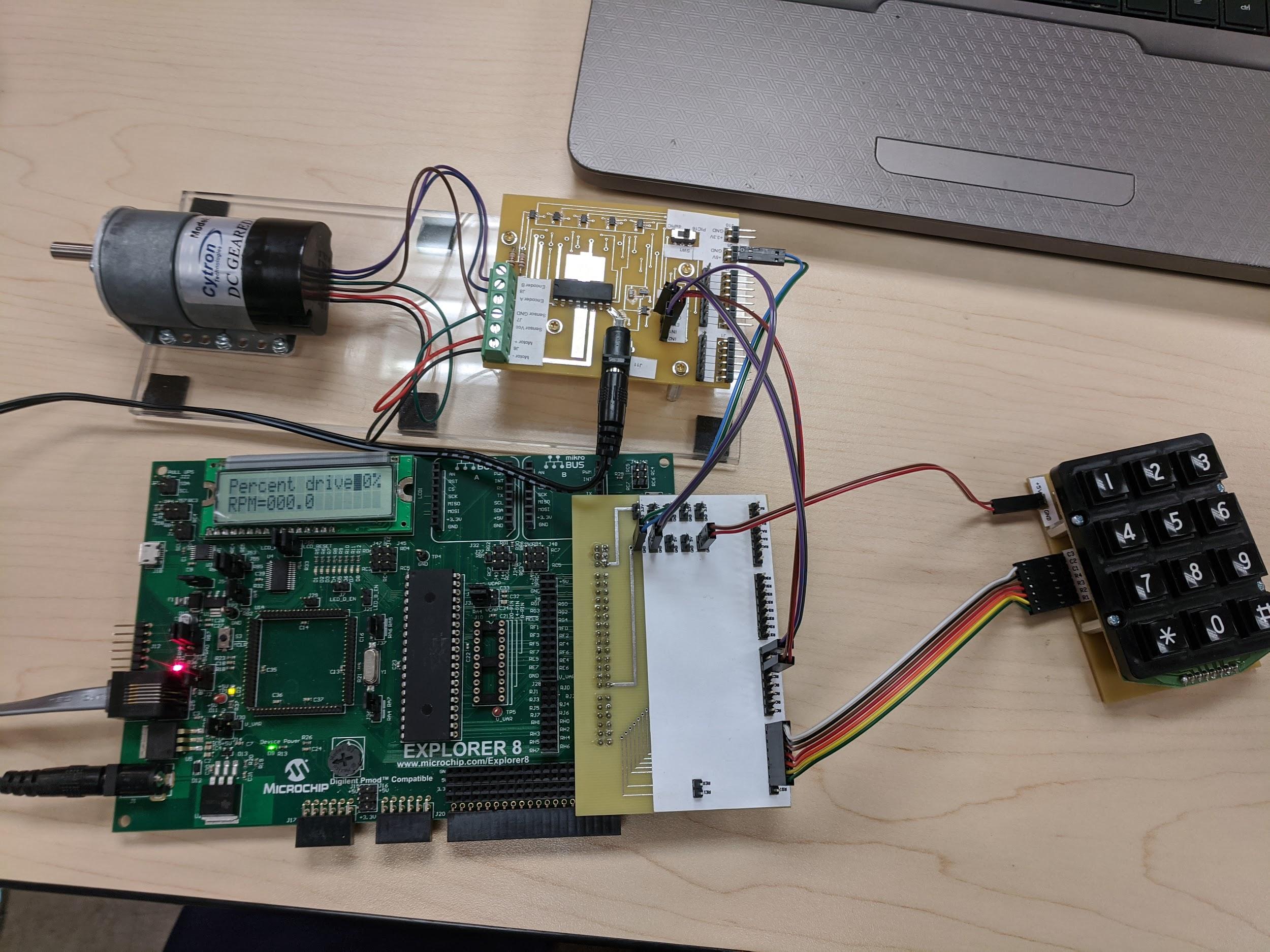
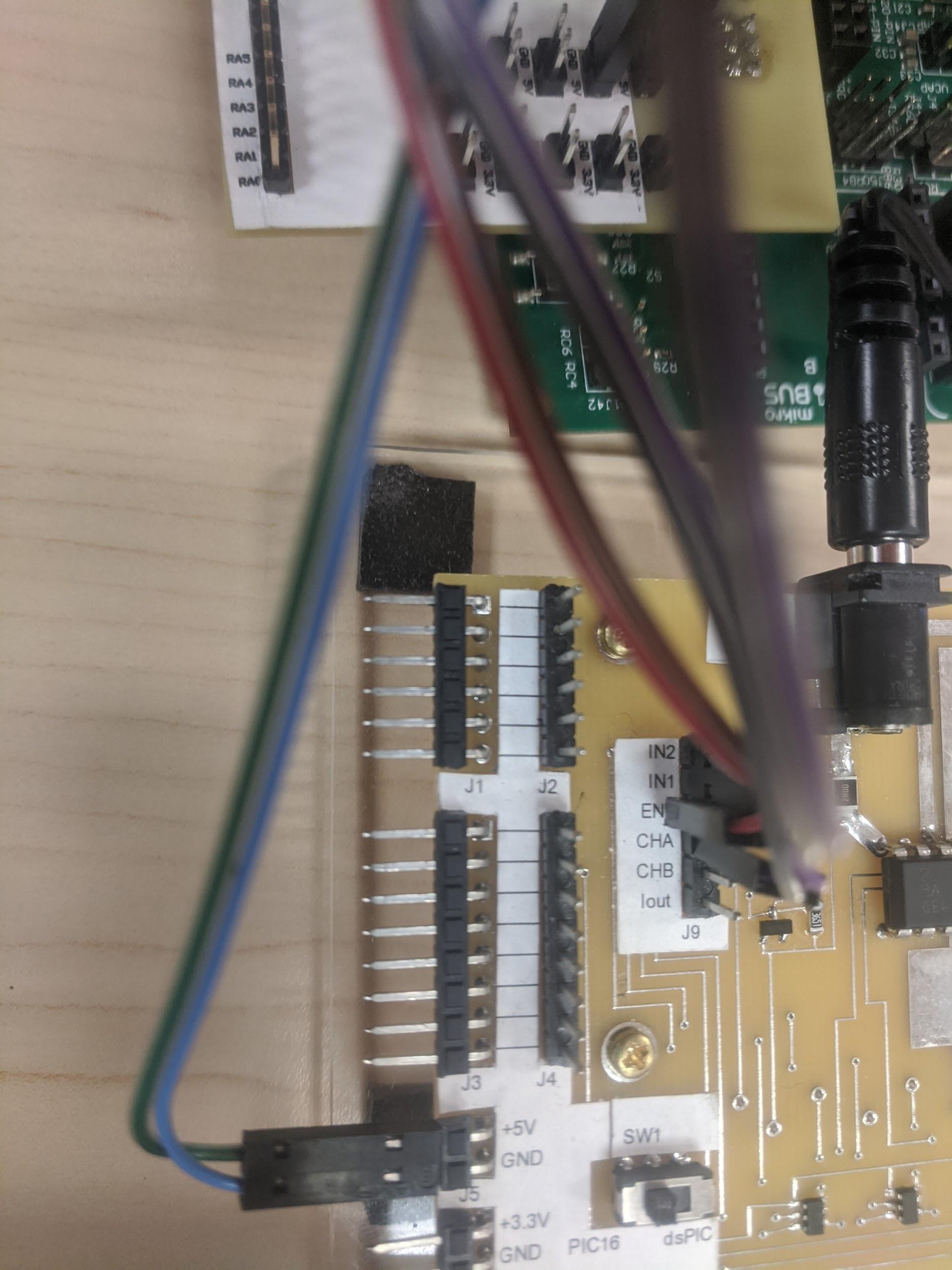
I used MPLAB X v5.20 and the compiler used was mpasm v5.84 to program and compile my code. I used PIC16F877A as my chip for this project. The chip was placed on the explorer 8 board. The lcd used was a Hitachi HD44780U LCD-II. To download the code to the chip I used a PICkit 3. Another important thing that I used for programming to the chip was the lcd subroutine code. This was used to write to the lcd. I also used a keypad which was connected to the explorer 8 by another board and a couple of jumper wires. I also connected a Cytron DC geared 30k motor to this board as well which worked as my motor. This was hooked up to another board that was provided to me. I also used an oscilloscope to test to make sure I was getting the correct number of rpms.

*Design Specifications:*

Create a motor control center that will allow a user to set a DC voltage to drive a motor. The more voltage applied to the motor the faster it will spin. Use the LCD display as a graphical interface on the explorer 8 board. The top line will display percent drive and two digits. The two digits must be an integer from 00 to 99 that can be entered by the user. The last entered percent value will be displayed until a new input is entered. On the bottom line it will display rpm = xxx.x. It should display the current rpms to the nearest tenth. For any rpm value greater than 50 there will be an allowable error of 1%. For any value between 25 to 50 rpms there will be an allowable error of 2%. Anything less than 25 rpms will not be required to have a percent error. The rpm value displayed shall not update any faster than .5 seconds and no slower than 6 seconds.

*Design:*

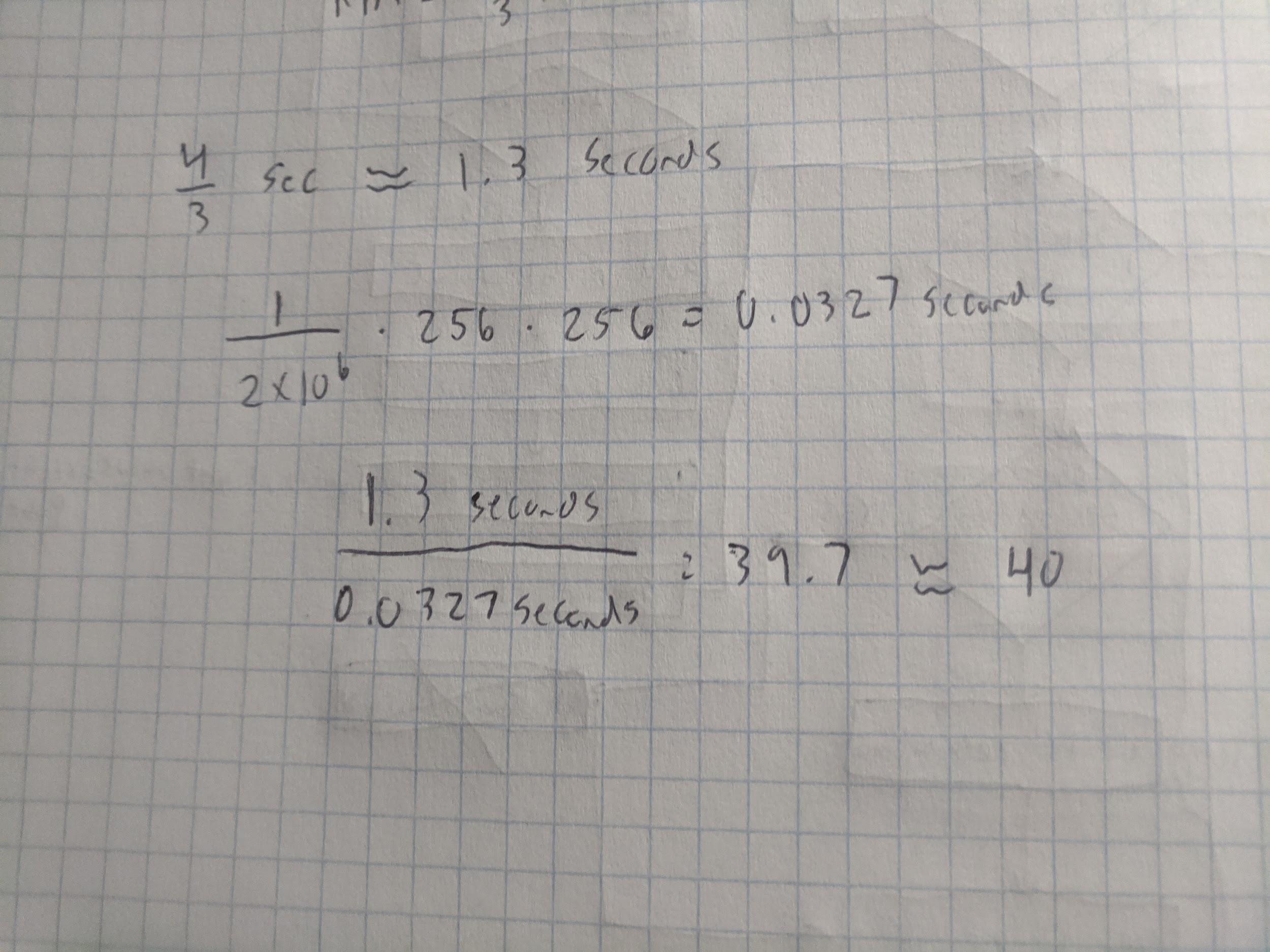
Wiring: The first part of my design was wiring my project correctly. The first thing I wired was the keypad which was pretty simple. It had to be connected to the +5v and GND to power the board and then I choose to hookup the inputs/outputs for the board to RD0 to RD6. The next thing I had to wire was the motor. Luckily the motor wiring was done for the most part and I only had to connect a couple things to the explorer 8 board. IN2 & IN1 were hooked up to +5v and GND. For the sake of this project it didn’t matter which one went to the 5v and which went to GND because I didn’t care which direction the motor spun. ENB was hooked up to RC2 because this is where the pwm signal would be generated. Channel a was hooked up to RC0 because that is where the external counter for tmr1 is. The motor was also powered by the board as well.

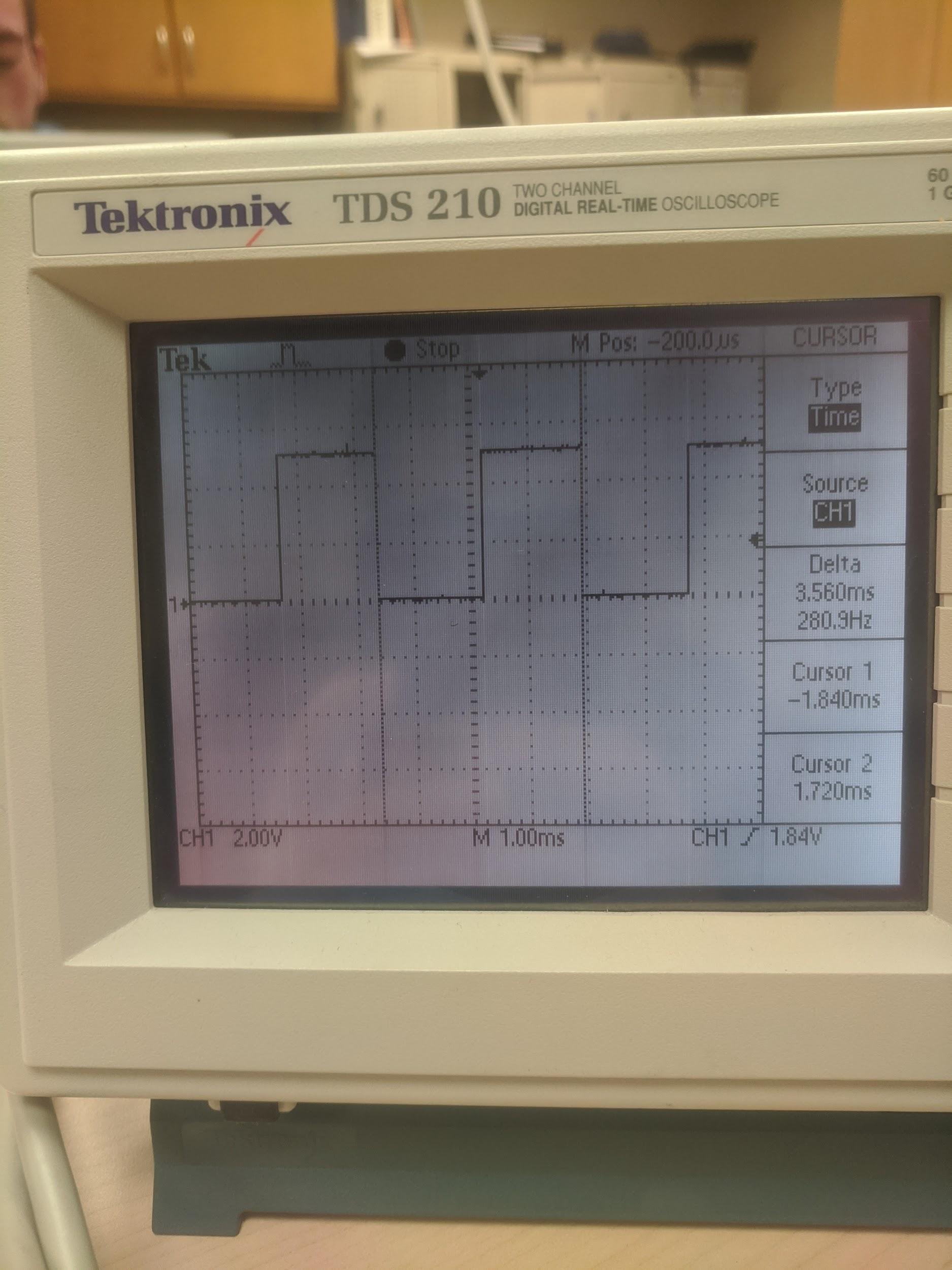


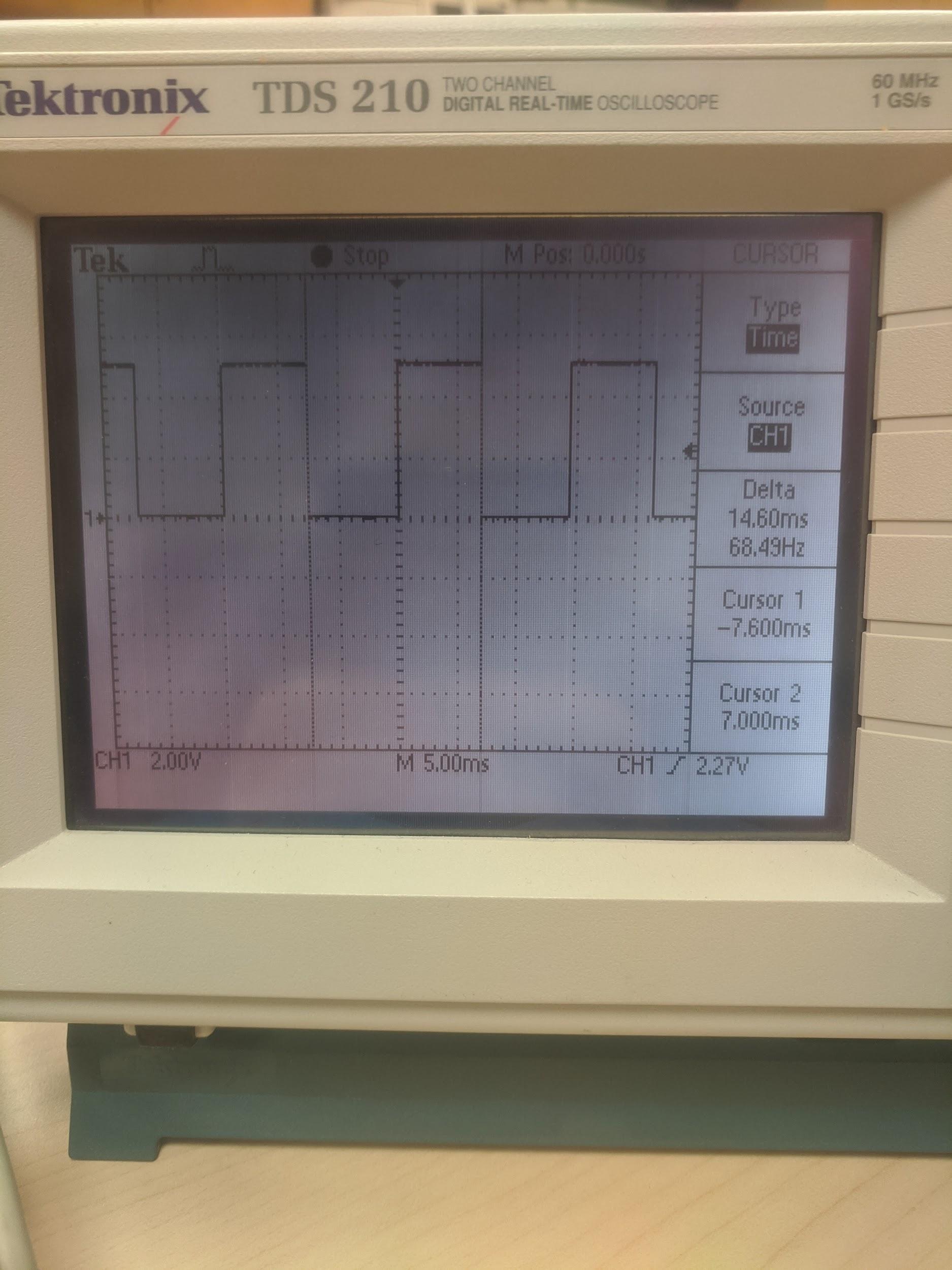
Code: My code starts out by initializing the include file and stuff in the chip like the oscillator and debug mode. Then I go through and define a bunch of variables for my shared variables I have total\_value, decimal, hundreds\_place, tens\_place, and ones\_place. I place total\_value here because it is also used in the motor\_run source file. The rest of the variables I placed there because it made my code more convenient. Next I had the variables that were not placed in the shared variable part. I had many variables that performed different tasks. I mostly had a bunch of flags and counters that were used for counting or signaling to move to the next part of code. I also had a couple of variables that would hold numbers until they were needed. After I initialized all of my variables I then set up my interrupt for tmr0. The only thing that would occur in the interrupt was that a flag would be set to 1 to signal that an interrupt had occurred and also the tmr0 overflow flag in intcon was also set back to 0. After the interrupt the program will then go through and set up the lcd. Then it will write the words percent drive and rpm = in a simple loop that will go through and call a table. The program then sets the cursor to the spot 0x8D on the board. Then it will go through and clear several variables to make sure that no values were preloaded into them because it could affect the program later on down the road. Then I set up tmr0 and tmr1. For tmr0 I loaded a prescaler value of 256 and enabled the portb pull up bit into the option\_reg. For tmr1 I set bit 0 to one to turn on tmr1 then I set bit 1 to one to turn on the external clock at RC0 and finally I set bit 2 to one as well so it didn’t synchronize the external clock input. After that starts my main loop that will be continuously polling and updating. It sets all of the columns to inputs and then waits to make sure no buttons are held. Then it will clear col\_flag which is used to signal if a button has been pressed. Then it will set up the tmr0 part which will be used to signal when to update the rpms. Since I am using a 30k motor I will be waiting 4/3 seconds or approximately 1.3 seconds. To set up the tmr0 to give me a 1.3 second delay I cleared out tmr0 because I want the full 256 loops. I also cleared out tmr1l and tmr1h because I want the count to reset every time I begin the delay so I can get an accurate rpm reading. I also clear the flag that signals that there is an interrupt. I then turn on the tmr0 interrupt by setting the global interrupt to 1 and tmr0 interrupt to 1 in intcon. The code will then sit in a loop waiting for either the interrupt to occur and wait for a button to be pushed. If a button is pushed it will load a value into val\_temp which will either be 0 for column 1, 1 for column 2, or 2 for column 2. If any of the buttons are pushed a flag will be set to one indicating a button was pushed. If no button is pushed and the interrupt occurs it will decrement the counter used to keep track if 1.3 seconds has occurred. If 1.3 seconds occurs and no button is pushed it will update the rpms. When a button is pushed it will switch to the row subroutine. It will set all of the rows to inputs and the columns to outputs. It will then go through and figure out which row is pushed and depending on which row is pressed it will add a different value for row 1 it won’t anything, for row 2 it will add 3, row 3 will add 6, and row4 will add 9. Based on what value in val\_temp it will call the correct value from the keypad table and then it will write the number in the correct spot on the board which is either in the left value which is location 0x8D on the lcd or it will write to the right value which is located at 0x8E. After the value is written to the board the same value is saved in either right or left value which will be used later on to load the correct value to the motor. After this the program will then return back to column section and wait to update the rpms or wait for a button to be pushed. However, for row 4 it will have to determine if star or pound was pressed because they perform different actions. If the star is pressed the keypad table will return a 1 and if pound is pressed then the keypad table will return a 2. If you bit test the value returned then you can determine which button was pressed. If pound is pressed it will take the left value and multiply it 10 times by adding it to itself and then add the right value. It will then call the motor\_run subroutine that will set up tmr2 and pwm so that you can change the motor speed with what value is displayed on the lcd board. If star is pressed then It will load the lcd with the value that was entered last to change the motor speed. If the zero is pressed it will just write a zero to the board. After the motor starts spinning it should start updating the rpm value. To update the rpm the program will take the value from tmr1 and load it into two values because tmr1 has 16 bits so I have two variables one that takes the upper 8 bits h\_number and number which takes the lower 8 bits. I then clear out the hundreds, tens, ones, and decimal place. It will then test number if it is odd or even and this done by testing bit 0 and if it is one then a 5 will be loaded into the decimal place holder. After this I must divide the number from tmr1 by 2. This is done by performing a rotate right. The division of two comes from the equation rpm = N(tmr1 value)/(4/3) \* 1/30 \* ⅓ \* 60/1 which simplifies to rpm = N/2. I then test to see if the h\_number is greater than 255 and if it is I set bit 7 in number to one to signal a division of 256. Then I go through a process of separating out each number by subtracting either 100, 10 or one. For every subtraction it will increment the correct decimal place. Once the value goes below 0 then the value subtracted will be added back. After each value is separate then the values will be written to board and then return to the main loop. For detailed flow charts refer to appendix A.

*Verification:*

Verification for the final project was pretty easy for the most part. I mostly just had to step through the program and make sure that the correct values were loaded into the correct spots. The first thing that I had to make sure that would work correctly was my keypad. I had to make sure that it would only write to the spots 0x8D and 0x8E, write from 0 to 9, and the star and enter would do what it was supposed to do. I tested this by running the debug mode on the keypad and stepping through the code making sure that when a number on the keypad was pressed that the correct number would be displayed. I also had to make sure that the keypad would write the value in the correct place and that if it wrote to 0x8D the cursor would go to 0x8E and if it was there then after writing that the cursor would go back to 0x8D. This ended up working. I also made sure that the number entered was stored later so that it could be used to input in the motor. After I got all of the numbers working correctly and made sure that the right values were in the right place by verifying them in the watches, I then had the task of setting up the enter or pound button and the star or previously entered value. In the enter button I had a subroutine that would total up the actual value displayed on the lcd and this was done by multiplying the left value by 10 and adding it to the right value. However, I ran into an issue that ASCII 1 is not the same value as a decimal 1, so I ended up having to subtract 0x30 from the value called from the table loaded into that variable. After I got the enter working the star button was pretty much working as well because it only loaded the previously entered value to the display so I just had to write the correct values to the board. After the keyboard was completely working I had to set up the motor. I made sure I set up the motor correctly first by creating a new project and working out of that. Once I got that working I added the source code from that project into my final project and set it up so it would work if I called a subroutine from that file. After that I had to set up the rpm portion of the project. I calculated the delay required and loading the correct values where they needed to be by solving it out.



I didn’t verify my this using the stopwatch, but I verified this later on down the road when I tested if I was getting the correct rpm value because if I get the correct rpm value then I know my calculations were correct and they did turn out to be. To test if I was getting the correct rpm values I had to use an oscilloscope. I used the probe and hooked it to channel a and grounded it to my board. These were my results:



To verify that the values are right you take the delta value and multiply it by ⅔ so for a delta value 280 multiplied by ⅔ would give you 186.6. The lcd displayed 186 and this fits the 1% error. For the second measurement ⅔ \* 68.9 = 45.9. The lcd displayed 44.5 and this fits the 2% error margin.

*Conclusion:*

There were I a lot of things that I learned in this lab and there are things I would have definitely done to make my life easier. This lab really tested my knowledge of the assembly language for this chip and It took a lot of effort to learn the new material for the final project. I learned how to spin a motor using pulse width modulation. I also learned how to set up timer 1 and timer 2 within the chip and use them to count or use them in pwm. This lab also improved my debugging skills as well because of the numerous times where I would run into small issues. I think this lab is a gateway into full fledged projects with these chips. I think If given any task and this chip I think that I could accomplish that task. This chip also has a lot of other things on it that we didn’t even touch. I think that the number of projects that could be done on using this chip is really limitless. Now with the knowledge I have from programming on this chip I think that I could move on to program on different chips that aren’t in the same family as the PIC16F877A.

Appendix A:

