

Scott Dedoncker Ass 2 FINAL SUB

by Scott Dedoncker

FILE	12968_SCOTT_DEDONCKER_SCOTT_DEDONCKER_ASS_2_FINAL_SUB_145116_2047539283.DOCX		
TIME SUBMITTED	24-MAY-2018 02:22PM (UTC+0100)	WORD COUNT	3541
SUBMISSION ID	89853772	CHARACTER COUNT	23158

HND Electrical & Electronic Engineering

PSDE 5013: Configuration & Programming of Embedded Systems



Assignment 2

DC Motor Controller

Scott Dedoncker

Valero Energy Electrical Apprentice

Date of Submission: 24/05/18

Word Count: 3577

Contents

1. System Hardware.....	3
1.1. Microcontroller PIC.....	3
1.2. Keypad Setup	4
1.3. Closed Loop PI Controller.....	4
1.4. E-Blocks PICmicro MCU Board	5
1.5. DC Motor.....	6
2. System Design	7
2.1. Timer 0	7
2.2. Timer 1	8
2.3. Timer 2	9
2.4. INTCON & OPTION registers	10
3. Software Design & Explanation.....	12
3.1. Header Files, Function Prototypes & Global Declarations.....	12
3.2. Interrupt Service Routine.....	13
3.3. Main Program	14
3.4. Main Program Super Loop	15
3.5. Display Speed Function	16
3.6. Display Values Function	17
3.7. Scan Keypad Function	18
3.8. Delay Function	19
4. Testing & Analysis	20
4.1. Simulation Testing of the PWM	20
4.2. Testing the Keypad.....	22
4.3. Interface Functionality Test	23
4.4. Practical Testing	24
5. Conclusions	27
References	28

Figure 1 - PIC16F877 Microcontroller	3
Figure 2 - Microcontroller Comparison (Piclist)	3
Figure 3 - 4x3 Keypad Matrix (Rubasinghe)	4
Figure 4 - HEDS-5540 (AVAGO Technologies)	4
Figure 5 - HP488 Board (E-Blocks).....	5
Figure 6 - Timer 0 Architecture (Microchip, 2003)	7
Figure 7 - Timer 1 Architecture (Microchip, 2003)	8
Figure 8 – T1CON Register (Microchip, 2003).....	8
Figure 9 – Timer 2 Architecture (Microchip, 2003).....	9
Figure 10 - T2CON Register (Microchip, 2003)	9
Figure 11 - OPTION_REG (The Bare Necessities, n.d.)	10
Figure 12 - INTCON Register (The Bare Necessities, n.d.).....	11
Figure 13 - Header Files, Function Prototypes & Global Declarations.....	12
Figure 14 - Interrupt service routine.....	13
Figure 15 - Main Program	14
Figure 16 - Super Loop	15
Figure 17 - Display Speed Function	16
Figure 18 - Display Values Function	17
Figure 19 - Scan Keypad Function	18
Figure 20 - Delay Function	19
Figure 21 - 300 RPM Sim	20
Figure 22 - 150 RPM Sim	21
Figure 23 - PWM Sim Results Table	21
Figure 24 - Keypad Simulation	22
Figure 25 - Keypad Testing	22
Figure 26 - System Function Testing	23
Figure 27 - LCD Screen Practical.....	23
Figure 28 - Practical Build	24
Figure 29 - 200 RPM Set point	25
Figure 30 - PWM at 200 RPM.....	25
Figure 31 - 2000 RPM Setpoint	26
Figure 32 - PWM at 2000 RPM	26

1. System Hardware

1.1. Microcontroller PIC

The first stage of this assignment was to select the correct microcontroller chip for this process. After considering multiple options the chip that was selected was the PIC16F877. This chips primary design is suited to parallel communication and was perfectly suited for this task. It is a 40 pin chip with multiple I/O's that can be mapped to suit. The top view with all labelled ports can be seen in the figure below. A table comparing features of this chip with others can also be seen in figure 2.

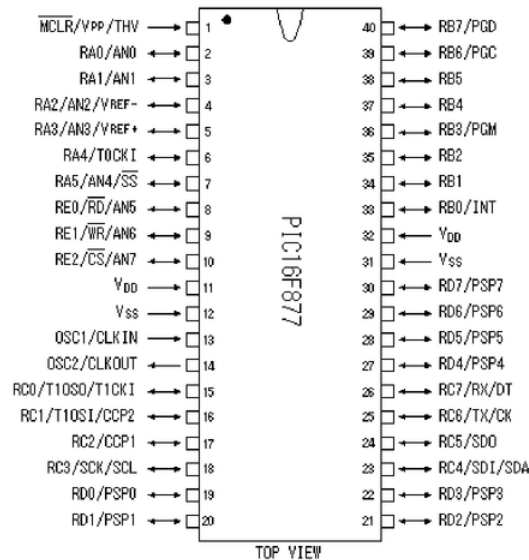


Figure 1 - PIC16F877 Microcontroller

Key Features	PIC16F873	PIC16F874	PIC16F876	PIC16F877
MAX Operating Frequency	20MHz	20MHz	20MHz	20MHz
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
I/O Ports	RA0-5 (6) RB0-7 (8) RC0-7 (8)	RA0-5 (6) RB0-7 (8) RC0-7 (8) RD0-7 (8) RE0-2 (3)	RA0-5 (6) RB0-7 (8) RC0-7 (8)	RA0-5 (6) RB0-7 (8) RC0-7 (8) RD0-7 (8) RE0-2 (3)
Timers	3	3	3	3
CCP	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	-	PSP	-	PSP
10-bit Analog-to-Digital Module	5 Channels	8 Channels	5 Channels	8 Channels
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions
Pins (DIP)	28 Pins	40 Pins	28 Pins	40 Pins

Figure 2 - Microcontroller Comparison (Piclist)

1.2. Keypad Setup

The exam criteria specified that a keypad used to control the speed set point and give the option to adjust the speed set point needed to be implemented. This keypad needed to be a 12-button pad with numbers 0-9 as well as an "*" and "#". The keypad design would use the typical 4x3 matrix keypad setup where there would be 3 vertical output pins covering the vertical keypad columns, and 4 horizontal output pins where the 4 horizontal columns would be covered. This can be seen in figure 3.

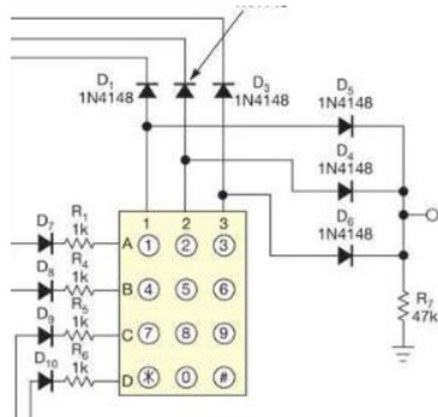


Figure 3 - 4x3 Keypad Matrix (Rubasinghe)

Good description of how the hardware (keypad) has to be connected to the MCU.

1.3. Closed Loop PI Controller

In order to control the speed of the motor to an accurate degree a closed loop PI controller would also be incorporated into the design. For this a HEDS 5540 incremental encoder was used and incorporated into the design. This would in turn give out a control burst of 500 pulses per revolution of the motor to monitor its speed. This is an incredibly accurate ratio and would give an accurate speed reading of the motor. An image from the manufacturer's data sheet can be seen below.

HEDS-5500/5540, HEDM-5500/5540

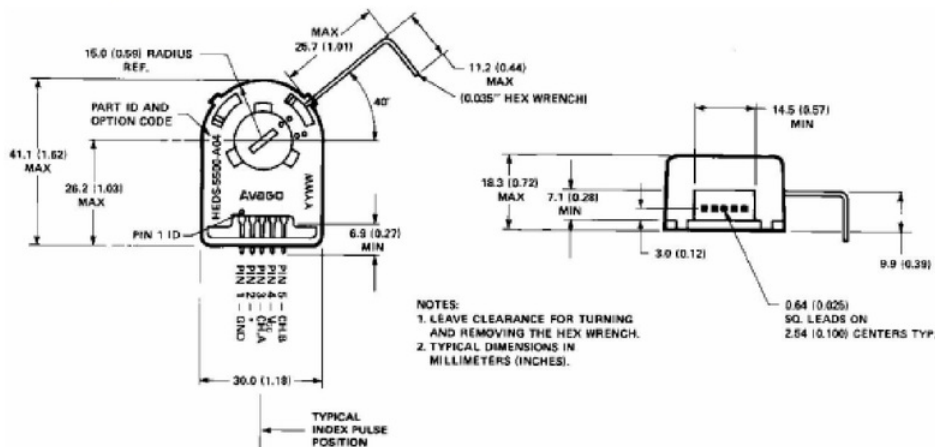


Figure 4 - HEDS-5540 (AVAGO Technologies)

1.4. E-Blocks PICmicro MCU Board

All of the features necessary for this system could be found on the E-blocks Micro PIC MCU Board, the HP488. It included the correct Pic microcontroller, various buttons and switches that could be adapted for the design, as well as an LCD screen to display a clear, real time interface of the motor's running variables. This board is ideal for the application of this assignment as it is flexible in how it can be used and programmed, with a multitude of ports, the LED screen accompanied by a 7 segment display(s), on board sensors as well as many more key features.

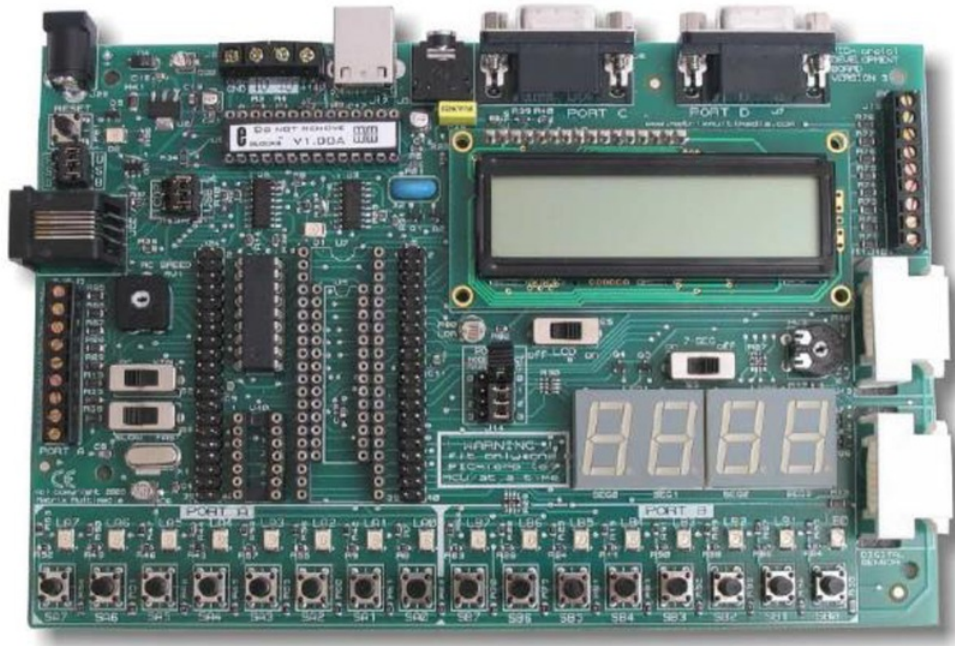
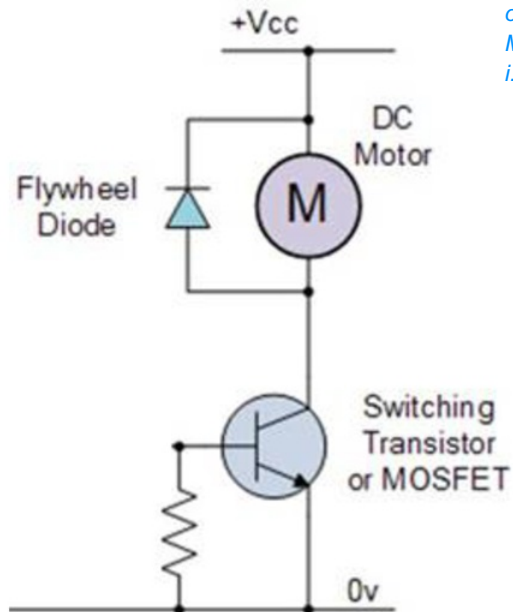


Figure 5 - HP488 Board (E-Blocks)

1.5. DC Motor

The final piece of hardware needed for this assignment was the DC motor that needed to have its speed monitored. In order for this to work and the devices to be compatible, a MOSFET transistor needed to be used to the motor to the required PWM while also activating the MOSFET's gate. This is due to the microcontroller and DC motor having different operating voltages. The MOSFET would make sure that the two voltages remain isolated from each other protecting each device. A flywheel diode would also be used to dissipate the back E.M.F from the motor. This can be seen in the figure below. This is what's known as a power MOSFET motor driver.

The gate is controlled by the MCU output PWM and the output of the MOSFET will work at the PS voltage i.e. 24V. This is only part explained.



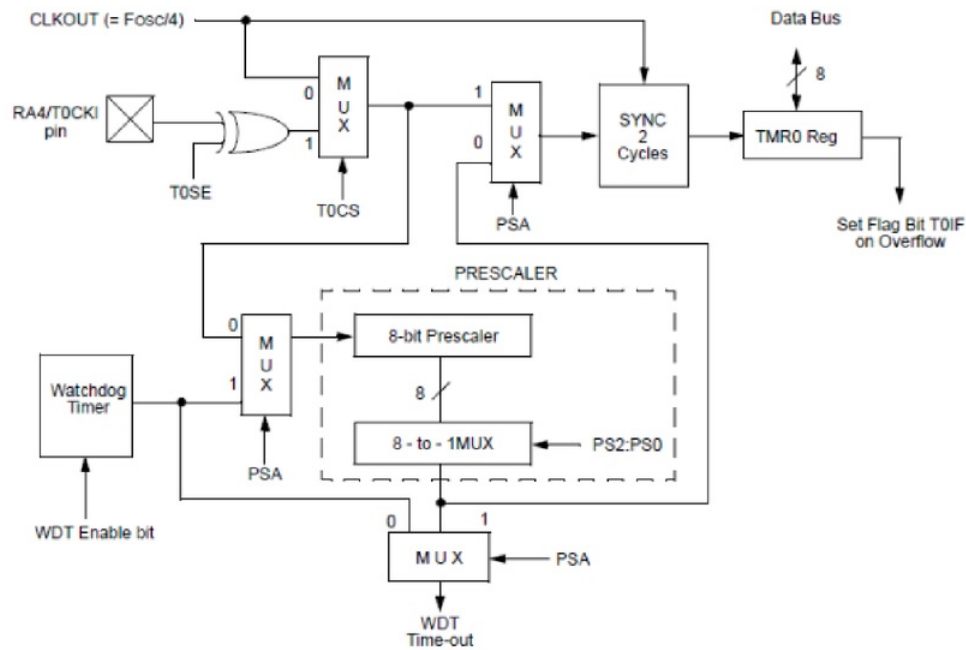
(DNA Tech, n.d.)

2. System Design

In order for the code to be correctly designed certain documentation had to be assessed including the correct registers and data sheets.

2.1. Timer 0

The first document that needed to be reviewed was the Timer 0 architecture and its internal PIC build. This document was taken from the manufacturer's data sheets. It shows the build of multiplexers and prescalers used to define the system. This timer would utilise the INTCON and OPTION_REG registers.



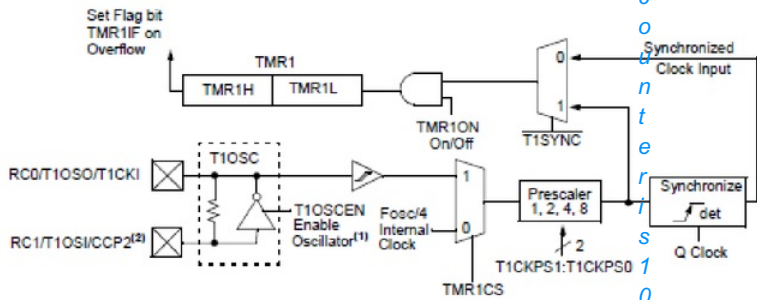
Note: T0CS, T0SE, PSA, PS2:PS0 are (OPTION_REG<5:0>).

Figure 6 - Timer 0 Architecture (Microchip, 2003)

2.2. Timer 1

Timer 1 would be used as the secondary timer function for the system. It would be used on the PIC16877 as a 16 bit counter. The maximum value it can read to is 65535. This timer differs from timer 0 in that it uses a 16 bit timer divided into two equalling two 8 bit timers. One of these 8-bit counters would store the higher values while the other 8-bit counter would store the lower value. This would supply a feedback into the PIC in port RCO. A diagram of the timer can once again be found in the manufacturer's data sheets. This timer would utilise the T1CON register also seen below. It would utilise a bit setting of:

T1CON = 0b00000111



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain

Figure 7 - Timer 1 Architecture (Microchip, 2003)

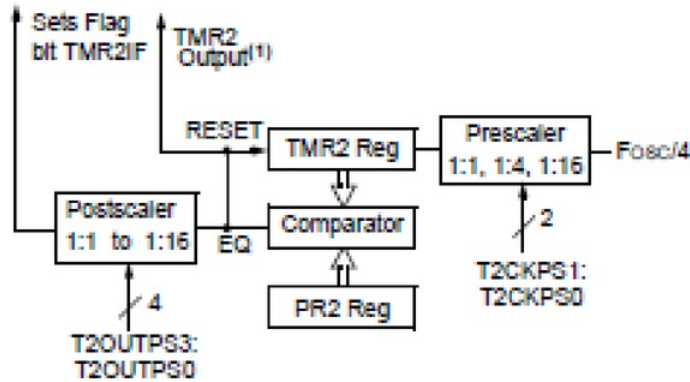
	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
	bit 7							bit 0
bit 7-6	Unimplemented: Read as '0'							
bit 5-4	T1CKPS1:T1CKPS0: Timer1 Input Clock Prescale Select bits 11 = 1:8 Prescale value 10 = 1:4 Prescale value 01 = 1:2 Prescale value 00 = 1:1 Prescale value							
bit 3	T1OSCEN: Timer1 Oscillator Enable Control bit 1 = Oscillator is enabled 0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)							
bit 2	T1SYNC: Timer1 External Clock Input Synchronization Control bit When TMR1CS = 1: 1 = Do not synchronize external clock input 0 = Synchronize external clock input When TMR1CS = 0: This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.							
bit 1	TMR1CS: Timer1 Clock Source Select bit 1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge) 0 = Internal clock (FOSC/4)							
bit 0	TMR1ON: Timer1 On bit 1 = Enables Timer1 0 = Stops Timer1							

Figure 8 – T1CON Register (Microchip, 2003)

2.3. Timer 2

The third timer used in the system was Timer 2. This would interface with the PWM for monitoring the DC motor speed. This counter had a max value of 256 and would therefore only utilise one 8-bit counter. The architecture for timer 2 can be seen below along with the register used to set up this timer. It would use the T2CON timer and be set to:

TMR2 = 0b00000101



Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

Figure 9 – Timer 2 Architecture (Microchip, 2003)

	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	
	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0	

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0:** Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0:** Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Figure 10 - T2CON Register (Microchip, 2003)

2.4. INTCON & OPTION registers

The program will need to utilise data from two registers in order to function properly. This is due to all interrupts having two bit values, an input and an output, and each will be supplied by a universally used register. One of the two registers utilised is the OPTION_REG register and its data. Bit 5 is the piece of data that will be derived from this table and used in the assignment, as it is the one using TMR0 and transitions on the RA4 pin, the one that will be connected to a button used to interrupt the cycle.

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RBPB	INTEDG	T0CS	T0SE	PSA	PS2	PS0
bit 7	bit 0						
bit 7	RBPB : PORTB Pull-up Enable bit 1 = PORTB pull-ups are disabled 0 = PORTB pull-ups are enabled by individual port latch values						
bit 6	INTEDG : Interrupt Edge Select bit 1 = Interrupt on rising edge of RB0/INT pin 0 = Interrupt on falling edge of RB0/INT pin						
bit 5	T0CS : TMR0 Clock Source Select bit 1 = Transition on RA4/T0CKI pin 0 = Internal instruction cycle clock (CLKOUT)						
bit 4	T0SE : TMR0 Source Edge Select bit 1 = Increment on high-to-low transition on RA4/T0CKI pin 0 = Increment on low-to-high transition on RA4/T0CKI pin						
bit 3	PSA : Prescaler Assignment bit 1 = Prescaler is assigned to the WDT 0 = Prescaler is assigned to the Timer0 module						
bit 2-0	PS2:PS0 : Prescaler Rate Select bits						
	Bit Value	TMR0 Rate	WDT Rate				
	000	1 : 2	1 : 1				
	001	1 : 4	1 : 2				
	010	1 : 8	1 : 4				
	011	1 : 16	1 : 8				
	100	1 : 32	1 : 16				
	101	1 : 64	1 : 32				
	110	1 : 128	1 : 64				
	111	1 : 256	1 : 128				

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figure 11 - OPTION_REG (The Bare Necessities, n.d.)

The second register used is the INTCON register as this would need to be used in conjunction with the first register. The INTCON register stands for interrupt control register, and is used to configure software interrupts, either starting or stopping processes in a system. The global interrupt enable/bit 7 is the most important part of this register as no interrupts will become active without this bit in effect. The 2 bits taken from this register will be bit 7 (the global interrupt enable) and bit 5 (timer 0 overflow enable). This will mean that when TIMER0 counts up to 256, it will intron make T0IF turn to logic 1. This will then mean that as T0IE is already logic 1 due to the TIMER0 being active, this will send a logic 1 signal through the system. This will cause the system to be interrupted.

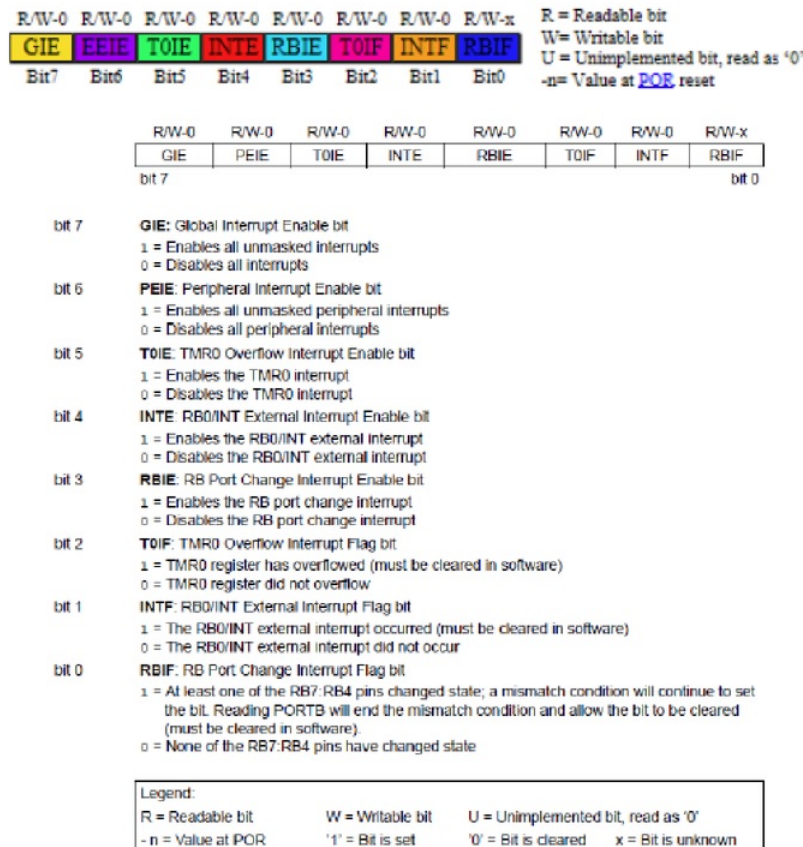


Figure 12 - INTCON Register (The Bare Necessities, n.d.)

3. Software Design & Explanation

The program was designed using two programs. The code was designed in MP Lab, and the circuit for simulation was designed using Proteus Project Suite.

PROTEUS

The code used the C programming language and was built using various functions and variables. An explanation of the code follows.

3.1. Header Files, Function Prototypes & Global Declarations

The first piece of code displayed below depicts the two header files used in this program. One is pic.h to recognise the C code, and the other file is a new header file, the LCD.h which is used to utilise the LCD screen of the microcontroller. This is how the motor speed and variables will be displayed on the LCD screen. The function prototypes are listed out next showing all their integer values that will need to be called in the system. The ScanKeypad function would use void instead of an integer value. All the global variables are then displayed, using two integer values and one float value. The variables would be called throughout the entire program.

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

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

int milliseconds=0, speed=0, fault=0;          //global variables
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;             //prac values: Kp = 0.01, Ki = 0.02;
```

*Reasonable explanation
which describes the initial
settings in the program.
These are in the correct
place in the program.*

Figure 13 - Header Files, Function Prototypes & Global Declarations

3.2. Interrupt Service Routine

The ISR (interrupt service routine) is what would be used throughout the program to pulse count and monitor the speed of the motor. There are three main variables in this piece of code; the TOIF, TMR1IF and TMR2IF. These are used routinely in this function with if statements to activate certain outcomes. The first if statement is used to detect the state of timer 0. If timer 0 (TOIF) is read as being active, or set to 1, the timer is reset and the counter is preloaded to 64. Milliseconds then begins to increment in steps of +10. Once 100 milliseconds is reached the sub if statement is activated. This sets both speed low and high byte to 0.

The second if statement is a flag for timer 1 and activates in the case of a fault where the system over counts. This simply sets fault to active.

The third if statement simply resets timer 2's flag to 0.

```
void interrupt ISR(void)                                //there are 3 possible interrupt sources
{
    if(TOIF==1)                                         //timer 0 is timing, interrupt every 10ms
    {
        TOIF = 0;
        TMR0 = 64;
        milliseconds+=10;
        if((milliseconds==100) && (fault==0))
        {
            speed_low_byte = TMR1L;
            speed_high_byte = TMR1H;
            TMR1L = 0;
            TMR1H = 0;
            milliseconds=0;
        }
    }

    if(TMR1IF==1)                                       //timer 1 is in event mode counting pulses
    {
        TMR1IF = 0;                                     //clear the interrupt flag
        fault = 1;
    }

    if(TMR2IF==1)                                       //timer 2 is for PWM
    {
        TMR2IF = 0;
    }
}
```

The program here is correct. The explanation is logical.

Figure 14 - Interrupt service routine

3.3. Main Program

The next section of code details the main program. This is where all other functions would be called to and the main program process carried out. The first line in this piece of code detects an integer value for the keypad to be set to 1, but with no specific raw input. The binary values for ADCON1, TRISB, TRISC, TRISD and PORTB are all then set. These will be used for the LCD display, the RC0 input counting with TMR1, and the output of RC2 for the PWM. The peripheral interrupts are then setup to their correct binary values followed by the two main registers of INTCON and OPTION_REG. Finally values for T1CON, T2CON and CCP1CON are all set up. This part of the main program primarily sets up all the registers before the main process.

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

    ADCON1 = 0x8E;
    TRISB = 0x00;
    TRISC = 0x01;

    TRISD = 0b11110000;

    PORTB = 0x00;
    LCD_Startup();

    PIE1 = 0b00000011;
    PIE2 = 0;
    PIR1 = 0;
    PIR2 = 0;

    INTCON = 0b11100000;
    OPTION_REG = 0b10000111;

    T1CON = 0b00000111;
    TMR1L = 0;
    TMR1H = 0;

    PR2 = 255;
    CCPR1L = 0;
    T2CON = 0b00000101;
    CCP1CON = 0b00001100;
```

*There should be
code comments
here to identify the
function of each line.
The significance of
these registers
should be identified.*

Figure 15 - Main Program

3.4. Main Program Super Loop

The next set of code utilises a super loop to process its internal code continuously while the system is running. The first line reads the pulses from RCO taking both the low and high speed bytes from the two 8-bit counters and multiplying them by the prescaler. This works out the current speed of the motor. The two 8-bit values are taken from the interrupt service routine. Following this are two if statements used to determine outcomes based on the state of RA1 and RA2. This is to determine the state of whether the start or stop buttons on the keypad have been pressed. If RA1 is active and RA2 not, then the motor will run. If otherwise then the motor will not run.

```
while(1)
{
    speed = (int)((float)((speed_low_byte + speed_high_byte*256)*6)/5);
    speed = speed * 3.1;

    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;
        enable_keypad=0;
    }

    // if((key=='$') && (enable_keypad==1))
    if(key=='$')
    {
        setpoint = 0;
    }

    if(key==-1)        enable_keypad=1;

    if(setpoint>2000)   setpoint = 2000;
    if(setpoint<0)     setpoint = 0;

    if(run==1)
    {
        error = setpoint - (float)speed;
        prop_part = error * Kp;
        int_accum += error * Ki;
        output = prop_part + int_accum;
    }
    else
    {
        error = 0;
        int_accum = 0;
        output = 0;
        RC2 = 0;
    }

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

    CCP1L = (int)output;

    DisplayValues(0, (int)setpoint, speed, (int)error);
    DisplayValues(1, (int)prop_part, (int)int_accum, (int)output);
}
```

Figure 16 - Super Loop

These function calls are significant. This is where the data arguments are called to the function prototypes which enable the data to be displayed on LCD.

The program code is correct but some of the code such as the set point constraints are not explained. The code should have comments to indicate the purpose of each line.

The 256 value is not the prescaler. It is the number to shift the 2 150's in the register 8 places for 10 bit resolution.

The Key variable is then assigned to which ever key is pressed on the keypad, hence the ScanKeypad function being called. The following if statements are used to correlate the keys pressed on the keypad to what is shown on the LCD screen. It first checks to see if a key has been pressed, and then assesses each depression of a key on the keypad one press at a time to derive clear functionality between the Keypad and LCD screen.

The latch 'enable-keypad'

The next section of code is part of the PI calculation. This only activates when the run variable is activated, meaning the motor is running. This takes the recorded speed away from the setpoint value and gives a value that is multiplied by kp. The error output is then given as the proportional value plus the accumulated value.

The next two if statements set the max and minimum run speeds of the motor to be displayed at 0 – 2300 RPM. The final two function calls are used to display on the LCD screen.

10 bit PWM.

3.5. Display Speed Function

The next function shown below is code that is no longer within the main program. This means this code would be function called into the program. Two integer variables of value and line are declared before the function begins. Then integer values for units, tens, hundreds, thousands and temperature are all declared followed by an unassigned char string with 17 values.

Calculations are then performed to determine 4 digit numbers as their respective units. Following this is an array string. This 8 bit array is used to represent each unit on the LCD Display. Each component has 48 added onto it to convert it into the ASCII equivalent value. String 4 then shows a space, followed by the letters RPM that would be displayed on the LCD screen after the speed value.

The final line of code function calls the DisplayString function which sends the data from this display function to line 1 of the LCD.

```
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);
}
```

This explains the function fairly well.

Code comments should be used.

Figure 17 - Display Speed Function

T
h
e
c
o
d
e
l
o
o
k
s
c
o
r
r
e
c
t
a
n
d
c
h
e
c
k
e
d
i
n
t
h
e

3.6. Display Values Function

The next function is the DisplayValues function and works on a very similar principle to the previous function. This function displays the top line of the LCD detailing error, speed and setpoint of the motor. This function otherwise works in the same principle to the display speed function.

```
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);    //send string data to LCD line 1
}
```

This is the final part of the display code for the program and should have some code comments to indicate the function of each section.

Figure 18 - Display Values Function

3.7. Scan Keypad Function

The ScanKeypad function is the next function shown below. This function simply determines which key is being depressed on the keypad and assigns it the correct value. It begins by declaring a local variable integer of key = -1. It then scans each vertical column of the keypad and assigns the correct number as such. After each vertical column is checked it is then disabled before moving onto the next column. At the end of the function it then returns the key variable.

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

    RD0 = 1;                                //scan column 1
    Delay(1);
    if(RD4 == 1)    key = 1;
    if(RD5 == 1)    key = 4;
    if(RD6 == 1)    key = 7;
    if(RD7 == 1)    key = '*';
    RD0 = 0;                                //disable column 1

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

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

    return key;
}
```

The reason for initial value of -1 value should be explained i.e. because 0 is used.

Figure 19 - Scan Keypad Function

3.8. Delay Function

The last function used in the program is the delay function. This uses a set amount of time cycles to delay the program. The value of t typically used is 100 as this is a solid value for the function to count to. Once the function reaches 100 it breaks out of the delay and the program cycle continues.

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

Figure 20 - Delay Function

It should be pointed out in the report that there were prototype stages developed i.e. Keypad scan, encoder feedback and PWM output driver which were all tested individually before being assembled together for the final outcome. This should be in the report.

4. Testing & Analysis

Once the program had been built it had to be thoroughly tested to make sure it was competent and working effectively.

4.1. Simulation Testing of the PWM

The first test performed was the simulation testing. This was where the PWM had to be function tested using Proteus. This was to see if the PWM would give the correct set point. The encoder is shown as the yellow waveform, and the PWM as the blue waveform. The equations used to check the outcomes of this testing were:

$$\text{Frequency} = f$$

$$\text{Revolutions per Minute} = \text{RPM}$$

$$\text{Pulses per Revolution} = \text{PPR}$$

$$\text{PPR} = 100$$

$$f = \frac{1}{(\text{Scope Division}) \times (\text{Scope Time})}$$

$$\text{RPM} = f \times \frac{60}{\text{PPR}}$$

Simulation 1 – 300 RPM

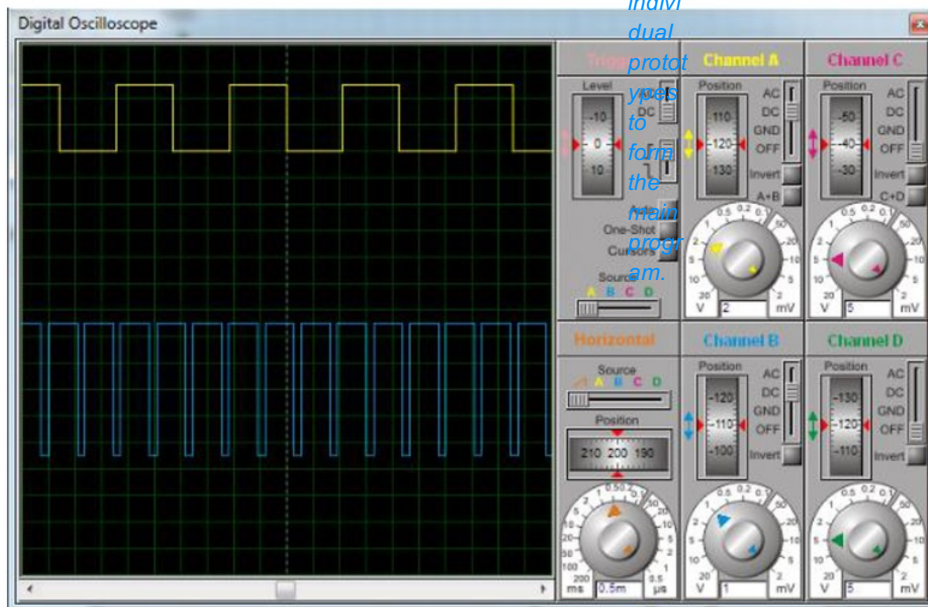


Figure 21 - 300 RPM Sim

Measured speed of 301 rpm

$$f = \frac{1}{4 \times 0.5 \text{ ms}} = 500 \text{ hz}$$

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

These calculations prove the PWM output for this value.

This should point out the prototypes and the method taken to test each part individually.

This test was done after assembling the individual

prototypes to form the main program.

Simulation 2 – 150 RPM

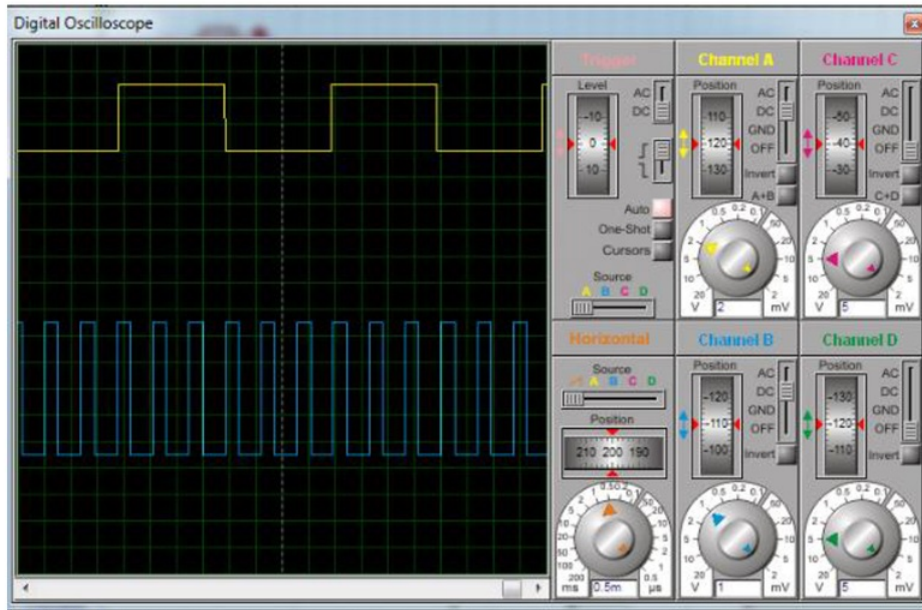


Figure 22 - 150 RPM Sim

Measured speed of 148 rpm

$$f = \frac{1}{8 \times 0.5 \text{ ms}} = 250 \text{ hz}$$

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

These calculations prove the PWM output for this value.

Both of these two simulation tests show that the PWM was operating correctly and giving an output accurate to what would be calculated in the real world.

Test	Setpoint	Calculated RPM	Actual RPM
1	300	300	301
2	150	150	148

Figure 23 - PWM Sim Results Table

The test of the PWM and encoder feedback looks good.

4.2. Testing the Keypad

The keypad also needed to be function checked to make sure that it would work correctly. This was done in simulation on Proteus. The circuit for the test can be seen below along with a testing table as well as outcomes.

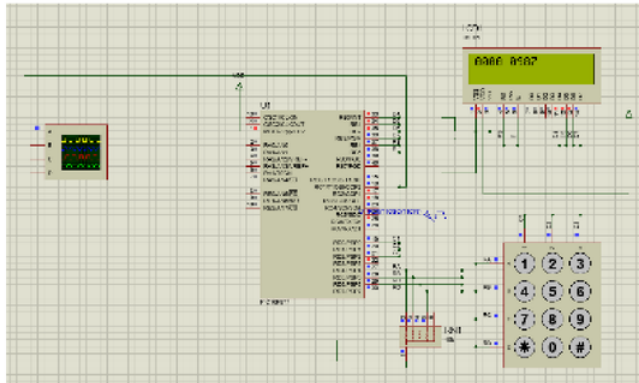


Figure 24 - Keypad Simulation

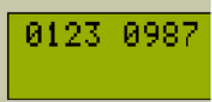
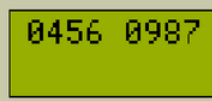
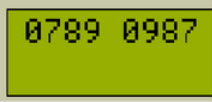
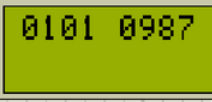
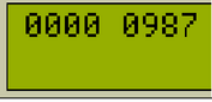
Test	Action	Expected Result	Outcome	Correct
1	Depressing keys of first row in order from left to right	Key numbers should display as "0 1 2 3" on LCD Screen		Y
2	Depressing keys of second row in order from left to right	Key numbers should display as "0 4 5 6" on LCD Screen		Y
3	Depressing keys of third row in order from left to right	Key numbers should display as "0 7 8 9" on LCD Screen		Y
4	Depressing the 0 key with a 1 before and after to show the 0 key working	Key numbers "1 0 1" should display on LCD Screen		Y
5	Depressing # key	Inputted values should clear		Y

Figure 25 - Keypad Testing

These test prove the keypad functionality.

The fully functioning CAD system is not here!

4.3. Interface Functionality Test

Now the full system needed to be functionality tested in order to prove that it was working effectively all in one system. This was important to do before testing the practical aspects of the motor system, as it was important to be sure that the system could be controlled. Below is another testing table with results showing how the system performed under function testing.

Test	Action	Expected Result	Outcome	Correct
1	LCD should display all parameters	All parameters and variables for the motor monitoring system should display on the LCD screen	As expected	Y
2	Enter a set point value using the keypad	The value entered should speed up the motor and sustain it at the entered speed value	As expected	Y
3	Depressing # key	Delete the set point of the motor	As expected	Y
4	Increase a load on the motor post set point entry	System should compensate for the new load and sustain the set point speed	As expected	Y

Figure 26 - System Function Testing

Below is a photo of the LCD screen displaying all parameters as it should in the practical experiment as evidence to back up the table in figure 26.



Figure 27 - LCD Screen Practical

The development board test looks good.

4.4. Practical Testing

Now that the program had been fully built, the PWM tested, the Keypad tested and simulated and the full system function tested for safety, it was time to run the system and monitor the practical results given from it. The board was setup with the motor and oscillator connected as can be seen in the figure below. The same equations as used previously would be used to confirm the measured results.

Frequency = f

Revolutions per Minute = RPM

Pulses per Revolution = PPR

PPR = 500

$$f = \frac{1}{(\text{Scope Division}) \times (\text{Scope Time})}$$

$$\text{RPM} = f \times 60 / \text{PPR}$$

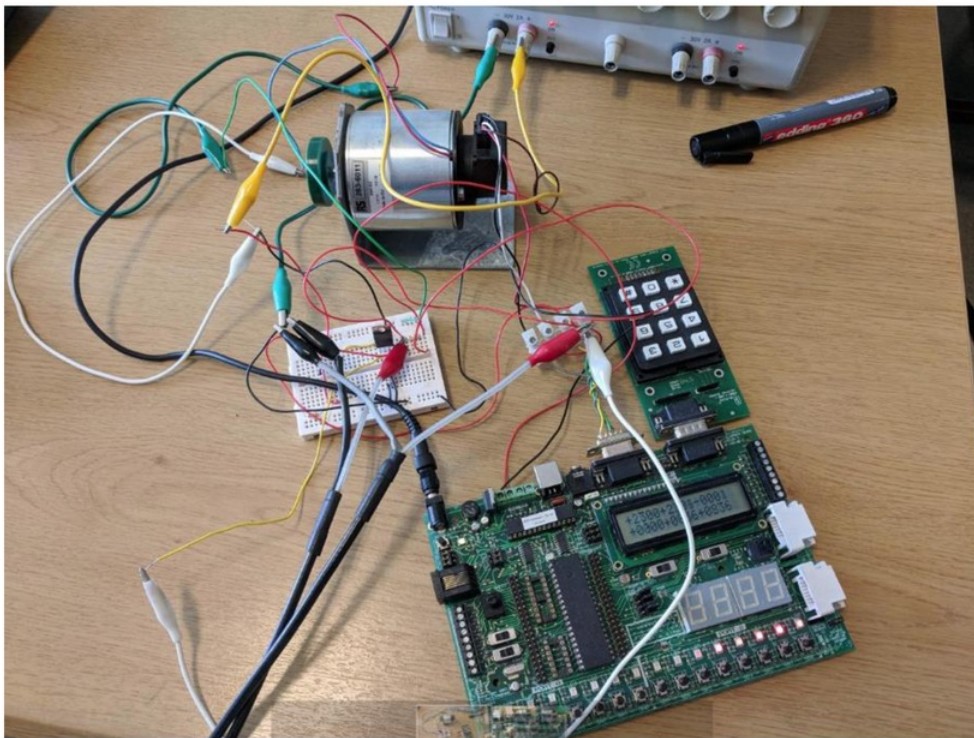


Figure 28 - Practical Build

Practical Test 1 – 200 RPM

The first practical test is for 200 RPM. The waveform for the 200 RPM set point can be seen below. It gives 1.2 divisions at 0.5ms per division. This means that:

$$f = \frac{1}{1.2 \times 0.5 \text{ ms}} = 1.6 \text{ kHz}$$

$$\text{RPM} = 1666 \times \frac{60}{500} = 199.92 \text{ RPM}$$

The calculated value of 199.92 RPM matches up with the simulated value of 200 RPM, negating discrepancies.

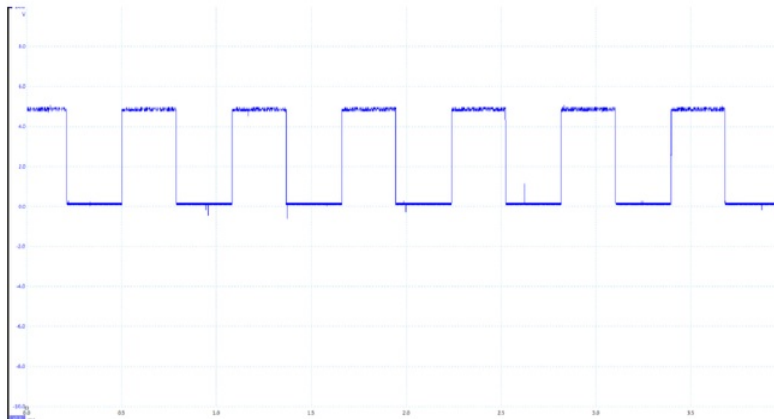


Figure 29 - 200 RPM Set point

The duty cycle for the PWM also needed to be calculated at 200 RPM. This was done using the actual PWM speed of 48:

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

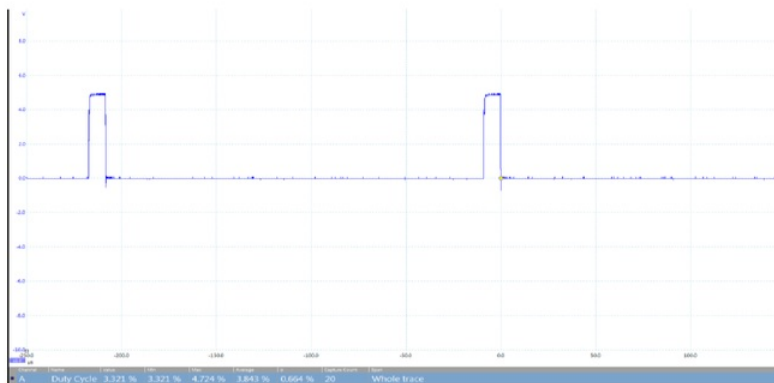


Figure 30 - PWM at 200 RPM

Practical Test 2 – 2000 RPM

The second practical test is for 2000 RPM. The waveform for the 2000 RPM set point can be seen below. It gives 1.2 divisions at 0.05ms per division. This means that:

$$f = \frac{1}{1.2 \times 0.05 \text{ ms}} = 16 \text{ kHz}$$

$$\text{RPM} = 16666 \times \frac{60}{500} = 1999.2 \text{ RPM}$$

The calculated value of 1999.2 RPM matches up with the simulated value of 2000 RPM, negating discrepancies.

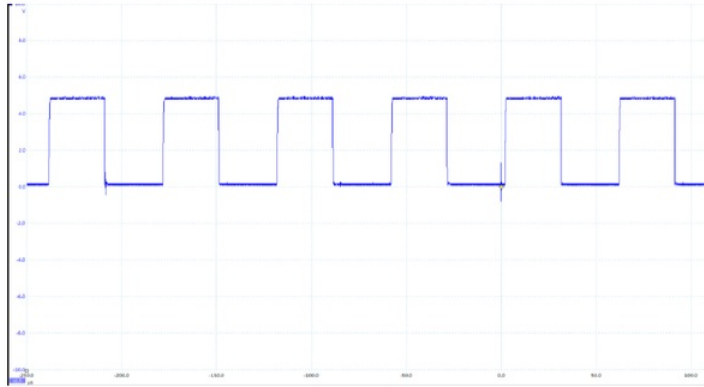


Figure 31 - 2000 RPM Setpoint

The duty cycle for the PWM also needed to be calculated at 2000 RPM. This was done using the actual PWM speed of 391:

$$\text{Duty Cycle} = 391 \times \frac{100}{1023} = 41.47\%$$

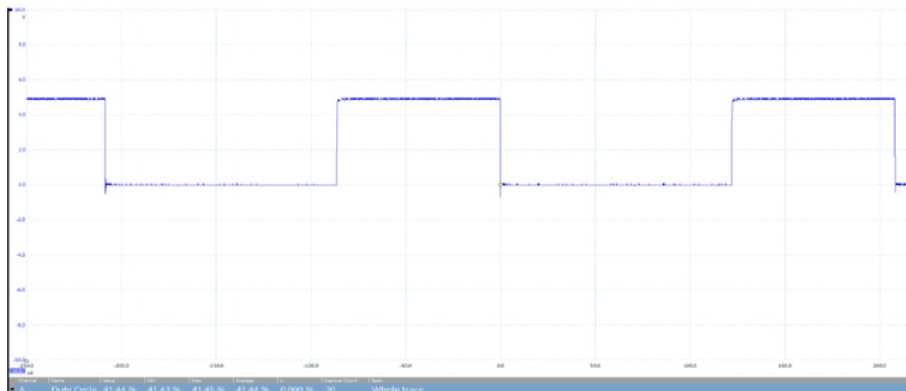


Figure 32 - PWM at 2000 RPM

This is not complete.

5. Conclusions

The overall aims of this assignment were to build a system that used a microcontroller to interface and interact with a DC motor. An incremental encoder, analogue circuit, keypad, LCD screen and various stop and start buttons also needed to be utilised.

This assignment required an appropriate design that met specification. This was met and the design of the system was a well thought out and planned design that met all criteria. It made use of on chip peripherals as well as implementing the provided PWM.

The C code was well written and worked effectively, also including comments that explained how the program would work and process. A detailed description of the code can also be seen in this assignment.

The PWM was tested to spec and met all specification tests as did the keypad. The whole system was then function tested for safety and worked as per was expected. The practical results gave accurate values that were checked and backed up by calculations.

References

AVAGO Technologies. (n.d.). *HEDS 5505 A06 Avago datasheet*.

DNA Tech. (n.d.). *DC Motor*. Retrieved from DNATECHINDIA: <http://www.dnatechindia.com/DC-Motor.html>

E-Blocks. (n.d.). *rsion 3 PICmicro MCU development board Version 3 PICmicro MCU development board*.

Microchip. (2003). *PIC16F87XA Datasheet*. Retrieved 4 1, 2017, from <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>

Piclist. (n.d.). *Hardware of the PIC16F877*. Retrieved from Piclist: http://www.piclist.com/images/www/hobby_elec/e_pic877.htm

Rubasinghe, A. (n.d.). *Connect a 4x3 matrix keyboard to a microcontroller using two I/O pins*. Retrieved from EDN Network: <https://www.edn.com/design/power-management/4413677/Connect-a-4-3-matrix-keyboard-to-a-microcontroller-using-two-I-O-pins>

*Reasonable
number of refs*

Scott Dedoncker Ass 2 FINAL SUB

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

36/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

Text Comment. Good description of how the hardware (keypad) has to be connected to the MCU.

PAGE 6

PAGE 7

Text Comment. The gate is controlled by the MCU output PWM and the output of the MOSFET will work at the PS voltage i.e. 24V. This is only part explained.

PAGE 8

PAGE 9

Text Comment. The counter is 10 bit which are 8 bits from the first 8 bit register and the other 2 bits from the second 8 bit register.

PAGE 10

PAGE 11

PAGE 12

PAGE 13

Text Comment. PROTEUS

Text Comment. Reasonable explanation which describes the initial settings in the program. These are in the correct place in the program.

Text Comment. The program here is correct. The explanation is logical.

Text Comment. There should be code comments here to identify the function of each line. The significance of these registers should be identified.

Text Comment. The 256 value is not the prescaler. It is the number to shift the 2 lsb 's in the register 8 places for 10 bit resolution.

Text Comment. The latch 'enable-keypad'

Text Comment. 10 bit PWM.

Text Comment. These function calls are significant. This is where the data arguments are called to the function prototypes which enable the data to be displayed on LCD.

Text Comment. The program code is correct but some of the code such as the set point constraints are not explained. The code should have comments to indicate the purpose of each line.

Text Comment. This explains the function fairly well.

Text Comment. Code comments should be used.

Text Comment. The code looks correct and checked in the demonstrations of prototypes.

Text Comment. This is the final part of the display code for the program and should have some code comments to indicate the function of each section.

Text Comment. The reason for initial value of -1 value should be explained i.e. because 0 is used.

Text Comment. It should be pointed out in the report that there were prototype stages developed i.e. Keypad scan, encoder feedback and PWM output driver which were all tested individually before being assembled together for the final outcome. This should be in the report.

Text Comment. This should point out the prototypes and the method taken to test each part individually.

Text Comment. This test was done after assembling the individual prototypes to form the main program.

Text Comment. The test of the PWM and encoder feedback looks good.

Text Comment. These test prove the keypad functionality.

Text Comment. The fully functioning CAD system is not here!

Text Comment. The development board test looks good.

Text Comment. This is not complete.

Text Comment. Reasonable number of refs

GRADING FORM: CSG-RUBIC

SCOTT DEDONCKER

0

KNOWLEDGE AND UNDERSTANDING



The introduction does explain the hardware and what the task is about. It lacks details and does not point out the methods used i.e. using prototypes to build and test the design in stages. 2.5/5 System hardware is covered and briefly explained. 3/5 The design and configuration is not explained very well and does not clearly point out the prototypes that were built and tested individually. The report focusses on mainly the overall result. The configuration method is only briefly explained. 8.5/20 Software design was demonstrated and was fully functioning. The design process using prototypes has not been pointed out very clearly. 4/10

18/40

ANALYSIS



The system was designed firstly using the CAD software. This was designed in stages using prototypes and each stage was tested. This resulted in all stages being combined into a working system. The report does not clearly point out this process and only part of the system on CAD is included in the report. The development board test has been carried out, but some of the tests are missing. 4/10 The conclusion does not make any comparisons or evaluations. 2/10

6/20

PRACTICAL APPLICATION AND DEP.



In reality the system was designed and tested in parts by developing prototypes for each section i.e. keypad encoder, Feedback encoder and PWM

6/20

output driver. This is not pointed in the report. The fully functioning CAD system is not included. The CAD and development boards systems were fully demonstrated in the lab, but only the development board is explained in this report. 3/10 Quality of work is below average. 3/10

SKILLS FOR PROFESSIONAL PRAC.



Overall report looks disorganised and done in a hurry. 3/10 Team working was good in the demonstrations 6/10

6/10