H62EDQ: SR Motor Project Report

Electrical and Electronic Design Project

Written by Ryan A, Nur' Hazirah Haji Abdul Rahman and Liu Chi

Table of Contents

1. Introduction

The project is about designing and constructing a Switch Reluctance Motor drive that will control the Motor. The designing part includes elements such as power electronics, analogue electronics, digital electronics, microprocessor interfacing and programming. This project builds on the objectives that have been laid out beforehand. The first three objectives serve the most important part of the project, which is to make the motor spin. After that, then one can proceed with the next objective. The objectives of the project in their order are as follows; Construct a method of limiting current into the stator windings, provide user interface for the drive, make the motor spin under simple software control with fix firing angles, detect and control the direction of the motor spin, make motor spin while varying firing angles, control torque produced, make the firing angles varied automatically as function of the speed to provide optimum torque, and lastly regulating speed automatically via a close loop system.

There are six parts to this report including the appendix. The first section will talk on the theory and the concepts that have been used when designing the gate drive. The second section touches on the hardware of the system such as design methods, calculations and component selections. Section 3 on the other hand is about the software and programming that have been implement in order to achieve the objectives required. The results of the testing and result of the final design can be found in section 4 where it will also include how some of the objectives were achieved. Lastly section 5 will be discussing on the project's outcomes and conclusions.

2. Theory

2.1 Current limit

The current limit circuits are based on a $0.1\,\Omega$ resistor and two significant chips i.e. LM393 comparator and LM555 timer. $0.1\,\Omega$ resistor is used to measure the current whereas LM393 is used to check the magnitude of the feedback current. NE555 timer is used to provide a delay. The delay forces the convertor's MOSFET off in order to decrease the current.

2.1.1 The principle of comparator

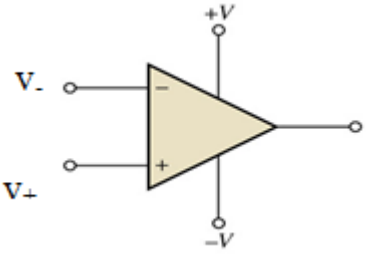


Figure 1 : Comparator

$v_0 = G(v_+ - v_-)$, G is the gain of the comparator.

$If\ V_+ > V_-,\ V_O = +V \quad If\ V_- > V_+,\ V_O = -V$

If the feedback current is larger than a certain value, the comparator will output signal to trigger the LM555 to cause delay.

2.1.2 The principle of 555 timer

In this project, monostable operation of 555 timer is used. **Figure 2** shows the circuitry of the monostable operation. In this operation, a negative trigger pulse of less than 1/3 VCC is inputted to pin 2, the output will be high. The voltage across the capacitor then increases exponentially for a period of t = 1.1 RAC, at the end of which time the voltage will equal to 2/3 VCC. The output will be in a low state. This procedure is illustrated by **Figure 3.**

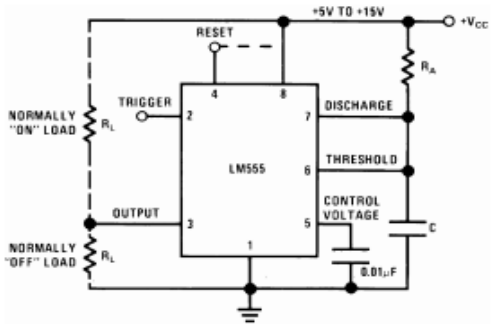


Figure 2: 555 timer circuitry

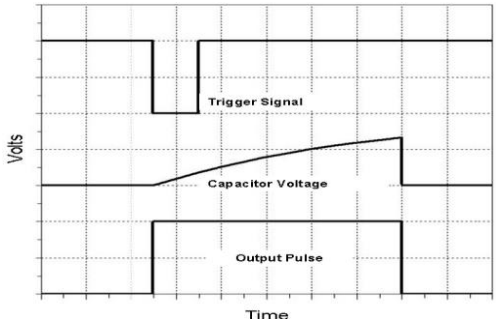


Figure 3: Timing diagram of the monostable operation

2.2 Converter operation

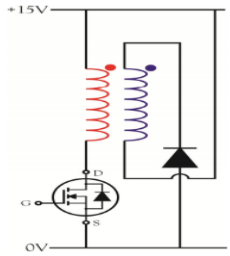


Figure 4: Converter

Figure 4 is the schematic of the designed converter to drive the motor. When MOSFET is switch ON, the primary inductor will save the energy. When the MOSFET is switched OFF, the secondary inductor will release its energy to the power supply. The advantages of this design are that the construction is relatively simple with high efficiency and the current is easy to access and measure. However, the disadvantage of this design is that the motor could not rotate as fast as others, because it needs time to save the energy.

2.3 Motor control methods

Basically, the motor speed and torque are controlled by firing angles ($\theta_{off}, \theta_{on}$) and the torque is also controlled by the current. The method for motor spinning is to energise the stator winding at the appropriate angle (θ_{on}) and for a certain duration ($\theta_{off} - \theta_{on}$). PIC will provide the different frequencies of square wave by setting the two firing angles. Combined with current limit signals and logic gates, the final signal is generated to drive the MOSFET.

2.4 Encoder operation

The AEDB-9140 are emitter/detector modules. Coupled with a codewheel, these modules translate the rotary motion of a shaft into a three channel digital output. Channel A is in quadrature with channel B (90 degrees out of phase) Channel A and channel B each produce 500 pulses per cycle. The Z pulse (which counts one full cycle) is used to decode the position of the rotor with respect to the stator. When the codewheel rotates in the clockwise direction viewing from top of the module, channel A will lead channel B. If the codewheel rotates in the opposite direction, channel B will lead channel A.

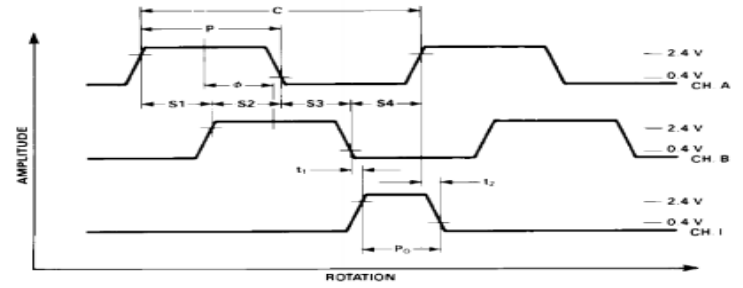


Figure 5: Three channels output waveforms

2.5 Speed calculation

One method to calculate the speed is to fix a certain period and record how many cycles the motor has rotated in this period. It could be illustrated by this formula: $\omega = \frac{count\ of\ cycles}{time}$. This method could be implemented by the software.

The other method is to fix the number of cycles and record the time. This method could be implemented by obtaining encoder A waveform frequency from oscilloscope. Every cycle Channel A will produce 500 pulses. The formula is given by: $\omega = \frac{1}{500} \times frequency$

2.6 Simulation of the 5V supply circuits

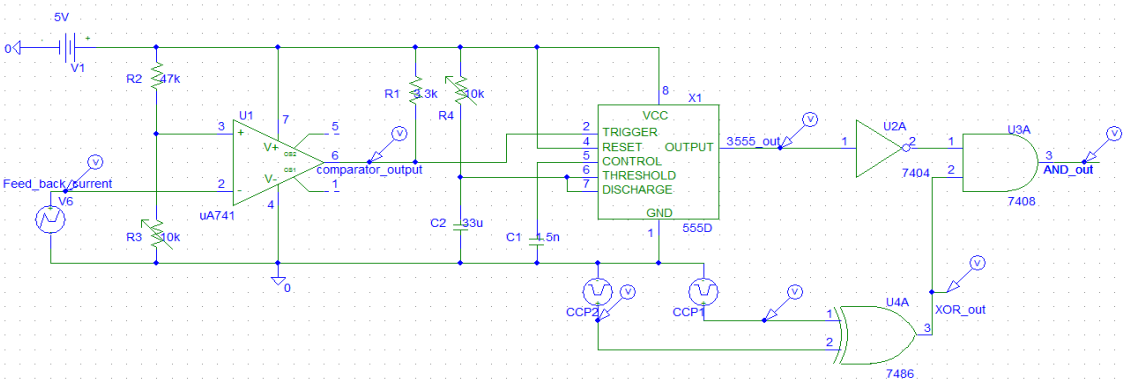


Figure 6: The schematic of the simulated 5V system.

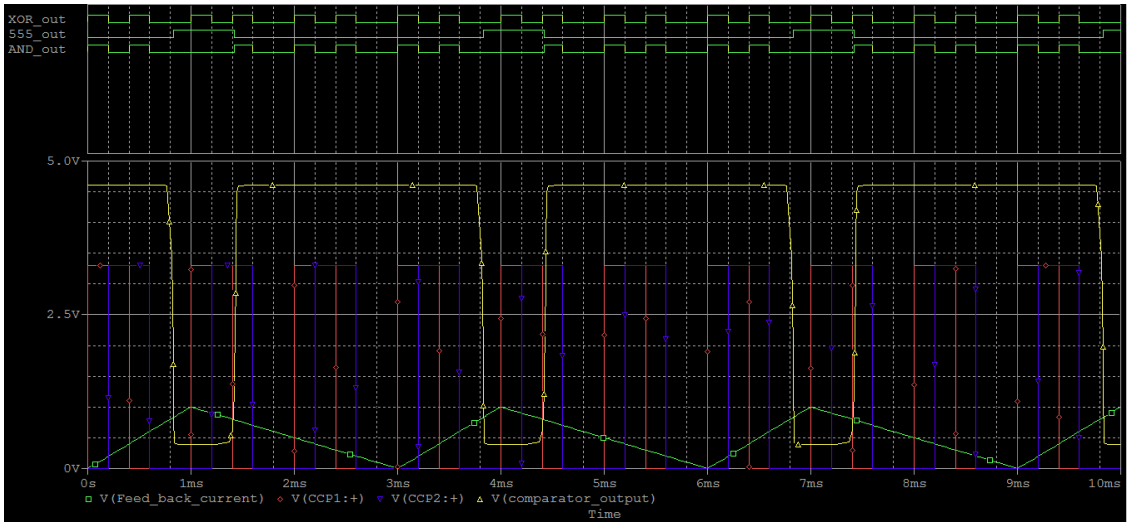


Figure 7: The result of simulation of the 5V system.

3. Hardware

The hardware of the project consisted of several components, all which play a vital role in the SR Motor Drive. The components involved are; Glue Logic, Gate Drive, Current Limit detector, a Delay and a Power Electronic Converter. The figure below shows that the 15V power supply has a decoupling capacitor with a value of 1000µF. It also shows that the 5V power supply has another two capacitors for decoupling, both with values of 100nF and 47µF. In addition, all the chips have a 100nF decoupling capacitor connected with their Vcc and ground. The red LED with the 1kΩ resistor indicates whether the power is on or off. In order for the converter to carry large amounts of current, we reinforced the tracks on the strip board and used double wires. Another red LED was used so that it could show us that the current limiter is functioning.

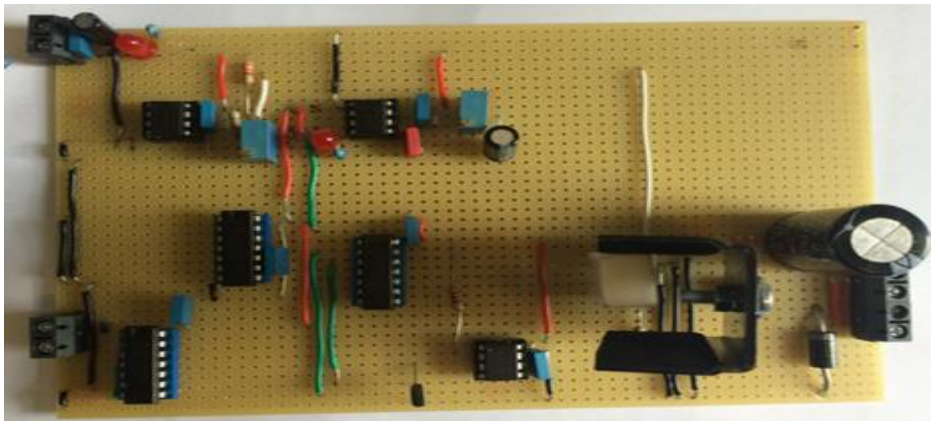


Figure 8: The layout of the strip board for the system

3.1 The Glue Logic

The glue logic consists of an XOR gate (SN74LS86), a NOT gate (SN74LS04) and an AND gate (SN74LS08). The purpose of the glue logic is to combine the signals from the PIC board and the delay signal from the Im555 timer. The delay signal from the Im555 timer chip goes through the NOT gate and into the AND gate while the signals CCP1 and CCP2 go into the XOR gate. **Table 1** below shows the truth table of the whole operation.

Table 1: The truth table of glue

CCP 1	CCP 2	XOR O/P	555	NOT 555	O/P
0	0	0	0	1	0
0	1	1	1	0	0
1	0	1	0	1	1
1	1	0	1	0	0

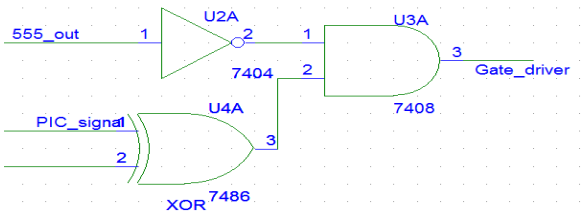


Figure 9: The schematics of glue logic

As you can see from the **table 1**, we require the output to be a 1 when both the signal from the XOR output and the NOT 555 is 1.

3.2 The Gate Driver

For the Gate Driver, a 100Ω resistor is placed in series with PIN 7. The reason for the 100Ω resistor is because of the following schematic:

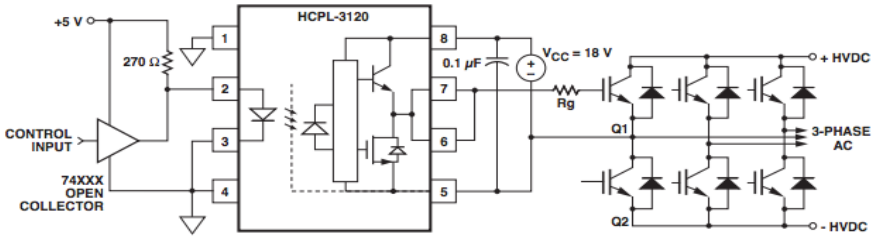


Figure 10: Design of the gate driver

The 100Ω resistor is acting like the resistor Rg, which is the output resistor. The input resistor, 270Ω, is to reduce the current entering the gate driver. Unlike this schematic, for PIN 3, our circuit has a MOSFET, in order to act like a switch. We also have a decoupling capacitor with value of 0.1µF connected between pins 8 and 5.

3.3 The Current Limit and Delay

The Current Limit and Delay consists of a comparator and a Im555 timer chip. Current from the coil, I(coil), from the Power Electric converter is connected to the inverting input of the comparator after a 0.1Ω resistor. The maximum current is 8A. In this case, the non-inverting input of the comparator is designed to be $V_+ = 0.1 \times 8 = 0.8V$. The non-inverting input of the comparator is connected to a potential divider with a 5V dc source. One fixed resistor of 47kΩ and a variable resistor of 10kΩ to give us a voltage of 0.8V. In this case, the actual value of the variable resistor should be$5 \times \frac{R}{R+47k} = 0.8V \rightarrow R = 8.95k\Omega$. The Im555 timer chip designed to have a variable resistor of 10kΩ and a 33µF capacitor, this is to give a time period of T = 1.1RC = 1.1 × 10k × 33uF = 363ms as a maximum delay.

3.4 Push Switches and LEDs

For the secondary board, we have decided to add push switches in order to maneuver through our user interface. For High-level detections and Low-level detections we have the following schematics:-

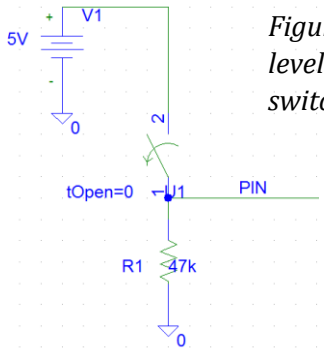
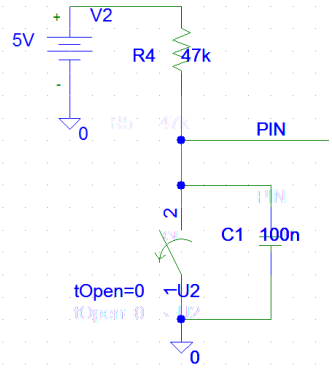


Figure 11: High-level Detection switch

Figure 12: Low-level Detection switch



For the Low level schematic, a capacitor is added in parallel with the switch to enable de-bouncing. This gives us a time period of T = RC = 47kΩ × 100nF = 4.7ms to de-bounce the signals into PIC Pins.

The purposes of the LEDs are to indicate whether the motor is running and to detect the direction of the motor.

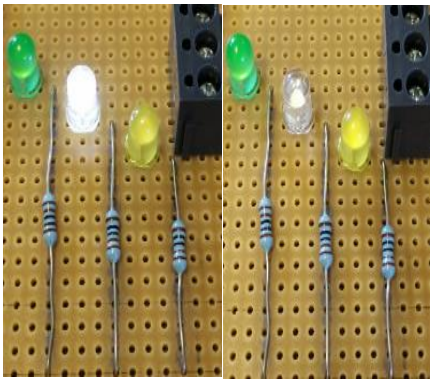


Figure 12: LED

Each LED from the **Figure 12** is connected to 1kΩ resistors to prevent large current from entering. In **Figure 12**, the green LED is ON for both figures, indicates that the motor is spinning. The white LED which is switched ON in the left figure indicates that the motor is spinning clockwise while the yellow light that is switched ON in the right figure indicates that the motor is spinning in anti-clockwise.

4. Software

Menu:

There are five parts to our menu. The first part is the main menu which is designed to show the current status of the motor and instructions of how to control the motor. The second part is designed to set the on angle. The third part is designed to set the off angle. The fourth part is designed to set the speed. The fifth part is designed to set the direction. The menu is controlled by 6 buttons and these buttons are connected with the pins and interrupts in order to control the motor intelligently. An integer variable "menu" is used to represent our modes. OLED Functions are used to display the information and messages.

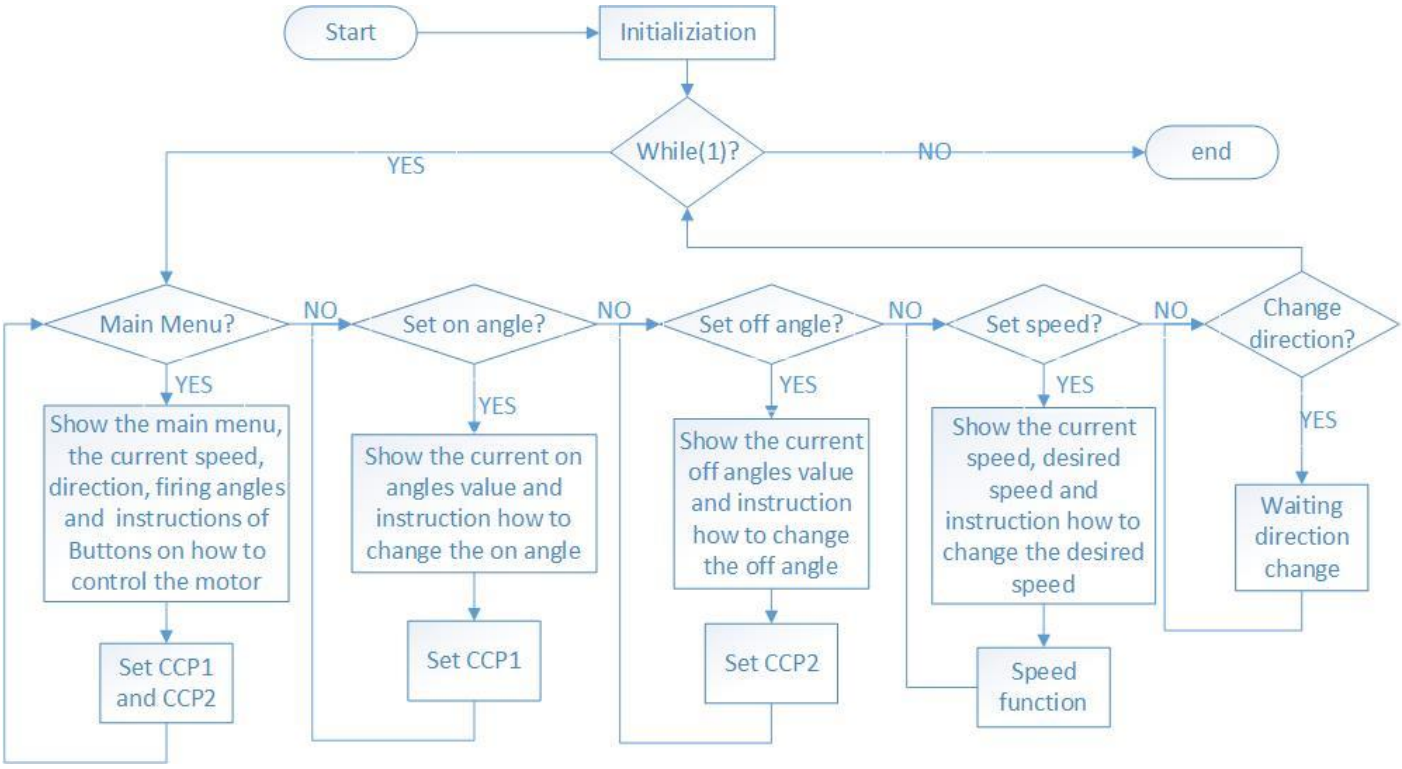


Figure 13: The flowchart of the menu

Table 2: The function of buttons of controlling the motor

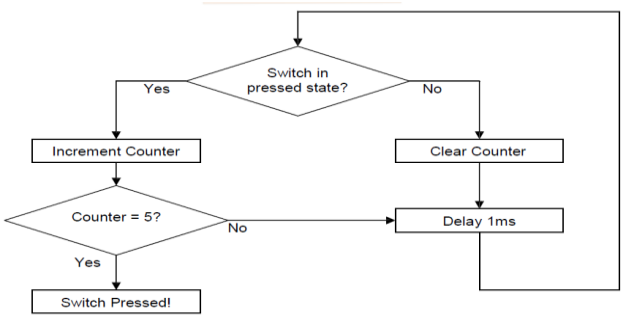
Menu	Button1(RB1)	Button 2(RB2)	Button3(RB4)	Button4(RB5)	Button5(RB6)	Button6(RB7)
Main menu	Change direction	Set +/-	Set on angle	Set off angle	Set speed	null
Set on angle	Change direction	Set +/-	On angle +/-1	On angle +/-10	Set off angle	Back to main menu
Set off angle	Change direction	Set +/-	Off angle+/-1	Off angle+/-10	Set on angle	Back to main menu
Set speed	Change direction	Set +/-	Speed +/-1	Speed +/-10	Speed +/-100	Back to main menu
Change direction	Change direction	Set +/-	null	null	null	null

A) PORT of buttons:

PORTB set as inputs, it is used to connect the buttons. The 6 buttons are connected with RB1, RB2, RB4, RB5, RB6, and RB7. The reason for using PORTB is that RB1 is connected with INT1 and RB2 is connected with INT2.

RB4~7 are connected with RBIF. These means when one of the voltage level of PORTB<7:4> PIN is changed, the interrupt will be triggered. In this case, for our switches' hardware it will save an OR gate. The INT1 and INT2 are set as high priority interrupts. RBIF is set as low priority interrupt. When the switches are pressed, the corresponding Pins will detect the voltage level changing and the interrupts thus its functions are triggered and implemented. Basic configurations of PORTB and Interrupts are shown in the appendix of InitializeSystem function.

B) Switch de-bouncing



In this project, switch de-bouncing is necessary. This method could be used in the low priority interrupt. For the high priority interrupt, it requires hardware de-bouncing.

Figure 14: the flow chart of switch de-bouncing

Motor spinning:

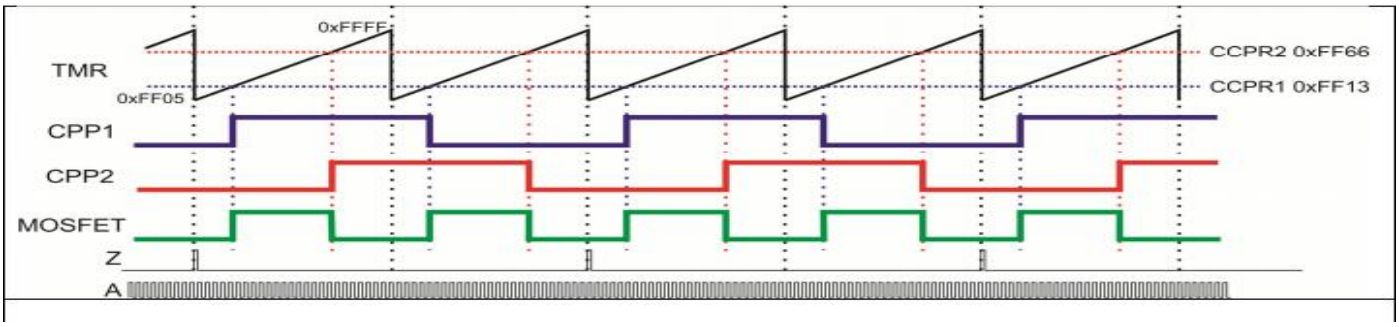


Figure 15: The timing diagram of the principles of CCP and Timer 1.

The CCP compare module and Timer 1 to generate the MOSFET signal. The principles of how the CCP and Timer 1 work are illustrated by **Figure 15**: Encoder A, connected with RC0 is used to as external clock for the timer 1. Encoder Z, connected with INT0, is used to correct the timer 0 settings when the motor passes to 0. The main principles of Timer 1 and CCP are that CCP1 and CCP2 will output square wave signals when the timer 1's value approaches to the setting values of CCP1 and CCP2 in a half cycle. RC0 and RB0 are set as input Configurations of TIMER1, CCP and INT0 are shown in the appendix. The calculation of how to transfer the firing angles to CCP are:

*CCPR2L = (off_angle*250/180)+5; CCPR1L = (on_angle*250/180)+5;// on_angle sets CCP1, off_angle sets CCP2*

In the TIMER1 interrupt and INT0 interrupt function, reset timer 1. Each cycle has 500 pulses of A. This means every half cycle (250 A pulses) and every cycle (500 pulses) reset timer1. The settings are in the appendix.

TMR1H = 0xFF; //TMR1=255 is 180 degree; TMR1L=0x05;//TMR1=5 is 0 degree

Motor speed calculation:

The method of speed calculation is to fix a certain period and record how many Z pulses occur in this period. The certain period is set by timer 0. An integer variable "count" is used to record the number of Z pulses.

Configurations of timer0 are shown in appendix T0CON = 0b00000101; // prescale 1:64;

Our PIC clock frequency are set to 16MHz.

The period=$65536 * \frac{1}{4M} * 64 = 1.048s$the speed=$\frac{COUNT*60}{1.048} rad/s$, to reduce the float calculation, $speed = 57 * count + count/4$. This calculation is produced in the timer0 interrupt. The accuracy of speed could be examined by the signal waveform of pulse A by oscilloscope.

Motor direction detection:

LATD6, LATD5 and LATD4 are used to represents the motor status. LATD6 represents the motor is running. LATD5 represents the motor direction is clockwise and the LATD4 represents anticlockwise.

TRISD=0; //Set PORTD as output; TRISA=0b00010000;// Set RA4 as input.

For motor running detection:

If the PIC captures the Z pulse, then in the INT0 interrupt, LATDbits.LATD6=1; It means the motor is running. PORTDbits.RD6 will output 3.3V. If the PIC detect the speed is 0, then in the timer0 interrupt, LATD=0; It means the motor is stopped. Set all the PORTD to 0.

For the direction detection:

If the PIC captures the Z pulse, then in the INT0 interrupt, and if the first captured pulse is B not A (B leads A):

if(PORTAbits.RA4==1)&&(PORTCbits.RC0==0)) {LATDbits.LATD5=1; LATDbits.LATD4=0;}

Turn on the LATD5, turn off the LATD4. It means the motor direction is clockwise. Otherwise turn off LATD5 and turn on LATD4. *LATDbits.LATD5=0; LATDbits.LATD4=1;* It means the motor direction is anticlockwise. This code could be checked by rotating the motor by hands.

Motor direction control:

The method of direction control is to cancel the reset of Timer 1 in the INT0 interrupt (Every cycle) when one switch is pressed. A "stop_flag" is used as a flag to cancel the reset stop_flag=1;. In this case, the motor would slow down and change the direction automatically. Wait for the motor slowing down. After detecting the direction is changed, recover the reset of Timer 1 in the INT0 interrupt(Every cycle) stop_flag=0;. The motor speeds up and rotates stably.

Torque control:

The software principles of torque control is to use DAC to generate 0-0.8V signal to the comparator to limit the current and use ADC to read the feedback current I. Measure the inductance and unaligned inductance (3mH) of the motor. By this formula: $T = \frac{1}{2} I^2 \frac{dL}{d\theta}$ to calculate the torque.

Angle advancement:

This could be worked out by the formula: $\theta\_advanced = L\_unaligned \times \frac{I_{desired}}{U} \times \omega\_actual$. In this case, U=15V, unaligned=3mH, speed could be calculated by software. I could be controlled by DAC. Using this formula, the angle will be figured out.

5. Results

In order to see how the speed of the motor responds to different firing angles, the motor was put on running and the off angles was varied while the on angle was fixed to 10°. The results has been tabulated in **table 3.**

On Angle (°)	Off Angle (°)	Firing angle (°)	Speed (rpm)
10	80	70	3721
10	70	60	3606
10	60	50	3549
10	50	40	3034
10	40	30	2461
10	20	10	916

Table 3: The results of firing angle vs speed

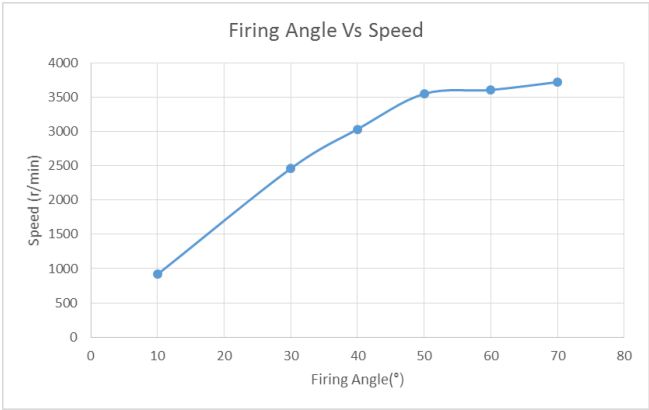


Figure 16: The graph of firing angles vs speed

It can be observe that speed is increasing with increasing firing angle. Initially the speed increases linearly with increasing firing angle however the result shows that for a larger firing angle, the increase in speed is relatively small.

Another test has also been carried out to see how the speed responds to change in delay. Delay is given by $1.1 \times C_{555timer} \times R_{VR}$. Therefore delay was varied by varying the variable resistor of the 555 timer circuit. The results are tabulated in the **table 4**

Table 4: Varying delay with corresponding speed

Capacitance (μF)	Resistance (Ω)	Delay (ms)	Firing Angle (°)	Speed(r/min)
33	270	9.8	60	3606
33	970	10.5	60	1087

It can be concluded that increasing delay time reduces the speed of the motor.

In the graphs below, all the waveform in yellow represent the waveform of the current generated by the motor measured by the 0.1Ω resistor.

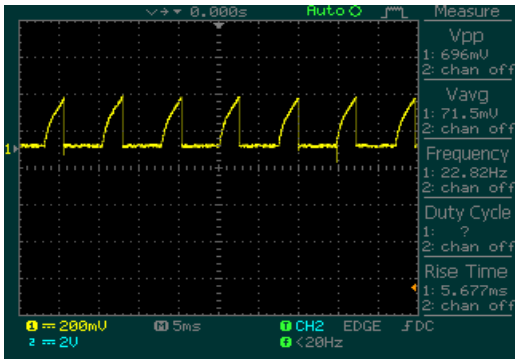


Figure 17: Waveform of the feedback current generate by the motor

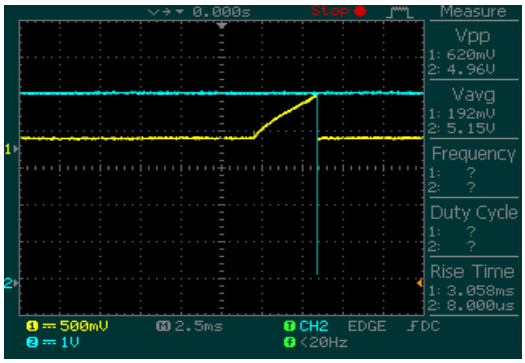


Figure 18 Channel 2 (blue) Output of the comparator at high speed

In **Figure 18**, feedback current (Channel 1) approximate value is =0.62/0.1=6.2A. The measurement may have errors because the Vpp value should be larger than 0.8V in order to produce the pulse as shown above.

For current exceeding 8A, the comparator will produce a negative output which is denoted by the vertical blue line. The flat region of the blue waveform represents the positive output (high level). This pulse signal will be past to the 555 timer and causes delay. At higher speed, these tiny pulses are unnoticeable.

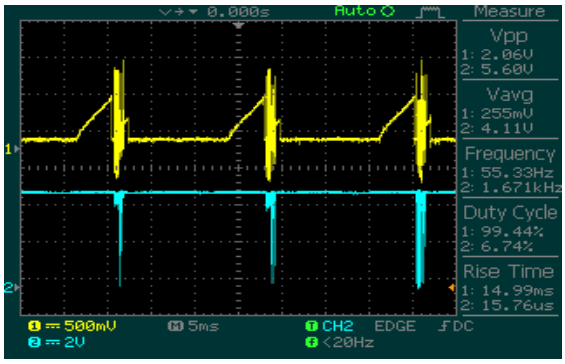


Figure 19: Channel 2(blue) is the output of comparator when the motor is forced to slow down by hands.

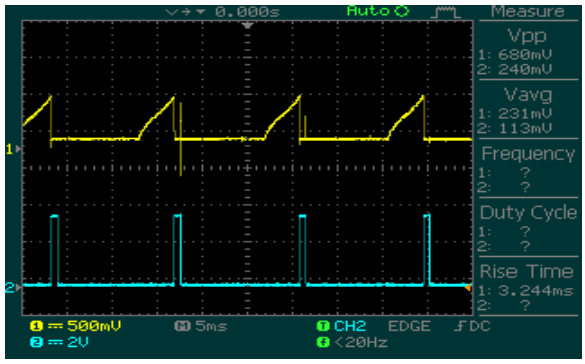


Figure 20: Channel 2(blue) output of the 555 timer

In **Figure 19** and **Figure 20**, the negative output is significant because the motor is slowing down and the current measured is much greater. The short pulse represents the delay and it can be observed that the delay is small. Theoretically, for feedback current larger than 8A, delay that occur will eventually causes a decrease in the feedback current.

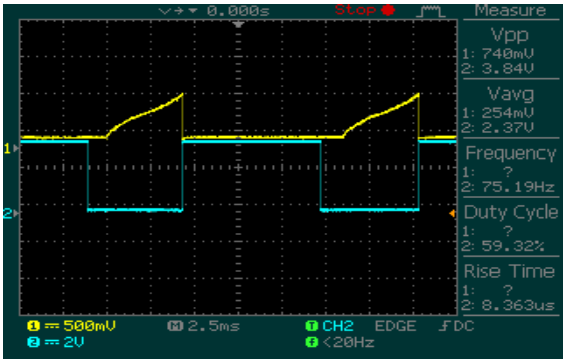


Figure 21: Channel 2(blue) output of the 555 timer with longer delay.

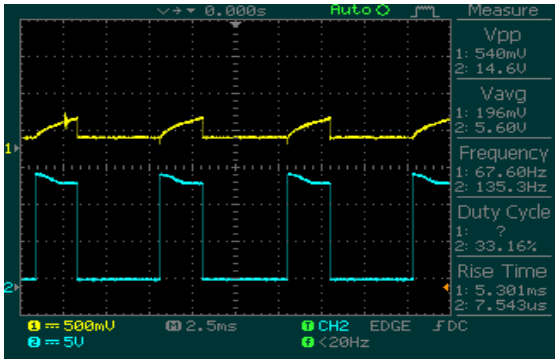


Figure 22: Channel 2(blue) output of gate driver.

In **Figure 21**, it can be observe that the delay is much longer than before. Theoretically, the delay time is given by the equation T=1.1RC. Thus value of the variable resistance of the timer is varied to vary the delay.

Theoretical Delay = 1.1 × 33μ × 210 = 7.62ms

Experimental Delay = (1/frequency) × duty_cycle = (1/75.19) × (59.32/100) = 7.89ms

There is only a slight difference between the theoretical and the experimental value of the delay and this is due to the imperfection in the connecting wires and the components.

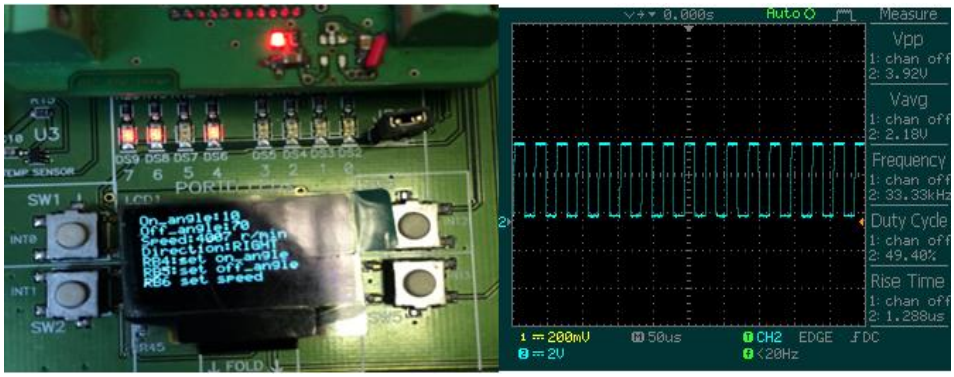


Figure 23: The examination of software code to calculate speed by using encoder A.

From the OLED screen in **Figure 23**, the calculation speed by the software is 4007 rad/min. The oscilloscope shows that the frequency of Channel A is 33.3kHz. Thus the calculation of the actual speed is $-\omega = \frac{33.33k \times 60}{500} =$ 4000 rad/min

This demonstrate that the calculation of the speed for the software is correct, though there is a 0.175% error. However, this error could be totally accepted.

From **Figure 19** to **Figure 21**, the waveforms of the comparator and 555 timer are satisfied the theory. It proves that our current limit circuits are working well. The objective 1 was achieved. Referring to the flowcharts, **Figure 12** and **Figure 23**, they indicates that our interface was working appropriately. The objective 2 was achieved. The motor was rotating stably by our highly reliable design of our software and hardware. The maximum speed that can be reach is about 4000 rpm shown in **Figure 23**. The objective 3 is completed successfully. For the objective 4, our LED showed the motor status correctly. In addition, by our advanced design of software, the motor could change the direction automatically. For the objective 5, firing angles and speed are varied intelligently by our well designed software. We would figure out the other three objectives if we have more time. However, principles and methods of how to work out the other three objectives are illustrated.

6. Discussion and Conclusion

The work and performance achieved:

For objective 1, the current limit circuit was working successfully for our protection system. A red LED allows users to monitor the current limit circuits and it ensures the safety our whole circuits. For objective 2, the interface was working properly. Its powerful functions provide convenience for users to control the motor and detect the motor status. The designed menu worked appropriately with the 6 buttons. These have laid a good foundation for achieving the other objectives. For objective 3, the motor was rotating stably and safely. This demonstrated the reliability of our hardware and software. In addition, the maximum speed of our motor could reach about 4000 rad/min. For objective 4, the motor could change direction automatically and our interface could display the direction of the motor spinning. However, the performance of the direction control was not stable and flexible as it made several mistakes. For the detection of direction, it also has made several mistakes when the motor was rotating very fast. For objective 5, the online variations of firing angles performance works well.

Challenges met and overcome:

When testing the motor spinning, presents of noises affect the circuits drastically. Different values of decoupling capacitors were added to the chips and power supply so as to eliminate and filter the noises. Both hardware and software switch de-bouncing were used to improve the user experience of the menu. High attention was paid when soldering on the strip board in case of short circuit.

Learning outcomes:

Working on this project requires a lot of research, trial and error, and basic understanding of how the machine works. We were able to apply and utilize our knowledge that we gain from our power supply electronic module, learnt how current limiter work and how to build a simple one, learnt how to introduce delay in a device, learnt how to spot fault better and reconfiguring any errors specifically in code writing and when soldering components to the boards. Other than covering the technical aspect of this project, it also builds on our team management skills, such as better delegations of work, brainstorming ideas and deciding on the best solutions which in turn help boost our decision making skill.

Improvements and suggestions:

The different typologies of converter would be used in the system. The new typology would increase the motor efficiency and make it rotate faster. All the switches would have both software and hardware de-bouncing. Matrix buttons would be implemented in order to save the input Pins of PIC. For the direction detection, hardware method which is using D flip-flop would replace the software method in order to ensure the accuracy of the detection. For torque control, R-2R DAC would be built and PORTD<4:0> would be set to output 0-0.8V signal to ensure the reliability of software. For the board of the hardware, less wires would be used to reduce the inductance which could causes the noises. More decoupling capacitors would be used to reduce the noises of the power supply and oscillations of chips. The space of the board would be used efficiently. More nodes would be added on the board for the detection of the circuits. More LEDs would be used to show the status of the circuits.

Comparison of project flow to plan:

When referring to our initial schedules that we had put up in the initial report documentation, unfortunately we may have overrun our set deadlines for some of the objectives thus resulting us to not able to complete all of them. However, we were able to achieve the objectives in the order that we have assigned. One area of concern that slowing us down was building the prototype as we had faced difficulties in making it to function however in the end we did manage to figure out the cause of it and had it reconfigured. Although there are three more objectives that has not been covered but we were quite satisfy with what we had achieved as most of the aspects of the gate driver is functioning well as we want to and as per required.

7. Appendix

```
#pragma config IESO=OFF, FCMEN=OFF, FOSC=INTIO67, PWRT=OFF, BOREN=OFF, WDTEN=OFF
#pragma config WDTPS=32768, MCLRE=ON, LPT1OSC=OFF, PBADEN=OFF, CCP2MX=PORTC
#pragma config STVREN=OFF, LVP=OFF, XINST=OFF, DEBUG=OFF, CP0=OFF, CP1=OFF
#pragma config CP2=OFF, CP3=OFF, CPB=OFF, CPD=OFF, WRT0=OFF, WRT1=OFF
#pragma config WRT2=OFF, WRT3=OFF, WRTB=OFF, WRTC=OFF, WRTD=OFF, EBTR0=OFF
#pragma config EBTR1=OFF, EBTR2=OFF, EBTR3=OFF, EBTRB=OFF
// Various includes for the display driver - not that these are MODIFIED from the originals
#include <stdio.h>
#include <xc.h>
#include <htc.h>
#include "oleds.h"
#include "oled_interface.h"
#include "oled_jib.h"
#include "oled_cmd.h"
#include "timers.h"
#include "Interrupts.h"
#include <p18f46K20.h>
int DAC1=0;
int on_angle;                       // firing angle
int count=0;                        // record Z pulses
int time=0;
int rotate_speed;                   // desired  speed
int speed=0;                        // actual speed
int menu;                           // modes of menu
char direction[]="LEFT ";
int off_angle;                      //firing angle
unsigned int flag;
int bool_direction;                 // bool for direction detection
unsigned int stop_flag;             // change direction flag: cancel the reset TMR1 value
float current;                      // feed back current
float d_current;                    // desired current
void InitializeSystem(void);
void ADC_Init(void);                // ADC init
unsigned char ADC_Convert(void);
```

```
void main(void)
{
	char buffer[20];// A buffer into which we place text for sending to the OLED
	InitializeSystem();	// initialise interrupts, PORTs, functions, etc
	// initialise the display
	oled_init();
	oled_clear();
	oled_refresh();
	// initialise variables.
	flag=0;// set +/-
	bool_direction=0;
	menu=0;
	on_angle=10;
	off_angle=70;
	stop_flag=0;
	rotate_speed=2000;
	count=0;
	time=0;
	while (1)
{
	while (menu==0)// main menu
	{
		CCPR2L = (off_angle*25/18)+5;
		CCPR1L = (on_angle*25/18)+5;
		oled_clear();
		sprintf(buffer, "On_angle:%d\n",on_angle);
		oled_puts_1x (buffer);
			sprintf(buffer, "Off_angle:%d\n",off_angle);
			oled_puts_1x (buffer);
			sprintf(buffer, "Speed:%d r/min\n",speed);
			oled_puts_1x (buffer);
			sprintf(buffer, "Direction:%s\n",direction);
			oled_puts_1x (buffer);
			sprintf(buffer, "RB4:set on_angle\n");
			oled_puts_1x (buffer);
```

```
                    sprintf(buffer, "RB5:set off_angle\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB6 set speed\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB1 set direction");
                    oled_puts_1x (buffer);
oled_refresh();
                    }//menu 0
                    while (menu==1)// set on_angle
                    {
                    oled_clear();
                    sprintf(buffer, "off_angle:%d\n",off_angle);
                    oled_puts_1x (buffer);
                    sprintf(buffer, "On_angle:%d\n",on_angle);
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB2 to set +/- angle\n");
                    oled_puts_1x (buffer);
                    if  (flag==0)
                    {
                    sprintf(buffer, "RB4:on_angle +1\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB5:on_angle +10\n");
                    oled_puts_1x (buffer);
                    }
                    else
                    {
                    sprintf(buffer, "RB4:on_angle -1\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB5:on_angle -10\n");
                    oled_puts_1x (buffer);
                    }
                    sprintf(buffer, "RB7:back to menu");
                    oled_puts_1x (buffer);
                    oled_refresh();
                    CCPR2L = (off_angle*25/18)+5; // set CCP2
```

```
                    CCPR1L = (on_angle*25/18)+5;// set CCP1
                    }//menu1
                            while (menu==2)// set off_angle?
                    {
                    oled_clear();
sprintf(buffer, "off_angle:%d\n",off_angle);
                    oled_puts_1x (buffer);
                    sprintf(buffer, "On_angle:%d\n",on_angle);
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB2 set +/- angle\n");
                    oled_puts_1x (buffer);
                    if  (flag==0)
                    {
                    sprintf(buffer, "RB4:off_angle +1\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB5:off_angle +10\n");
                    oled_puts_1x (buffer);
                    }
                    if (flag==1)
                    {
                    sprintf(buffer, "RB4:off_angle -1\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB5:off_angle -10\n");
                    oled_puts_1x (buffer);
                    }
                    sprintf(buffer, "RB7:back to menu");
                    oled_puts_1x (buffer);
                    oled_refresh();
                    CCPR2L = (off_angle*25/18)+5;// set CCP2
                    CCPR1L = (on_angle*25/18)+5;// set CCP1
                    }// menu2
                    while (menu==3)// set speed
                    {
                    oled_clear();
                    sprintf(buffer, "actual_speed:%d\n",speed);
```

```
oled_puts_1x (buffer);
sprintf(buffer, "Speed:%d\n",rotate_speed);
oled_puts_1x (buffer);
                    sprintf(buffer, "RB2 to set +/- speed\n");
                    oled_puts_1x (buffer);
                    if (flag==0)
                    {
                    sprintf(buffer, "Increase speed\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB4:speed +1\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB5:speed +10\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB6:speed +100");
                    oled_puts_1x (buffer);
                    }
                    if (flag==1)
                    {
                    sprintf(buffer, "decrease speed\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB4:speed -1\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB5:speed -10\n");
                    oled_puts_1x (buffer);
                    sprintf(buffer, "RB6:speed -100");
                    oled_puts_1x (buffer);
                    }
                    oled_refresh();
                    // speed_function();
                    }//menu3
                    while (menu==4)// direction change menu
                    {
                    stop_flag=1;
                    oled_clear();
                    oled_puts_2x ("waiting for changing");
```

```
                        oled_refresh();
                        if (LATDbits.LATD4==1)// current direction detection
                        {
                        while((LATDbits.LATD4==1)&&(speed<200));// wait  direction changing
                                if(LATDbits.LATD5==1)
                                {
                                menu=0;stop_flag=0;break;
                                }
                        }//end if
                        if (LATDbits.LATD5==1)// current direction detection
                        {
                        while((LATDbits.LATD5==1)&&(speed<200));// wait  direction changing
                                if(LATDbits.LATD4==1)
                                {
                                menu=0; stop_flag=0;break;// recover reset TMR1 in INT0 go to menu
                                }
                        }
                        }//menu4
}//while 1
}//end main
//#pragma interrupt InterruptServiceHigh  // "interrupt" pragma also for high priority
void interrupt ServiceHigh(void)
{
if (INTCON3bits.INT2IF)
{
INTCON3bits.INT2IF=0;
if (PORTBbits.RB2==0) flag=(flag+1)%2;//RB2 SET+/-
}//INT2
if (INTCONbits.INT0IF)
{
        INTCONbits.INT0IF = 0;
        if ((PORTAbits.RA4==1)&&(PORTCbits.RC0==0))// If B leads A?
                        {
                                direction[0]='L';       direction[1]='E';       direction[2]='F';
        direction[3]='T';       direction[4]=' ';
```

```
                    LATDbits.LATD5=1;LATDbits.LATD4=0;// direction status display
                }
        else {
                direction[0]='R';       direction[1]='I';       direction[2]='G';       direction[3]='H';
        direction[4]='T';
                LATDbits.LATD5=0;LATDbits.LATD4=1;// direction status display
                }

                if(stop_flag==0)// cancel or recover the reset?
                {
                        TMR1H = 0xFF;
        TMR1L = 0x05;
                }//INT0
        count++;// record Z pulses
        LATDbits.LATD6=1;// motor running
}
if  (PIR1bits.TMR1IF)
  {
    PIR1bits.TMR1IF = 0;        // clear (reset) flag
    LATDbits.LATD7 = 1;
    TMR1H = 0xFF;     // reset TMR1
                TMR1L=0x05;
        }
if (INTCON3bits.INT1IF)//RB1 pressed, direction changed.
{
        INTCON3bits.INT1IF=0;
        menu=4;
}
}//high interrupt
void interrupt low_priority ServiceLow(void)
{
int Switch_Count=0;// switch debouncing
if (INTCONbits.TMR0IF)
        {
                INTCONbits.TMR0IF=0;
```

```
        TMR0H = 0;              //65536
   TMR0L = 0;
        if (speed==0) {LATD=0;}         // motor is stopped
        speed=count*57+count/4;// caculate speed
        count=0;//reset count
        }


if ((menu==0)&&(INTCONbits.RBIF))// main menu;
  {
                INTCONbits.RBIF=0;
        if (PORTBbits.RB4 == 1)         // set on anlge
                {
                        Switch_Count=0;
                        do                                      // switch debouncing
                {
                        if (PORTBbits.RB4 == 1)
                                Switch_Count++;
                        else
                                break;
                        Delay10TCYx(125); // delay 5ms.
                } while (Switch_Count < 5);
                while (PORTBbits.RB4 == 1);

                if (Switch_Count==5)
                menu=1;// goto set on angle menu
                }
                if (PORTBbits.RB5 == 1)         // set off anlge
                {
                        Switch_Count=0;
                        do                                      // switch debouncing
                        {
                        if (PORTBbits.RB5 == 1)
                                Switch_Count++;
                                else
```

```
                    break;
                    Delay10TCYx(125);
                } while (Switch_Count < 5);
            while (PORTBbits.RB5 == 1);
            if (Switch_Count==5)   menu=2;//goto set ff angle menu
            }
        if (PORTBbits.RB6 == 1)// set speed
            {
                    Switch_Count=0;
                        do                                          // switch debouncing
                        {
                        if (PORTBbits.RB6 == 1)
                        Switch_Count++;
                        else
                            break;
                            Delay10TCYx(125); // delay 1250 cycles or 5ms.
                        } while (Switch_Count < 5);
                while (PORTBbits.RB6 == 1);
                if (Switch_Count==5)   menu=3; // goto set speed
            }
        }//main menu
if (INTCONbits.RBIF)// child menu
{
if (menu==1)// set on_angle
  {
            INTCONbits.RBIF = 0;  // clear (reset) flag
                if (PORTBbits.RB4 == 1)
                    {
                        Switch_Count=0;
                        do
                        {
                        if (PORTBbits.RB4 == 1)
                        Switch_Count++;
                        else
                            break;
```

```
                Delay10TCYx(125);
            } while (Switch_Count < 5);
            while (PORTBbits.RB4 == 1);
        if (Switch_Count==5)
            {
            if (flag==0)on_angle=on_angle+1; else on_angle=on_angle-1; // on
angle+/-1
            }
        }

    if (PORTBbits.RB5 == 1)
        {
            Switch_Count=0;
            do
            {
            if (PORTBbits.RB5 == 1)
            Switch_Count++;
            else
                break;
                Delay10TCYx(125);
            } while (Switch_Count < 5);
                while (PORTBbits.RB5 == 1);
        if (Switch_Count==5)
        {
            if (flag==0)on_angle=on_angle+10; else on_angle=on_angle-10;  // // on
angle+/-10
        }
        }
    if (on_angle>90) on_angle=90;//restrict on angle value
    if (on_angle<0)  on_angle=0;//restrict  on angle value
    if (PORTBbits.RB7 == 1)// back to menu
        {
            Switch_Count=0;
            do
            { // monitor switch input for 5 lows in a row to debounce if (Switch_Pin
== 0)
```

```
                                        if (PORTBbits.RB7 == 1)
                                        Switch_Count++;
                                        else
                                                break;
                                                Delay10TCYx(125); // delay 1250 cycles or 5ms.
                                        } while (Switch_Count < 5);
                                        while (PORTBbits.RB7 == 1);
                                        if (Switch_Count==5) menu=0;
                                }
                        if (PORTBbits.RB6 == 1)// set off angle
                {
                                        Switch_Count=0;
                                        do
                                        { // monitor switch input for 5 lows in a row to debounce if (Switch_Pin
== 0)
                                        if (PORTBbits.RB6 == 1)
                                        Switch_Count++;
                                        else
                                                break;
                                                Delay10TCYx(125); // delay 1250 cycles or 5ms.
                                        } while (Switch_Count < 5);

                                while (PORTBbits.RB6 == 1);
                        if (Switch_Count==5) menu=2;// goto set off angle menu;
                }
        }//menu1
        if (menu==2)// set off angle
  {
                                INTCONbits.RBIF = 0;
                                if (PORTBbits.RB4 == 1)
                                {       Switch_Count=0;
                                        do
                                        {
                                        if (PORTBbits.RB4 == 1)
                                        Switch_Count++;
```

```
                    else
                        break;
                        Delay10TCYx(125);
                    } while (Switch_Count < 5);
                    while (PORTBbits.RB4 == 1);
                    if(Switch_Count == 5)
                    {
                    if (flag==0)off_angle=off_angle+1; else off_angle=off_angle-1; //  set
off_angle+/-1
                    }
                }
            if (PORTBbits.RB5 == 1)
                {
                    Switch_Count=0;
                    do
                    {
                    if (PORTBbits.RB5 == 1)
                    Switch_Count++;
                    else
                        break;
                        Delay10TCYx(125);
                    } while (Switch_Count < 5);
                while (PORTBbits.RB5 == 1);
                    if(Switch_Count == 5)
                    {
                    if (flag==0)off_angle=off_angle+10; else off_angle=off_angle-10; // set
0ff_angle+/-10
                    }
                }
            if (off_angle>90) off_angle=90;
            if (off_angle<0)  off_angle=0;
                    if (PORTBbits.RB6 == 1)// set on_angle
        {
                    Switch_Count=0;
                    do
```

```
                    {
                    if (PORTBbits.RB6 == 1)
                    Switch_Count++;
                    else
                        break;
                        Delay10TCYx(125);
                    } while (Switch_Count < 5);

                while (PORTBbits.RB6 == 1);
            if (Switch_Count==5) menu=1;// go to set on  angle menu
        }
                    if (PORTBbits.RB7 == 1)
                    {
                        Switch_Count=0;
                        do
                        {
                        if (PORTBbits.RB7 == 1)
                        Switch_Count++;
                        else
                            break;
                            Delay10TCYx(125);
                        } while (Switch_Count < 5);
                        while (PORTBbits.RB7 == 1);
                        if (Switch_Count==5) menu=0;// go to main menu
                    }
}//menu2
if (menu==3)// set speed
{
            INTCONbits.RBIF = 0;
if (PORTBbits.RB4 == 1)
                    {
                        Switch_Count=0;
                        do
                        {
                        if (PORTBbits.RB4 == 1)
```

```
                    Switch_Count++;
                    else
                        break;
                        Delay10TCYx(125);
                    } while (Switch_Count < 5);

                    while (PORTBbits.RB4 == 1);
                    if (Switch_Count==5)
                    {
                    if(flag==0)rotate_speed=rotate_speed+1;
else rotate_speed=rotate_speed-1; // flag=0 speed+1 flag=1 speed-1
                    }
                }
            if (PORTBbits.RB5 == 1)
                {
                    Switch_Count=0;
                    do
                    {
                    if (PORTBbits.RB5 == 1)
                    Switch_Count++;
                    else
                        break;
                        Delay10TCYx(125); // delay 1250 cycles or 5ms.
                    } while (Switch_Count < 5);
                        while (PORTBbits.RB5 == 1);
                    if (Switch_Count==5)
                    {
                    if (flag==0) rotate_speed=rotate_speed+10; else
rotate_speed=rotate_speed-10;// speed+/-10
                    }
                }
            if (PORTBbits.RB6 == 1)
                {
                    Switch_Count=0;
                    do
```

```
                    {
                    if (PORTBbits.RB6 == 1)
                    Switch_Count++;
                    else
                        break;
                        Delay10TCYx(125); // delay 250 cycles or 5ms.
                    } while (Switch_Count < 5);
                    while (PORTBbits.RB6 == 1);
                    if (Switch_Count==5)
                    {
            if (flag==0)  rotate_speed=rotate_speed+100; else rotate_speed=rotate_speed-100;
                    }
                }
                if (rotate_speed<=1000) rotate_speed=1000;
                if (rotate_speed>=6000) rotate_speed=6000;
            if (PORTBbits.RB7 == 1)
                {
                    Switch_Count=0;
                    do
                { // monitor switch input for 5 lows in a row to debounce if (Switch_Pin == 0)
                    if (PORTBbits.RB7 == 1)
                    Switch_Count++;
                    else
                        break;
                        Delay10TCYx(125); // delay 1250 cycles or 5ms.
                    } while (Switch_Count < 5);
                    while (PORTBbits.RB7 == 1);
                    if (Switch_Count==5) menu=0;// go back to main menu
                }
        }//menu3
}//Child menu
}//low
// Intialise the system -
void InitializeSystem(void)
{
```

```
        ANSEL=0x00;
        OSCCON = 0b01110000;
        OSCTUNEbits.PLLEN = 0;                                  // change to 1 to turn on the PLL CLK * 4
        // Setup I/O Ports.
        TRISA = 0b00001000; //PORTA IO SETTING
        TRISB = 0b11111111;  //PORTB IO setting
        TRISC = 0b00000001;// PORTC IO setting
        TRISE = 0b00000111;// Set tristate on E (may be overridden in ADC_INIT)
        TRISD = 0b00001000;  // TRISD is LED input/output
        LATD = 0;                   // turn off LEDS
        IOCB=0xF0;                      // IOCB register Enable RBIF
        oled_res = 1;                                           // do not reset LCD yet
        oled_cs = 1;                                            // do not select the LCD
        INTCON2bits.RBPU = 0;               // enable PORTB internal pullups
        WPUBbits.WPUB0 = 0;                 // disable PORTB RB0 pull-up
        WPUBbits.WPUB1 = 1;                 // Enable PORTB RB1
        WPUBbits.WPUB2 = 1;
         WPUBbits.WPUB3 = 0;
        WPUBbits.WPUB4 = 0;
        WPUBbits.WPUB5 = 0;
         WPUBbits.WPUB6 = 0;
        WPUBbits.WPUB7 = 0;
        // Setup TMR1
        // Configure Timer 1
        T1CON = 0b00000011;   // TIMER 1 ENABLED- EXETERNAL CLOCK ENABLED-no prescale
T1CONbits.T1CKPS0 = 0;          //PRESCALER 1:1
        T1CONbits.T1CKPS1 = 0;
// Set up Interrupts for timer
INTCONbits.TMR0IF = 0;        // clear roll-over interrupt flag
INTCON2bits.TMR0IP = 0;       // Timer0 is low priority interrupt
INTCONbits.TMR0IE = 1;        // enable the Timer0 interrupt.
// Set up timer itself
        T0CON = 0b00000101;         // prescale 1:64
TMR0H = 0;                  // clear timer - always write upper byte first
TMR0L = 0;
```

```
T0CONbits.TMR0ON = 1;         // start timer
        // Configure MSSP for SPI master mode
        SSPCON1 = 0b00110001;                                   // clock idle high, Clk /64
        SSPSTAT = 0b00000000;
        TMR1H = 0xFF;                  // clear timer - always write upper byte first
        TMR1L = 0xFF-250;
        CCP1CON = 0b00000010;                                   // compare mode, toggle output on match
        CCP2CON = 0b00000010;                                   // compare mode, toggle output on match
        CCPR1H=0xFF;
        CCPR2H=0xFF;
        CCPR1L = 0x13;                                          //TURN ON ANGLE = 10 DEG
        CCPR2L = 0x66;                                                  //TURN OFF ANGLE= 70 DEG
        T1CONbits.TMR1ON = 1;           // start timer 1
        RCONbits.IPEN = 1;              // Enable priority levels on interrupts
        PIR1 = 0;
        PIE1bits.TMR1IE = 1;   // Enable TIMER1 Interrupt
        IPR1 = 0;
        IPR2 = 0;
        IPR1bits.TMR1IP = 1;   // high priority
        PIR2=0;
INTCON2bits.INTEDG0 = 0;    // interrupt on falling edge of INT0 (switch pressed)
INTCONbits.INT0IF = 0;     // ensure flag is cleared
INTCONbits.INT0IE = 1;
        INTCON2bits.INTEDG1 = 0;    // interrupt on falling edge of INT1 (switch pressed)
INTCON3bits.INT1IF = 0;     // ensure flag is cleared
INTCON3bits.INT1IE = 1;     // Enable INT1 Interrupt
        INTCON3bits.INT1IP=1;      // high priority
        INTCON2bits.INTEDG2 = 0;    // interrupt on falling edge of INT2 (switch pressed)
INTCON3bits.INT2IF = 0;     // ensure flag is cleared
INTCON3bits.INT2IE = 1;                 // Enable INT2 Interrupt
        INTCON3bits.INT2IP=1;      // high priority
        INTCONbits.RBIE=1;  // Enable RBIF Interrupt
        INTCONbits.RBIF=0;
        INTCON2bits.RBIP=0;// low priority
        INTCONbits.PEIE = 1;
```

```
        INTCONbits.GIEH = 1;// Interrupting enabled.
} // end InitializeSystem
void ADC_Init(void)
{ // initialize the Analog-To-Digital converter.
        ANSEL = 0;       //turn off all other analog inputs
        ANSELH = 0;
        ANSELbits.ANS5 = 1;    // turn on RE0 analog
   ADCON2 = 0b00111000;
        ADCON0 = 0b00010101;
}
unsigned char ADC_Convert(void)
{ // start an ADC conversion and return the 8 most-significant bits of the result
   ADCON0bits.GO_DONE = 1;          // start conversion
   while (ADCON0bits.GO_DONE == 1);    // wait for it to complete
   return ADRESH;                   // return high byte of result
}
```