

EWDyer_PSDE5013- Assignment_2

by Emyr Dyer

FILE	7311_EMYR_DYER_EWDYER_PSDE5013- ASSIGNMENT_2_145116_726629688.DOCX		
TIME SUBMITTED	27-MAY-2018 10:41PM (UTC+0100)	WORD COUNT	3020
SUBMISSION ID	89979919	CHARACTER COUNT	21191

HND Electrical & Electronic Engineering

DC Motor Controller

PSDE5013 – Configuration and
Programming of Embedded System

Author: Emyr Wyn Dyer
Submission Date: 27/05/2018
Word Count: 3020

CONTENTS

1	Introduction	1
2	System Hardware	2
2.1	PIC16F877 Microcontroller	2
2.2	Encoder Feedback	3
2.3	Keypad	3
2.4	HP488 Development Board	4
2.5	Start/Stop	4
2.6	LCD Display	4
2.7	24V DC Motor	5
3	Microcontroller Configuration	6
3.1	Option Register	6
3.2	Intcon Register	6
3.3	Ports	6
3.4	Timer 0	7
3.5	Timer 1	8
3.6	Timer 2	8
3.7	PWM	8
4	Software Design and Coding	9
5	Testing and Analysis	14
5.1	Simulation	14
5.2	Practical	16
5.2.1	Build	16
5.2.2	General Test	16
5.2.3	Setpoint and PWM at 200rpm	17
5.2.4	Setpoint and PWM at 1000RPM	18
5.2.5	Setpoint and PWM at 2300 RPM	19
5.2.6	Observation	19
6	Discussion and Conclusions	20
7	Works Cited	21

Figures

Figure 2.1: Pinout	2
Figure 2.2: HEDS/HEDM Encoder	3

Figure 2.3: HEDS/HEDM Wiring Schematic.....	3
Figure 2.4: 3x4 Matrix Keypad linked to PIC Schematic.....	3
Figure 2.5: HP488 Dev Board	4
Figure 3.1: Timer 0	7
Figure 4.1: Headers, Function Prototypes, and Global Variables	9
Figure 4.2: ISR	9
Figure 4.3: Main Program	10
Figure 4.4: DisplaySpeed	11
Figure 4.5: DisplayValues	12
Figure 4.6: ScanKeypad and Delay	13
Figure 5.1: 150 Setpoint.....	14
Figure 5.2: 300 Setpoint.....	14
Figure 5.3: 350 Setpoint.....	14
Figure 5.4: Practical Setup	16
Figure 5.5: LCD Display.....	16
Figure 5.6: Setpoint @ 200RPM.....	17
Figure 5.7: PWM @ 200	17
Figure 5.8: Setpoint @ 1000RPM.....	18
Figure 5.9: PWM @ 1000	18
Figure 5.10: Setpoint @ 2300RPM.....	19
Figure 5.11: PWM @ 2300	19

1 INTRODUCTION

The aim of this investigation is to determine the viability of using a PIC to control a DC motor using PWM, with a Duty Cycle determined by Feedback read by an encoder and processed by a PI Control.

*Far too
brief. This
could have
pointed out
the method
of
developme
nt using
prototyping.*

2 SYSTEM HARDWARE

With the objective being the development of a DC Motor control using a Microcontroller programmed in C, the following elements are required to facilitate development;

- PIC16F877 Microcontroller – A Controller Chip mounted to a development board which has ports for I/O and PWM especially.
- 9 Digit Keypad – for manual control of the setpoint for speed.
- 2 Line LCD screen – to provide the operator a visual indicator of selected speed as well as measured speed.
- Variable Power Supply – For powering of DC Motor.
- Encoder – Used to measure the rotational speed of the DC Motor.
- DC Motor – to be test the ability of the Microcontroller and C Program to control speed.

2.1 PIC16F877 MICROCONTROLLER

In consulting the accompanying datasheet for the PIC, all the essential details for effective development of the C program can be found, such as the Pin Schematic as shown in Figure 2.1 which shows the connections for the required ports for PWM, Start/Stop, Encoder feedback (Microchip, 2018)Figure 2.1.

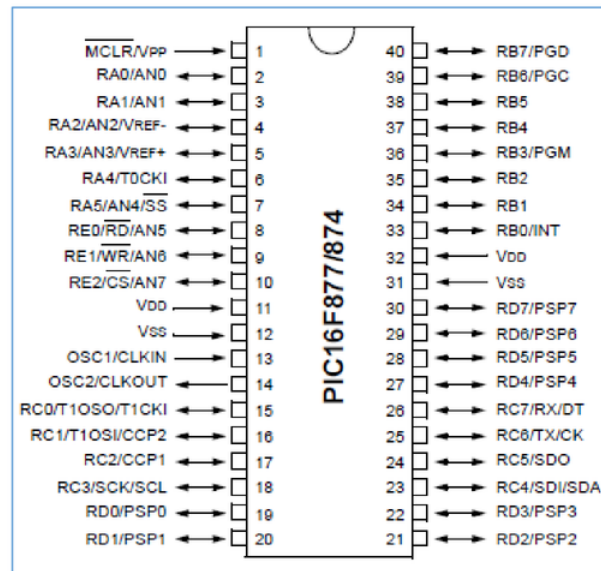


Figure 2.1: Pinout

2.2 ENCODER FEEDBACK

To effectively allow Proportional Integral Control (PI), it is necessary to have a means of detecting the instantaneous rotational speed of the motor within the control loop. The best means to allow this is the use of an Incremental Encoder, which in the case of the design will be an Avago manufactured Optical Encoder of the type shown in **Error! Reference source not found.**. The model selected for the job of reading the rotational speed is capable of outputting 500 pulses per revolution, though other models do offer resolutions of up to 1024 pulses.



Figure 2.2: HEDS/HEDM Encoder

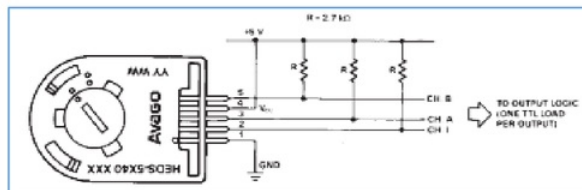


Figure 2.3: HEDS/HEDM Wiring Schematic

2.3 KEYPAD

Although different ports are used in the actual design to read in the setpoint, Figure 2.4 shows how a Matrix Keypad can be linked to a PIC, however there should be four additional resistors on A, B, C, and D for correct output. Keypresses are correctly identified using Multiplexing (done in code) where the cross-referencing of the row and column identifies the button pressed. So when '5' is pressed, a circuit is completed for Column 2 and Row B.

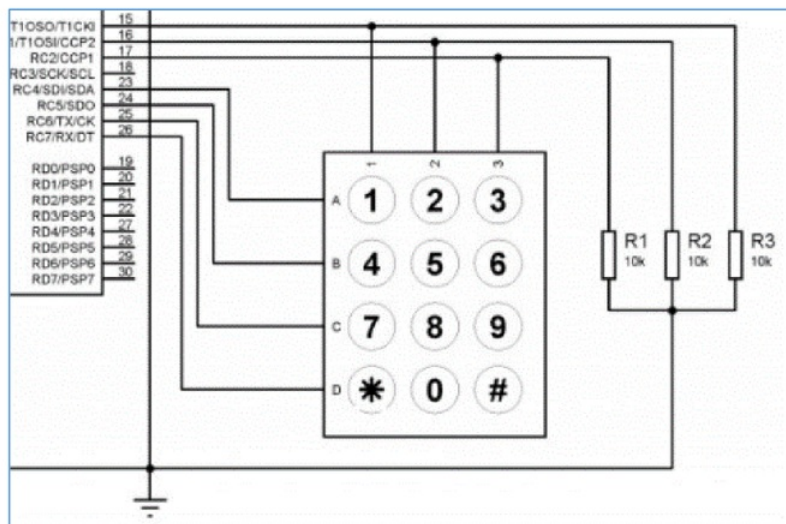


Figure 2.4: 3x4 Matrix Keypad linked to PIC Schematic

2.4 HP488 DEVELOPMENT BOARD

The Development Board shown in Figure 2.5 will be used to host the PIC16F877 chip. Board features used are identified in Table 2.1 as found in the accompanying device datasheet (Ltd, Matrix Technology Solutions, 2018).

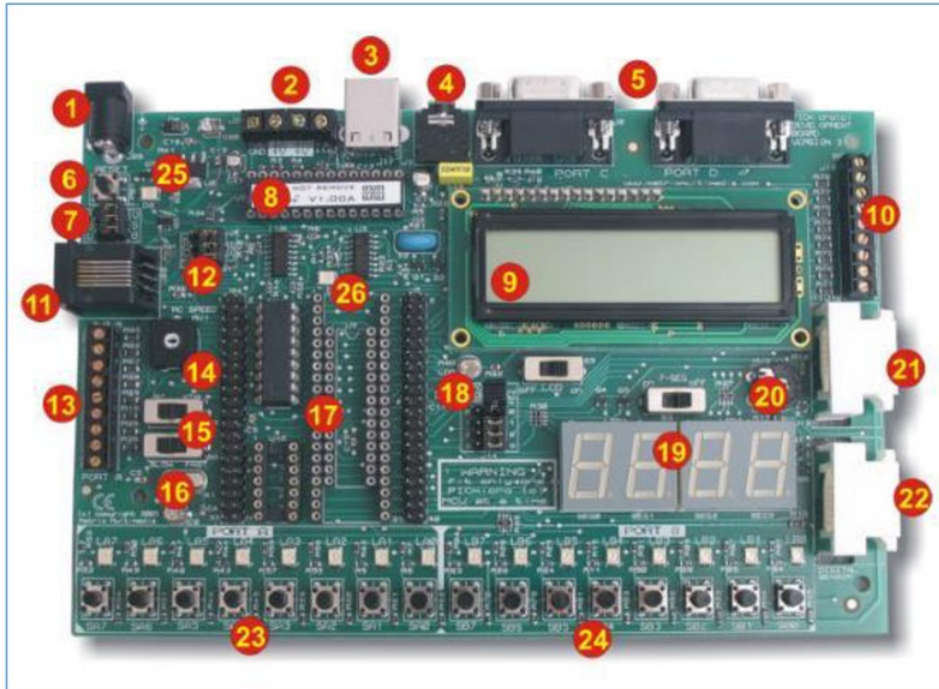


Figure 2.5: HP488 Dev Board

Element	Description
1	Power Supply connector
2	5V outputs for E-blocks connection
5	E-blocks connectors for ports C and D
9	2-line 16-character alphanumeric display
17	Sockets for programming 8, 14, 18, 28 and 40 pin PICmicro devices
24	PORT B switches and LEDs

Table 2.1: Elements Used

2.5 START/STOP

For simplicity, existing pushbuttons on the development board (PORT B) are used for the Start and Stop function in the program, other pushbuttons may be used in practical implementation, possibly on the numeric keypad, or separate DOL type pushbuttons.

2.6 LCD DISPLAY

The onboard 2 Line LCD display is sufficient for displaying the set speed and the measured speed. The Display is connected to the PIC directly through PORT B.

2.7 24V DC MOTOR

The final element is the Motor itself. As the motor operates at a higher voltage of 24V, which would damage the PIC and Board which runs at a lower 5V, so a MOSFET is employed to handle the direct switching of the 24V DC as dictated by the 5V PWM of the PIC.

The explanations are good. The interface for driving the PWM and receiving the encoder pulses should be included here.

3 MICROCONTROLLER CONFIGURATION

3.1 OPTION REGISTER

Used to configure Prescaling, Interrupts and Transition Detection (Rising/Falling Edge). To set the register for the job required, the following setting is used, 0b10000111.

The set bits are the following;

- Prescaler Rate Select bits (bit 2-0) – Setting dictates that a 1:256 TMRO Rate is used and further adjusted later in code.
- Prescaler Assignment bit (bit 3) – The Prescaler is assigned to the TMRO module, thus the 1:256 Rate.
- TMRO Source Edge Select bit (bit 4) – The Incrementation is carried out on a transition of low-to-high signal input on RA4.
- TMRO Clock Source Select bit (bit 5) – The Internal CLKOUT is used as opposed to detected input on RA4.
- Interrupt Edge Select bit (bit 6) – The Interrupt is detected on the falling edge of the signal from RB0.
- PORT B Pull-up Enable bit (bit 7) – The Pull-ups are disabled on PORT B.

3.2 INTCON REGISTER

This Register exclusively handles the Interrupt Conditions for Timer 0, PORT B change as well as External Peripheral Interrupts. For the design, the setting used is 0b11100000.

The following bits have been set;

- RB Port Change Interrupt Flag bit (bit 0) – Pins RB4 to RB7 do not have a changed state.
- RB0/INT External Interrupt Flag bit (bit 1) – No External Interrupt.
- TMRO Overflow Interrupt Flag bit (bit 2) – No Overflow of TMRO register.
- RB Port Change Interrupt Enable bit (bit 3) – Port Change Disabled.
- RB0/INT External Interrupt Enable bit (bit 4) – External Interrupt Disabled.
- TMRO Overflow Interrupt Enable bit (bit 5) – TMRO Interrupt Enabled.
- Peripheral Interrupt Enable bit (bit 6) – Peripheral Interrupts are permitted.
- Global Interrupt Enable bit (bit 7) – All Unmasked Interrupts are enabled.

With this configuration, only Timer 0 and Peripheral Interrupts may operate as interrupts.

3.3 PORTS

The following Ports are used for the design, which are and set using TRIS;

- PORT A – Unused, so set to 0xFF
- PORT B – LCD Display output, so set to 0x00
- PORT C – Timer 1 Counting and PWM, so set to 0x01 (RC0 input, RC2 output)
- PORT D – Keypad input, so set to 0b11110000

3.4 TIMER 0

With the Option Register and the Intcon configured, the only element that remains to be configured before using Timer 0, is the value for the T0IF, which is the Timer 0 Overflow Flag, which was enabled in the INTCON.

Design will require an interval of 10ms for the interrupt to provide a sensible operation. As the PIC's clock speed (Fosc) is 19.6608Mhz, some mathematical processing of the frequency must be carried out to gain the sensible 10ms time interval.

$$CLKOUT = \frac{Fosc}{4} = 4.9152MHz$$

A Max Prescaler of 256 is used;

$$\frac{CLKOUT}{Prescaler} = \frac{4.0152 \times 10^6}{256} = 19.2KHz$$

From this new frequency with prescaler applied the Time Period is found;

$$\frac{1}{19.2 \times 10^3} = 52\mu s$$

With the number of counts being 1/100 for every second equalling 192, a 10ms interrupt is found. So, in code to correctly set the interrupt, the number of counts is deducted from the max prescaler;

$$256 - 192 = 64$$

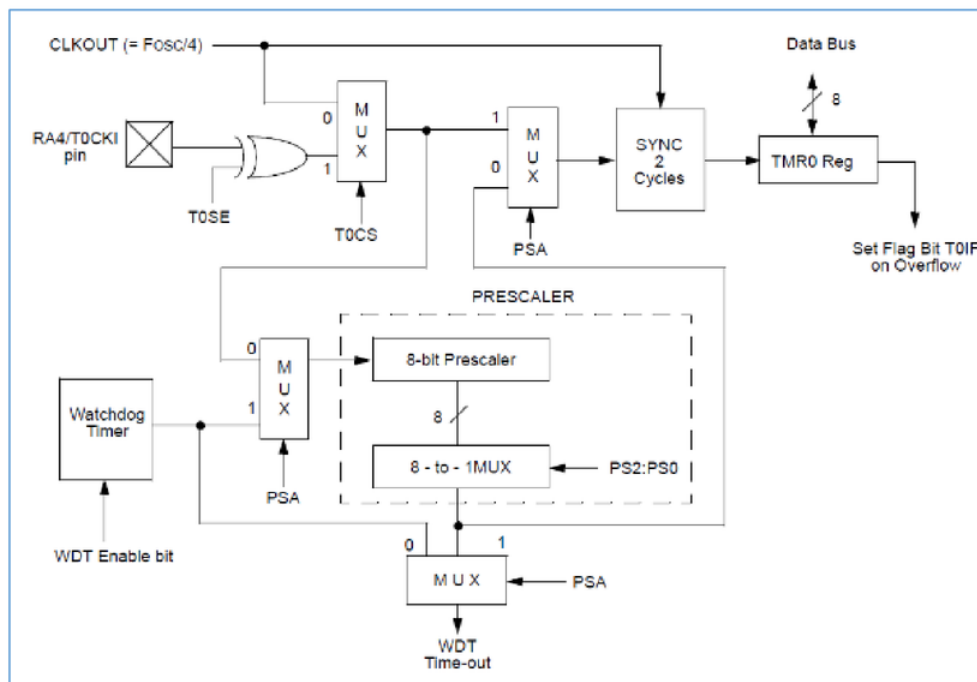


Figure 3.1: Timer 0

3.5 TIMER 1

Timer 1 is utilised in the design and can count to 65535 (2^{16}) as the timer is 16-bit, which is the result of two 8-bit registers being used together to store a high and low byte from the whole 16-bit count. Feedback from the DC motor is picked up by the RC0 pin for Timer 1.

To properly configure, PIE1 is set to 0b00000001 and PIR1 is set to 0b00000000 for overflow, and the following bits are set as 0b00000111 for the Timer 1 Configuration Register (T1CON);

- Prescaler Rate Select bits (bit 2-0) – Setting dictates that a 1:256 TMR0 Rate is used and further adjusted later in code.
- Timer 1 On bit (bit 0) – Timer 1 is enabled.
- Timer 1 Clock Source Select bit (bit 1) – RC0 pin used as clock source
- Timer 1 External Clock Input Synchronisation Control bit (bit 2) – The external clock source is not synchronised with the internal clock.
- Timer 1 Oscillation Enable Control bit (bit 3) – Oscillator is off.
- Timer 1 Input Clock Prescale Select bits (bit 4-5) – 1:1 Prescale value used with scale adjustment handled in code.
- (bit 6-7) – Unused.

3.6 TIMER 2

The core function of Timer 2 is to operate with the PWM to control the speed of the DC motor. The maximum count for Timer 2 is 256 (2^8) due to the 8-bit register used.

Timer 2 is configured as 0b00000101, with each bit setting as follows;

- Timer 2 Clock Prescale Select bits (bits 0-1) – Prescaler set to 4.
- Timer 2 On bit (bit 3) – Timer 2 on.
- Timer 2 Output Postscale Select bits (bits 3-6) – Postscale of 1:1
- (bit 7) – Unused.

3.7 PWM

The PWM output from the PIC has a resolution of 10-bit, so with a binary value of 0 the duty cycle will be 0% and at 1024 (2^{10}) the duty cycle will be 100%. The PWM module that will be used is CCP1, so for the 10-bit PWM to be utilized.

The following are registers used due the selection of CCP1 as opposed to CCP2;

- CCP1CON – Control register for CCP1.
- CCPPR1L – CCP Register Low.
- T2CON – Control register for Timer 2.
- PR2 – Period register for Timer 2 modules.

To determine the duty cycle, the pin RC2 will be compared to PR2, then Timer 2 to the CCP Low Register.

TRISC needs to be set so that the CCP1 pin is set as an output due to Multiplexing present on PORT C. For the PWM to operate, the configuration 0b00001100 is used for CCP1CON and both PIE2 and PIR2 are set to 0b00000000.

*T1CON should
be included to
explain the
event mode
operation with*

4 SOFTWARE DESIGN AND CODING

The top of the code are the Headers which include pic.h and LCD.h. These headers provide pre-defined functions that are used with the PIC and the peripheral LCD on the development board.

Under Function Prototypes are described the Functions to be defined at the foot of the program, along with the types of data that they read in and send back to the calling code element.

Under Global Variables are defined the integers and floats used in the developed code. Here the PI values for program are defined.

```
/////////////////////////////////Headers/////////////////////////////////
#include <pic.h>
#include "LCD.h"

/////////////////////////////////Function Prototypes/////////////////////////////////
void DisplaySpeed(int line, int value);
void DisplayValues(int line, int value1, int value2, int value3);
int ScanKeypad(void);
void Delay(int t);

/////////////////////////////////Global Variables/////////////////////////////////
int milliseconds=0, speed=0, fault=0;
int speed_low_byte=0, speed_high_byte=0, run=0;
float setpoint=0, error=0, output=0, prop part=0, int_accum = 0, Kp = 0.5, Ki = 0.1;
```

Figure 4.1: Headers, Function Prototypes, and Global Variables

The Interrupt Service Routine is not called or sends data directly but interjects during operation under specific conditions. In the case of this ISR, there is a 10ms interrupt operating off Timer 0, there is an interrupting for pulse counts, and there is an interrupt for the PWM. In the first interrupt statement, the Timer is pre-set at 64, and each time the interrupt operates, the millisecond value is incremented by 10ms until 1000ms by when the 10-bit speed value is recovered through two 8-bit values and counters are reset to 0.

```
/////////////////////////////////ISR/////////////////////////////////
void interrupt ISR(void) //ISR contains three interrupt methods.
{
    if(T0TF==1) //Timer 0. Interrupts at 10ms intervals.
    {
        T0IF = 0;
        TMR0 = 64; //Calculated Pre scale factor.
        milliseconds+=10;
        if((milliseconds==1000) && (fault==0))
        {
            speed_low_byte = TMR1L; //As speed is a 10 bit binary value, two 8 bit bytes are used.
            speed_high_byte = TMR1H;
            TMR1L = 0;
            TMR1H = 0;
            milliseconds=0;
        }
    }

    if(TMR1IF==1) //Timer 1. Pulses are counted in event mode.
    {
        TMR1IF = 0; //Reset interrupt flag.
        fault = 1;
    }

    if(TMR2IF==1) //Timer 2. PWM.
    {
        TMR2IF = 0;
    }
}
```

Figure 4.2: ISR

Starting off with the beginning of the Main element of the program, the keypad is enabled, and the registers are configured as previously discussed in Microcontroller Configuration. Code is further discussed after Figure 4.3.

```

//main program
void main(void)
{
    int enable_keypad=1, key;
    TRISA = 0xFF;

    ADCON1 = 0x0F; //ADCON1 = 0000 1110.
    TRISA = 0x00; //LCD display outputs.
    TRISC = 0x01; //RC0 set as input for Timer 1 and RC2 set as PWM output.
    TRISD = 0b11110000; //Keypad.

    PORTB = 0x00;
    LCD_Startup();

    PIE1 = 0b00000011; //Peripheral Interrupt Enable registers.
    PIR1 = 0;
    PIR2 = 0; //Reset Peripheral Interrupts Registers.

    TMRCON = 0b11100000; //Configure TIMER 0 in timer mode.
    OPTION_REG = 0b1000111;

    T1CON = 0b00000111; //Configure TIMER 1 in event mode from RC0.
    TMR1L = 0;
    TMR1H = 0;

    PR2 = 255;
    CCP1L = 0; //0% DUTY CYCLE (Off).
    T2CON = 0b00000101; //set up for Timer 2 in PWM mode using RC2.
    CCP1CON = 0b00001100;

    while(1)
    {
        speed = (int)((float)((speed_low_byte + speed_high_byte*256)*6)/50); //running speed

        if((RA1==1)&&(RA2==0)) run = 1;
        if((RA1==0)&&(RA2==1)) run = 0;

        key = ScanKeypad();

        if((key!=1) && (enable_keypad==1))
        {
            setpoint = setpoint * 10 + (float)key; //append new digit to setpoint.
            enable_keypad=0;
        }

        if(key=='#')
        {
            setpoint = 0; //Reset setpoint
        }

        if(key==1) enable_keypad=1;

        if(setpoint>2300) setpoint = 2300; //Speed must not go above 2300.
        if(setpoint<0) setpoint = 0; //Speed cannot go below 0.

        if(run==1)
        {
            error = setpoint - (float)speed; //Difference between set and actual speed.
            prop_part = error * Kp; //PI Controller
            int_accum += error * Ki;
            output = prop_part + int_accum; //PI Output
        }
        else
        {
            error = 0;
            int_accum = 0;
            output = 0;
            RC2 = 0;
        }

        if(output < 0) output = 0;
        if(output > 1023) output = 1023;

        CCP1L = (int)output/4;

        //Passes variable data onto the function for displaying on the LCD.
        DisplayValues(0, (int)setpoint, speed, (int)error);
        DisplayValues(1, (int)prop_part, (int)int_accum, (int)output);
    }
}

```

Figure 4.3: Main Program

The Mathematical processes take place in the While loop where the 10-bit speed value is assembled from the high and low byte words procured and stowed as a Global Variable by the ISR. Run state is then identified depending on inputs on pins RAO and 1.

A key is taken from the ScanKeyPad function and if there is a valid input, it is amended to the setpoint by shunting the previous value held by multiplying by 10 and then disabling the Keypad until next loop through.

Good explanation of keypad entry

Two if statements ensure that the DC Motor setpoint does not exceed 2300 or goes 'below' 0.

If the DC Motor is in the 'run' state, the setpoint is compared to the current actual speed as stored by the program, and then calculates the error between the two values. The Proportional and Integral values are then calculated using the error and summed into an Output.

Only divided by 4 for value of 256 for testing. This was later changed to 1024 for 10 bit operation.

The Output is then divided by 4 and stored as CCPR1L (Duty Cycle) for the PWM.

The calculated values along with the setpoint are sent onto the DisplayValues Function to be shown on the LCD.

The next bits of code are in the Functions, starting with DisplaySpeed. This Function breaks up the read 'value' in Thousands, Hundreds, Tens, and Units through division and remainder operations. Those values are then put into an alphanumeric string to show the speed value along with RPM, then returned to DisplayString with which line to write to on the LCD.

```
//////////////////////////Functions//////////////////////////////////////////
void DisplaySpeed(int line, int value)
{
    int units, tens, hunds, thous, temp;
    unsigned char string[17];

    thous = value /1000;
    hunds = value %1000/100;
    tens = value %1000%100/10;
    units = value %1000%100%10;

    string[0] = thous + 48;
    string[1] = hunds + 48;
    string[2] = tens + 48;
    string[3] = units + 48;
    string[4] = ' ';
    string[5] = 'R';
    string[6] = 'P';
    string[7] = 'M';
    string[8] = 0;

    DisplayString(line, string);    //Return string to Line 1 of LCD
}
```

Figure 4.4: DisplaySpeed

DisplayValues is explained after Figure 4.5.

```
void DisplayValues(int line, int value1, int value2, int value3)
{
    int units, tens, hunds, thous, temp;
    unsigned char string[17];

    if(value1>=0)    string[0] = '+';
    else
    {
        string[0] = '-';
        value1 = value1*(-1);
    }

    thous = value1 /1000;
    hunds = value1 %1000/100;
    tens = value1 %1000%100/10;
    units = value1 %1000%100%10;

    string[1] = thous + 48;
    string[2] = hunds + 48;
    string[3] = tens + 48;
    string[4] = units + 48;

    if(value2>=0)    string[5] = '+';
    else
    {
        string[5] = '-';
        value2 = value2*(-1);
    }
    thous = value2 /1000;
    hunds = value2 %1000/100;
    tens = value2 %1000%100/10;
    units = value2 %1000%100%10;

    string[6] = thous + 48;
    string[7] = hunds + 48;
    string[8] = tens + 48;
    string[9] = units + 48;

    if(value3>=0)    string[10] = '+';
    else
    {
        string[10] = '-';
        value3 = value3*(-1);
    }
    thous = value3 /1000;
    hunds = value3 %1000/100;
    tens = value3 %1000%100/10;
    units = value3 %1000%100%10;

    string[11] = thous + 48;
    string[12] = hunds + 48;
    string[13] = tens + 48;
    string[14] = units + 48;
    string[15] = 0;

    DisplayString(line, string);    //Return string to Line 2 of LCD
}
```

Figure 4.5: DisplayValues

After declaring int and char variables, an if statement is used to apply a +ve or -ve symbol depending on whether value1 is greater than or less than 0.

Then just like in DisplaySpeed, the value is broken up into Thousands, Hundreds, Tens, and Units, then appended to a string. It repeats this process for two other values and sends to Line 2 of the LCD as a returned concatenated string.

The final two elements are the ScanKeypad and the Delay Functions. The ScanKeypad functions monitors keypresses on the Keypad peripheral linked to PORT D and cross references the circuits completed on specific columns and rows. The single value is returned to the calling element.

The Delay Function simply introduces a timed delay depending on the value sent with the call.

```
int ScanKeypad(void)
{
    int key=-1;

    RD0 = 1; //scan column 1
    Delay(1);
    if(RD4 == 1) key = 1; //scan row A
    if(RD5 == 1) key = 4; //scan row B
    if(RD6 == 1) key = 7; //scan row C
    if(RD7 == 1) key = '+'; //scan row D
    RD0 = 0; //disable column 1

    RD1 = 1; //scan column 2
    Delay(1);
    if(RD4 == 1) key = 2; //scan row A
    if(RD5 == 1) key = 5; //scan row B
    if(RD6 == 1) key = 8; //scan row C
    if(RD7 == 1) key = 0; //scan row D
    RD1 = 0; //disable column 2

    RD2 = 1; //scan column 3
    Delay(1);
    if(RD4 == 1) key = 3; //scan row A
    if(RD5 == 1) key = 6; //scan row B
    if(RD6 == 1) key = 9; //scan row C
    if(RD7 == 1) key = '#'; //scan row D
    RD2 = 0; //disable column 3

    return key;
}

void Delay(int t)
{
    int i;
    for(i=0;i<t;i++);
}
```

Figure 4.6: ScanKeypad and Delay

This is self explanatory with columns and rows. The column

5 TESTING AND ANALYSIS

5.1 SIMULATION

To verify the code for the PWM functioned as intended, a simulation was carried out in Proteus using a Virtual Oscilloscope. Due to limitations with the Proteus application, the maximum testable speed was 360 rpm, so incremental tests of 150, 300, and 350 were recorded to achieve at least three discernible results.

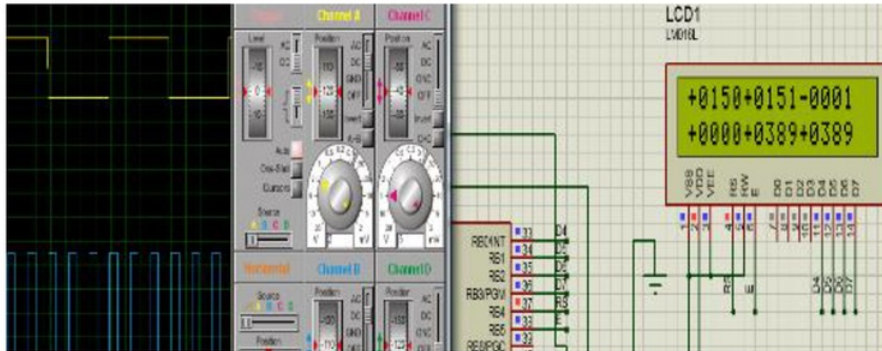


Figure 5.1: 150 Setpoint

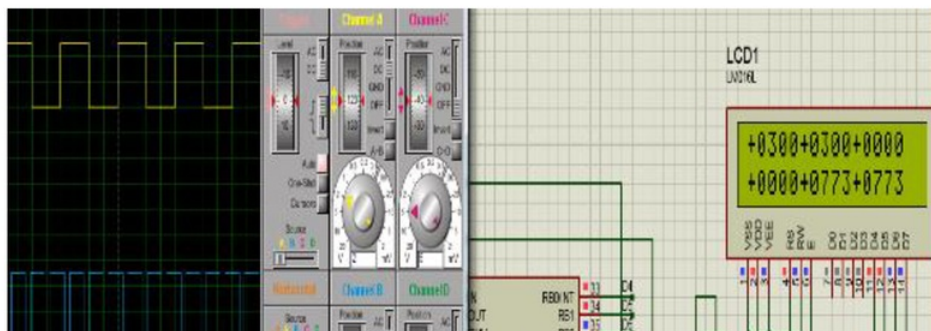


Figure 5.2: 300 Setpoint

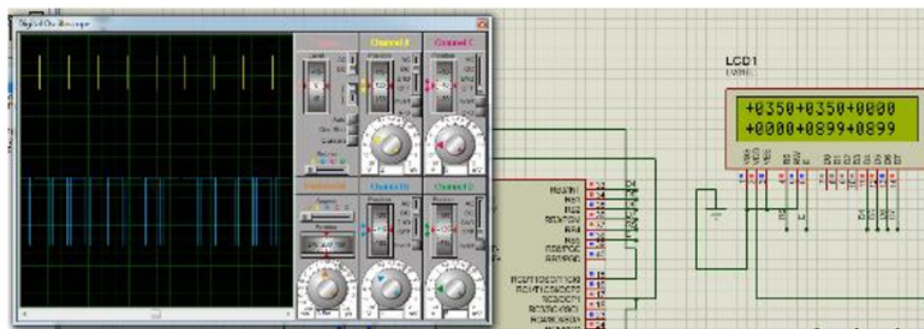


Figure 5.3: 350 Setpoint

From the three results, calculations can be carried out to verify the accuracy of the program.

$$Frequency = \frac{1}{divisions \times timebase}$$

$$PPR = 100$$

$$RPM = Frequency \times \frac{60}{Pulses Per Revolution}$$

Using these equations, the first test was calculated thus;

For 150RPM

$$Frequency = \frac{1}{4} = 250Hz$$

$$PPR = 100$$

$$250 \times \frac{60}{100} = 150RPM$$

For 300RPM

$$Frequency = \frac{1}{2} = 500Hz$$

$$PPR = 100$$

$$500 \times \frac{60}{100} = 300RPM$$

For 350RPM

$$Frequency = \frac{1}{1.71429} = 583.33Hz$$

$$PPR = 100$$

$$583.33 \times \frac{60}{100} = 350RPM$$

As these calculated values match the simulated. It can be determined that the PWM function works.

The next step is the Practical Test.

*Good testing with CAD system
using various set points.*

5.2 PRACTICAL

5.2.1 Build

The Practical build was setup using components listed in System Hardware, and connected to a variable Power supply to power the motor and board.

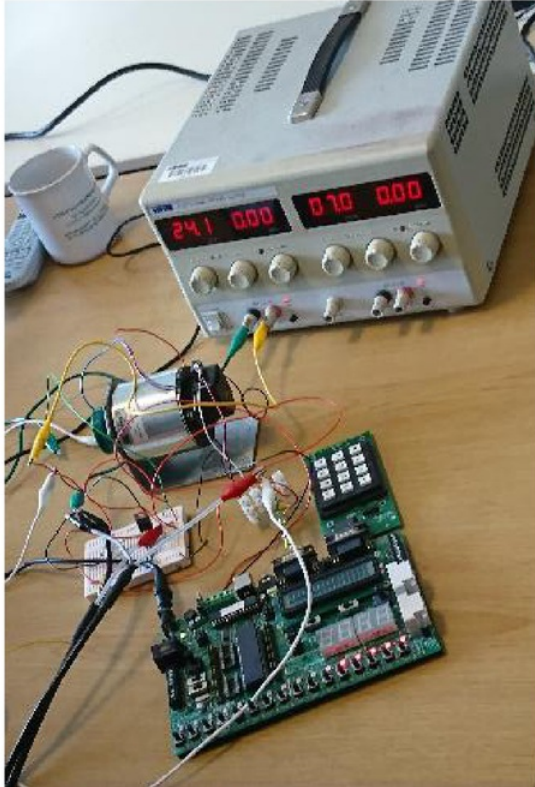


Figure 5.4: Practical Setup



Figure 5.5: LCD Display

Good image which shows the setup under test.

5.2.2 General Test

The following test plan was written to match results against criteria for success.

Test Method	Expected Result	Observed Result
1. Choose Setpoint	Motor should run at selected speed and maintain speed.	Pass
2. Press # to Stop	Speed is set to 0.	Pass
3. PWM Shift	PWM shift according to new Setpoints	Pass/Evidence in Figures
4. Apply physical braking to shaft of motor	Increase in Duty Cycle to compensate to maintain Setpoint	Pass
5. LCD Display	Program values show on 2 Lines	Pass

Table 5.1: Practical Test

Good use of table for analysis.

Further tests were required to ensure speed truly was accurate across ranges.

5.2.3 Setpoint and PWM at 200rpm

Setpoint 200 RPM

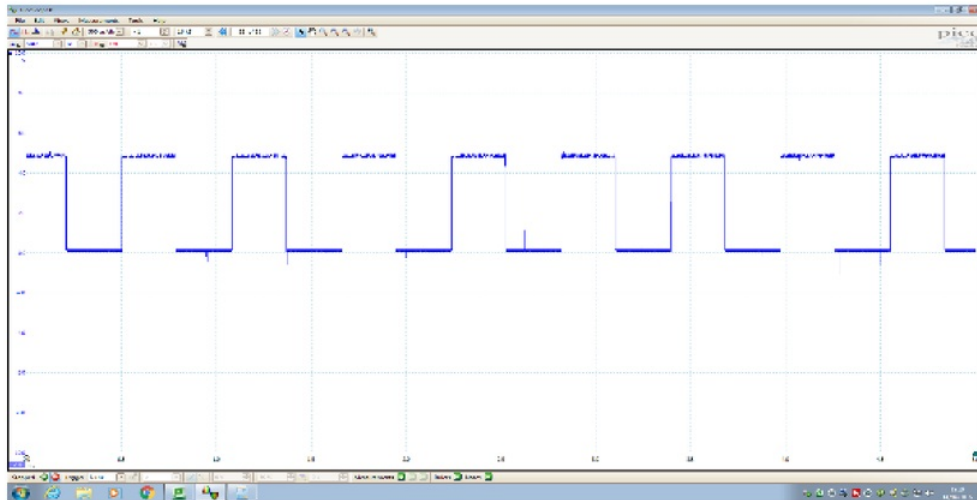


Figure 5.6: Setpoint @ 200RPM

$$Frequency = \frac{1}{1.2(div) \times 0.5(t \text{ per div})} = 1666Hz$$

$$RPM = 1666 \times \frac{60}{500} = 200RPM$$

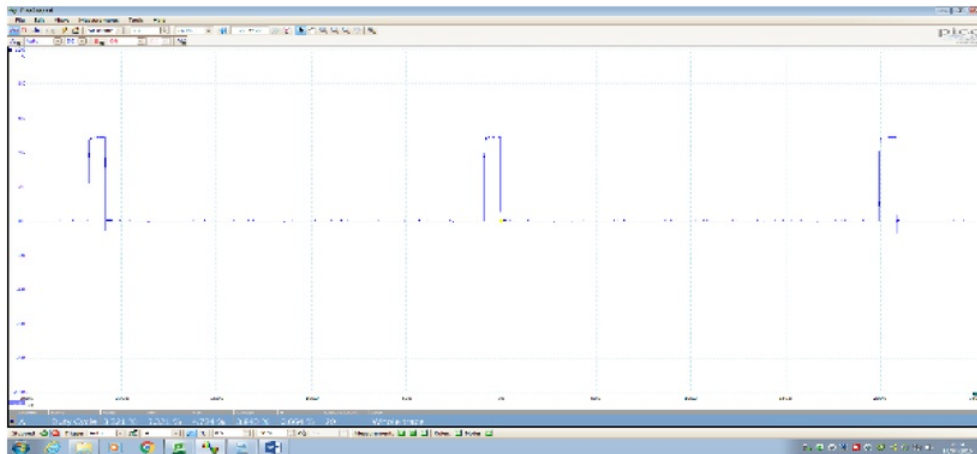


Figure 5.7: PWM @ 200

$$Duty Cycle = 48 \times \frac{100}{1023(100\% \text{ Duty})} = 4.7\%$$

5.2.4 Setpoint and PWM at 1000RPM

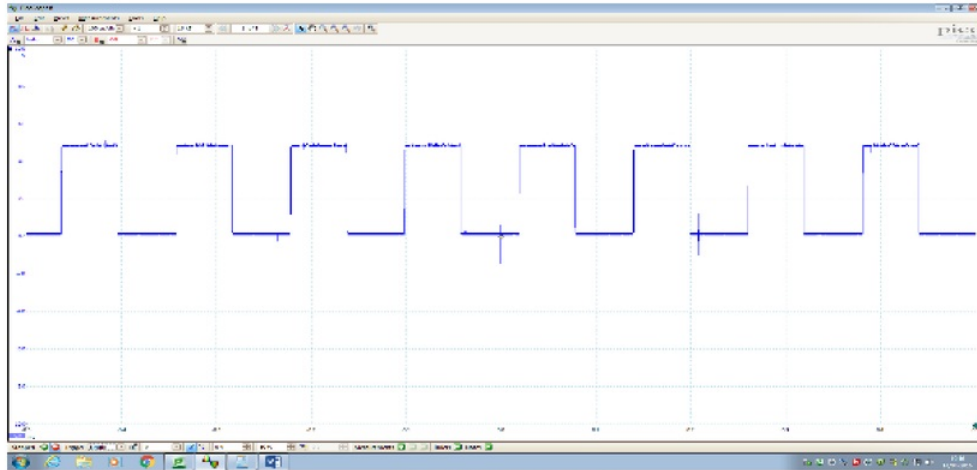


Figure 5.8: Setpoint @ 1000RPM

$$\text{Frequency} = \frac{1}{1.2(\text{div}) \times 0.1(\text{t per div})} = 8333\text{Hz}$$

$$\text{RPM} = 8333 \times \frac{60}{500} = 999.6\text{RPM}$$

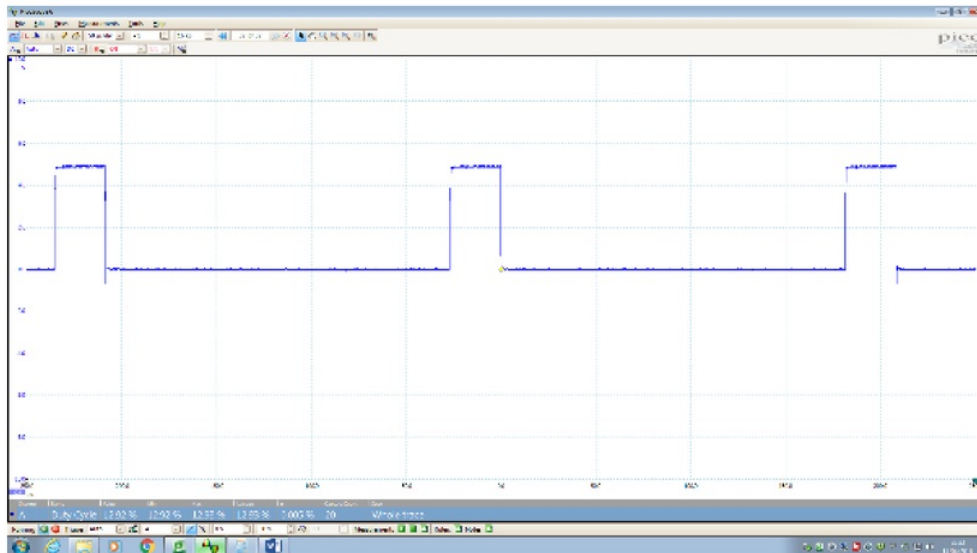


Figure 5.9: PWM @ 1000

$$\text{Duty Cycle} = 133 \times \frac{100}{1023(100\% \text{ Duty})} = 13.3\%$$

5.2.5 Setpoint and PWM at 2300 RPM

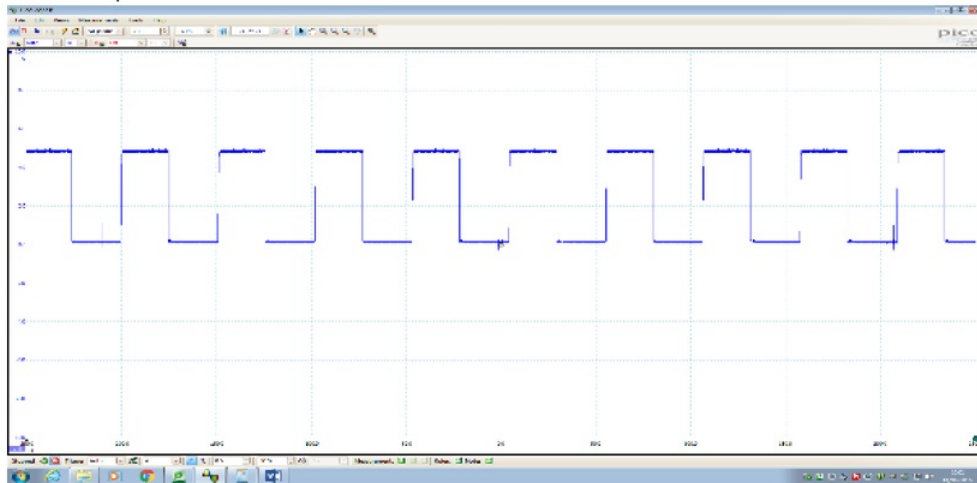


Figure 5.10: Setpoint @ 2300RPM

$$Frequency = \frac{1}{1.0(div) \times 0.05(t \text{ per div})} = 19000Hz$$

$$RPM = 19000 \times \frac{60}{500} = 2280RPM$$

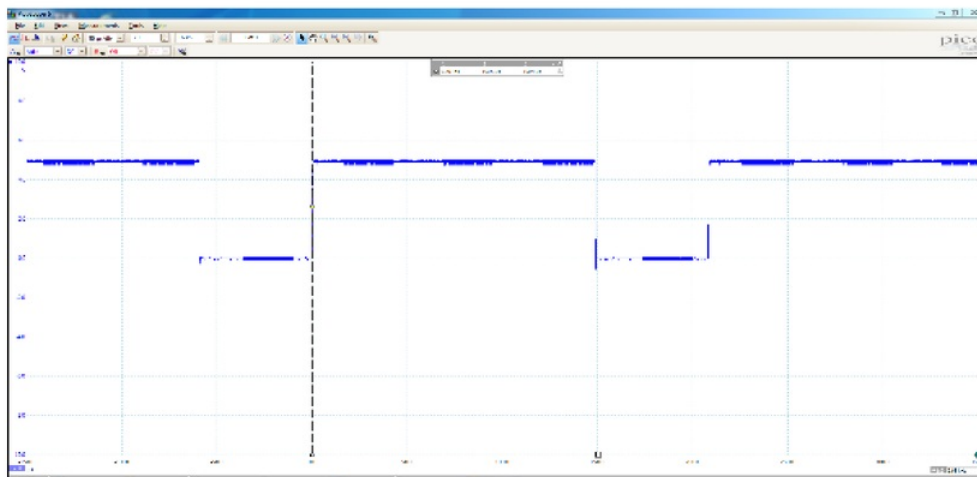


Figure 5.11: PWM @ 2300

$$Duty Cycle = 744 \times \frac{100}{1023(100\% \text{ Duty})} = 72.72\%$$

5.2.6 Observation

What is seen from the practical results is that the PIC and code functions as designed, with the Duty Cycle being adjusted by new setpoints and the DC Motor speeding up accordingly and maintaining speed using the PI controller.

Good test results for the dev board practical.

6 DISCUSSION AND CONCLUSIONS

In the development of the PIC based Controller Programmed in the C Programming Language, it is shown that using a PIC is an effective means of controlling DC Motor speed.

While the implementation made use of a development board, it is possible to now design a control panel featuring only a keypad, LCD display and PIC controller linked to the Encoder and Motor through a MOSFET for protection. *MOSFET is needed as*

interface for motor which operates at 24V
It was noted there were limitations in the Proteus software in simulating a speed greater than 360RPM, however it is possible that the Multisim application produced by National Instruments could provide a viable alternative. Picoscope software and hardware captured the performance of the system accurately enough, to draw the conclusion about the performance of the practical setup.

An excellent advantage of the solution developed here, is that due to the portability of the code, setups can be made using Arduinos, Raspberry Pis and other Microcontrollers/Small form PCs.

A good point here.

7 WORKS CITED

Ltd, Matrix Technology Solutions. (2018, 04 20). *PICmicro MCU development board (version 3) - Matrix TSL*. Retrieved from Matrix:

<http://www.matrixtsl.com/resources/getresource.php?id=175>

Microchip. (2018, 04 20). *PIC16F87X - Microchip Technology*. Retrieved from Microchip:

<http://ww1.microchip.com/downloads/en/DeviceDoc/30292D.pdf>

EWDyer_PSDE5013-Assignment_2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

55/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

Text Comment. Far too brief. This could have pointed out the method of development using prototyping.

PAGE 5

PAGE 6

PAGE 7

PAGE 8

Text Comment. The explanations are good. The interface for driving the PWM and receiving the encoder pulses should be included here.

PAGE 9

PAGE 10

PAGE 11

Text Comment. T1CON should be included to explain the event mode operation with TMR0.

PAGE 12

Text Comment. Good explanation of code

PAGE 13

PAGE 14

Text Comment. Good explanation of keypad entry.

Text Comment. Only divided by 4 for value of 256 for testing. This was later changed to 1024 for 10 bit operation.

PAGE 15

PAGE 16

Text Comment. Good. The functions do operate exactly the same except with more arguments.

Text Comment. This is self explanatory with columns and rows. The columns are driven high as outputs and the row inputs are then monitored.

PAGE 17

PAGE 18

Text Comment. Good testing with CAD system using various set points.

PAGE 19

Text Comment. Good image which shows the setup under test.

Text Comment. Good use of table for analysis.

PAGE 20

PAGE 21

PAGE 22

Text Comment. Good test results for the dev board practical.

PAGE 23

Text Comment. MOSFET is needed as interface for motor which operates at 24V

Text Comment. A good point here.

PAGE 24

GRADING FORM: CSG-RUBIC

RESEARCH-INFORMED LITERATURE

 n/a

KNOWLEDGE AND UNDERSTANDING

 The introduction is very brief. 1/5 The hardware description is good. The interfacing for PWM and encoder input should also be included here. 2.5/5 The prototyping process has not been explained very well. There should be images of the prototypes leading to the final system in the CAD. The final design in CAD and the dev board are present with testing. The configuration is mostly explained. 11/20 The software design is explained clearly. There are minor errors and some minor parts missing. 5.5/10

20/40

ANALYSIS

 The tests are complete. The system is tested as a complete system and does not explain the individual tests done with prototypes. However the final systems both CAD and dev board are tested. The tests should include the test under external force of disturbance. 6.5/10 The conclusion should include the prototype method of design development. The tests are good but this section should also include the performance of the system with external disturbance. 5.5/10

12/20

PRACTICAL APPLICATION AND DEP.



Software design is explained, but does not explain the prototyping process evolution for the practical application. 5.5/10 Quality of work is good. All aspects of the build have been good. The prototyping process would have given a better idea of the system development which would enhance the report. 5.5/10

11/20

SKILLS FOR PROFESSIONAL PRAC.



The report is good and easy to follow, but there are missing parts. 5.5/10 Team work is good 6.5/10

12/20