Temple University

College of Engineering

Department of Electrical and Computer Engineering (ECE)

Student Lab Report Cover Page

Course Number : 3613

Course Section : 001 / 002

Experiment # : Lab #12 Final

Student Name (print) : Von Kaukeano

TUid# : 915596703

Date : 11/21/19

Grade : _____ **/100**

TA Name : Sung Choi

Introduction

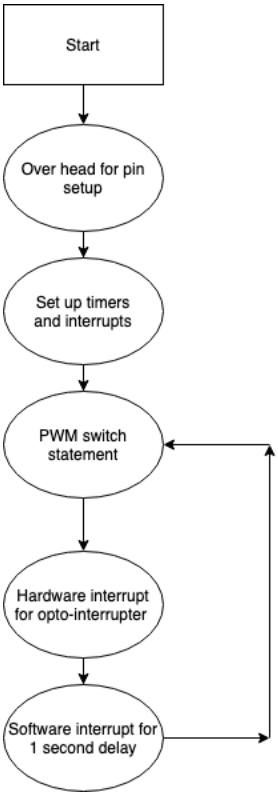
This is the lab final for Processors Lab. The goal is to write code for a motor controller using variable PWM and have an opto-interrupter measure the revolutions of the drive shaft. The operation of the opto-interrupter is to provide a hardware interrupt whenever the sensor is flagged on a rising edge by the wheel attached to the drive shaft of the motor. There is a software interrupt created by a timer, this allows to count the times it passes through and another every second to add the total number it counts to calculate RPM. The RPM, duty cycle percentage, and mode based on the switches then must be output to the LCD screen

Procedure

The frame for the motor and opto-interrupter was kindly given to us as a 3D printed model which prevented us from creating one from scratch. This helps prevent errors in cycles of the wheel. After receiving the frame the code can be broken down into three major parts.

- Hardware interrupt for opto-interrupter and count pulses.
- Timer to create variable PWM duty cycle based on switches thrown.
- Software interrupt to display the count total on the screen using 1 second timer.

A case statement was the best choice to select the duty cycle of the PWM. This would allow for use of the switches on port B for a binary 0-10 input for each of the specified duty cycles and constantly poll them.



```
	Z = 0X0F; //Masking
	DDRB = 0X00; //PORTB INPUT
	PORTB = 0XFF; // PULLUP RESISTORS ACTIVATED
	DDRE |= (1<<PINE2);
	DDRD |= (1<<PIND7);
	PORTD |= (1<<PIND7);

	// Initialize PWM to 50% duty cycle round(255/127)
	// Initialize counter
	TCNT1 = 0;
	OCR2A = 0;
	// Non-inverting Fast PWM Mode 7
	// START the PWM timer with 64 Pre-scaler
	// I/O Clock @ 16 MHz
	//
	TCCR1B |= (1 << CS11);
	TCCR2A |= (1 << WGM21)|(1<<WGM20)|(1 << COM2A1);
	TCCR2B |= (1<<CS42)|(1 << CS40);
```

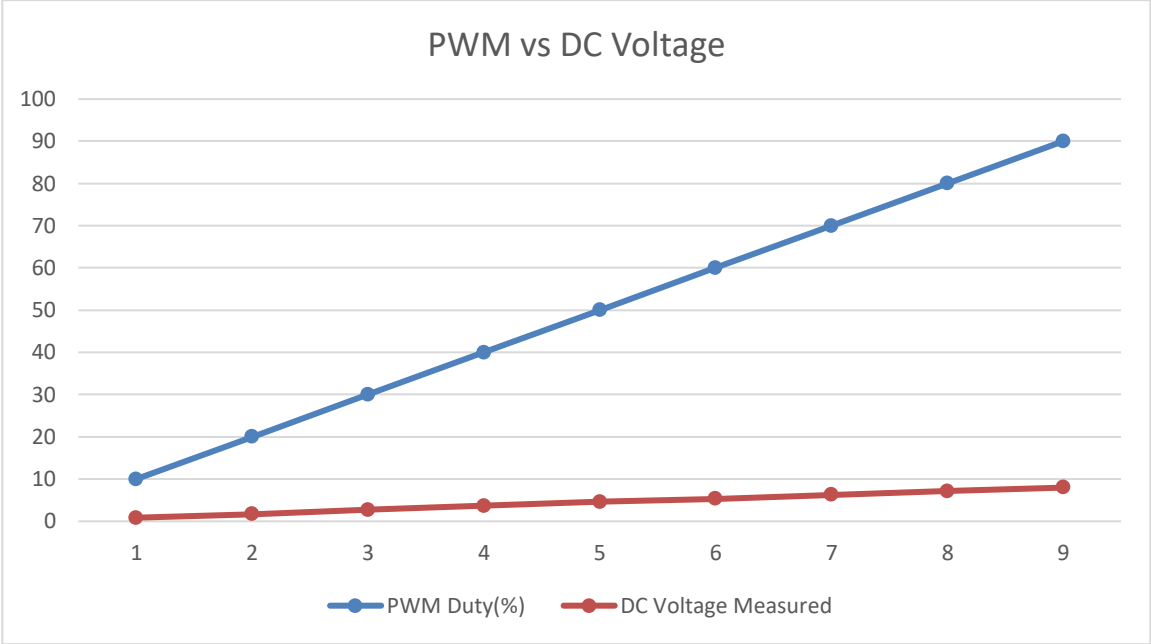
This was the most important piece to the code. I used the clk signal to setup fast pwm mode. The fast pwm mode was nice because .1 * 255 was 10% and .9 * 255 was 90%. This was convenient in setting up the polling for the switches. The other important pieces were the interrupts. The fast pwm is output to PD7 according to the datasheet. This means that there must be a wire going from PD7 to PE2 to run the code.******

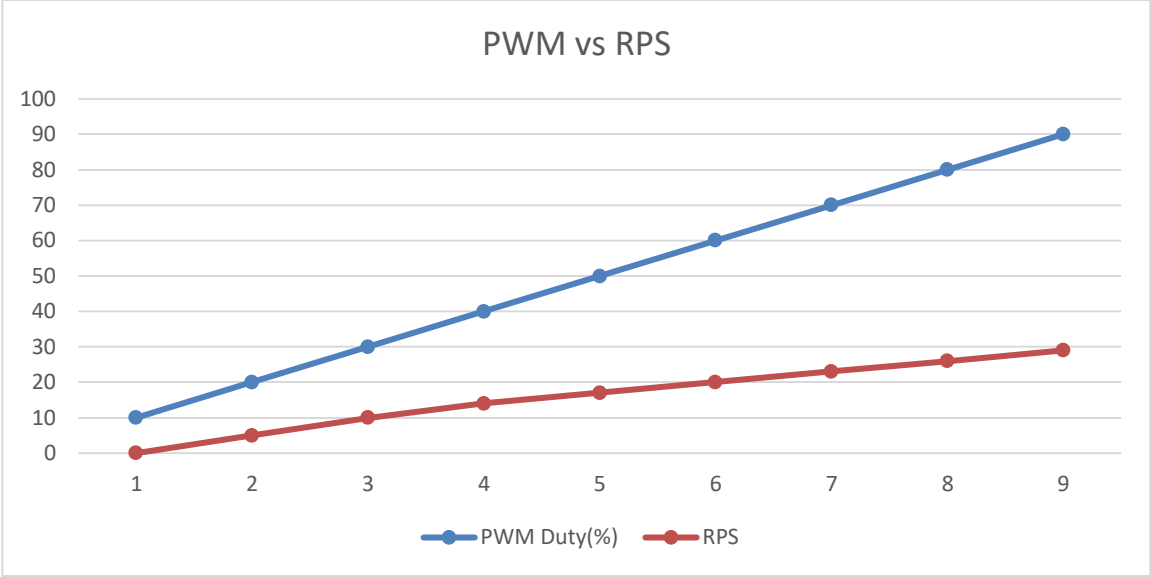
```
// External Interrupt on Port D Pin 2
ISR(INT0_vect, ISR_BLOCK){
	COUNTER++;// Increment counter when pin changes
}

ISR(TIMER1_OVF_vect){
	OVERFLOW++;
	COUNTER = (COUNTER/2);
	if(OVERFLOW >=33){
		OLED_SetCursor(4, 0);//set the cursor position to (0, 0)
		OLED_Printf("RPS:");
		OLED_DisplayNumber(C_DECIMAL_U8,COUNTER,3); //Print out some text
		OVERFLOW = 0;
		COUNTER=0;
	}
}
```

The first one counts the number of times the octo-interrupter reads a rising edge. The second one is an overflow timer that displays the count after 33 times which was approximately a 1 second delay.

Result and analysis

Switch	PWM Duty(%)	DC Voltage Calc	DC Voltage Measured	RPS
1	10	0.9	0.86	0
2	20	1.8	1.71	5
3	30	2.7	2.73	10
4	40	3.6	3.68	14
5	50	4.5	4.62	17
6	60	5.4	5.29	20
7	70	6.3	6.26	23
8	80	7.2	7.11	26
9	90	8.1	7.98	29





****BE ADVISED***

If you want to run this code you must hardwire PD7 to PE2. PE2 is hardwired directly to the motor. PD7 is the pin that the Fast PWM is output to according to the datasheet. If you are running the code to check for correctness, this must be wired.

https://drive.google.com/file/d/1N3AjFCsrtLTavWUdxgUOz6SZltg_1wJe/view?usp=sharing

Discussion and Conclusion

This lab was extremely helpful in understanding timers and interrupts. It was difficult to get to work, but reading the datasheet and running examples have helped me tremendously. The PWM percentages were the hardest part because the way we learned timers in class were not sufficient enough for practical use. I explored delays, timers, and fast PWM mode to figure this lab out. Delays do not work well with the LCD but work with PWM.

Code (if required)

```
#include <avr/io.h>
#include "avr/iom324pb.h"
#define F_CPU 16000000UL
#include <util/delay.h>
#include <avr/interrupt.h>
#include "SSD1306.h"
#include "i2c.h"
```

```
unsigned char Z;
unsigned char INPUT;
volatile int COUNTER = 0;
volatile int OVERFLOW = 0;

int main(void)
{
	OLED_Init();  //initialize the OLED
	_delay_ms(1);
	OLED_Clear(); //clear the display (for good measure)

	Z = 0X0F; //Masking
	DDRB = 0X00; //PORTB INPUT
	PORTB = 0XFF; // PULLUP RESISTORS ACTIVATED
	DDRE |= (1<<PINE2);
	DDRD |= (1<<PIND7);
	PORTD |= (1<<PIND7);

	DDRD |= (0<<PORTD2);// Port D input
	PORTD |= (1<<PORTD2);// Set Pull up resistor PD2 (INT0)
	EIMSK=(1<<INT0);
	TIMSK1 = (1 << TOIE1);
	EICRA=0X03; //INT0 WILL ACTIVATE Rising EDGE TRIGGER

	// Initialize PWM to 50% duty cycle round(255/127)
	// Initialize counter
	TCNT1 = 0;
	OCR2A = 0;
	// Non-inverting Fast PWM Mode 7
	// START the PWM timer with 64 Pre-scaler
	// I/O Clock @ 16 MHz
	//
	TCCR1B |= (1 << CS11);
	TCCR2A |= (1 << WGM21)|(1<<WGM20)|(1 << COM2A1);
	TCCR2B |= (1<<CS42)|(1 << CS40);

	sei();// Set Interrupt Flag
	while (1){

		INPUT = PINB & Z;
		switch(INPUT){

			case(0x01):
			OCR2A = 25;
			break;

			case(0x02):
			OCR2A = 51;
			break;
```

```
                        case(0x03):
                        OCR2A = 76;
                        break;

                        case(0x04):
                        OCR2A = 102;
                        break;

                        case(0x05):
                        OCR2A = 127;
                        break;

                        case(0x06):
                        OCR2A = 153;
                        break;

                        case(0x07):
                        OCR2A = 178;
                        break;

                        case(0x08):
                        OCR2A = 204;
                        break;

                        case(0x09):
                        OCR2A = 229;
                        break;

                        default:
                        OCR2A = 0;
                }

        }
}
// External Interrupt on Port D Pin 2
ISR(INT0_vect, ISR_BLOCK){
        COUNTER++;// Increment counter when pin changes
}

ISR(TIMER1_OVF_vect){
        OVERFLOW++;
        COUNTER = (COUNTER/2);
        if(OVERFLOW >=33){
                OLED_SetCursor(4, 0);//set the cursor position to (0, 0)
                OLED_Printf("RPS:");
                OLED_DisplayNumber(C_DECIMAL_U8,COUNTER,3); //Print out some text
                OVERFLOW = 0;
                COUNTER=0;
        }
```

```
}
```