

Temple University
College of Engineering
Department of Electrical and Computer Engineering (ECE)

Student Lab Report Cover Page

Course Number : 3613

Course Section : 002

Experiment # : Final Project

Student Name (print) : Robert Bara (Partner Zenon Matychak)

TUId# : 915614617

Date : 12/14/20

Grade : _____ /100

TA Name : Sung Choi

Processor System Laboratory Final

Part I. Report

1. Introduction

The following project applies our knowledge of DC motors by creating a circuit that will run a DC motor by sending pulse width modulated signals. The DC motor is connected to an optical sensor which will then help us calculate the rotations per minute and display them using our OLED display. Important background information for this project includes knowledge of labs 11 and 12 for the implementation of a motor, as well as working with the OLED display, working with hardware interrupts, and understanding timers/prescalers and their effects on the square wave generated and observed within the data visualizer.

2. Method and Procedure

Upon reading the manual, I developed the following flowchart as a starting point to build our code upon. Zen and I worked together on the code, I primarily tackled the OLED display, ISR Timer1 functions. Together both of us created the main function and Zen wrote additional comments and took the video shown later in the report. Flow Chart:

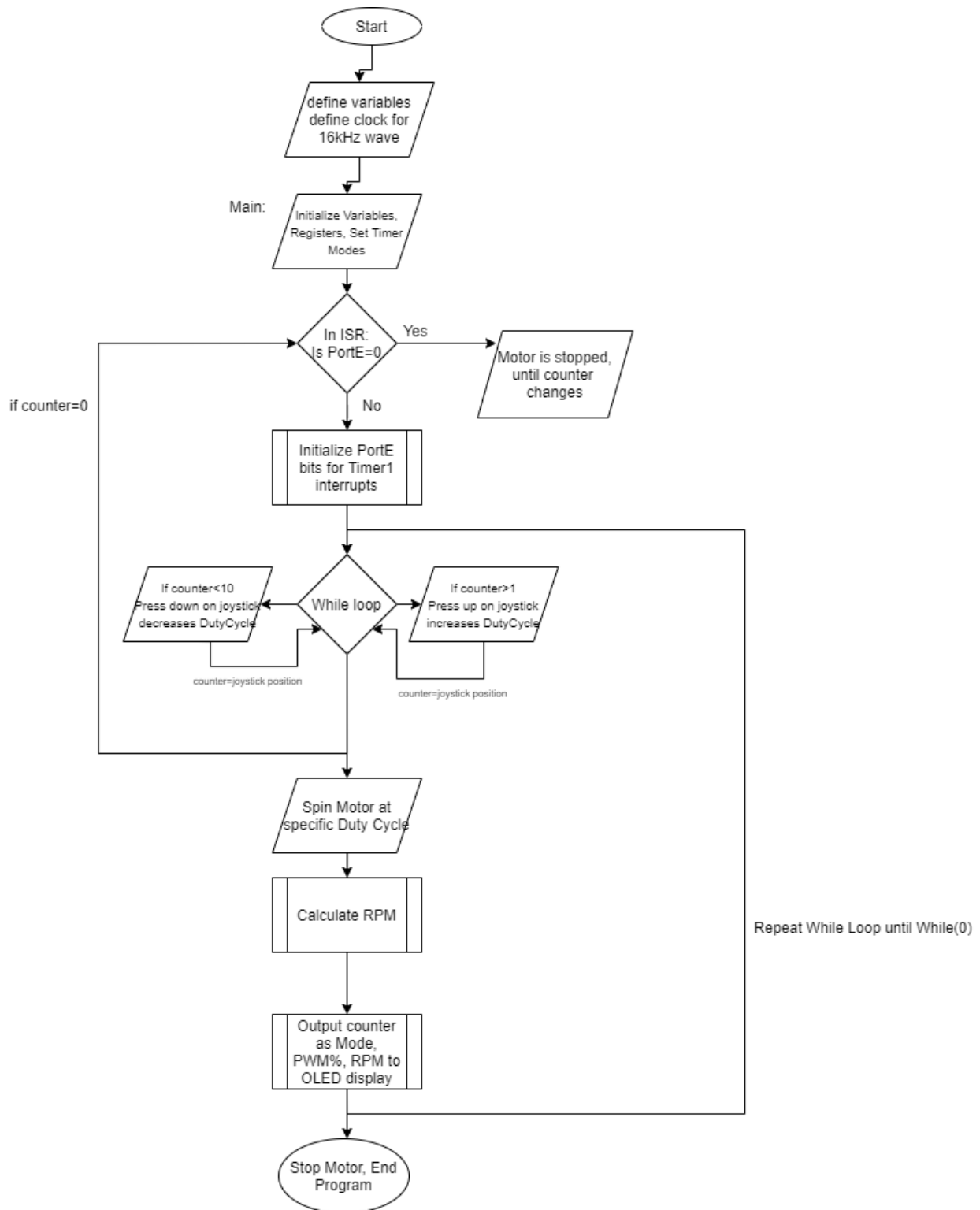


Figure 1. Flow Chart for Code and Hardware Results

To get the program to run properly, the motor must properly be assembled as stated in the manual, first by connecting the 3wires from the motor to their respective connections (white-pd2, red-

vcc, black-gnd). The motor should also be screwed into the stand with the wheel attached, and connecting the red and blue wires into the DC motor pins on the 324PB with two female-to-female jumpers. A male-to-male jumper should be connected from PE2 to PB2 to observe and deliver the proper pulse width modulated signals to the motor and properly observe the waves within Atmel's data visualizer. The OLED display can be connected to the three pins above the joystick as shown in the manual, which is connected to PortB. Finally, after starting to run the code in Atmel, the battery should be connected using the battery clip with the red being the positive VCC and black being GND into the M_Power pins on the 324PB.

The choice of choosing the values for the various timers used throughout our code came from observing the 324PB manual, by choosing to toggle the compares and yield a prescaler of 1024 for timer0 and no timer for timer1, furthermore the interrupts are enabled on a falling edge.

While the entire code may be found in the appendix below, two segments worth looking at is our ISR Timer1 functions which use priority logic to stop the wheel from spinning by setting PortE to 0 when the joystick is in the 0 position.

```
ISR(TIMER1_COMPA_vect){    //ISR for when timer1A interrupts

    if(count_int==0)

        PORTE=0x00;

    else

        PORTE &= ~(1<<DCmotor);    //turn off PE2 pin
}

ISR(TIMER1_COMPB_vect){    //ISR for when timer1B interrupts

    if(count_int==0)

        PORTE=0x00;

    else

        PORTE |= (1<<DCmotor);    //turn on PE2 pin
}
```

Our while loop may also need further clarification, as this is where in Main we are setting the joystick equal to a counter, which is then calculating the PWM signal and sending it to the motor and OLED display

```
while (1) {    //infinite loop

char joystick = (0b11110000&PINB); //get high 4 bits of PORTB (contains up and down button)
```

```

joystick = joystick^0xf0; //invert value of up and down button (Pullup resistors pull pins high)

if(lastInput != joystick){ //check if PORTB has been updated

if(joystick&(1<<up_count)){ //check for up button being pressed

    if(count_int < 10) //if the PWM value is less than 10,
        count_int++; //YES: increment it
    }

if(joystick&(1<<down_count)){ //check for down button being pressed

    if(count_int > 0) //if the PWM value is more than 0,
        count_int--; //NO: decrement it
    }

dutyCycle = count_int*10; //update dutyCycle variable for OLED display

lastInput = joystick; //keep the latest PORTB input
}

OCR1A = 0xFFFF* (dc_adj*dutyCycle)/10; //Duty Cycle LOW Activation, divide by 100 to get
Percentage (dutyCycle/10 = count(Mode)*dc_adj)

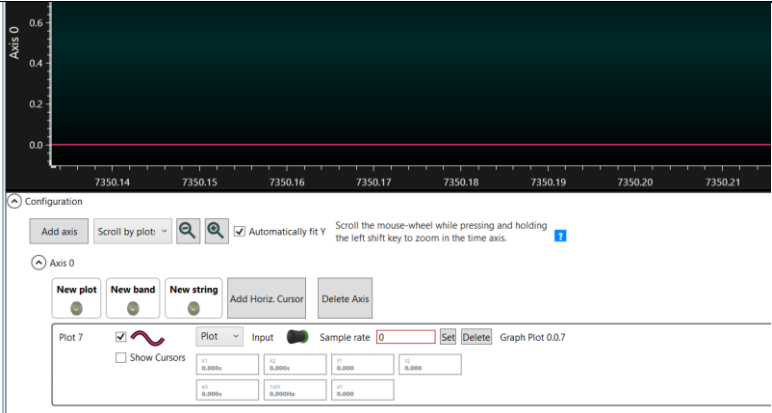

OLED_display(); //update OLED display
}

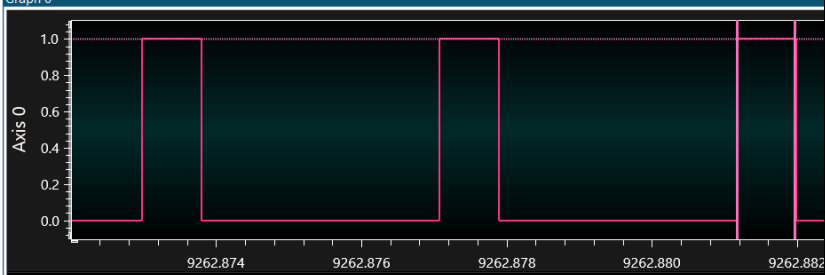





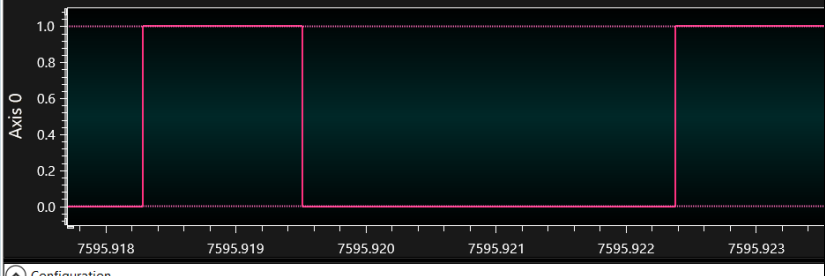





```





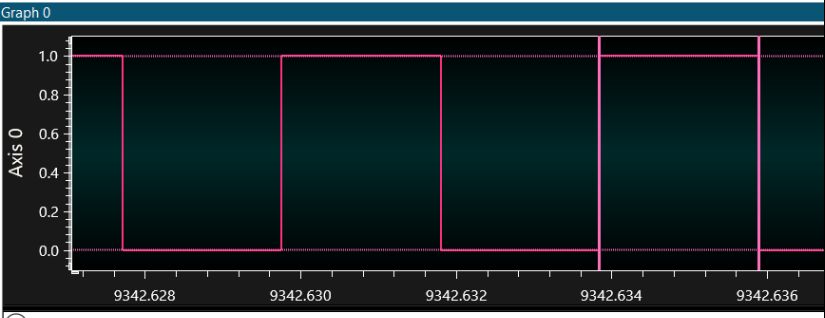




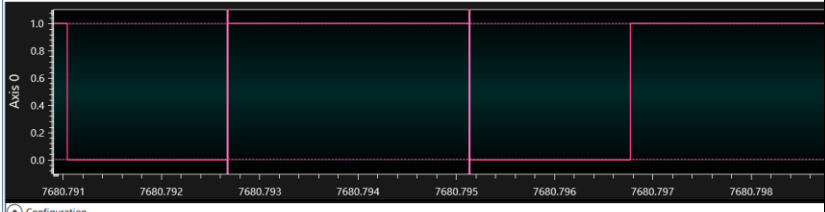





3. Result and Discussion

I was a little bit confused about the formatting of this report, so our results are primarily showcased below, while the link to the video is located under the Part II performance section of the report. For the calculated RPM values, I had trouble understanding the document on canvas. I was going to use Matlab to do the calculations, but the number of loops confused me, and I was unsure about how to calculate C to find the RPM value and then take the percentage error from the experimental results. I tried asking other classmates I had contact with for what they did, but no one had a clear answer, so I tried my best to answer/explain my results using intuition of the circuit and hardware results.

Calculating and Displaying Result

PWM Counter Value	PWM Duty Cycle (%)	On-Time Pulse Length (ms) / Time Period, T (ms)	Image of Motor PWM
0	0	0ms/4ms	
1	10	413us/4ms	

2	20	794us/4ms	<div><div>Graph 0</div><div><div>Configuration</div><div><div>Add axis</div><div>Scroll by plot:  </div><div><input checked="" type="checkbox"/> Automatically fit Y <div>Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis. </div></div></div><div><div>Axis 0</div><div><div>New plot</div><div>New band</div><div>New string</div><div>Add Horiz. Cursor</div><div>Delete Axis</div></div><div><div>Plot 7</div><div><input checked="" type="checkbox"/> </div><div>Plot</div><div>Input </div><div>Sample rate 0</div><div>Set</div><div>Delete</div><div>Graph Plot 0.0.7</div></div><div><div><input checked="" type="checkbox"/> Show Cursors</div><div><div>X1</div><div>9.263ks</div><div>dx</div><div>794.7us</div></div><div><div>X2</div><div>9.263ks</div><div>1/dx</div><div>1.258kHz</div></div><div><div>Y1</div><div>1000m</div><div>dy</div><div>0.000</div></div><div><div>Y2</div><div>1000m</div></div></div></div></div></div>
3	30	814us/4ms	<div><div>Graph 0</div><div><div>Configuration</div><div><div>Add axis</div><div>Scroll by plot:  </div><div><input checked="" type="checkbox"/> Automatically fit Y <div>Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis. </div></div></div><div><div>Axis 0</div><div><div>New plot</div><div>New band</div><div>New string</div><div>Add Horiz. Cursor</div><div>Delete Axis</div></div><div><div>Plot 7</div><div><input checked="" type="checkbox"/> </div><div>Plot</div><div>Input </div><div>Sample rate 0</div><div>Set</div><div>Delete</div><div>Graph Plot 0.0.7</div></div><div><div><input checked="" type="checkbox"/> Show Cursors</div><div><div>X1</div><div>7.554ks</div><div>dx</div><div>814.1us</div></div><div><div>X2</div><div>7.554ks</div><div>1/dx</div><div>1.228kHz</div></div><div><div>Y1</div><div>0.000</div><div>dy</div><div>1000m</div></div><div><div>Y2</div><div>1000m</div></div></div></div></div></div>

4	40	1.642ms/4 ms	<div><p>Graph 0</p><p>Configuration</p><p>Add axis Scroll by plot:   <input checked="" type="checkbox"/> Automatically fit Y Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis. </p><p>Axis 0</p><p>New plot New band New string Add Horiz. Cursor Delete Axis</p><p>Plot 7 <input checked="" type="checkbox"/>  Plot Input  Sample rate 0 Set Delete Graph Plot 0.0.7</p><p><input checked="" type="checkbox"/> Show Cursors</p><table><tr><td>X1</td><td>X2</td><td>Y1</td><td>Y2</td></tr><tr><td>7.622ks</td><td>7.622ks</td><td>0.000</td><td>1000m</td></tr><tr><td>dx</td><td>1/dx</td><td>dY</td><td></td></tr><tr><td>1.642ms</td><td>609.1Hz</td><td>1000m</td><td></td></tr></table></div>	X1	X2	Y1	Y2	7.622ks	7.622ks	0.000	1000m	dx	1/dx	dY		1.642ms	609.1Hz	1000m	
X1	X2	Y1	Y2																
7.622ks	7.622ks	0.000	1000m																
dx	1/dx	dY																	
1.642ms	609.1Hz	1000m																	
5	50	2.054ms/4 ms	<div><p>Graph 0</p><p>Configuration</p><p>Add axis Scroll by plot:   <input checked="" type="checkbox"/> Automatically fit Y Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.</p><p>Axis 0</p><p>New plot New band New string Add Horiz. Cursor Delete Axis</p><p>Plot 7 <input checked="" type="checkbox"/>  Plot Input  Sample rate 0 Set Delete Graph Plot 0.0.7</p><p><input checked="" type="checkbox"/> Show Cursors</p><table><tr><td>X1</td><td>X2</td><td>Y1</td><td>Y2</td></tr><tr><td>9.343ks</td><td>9.343ks</td><td>0.000</td><td>1000m</td></tr><tr><td>dx</td><td>1/dx</td><td>dY</td><td></td></tr><tr><td>2.054ms</td><td>486.9Hz</td><td>1000m</td><td></td></tr></table></div>	X1	X2	Y1	Y2	9.343ks	9.343ks	0.000	1000m	dx	1/dx	dY		2.054ms	486.9Hz	1000m	
X1	X2	Y1	Y2																
9.343ks	9.343ks	0.000	1000m																
dx	1/dx	dY																	
2.054ms	486.9Hz	1000m																	
6	60	2.463ms/4 ms	<div><p>Graph 0</p><p>Configuration</p><p>Add axis Scroll by plot:   <input checked="" type="checkbox"/> Automatically fit Y Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis. </p><p>Axis 0</p><p>New plot New band New string Add Horiz. Cursor Delete Axis</p><p>Plot 7 <input checked="" type="checkbox"/>  Plot Input  Sample rate 0 Set Delete Graph Plot 0.0.7</p><p><input checked="" type="checkbox"/> Show Cursors</p><table><tr><td>X1</td><td>X2</td><td>Y1</td><td>Y2</td></tr><tr><td>7.681ks</td><td>7.681ks</td><td>0.000</td><td>1000m</td></tr><tr><td>dx</td><td>1/dx</td><td>dY</td><td></td></tr><tr><td>2.463ms</td><td>406.1Hz</td><td>1000m</td><td></td></tr></table></div>	X1	X2	Y1	Y2	7.681ks	7.681ks	0.000	1000m	dx	1/dx	dY		2.463ms	406.1Hz	1000m	
X1	X2	Y1	Y2																
7.681ks	7.681ks	0.000	1000m																
dx	1/dx	dY																	
2.463ms	406.1Hz	1000m																	

7	70	2.870ms/4 ms	<div><div>Graph 0</div><div>Configuration</div><div><div>Add axis</div><div>Scroll by plot:  </div><div><input checked="" type="checkbox"/> Automatically fit Y <small>Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.</small> </div></div><div>Axis 0</div><div><div>New plot</div><div>New band</div><div>New string</div><div>Add Horiz. Cursor</div><div>Delete Axis</div></div><div><div>Plot 7</div><div><input checked="" type="checkbox"/> </div><div><input checked="" type="checkbox"/> Show Cursors</div><div>Plot <div>▼</div></div><div>Input </div><div>Sample rate <div>0</div> <div>Set</div> <div>Delete</div></div><div>Graph Plot 0.0.7</div></div><div><div>X1</div><div>7.851ks</div><div>X2</div><div>7.851ks</div><div>Y1</div><div>0.000</div><div>Y2</div><div>1000m</div></div><div><div>dX</div><div>2.870ms</div><div>1/dX</div><div>348.5Hz</div><div>dY</div><div>1000m</div></div></div>
---	----	--------------	--



I calculated the voltage by observing the 3.3V from device programmer in Atmel, from this I subtracted $9V - 3.3V = 5.7V$ and then multiplied this by the PWM%. I would have measured the voltage experimentally of the pulse width modulation, but I left my multimeter up in my home in the Poconos, and I did not know if there is a way to measure the voltage in Atmel. The RPM values were taken experimentally.

Joystick Up-Counter	PWM Duty Cycle (%)	DC Voltage (Calculated) [V]	RPM or RPS
0	0	0	0
1	10	0.57	750
2	20	1.14	3480
3	30	1.71	5355
4	40	2.28	6320
5	50	2.85	7065
6	60	3.42	7565
7	70	3.99	8125
8	80	4.56	8320
9	90	5.13	8720
10	100	5.7	8950

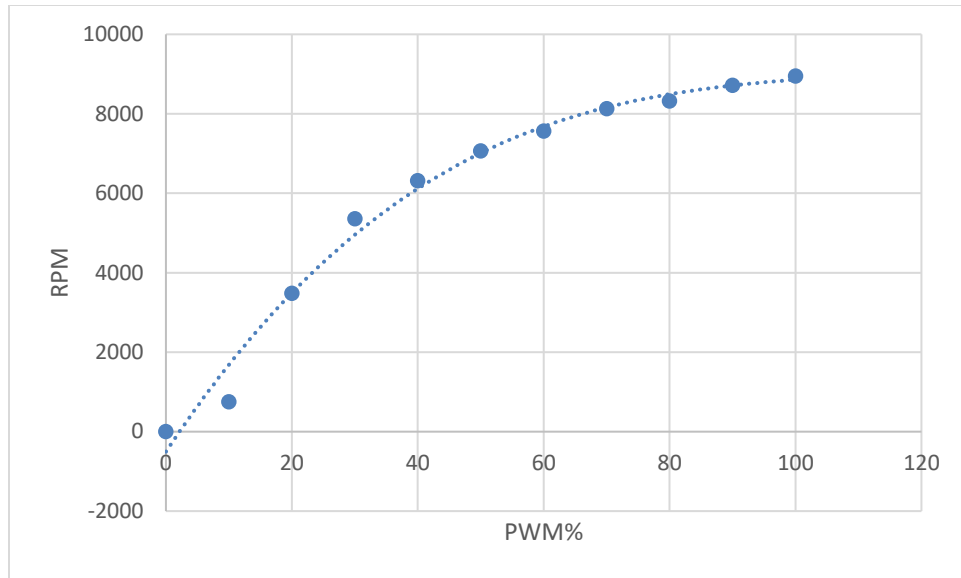


Figure 2. Measured RPM vs PWM%

The graph and table measuring the relationship between PWM% and RPM also makes sense to be an increase since we are using more voltage from the battery upon creating a larger duty cycle. If you examine the square wave results from the data visualizer, you can see that the off time decreases from the 16kHz signal, upon increasing the on time of the signal. This means that in order to keep a 16kHz signal, the RPM will stabilize over time because there is less of an increase for the signal to be on, and it will limit the motor from rotating any faster, unless the wave is changed from 16kHz.

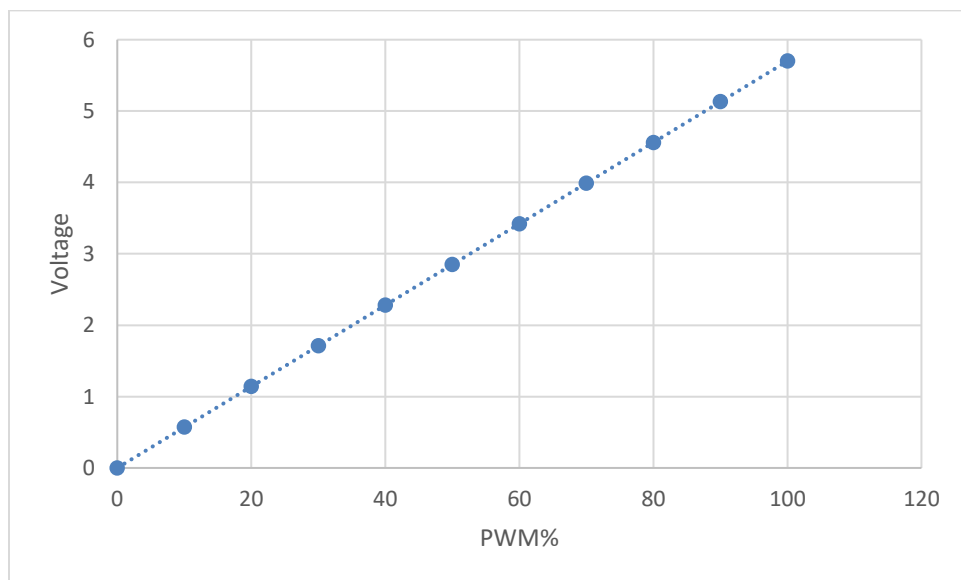


Figure 3. Calculated Voltage vs PWM%

The table and graph relating voltage to PWM% makes sense since as the duty cycle is increasing, we are increasing the amount of time the signal is on, which is done by increasing the voltage which will also spin the motor faster.



Figure 4. OLED Example 1



Figure 5. OLED Example 2

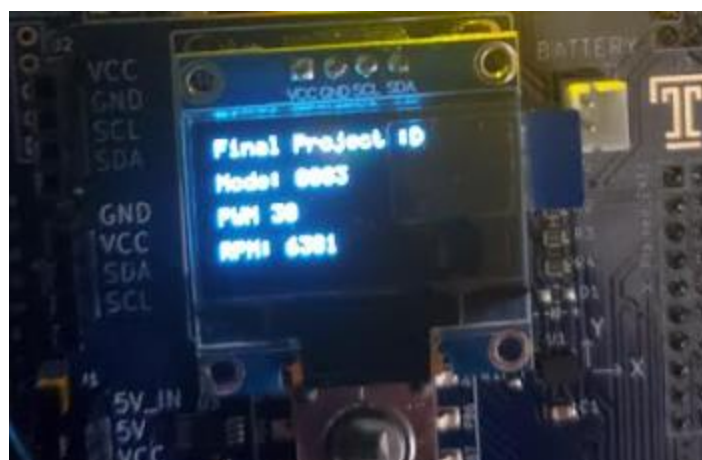


Figure 6. OLED Example 3

The three images above are three examples upon running the code and using the joystick to change the mode and duty cycle, which then spins the motor and can be examined on the data visualizer, as seen above in the last column of table 1.

4. Conclusion

Overall, we were able to complete the final project and successfully display the results from the motor onto the OLED display, by using the joystick as an input. The project required an understanding of hardware interrupts combined primarily with how PWM signals are achieved through the use of timers. One concept this project solidified for me is how running a motor is just dependent on manipulating the rate of voltage inputted into it.

5. Appendix

Full code with full comments

```
#include <avr/io.h>

#include <avr/io.h>

#define F_CPU 16000 //for a 16kHz squarewave

#include <avr/io.h>

#include <avr/interrupt.h>

#include "i2c.h"

#include "SSD1306.h"


#define up_count PB5 //pin corresponding to "up" position of RHC button

#define down_count PB7 //pin corresponding to "down" position of RHC button

#define DCmotor PE2 //pin to connect motor interrupt to PB2

#define SensorPin PD2 //opto-interrupter sensor pin


volatile char dutyCycle = 0; //global variable containing selected motor speed value

volatile char lastInput = 0; //variable to store the last state of PORTB

int count_int = 0; //set pwm value to 0 (no pwm) initially

volatile int counter = 0; //track how many times timer0 has overflowed

volatile int Sensor_counter=0; //count of opto-interrupter reading

volatile double RPM=0; //count of opto-interrupter reading
```

```

void OLED_display(){ //this will update the OLED display with the most current info

    OLED_SetCursor(0, 0);

    OLED_Printf("Final Project :D");

    OLED_SetCursor(2, 0); //displaying mode which is equal to the joystick position counter (0-10)

    OLED_Printf("Mode: ");

    OLED_DisplayNumber(C_DECIMAL_U8,count_int,2);

    OLED_SetCursor(4, 0); //displaying PWM%

    OLED_Printf("PWM%: ");

    OLED_DisplayNumber(C_DECIMAL_U8,dutyCycle,3);

    OLED_SetCursor(6, 0); //displaying RPM

    OLED_Printf("RPM: ");

    OLED_DisplayNumber(C_DECIMAL_U8,RPM,4);

}

ISR(TIMER0_COMPA_vect){ //ISR for when timer0 overflows

    counter++; //increase counter variable

    int k=2; //number of sensor reading per 1 revolution, if you have 2 wheel opening, change to 2

    if(counter >= 200){ //when counter equals 200

        //calculate RPM with eqn
        ((trig_counts*(milliseconds_in_1_second/(milliseconds_for_1_clock_cycle*number_of_loop_iteerations)))/2_triggers_in_1_revolution)

        RPM = ((Sensor_counter/((256*0.0625*0.000001*1024)*200))*60)/k; //RPM=((sensor_counter/((timer value)*number_of_loop))*seconds for 1 minute)/number_of_sensor_counts_per_1_revolution);

        Sensor_counter = 0; //reset trigger_count variable

        counter = 0; //reset timer0 counter variable

    }

}

ISR(INT0_vect, ISR_BLOCK){ //ISR for when PD2 detects a falling edge

```

```
    Sensor_counter++;    //increase trigger counter variable
}
```

```
double dc_adj=0.1; //duty cycle adjusting parameter
```

```
ISR(TIMER1_COMPA_vect){    //ISR for when timer1A interrupts
    if(count_int==0)
        PORTE=0x00;
    else
        PORTE &= ~(1<<DCmotor);    //turn off PE2 pin
}
```

```
ISR(TIMER1_COMPB_vect){    //ISR for when timer1B interrupts
    if(count_int==0)
        PORTE=0x00;
    else
        PORTE |= (1<<DCmotor);    //turn on PE2 pin
}
```

```
int main(void){
    //set data direction of used ports
    DDRB = 0x00; //set port b pins to inputs
    PORTB = 0xA0; //set port b pins to have pull-up resistors
    OLED_Init(); //INITIALIZE THE OLED
    OLED_Clear(); //CLEAR THE DISPLAY

    //configuration
    DDRE |= (1<<DCmotor); //set motor pin to output
    OCR1A=0xFFFF; //full speed: On-time of PWM
}
```

```

OCR1B=0xFFFF;

// initialize timer1

TCCR1A |= (1<<COM1A0)|(1<<COM1B0); //enable timer1a and timer1b comparison
TCCR1B |= (1<<CS10); //enable timer1 with no pre-scaler
TIMSK1 |= (1<<OCIE1A)|(1<<OCIE1B); //enable timer1A and timer1B interrupts
DDRD &= ~(1<<SensorPin); //set trigger pin to be an input
PORTD |= (1<<SensorPin); //Set trigger pin to have a pull-up resistor


//initialize timer0

TCCR0A |= (1<<WGM01); //set timer0 to be CTC
TCCR0B |= (1<<CS00)|(1<<CS02); //set timer0 to a 1024 pre-scaler
TIMSK0 |= (1<<OCIE0A); //set timer0A to have an interrupt
OCR0A = 0xff; //set timer0A to interrupt when it reaches 0xff


//initialize INT0 (PD2)

EIMSK |= (1<<INT0); //enable int0 interrupt
EICRA |= (1<<ISC01); //set int0 to falling edge


sei();


while (1) { //infinite loop
char joystick = (0b11110000&PINB); //get high 4 bits of PORTB (contains up and down button)

joystick = joystick^0xf0; //invert value of of up and down button (Pullup resistors pull pins high)

if(lastInput != joystick){ //check if PORTB has been updated

if(joystick&(1<<up_count)){ //check for up button being pressed

if(count_int < 10) //if the PWM value is less than 10,

count_int++; //YES: increment it

}

}
}

```



```

if(joystick&(1<<down_count)){ //check for down button being pressed

    if(count_int > 0) //if the PWM value is more than 0,

        count_int--; //NO: decrement it

    }

    dutyCycle = count_int*10;//update dutyCycle variable for OLED display

    lastInput = joystick; //keep the latest PORTB input

}

OCR1A = 0xFFFF* (dc_adj*dutyCycle)/10; //Duty Cycle LOW Activation, divide by 100 to get
Percentage (dutyCycle/10 = count(Mode)*dc_adj)

OLED_display(); //update OLED display

}
}

```

Part II. Performance

Video

Zen made the following video which showcases the OLED display properly updating the PWM% upon using the joystick, which also triggers the “mode”, or counter 0-10 to show which position the joystick is in. From this, the motor spins based upon the PWM% and the optical sensor calculates the RPM value which gets displayed on the last line of the OLED display, you can also see the data visualizer increasing each PWM signal when the duty cycle is getting changed. Further information about the PWM and RPM results can be observed in the results and discussion section earlier in the report.

Youtube Link to Hardware Demo:

https://www.youtube.com/watch?v=VpjX7ROkv4w&ab_channel=ZenonMatychak

ECE3613 Processor System Laboratory Rubric

Final Project

Section: 001 / 002

Name: _____

Report Section	Part	Contents	Full Points	Earned Points	Comment
Cover Page			5		Lab report Cover page
Introduction			15		Objective (5pts) and background information(10)
Procedure		Flowchart	10		
		Method	30		<ul style="list-style-type: none"> Complete description of your code design and key code section explanation (breakdown each section and explain) –include important instructions and functions such as timer, interrupt, port operations so on (20pts) Demonstrate the timer calculation and interrupt setup (10pts) Note: Do not put the entire code in here.
		Hardware Design	20		<ul style="list-style-type: none"> Complete description of your hardware system design and components specification
Result	I	Result	30		<ul style="list-style-type: none"> Joystick control result to generate different duty cycles (%), use Data Visualizer reading for the duty cycles – show 0% to 100 % duty cycle cases up and down, include Table 1 (10pts), Graph 1 result (10pts) and explain the result (10pts)
	II	Result	40		<ul style="list-style-type: none"> Optical Sensor Reading to count the revolutions for DC motor and computation for RPM – you must include Table 2 (10pts), Graph 2: Duty Cycles vs. RPM (10pts), and explanation of the result (10pts) OLED readings for the RPM or RPS – select 3 examples of duty cycles and include the pictures of the reading with description (10pts)
Discussion		Evaluation	10		Result Verification and Analysis of the result I : Compare the expected values and readings (error % from the expected)

		Evaluation	10		Result Verification and Analysis of the result II : Compare the expected values and readings (error % from the expected)
Conclusion			10		Summary of the lab final project
Appendix		Code	10		Code with full comments
Video Link			10		Show the full operation – Include all duty cycles running operation with the joystick operation (up and down both)
Total			200		