

assignment 2

by Joseff Jones

FILE	2874_JOSEFF_JONES_ASSIGNMENT_2_145116_853566698.DOCX		
TIME SUBMITTED	22-MAY-2018 03:34PM (UTC+0100)	WORD COUNT	3841
SUBMISSION ID	89735496	CHARACTER COUNT	23056

HND Electrical and Electronic Engineering

PSDE5013 – Configuration and Programming
of Embedded Systems

Assignment 2 – DC Motor Controller

Joseff Jones

Date of Submission: 22/5/18

Word count: 3841

Table of Contents

1. Introduction	4
2. System Hardware	5
3. Microcontroller Configuration	9
4. Software Design and Coding	13
5. Testing and Analysis of Results	22
5.1 - Simulation Results	22
5.2 - Practical Results	25
6. Discussion and Conclusions	30
7. Appendices & Bibliography	31
References	31

Table of Figures

Figure 1 - Microcontroller DC Motor System Block Diagram.....	4
Figure 2 - PIC16F877 Microchip Basing Diagram	5
Figure 3 - Keypad Circuit Diagram and Output Arrangement.....	6
Figure 4 - HP488 Development Board	7
Figure 5 - HEDS-5540 Encoder	8
Figure 6 - OPTION_REG Register Datasheet Information	9
Figure 7 - INTCON_REG Register Datasheet Information	10
Figure 8 - Timer2 Datasheet Information	12
Figure 9 - CCP Module Block Diagram for PWM	12
Figure 10 - Encoder Circuit.....	13
Figure 11 - Encoder Oscilloscope Circuit.....	13
Figure 12 - Encoder Circuit with Keypad.....	14
Figure 13 - Proteus Final Circuit	15
Figure 14 - Final Design Code (1/8).....	15
Figure 15 - Final Design Code (2/8).....	16
Figure 16 - Final Design Code (3/8).....	17
Figure 17 - Final Design Code (4/8).....	17
Figure 18 - Final Design Code (5/8).....	18
Figure 19 - Final Design Code (6/8).....	19
Figure 20 - Final Design Code (7/8).....	20
Figure 22 - Final Design Code (8/8).....	21
Figure 23 - Practical Result @ Set-Point 200RPM.....	25
Figure 24 - Practical Result @ 200RPM PWM.....	25
Figure 25 - Practical Result @ Set-Point 1000RPM.....	26
Figure 26 - Practical Result @ 1000RPM PWM.....	26
Figure 27 - Practical Result @ Set-Point 2000RPM.....	27
Figure 28 - Practical Result @ 2000RPM PWM.....	27
Figure 29 - Practical Result @ Set-Point 2300RPM.....	28
Figure 30 - Practical Result @ 2300RPM PWM.....	28

1. Introduction

This assignment had a number of learning outcomes and assessment criteria to meet, the primary criteria to meet was to design an embedded system which utilized a microcontroller which was interfaced to a DC motor, LCD display, analogue sensing circuit, two pushbuttons, incremental encoder and a keypad. The keypad was used to enter a speed set-point, the LCD was used to display the speed set-point, and the incremental encoder was used to provide feedback for the speed of the motor – current sensing came in the form of a voltage proportional to the current of the armature. The motor needed to be provided with PWM speed control with current feedback and was turned off and on using the pushbuttons. The circuit in the figure below was obtained from the assignment brief and visually shows how each part of the system is connected.

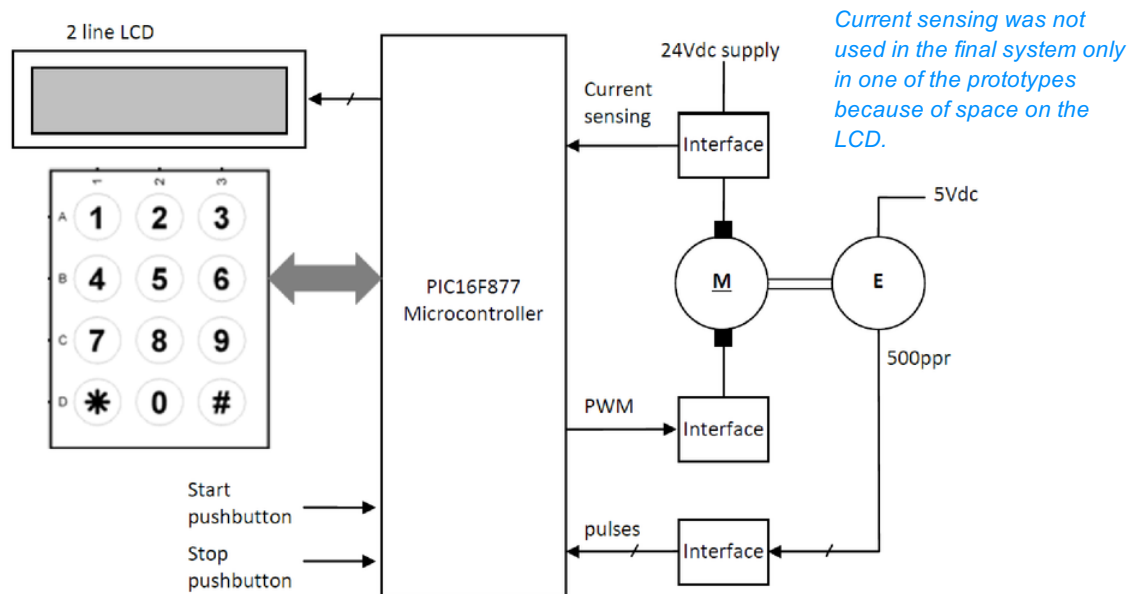


Figure 1 - Microcontroller DC Motor System Block Diagram

The programming language that was used for the program was C, it was chosen as it is a suitable programming language for embedded systems, is relatively simple, and can perform all of the necessary functions to complete the task at hand. The program that was used to write the code for the system was MPLab due to its intuitiveness, and the program that was used to design the circuits in order to simulate the system was Proteus as it proves an excellent program for real-time circuit simulation.

Good intro

2. System Hardware

Several pieces of hardware were used within the assignment to develop the desired circuit, the specific chosen devices for each hardware type is shown in this section, where a description is given about the functionality and uses of each individual device.

Microcontroller

Due to its complexity, the assignment demanded a microcontroller with specific attributes such as PWM functionality and a plethora of I/O ports; the microcontroller that was chosen based on the desired attributes was the PIC16F877, which had all the desired functionalities along with several extra on-chip features. The basing diagram for the PIC was obtained from the manufacturer's datasheet at (PIC16F87XA Data Sheet, 2003) and is shown in the figure below.

40-Pin PDIP

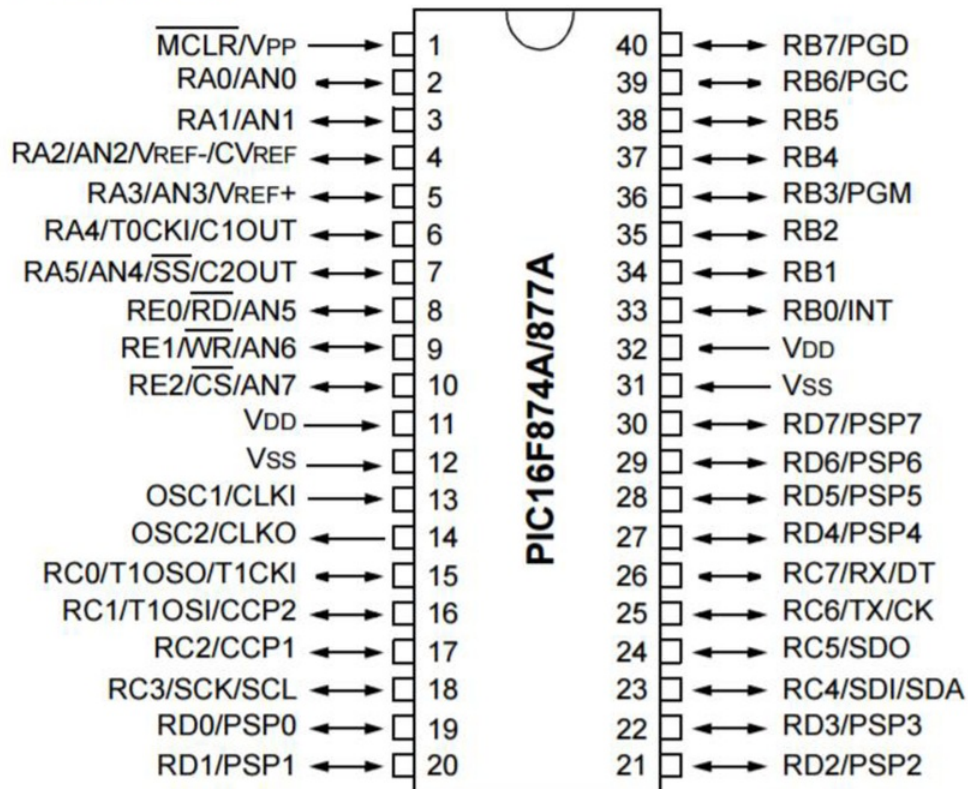


Figure 2 - PIC16F877 Microchip Basing Diagram

Keypad

One of the specifications of the assignment was to have a way of inputting a speed set-point for the motor to adjust to – to achieve this, a 12-key keypad was used. The keypad that was used was setup as a 3x4 input device where the digits 0-9 were used to specify the desired set-point, an asterisk (*) which wasn't used, and a hashtag (#) key which was used to set the speed set-point as 0RPM. An interface was developed which allowed the keypad to communicate key-presses to the microcontroller.

The data sheet was accessed (found at (eblocks Keypad Board, n.d.)) and a circuit diagram and output arrangement was obtained highlighting the keypad's characteristics, found in the figure below.

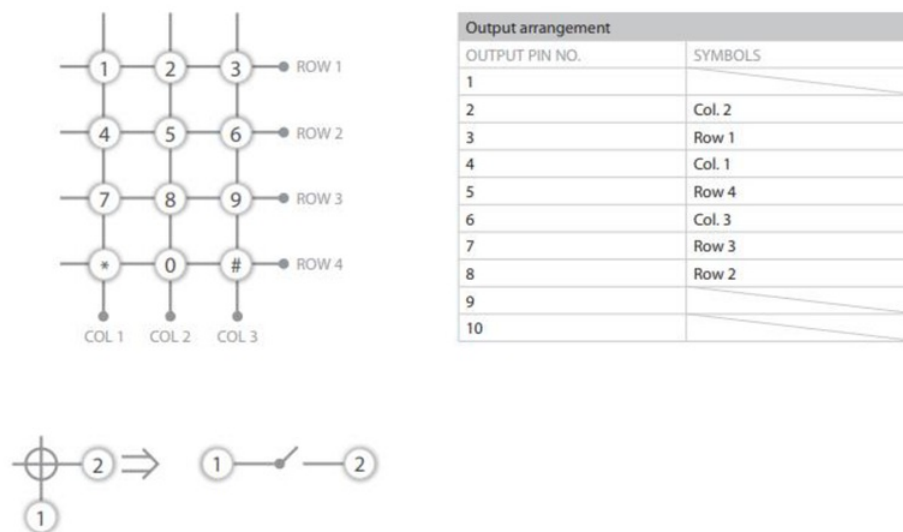


Figure 3 - Keypad Circuit Diagram and Output Arrangement

Start/Stop Pushbuttons

Two pushbuttons were used to allow a start/stop function for the motor – these pushbuttons were connected to PORT-A on the microcontroller.

Closed-Loop Feedback

In order for the motor to take the input value from the keypad and then accurately replicate that value as an RPM, a closed-loop PI (proportional-integral) controller was used. In order to function properly as a closed-loop PI controller, the system needed to measure the RPM of the motor itself so that it could be used as feedback – an incremental encoder was chosen to achieve this. The encoder works using a single LED (light-emitting diode); “the light is collimated into a parallel beam by means of a single polycarbonate lens located directly over the LED. Opposite the emitter is the integrated detector circuit.” (Avago, 2012).

Development Board & LCD

The LCD screen that was used for the practical build was attached to the development board that was used. The development board that was used was the HP488, which along with the LCD has an in-built removable crystal, oscillator mode, a number of sockets for programming 8, 14, 18, 28 and 40 pin PICmicro devices and on-board light sensors along with a number of other features.

The LCD was used to display relevant parameters of the motor’s performance using 32 characters spread across 2 lines – sufficient to display all relevant parameters. The development board is shown in the figure below and was obtained from (MATRIX).

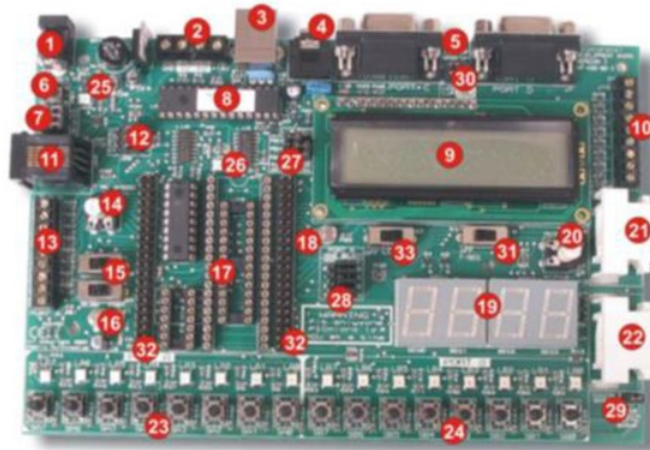


Figure 4 - HP488 Development Board

DC Motor

The DC motor that was used to complete the assignment had an operating voltage far greater than the output voltage of the microcontroller – the DC motor’s operating voltage was 24VDC and the output voltage of the microcontroller was 5V. In order to get around this safely, the two voltages were independently isolated from one another; this was achieved using a transistor as a switch to output a PWM wave from the microcontroller which was amplified for the DC motor, all while keeping the two devices isolated. The DC motor featured a flywheel diode which was used to dissipate excess back-EMF.

Encoder Feedback

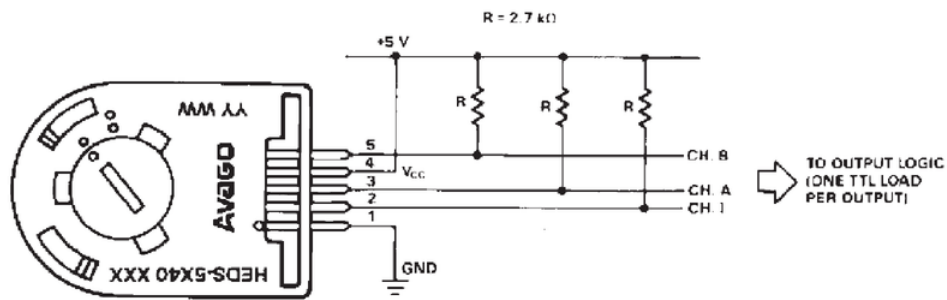


Figure 5 - HEDS-5540 Encoder

In order to accurately control the speed of the motor, a PI controller was used – this featured the use of an encoder. The encoder that was used was the HEDS-5540 encoder, its pin-out and connections can be seen in the figure above.

*The hard are description is reasonable,
but the MOSFET interface is missing.*

3. Microcontroller Configuration

The PIC16F87XA datasheet was used extensively in this section of the report as it provided all of the necessary and relevant information that made the functioning final circuit possible such as information on timers, registers, ports and the PWM functionality.

OPTION_REG Register

The OPTION_REG register is a read/write register which can be programmed to configure the TMR0 pre-scaler, which was also used. In the coding of the final design, the OPTION_REG is programmed to be 0b10000111, which enables the PORTB Pull-up, the pre-scaler assignment and the pre-scaler rate selectors. Information on the OPTION_REG register was used from the datasheet, and the most relevant of that information is shown in the figure below.

REGISTER 2-2: OPTION_REG REGISTER (ADDRESS 81h, 181h)

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RBP _U	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7								bit 0
bit 7	RBP_U : 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 (CLKO)							
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

Note: When using Low-Voltage ICSP Programming (LVP) and the pull-ups on PORTB are enabled, bit 3 in the TRISB register must be cleared to disable the pull-up on RB3 and ensure the proper operation of the device

Figure 6 - OPTION_REG Register Datasheet Information

INTCON_REG Register

The INTCON_REG Register is similar to the OPTION_REG register in that it's a read/write register that directly affects TMR0, but the way it affects TMR0 is different. The INTCON_REG register is programmed to be 0b11100000 which enables the global interrupt, peripheral interrupt and TMR0 overflow interrupt, as can be seen in the figure below.

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x	
	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0	
bit 7	GIE: Global Interrupt Enable bit 1 = Enables all unmasked interrupts 0 = Disables all interrupts							
bit 6	PEIE: Peripheral Interrupt Enable bit 1 = Enables all unmasked peripheral interrupts 0 = Disables all peripheral interrupts							
bit 5	TMR0IE: TMR0 Overflow Interrupt Enable bit 1 = Enables the TMR0 interrupt 0 = Disables the TMR0 interrupt							
bit 4	INTE: RB0/INT External Interrupt Enable bit 1 = Enables the RB0/INT external interrupt 0 = Disables the RB0/INT external interrupt							
bit 3	RBIE: RB Port Change Interrupt Enable bit 1 = Enables the RB port change interrupt 0 = Disables the RB port change interrupt							
bit 2	TMR0IF: TMR0 Overflow Interrupt Flag bit 1 = TMR0 register has overflowed (must be cleared in software) 0 = TMR0 register did not overflow							
bit 1	INTF: RB0/INT External Interrupt Flag bit 1 = The RB0/INT external interrupt occurred (must be cleared in software) 0 = The RB0/INT external interrupt did not occur							
bit 0	RBIF: RB Port Change Interrupt Flag bit 1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software). 0 = None of the RB7:RB4 pins have changed state							

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 7 - INTCON_REG Register Datasheet Information

Timer0

Once the OPTION_REG and INTCON_REG registers were configured, timer0 and its interrupt could be used. In order to correctly operate, timer 0 was configured with an interrupt interval of 10ms. Since the on-board oscillator was fluctuating at a rate of 19.6608MHz, a number of calculations were carried out in order to achieve that interrupt interval of 10ms using the max pre-scaler and preload; the calculation is:

$$F_{osc} = 19.6608MHz$$

$$Countable\ clock\ speed = \frac{F_{osc}}{4} = \frac{19.6608M}{4} = 4915200Hz$$

$$\frac{1}{4915200} * 256 = 52us$$

$$Max\ prescaler = 256$$

$$\frac{\frac{F_{osc}}{4}}{256} = \frac{4915200}{256} = 19200Hz$$

$$Period = \frac{1}{19200}$$

$$Number\ of\ counts\ every\ 10ms = 192\ (to\ give\ 10ms\ interrupt)$$

$$Preload\ timer\ 0\ with\ 256 - 192 = 64\ (to\ interrupt\ every\ 10ms)$$

Timer 1

Timer1 is another timer that was used for the final design. Timer1 is a 16-bit counter with a pre-scaler, as opposed to timer0 which is 8-bit; the timer is made up of two 8-bit registers called TMR1H and TMR1L and each register increments from 0000h to FFFFh, incrementing on every rising edge of the clock input.

Timer2

Timer2 is another 8-bit counter with a pre-scaler, but this one has a post-scaler. It was used in the final design for PWM control due to its functionalities. Relevant information on timer2 was obtained from the manufacturer's datasheet, that information is shown in the figure below.

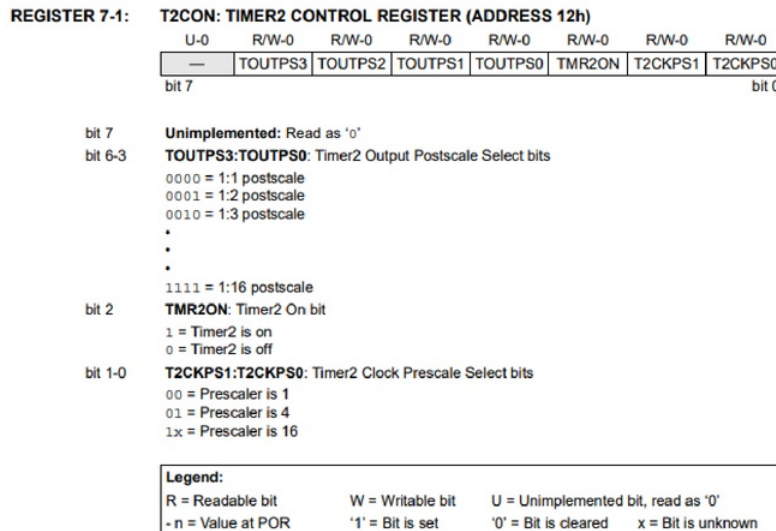


Figure 8 - Timer2 Datasheet Information

PWM

In PWM mode, the CCPx pin produces the PWM output and has a resolution of up to 10-bits. CCP is the PIC16F877's on-board PWM module. Because the CCPx pin is used, the CCP1CON register needed to be configured. The figure below which was obtained from the datasheet shows a simplified block diagram of the CCP in PWM mode. The PIC16F877 uses a crystal oscillator of 20MHz and since it executed instructions on every fourth CPU cycle, the board run at 5MIPS (millions of instructions per second).

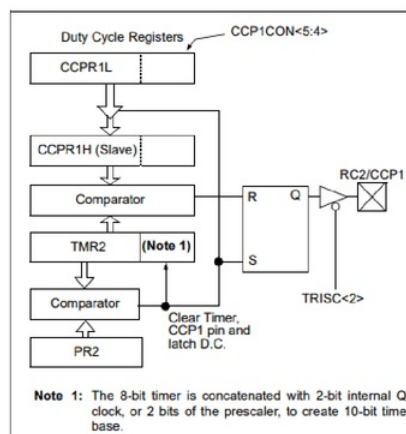


Figure 9 - CCP Module Block Diagram for PWM

4. Software Design and Coding

Prior to completing the final design, a number of stages where different individual parts of the final design were developed and tested.

Encoder

Prototypes have been identified here

The first stage of the design process to be designed and tested was the encoder – this involved the fundamental components of the final design - the PIC16F877 and the LCD; a pulse generator was also included in this part to simulate the input of an encoder. The figure below shows the setup on Proteus along with the LCD display showing roughly 1kHz (which is the value produced by the pulse generator); the keypad isn't used at this point because its functionality hadn't been coded in.

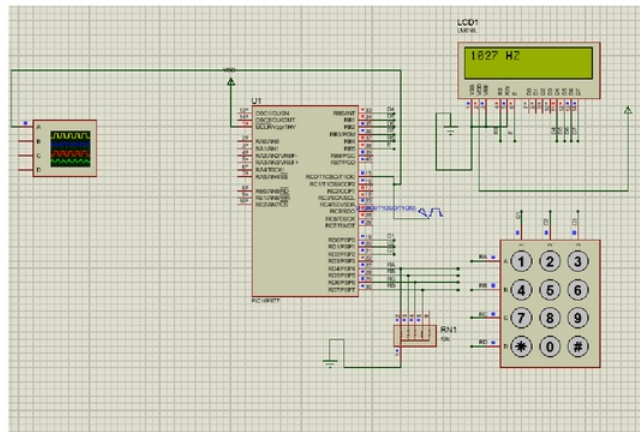


Figure 10 - Encoder Circuit

An oscilloscope was also used to measure the frequency output of the encoder, the output of which can be seen in the figure below.

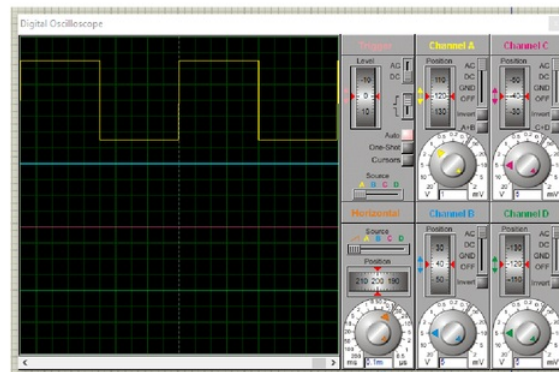


Figure 11 - Encoder Oscilloscope Circuit

The output shown on the oscilloscope shows ten columns of 0.1ms, giving a total of 1ms. This gives a frequency of $f = \frac{1}{P} = \frac{1}{1 \times 10^{-3}} = 1000\text{Hz}$ which is the expected value.

Prototype has been designed and tested

Encoder & Keypad

The next stage of development involved developing the code to incorporate the keypad to allow the frequency to be altered while the program was running using the keypad. The '#' key on the keypad could be used to reset the frequency, and the max frequency was 2000Hz. The figure below shows the keypad in use, as a frequency of 855Hz was just input using it.

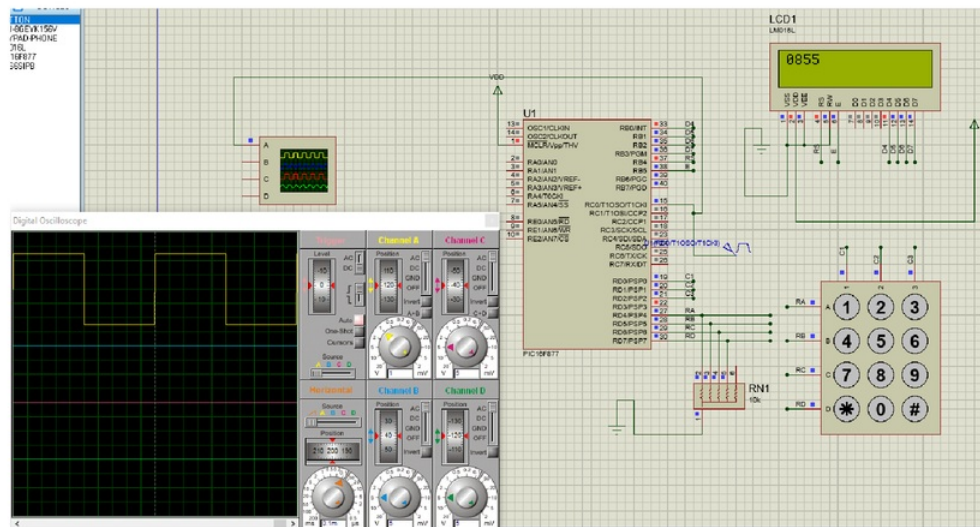


Figure 12 - Encoder Circuit with Keypad

The LCD is showing the output from the keypad, but the oscilloscope is still showing the output from the encoder, showing that the keypad is in fact producing the 855Hz output.

Key
pad
desi
gn
and
test
is
goo
d

Final Circuit

The final circuit of the assignment was designed on Proteus to be simulated; that final circuit is shown in the figure below.

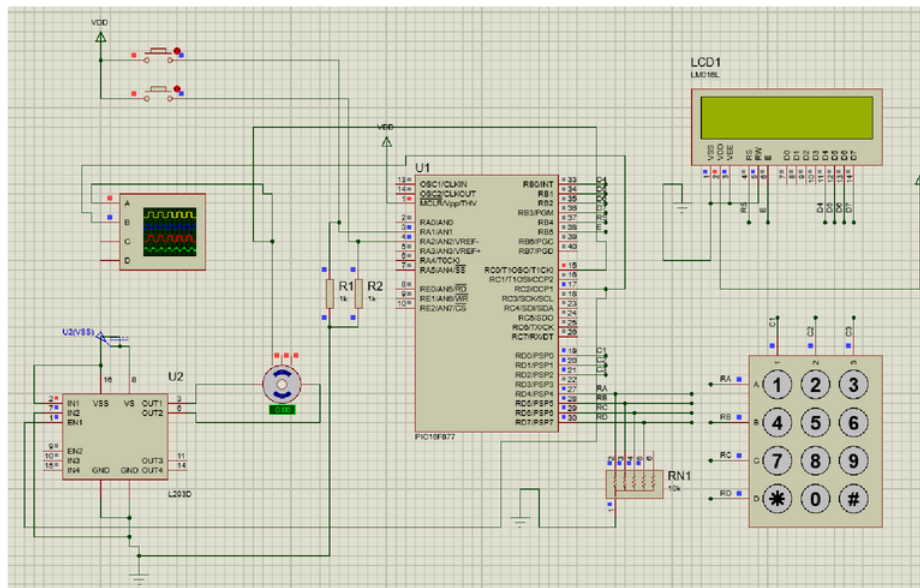


Figure 13 - Proteus Final Circuit

Design looks good.

The code that was written in order to operate the simulation and the practical board, that code is shown in the following figures.

```
#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;
```

Figure 14 - Final Design Code (1/8)

The above figure shows the first part of the code. The first line includes PIC.h which is there because a PIC is being used and it makes it simpler to write code for the PIC. The next line includes LCD.h which is where a library of functions are included for the LCD specifically.

Below this are four lines which include a number of functions to be declared. These functions are beefed out later in the main program.

The three lines of code at the bottom are where the global variables are declared – variables which are set to 0 are generally either user-defined or are calculated by the PIC; the kp and ki values were values that were found to be optimal for the system.

```

////////////////////////////////// ISR //////////////////////////////////////////
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;
    }
}

```

Figure 15 - Final Design Code (2/8)

The next part of the code is the ISR (interrupt service routine) and is used to count pulses and then interrupt when certain conditions are met. The conditions are determined by TOIF, TMR1IF and TMR2IF – these are the interrupts for timer0, timer1 and timer2. If timer0 overflows and the flag is set - TOIF then resets the flag, the count is then pre-loaded to 64 and the milliseconds are incremented by +10. After this, an if statement is used to check if milliseconds are set to 100 and there's no fault, and then speed_low_byte and speed_high_byte are set to TMR1L and TMR1H's respected values.

*Good
explanation.*

```

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

    ADCON1 = 0x8E;          //Configure ADCON1 = 0000 1110
    TRISB = 0x00;          //used for LCD display
    TRISC = 0x01;          //input RC0 used for counting with TMR1
                          //output RC2 used for PWM with TMR2
    TRISD = 0b11110000;

    PORTB = 0x00;
    LCD_Startup();

    PIE1 = 0b00000011;     //Peripheral interrupt enable register for TMR1 & TMR2
    PIE2 = 0;
    PIR1 = 0;              //clear Peripheral interrupts
    PIR2 = 0;              //clear Peripheral interrupts

    INTCON = 0b11100000;    //setup for TIMER 0 in timer mode
    OPTION_REG = 0b10000111;
}

```

Figure 16 - Final Design Code (3/8)

Commented code is good.

The next part of the code is the main program.

Firstly, enable_keypad is set to an integer value of 1 indicating that it's activated and key is set with no pre-determined value as it is determined by the user's input. The code below that is for setting the ADCON register and also setting tristate ports B, C and D as either inputs or outputs (done in hexadecimal). Port B is then set to zero and the LCD_Startup() function is fetched. PIE1, PIR1, PIE2 and PIR2 are registers that are set up before INTCON and OPTIONS_REG, which were explained in detail in the microcontroller configuration section of the report.

```

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

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

```

Figure 17 - Final Design Code (4/8)

In the same section of main code are seven more lines of code. T1CON is there to setup for timer1 in event mode, TMR1L and TMR1H are set to 0. PR2 was set to 255 and the duty cycle (CCPR1L) was set to 0%. T2CON is used for setting up timer2 during PWM mode and uses the binary code 0b00000101 to set up necessary functions. CCP1CON is used for a number of functions such as setting up PWM mode and the extra two least significant bits for PWM and providing 10 bits for PWM.

Good explanation of the initial conditions and identifying how the 10 bit PWM is configured.

```

while(1)
{
    speed = (int)((float)((speed_low_byte + speed_high_byte*256)*6)/5);
    // speed = (int)((float)(speed_low_byte + speed_high_byte*256));
    speed = speed * 31;
    //DisplaySpeed(0,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;
        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;

    CCPR1L = (int)output;

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

```

Figure 18 - Final Design Code (5/8)

The next part of the code is the super-loop, which runs continuously. The first line of code determines the speed of the motor at its current RPM using the calculation $((\text{speed_low_byte} + \text{speed_high_byte} * 256) * 6) / 5$. The next two lines check to see if the start or stop pushbuttons have been pressed and gives the buttons their functionalities. The line of code below this states that the 'key' variable is set to the value obtained from ScanKeypad(); the value which is obtained is then put through a number of if statements to determine what to do with the input value - the first if statement checks that a key has been pressed and the keypad is enabled, then it formats the input key presses so that it multiplies the already held value by 10 each time the key is pressed, and then adds the pressed integer key. The next if statement checks whether the # key is pressed, and then sets the set-point to 0 if it has been. The next if statement checks that no key is currently being pressed before accepting another key press. The next two if statements cap the max set-point to 2000 and restrict the lowest set-point to 0. The next if statement is what controls the functionality for the closed-loop PI; if the system is already running, the 'error' value is calculated by deducting the measured speed away from the set-point, the 'proportional' value is calculated by multiplying the error by Kp, the integral accumulation part is done by storing the error and multiplying it by ki, the output is the proportional part multiplied by the integral accumulation. If the system is not already running, the error, int_accum, output and RC2 are set to 0. The next two lines of code are what cap the motor's speed to a maximum and minimum value. The DisplayValues are there to configure the LCD display.

```

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);
}

```

Figure 19 - Final Design Code (6/8)

The DisplaySpeed() function is what is used to display the speed of the more on the LCD. The variables in the function are line and value. Five local variables are declared in the function – units, tens, hunds, thous and temp. In order to convert each integer into a ASCII equivalent, each of the units, tens, hundreds and thousands have 48 added to them. “RPM” is then input at the end of the DisplayString().

```

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;
}

```

Figure 20 - Final Design Code (7/8)

The DisplayValue() function works similarly to the DisplaySpeed() function, except it uses an if function that determines whether the value is positive or negative. It is used to display important information on the LCD such as prop_part, int_accum, output, set-point, speed and error.

Explanation is

*a little brief.
There are 3
sets of display
on each line of
the LCD
defined by the
arguments of
the prototype
function.*

```

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;
}

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

```

Figure 21 - Final Design Code (8/8)

The final part of the code includes the ScanKeypad() function along with the Delay() function.

The ScanKeypad() function is there to constantly scan for when the user presses a key and deduces which key has been pressed based on which column the key is on and then which row.

The Delay() function utilizes a for loop to delay the system for a designated value or time.

This is also a little brief. The columns are enabled and the related row is scanned. This is done in turn by column.

5. Testing and Analysis of Results

Testing was performed on both the simulative and practical results.

5.1 - Simulation Results

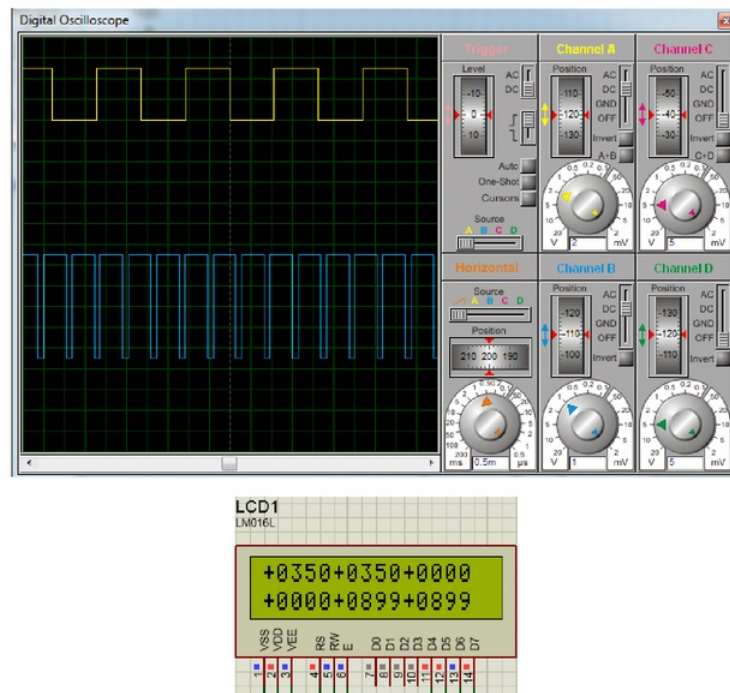
In order to simulate the desired outputs the keypad was used to set the motor to a specific speed and then the oscilloscope and LCD were used to verify that the system was working. Calculations were performed to determine RPM and frequency to compare with simulated results; the calculations for RPM and frequency are:

$$RPM = frequency \times \left(\frac{60}{PPR} \right)$$

The PPR (pulses per revolution) was set to 100.

$$Frequency = \frac{1}{(division\ on\ scope) * (timebase\ on\ scope)}$$

350RPM

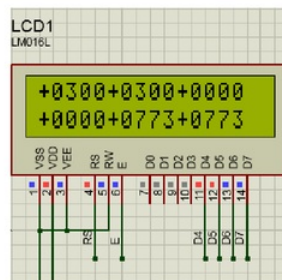
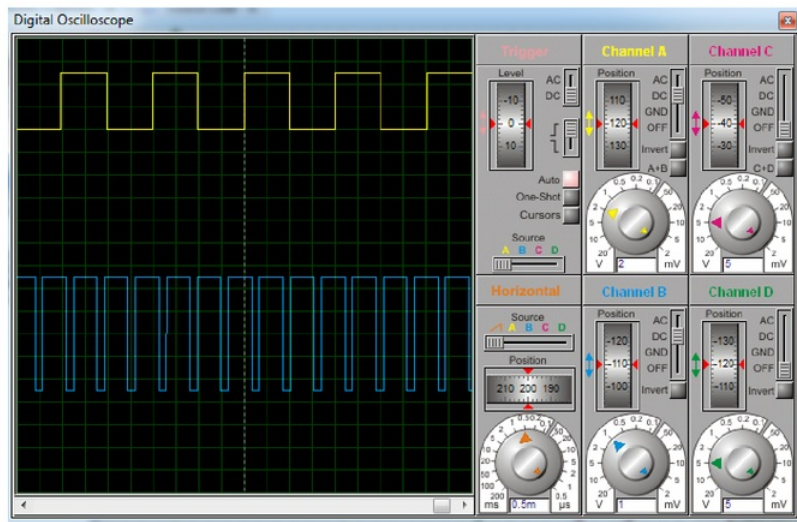


At the 350RPM set-point, the measured speed was 350RPM

The frequency was calculated to be $\frac{1}{1.7m} = 588\text{Hz}$

Calculated RPM = $588 * \frac{60}{100} = 352.8\text{RPM}$

300RPM

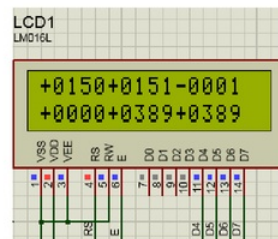
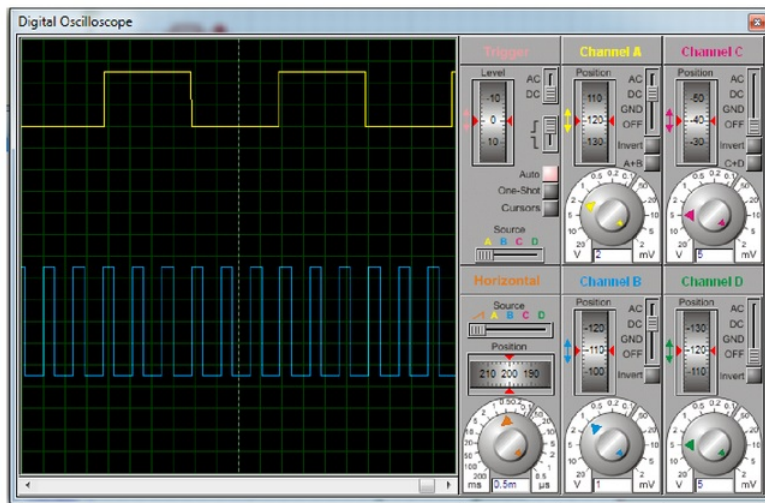


At the 300RPM set-point, the measured speed was 300RPM

The frequency was calculated to be $\frac{1}{2m} = 500\text{Hz}$

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

150RPM



At the 150RPM set-point, the measured speed was 151RPM

The frequency was calculated to be $\frac{1}{4m} = 250\text{Hz}$

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

5.2 - Practical Results

The image of the development board under test should be included here.

The practical build was setup and the microcontroller was programmed. Picoscope software was used to measure the PWM and set-point waveform, and the frequency, RPM and duty cycle were calculated to verify the output was correct.

Set-point 200 RPM

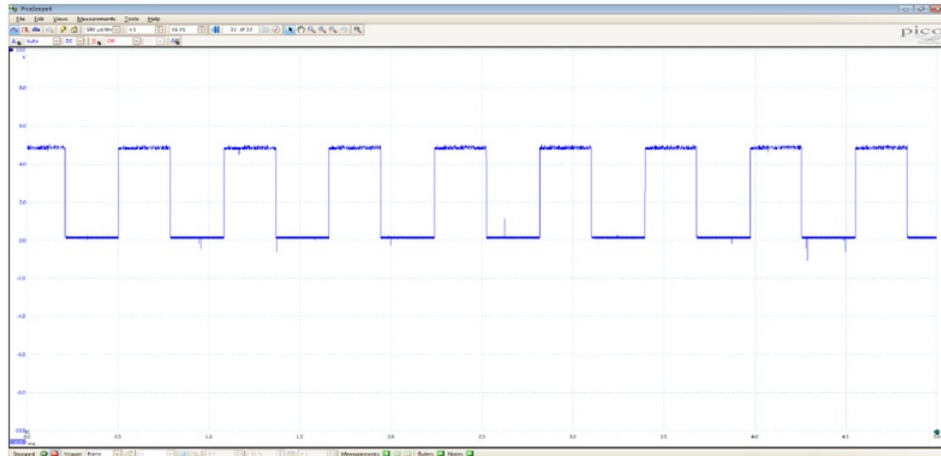


Figure 22 - Practical Result @ Set-Point 200RPM

1.2 divisions at 0.5ms per division = 0.0006

$$F = 1 / 0.0006 = 1666\text{HZ}$$

$$\text{RPM} = 1666 * (60 / 500) = 200$$

PWM at 200 RPM

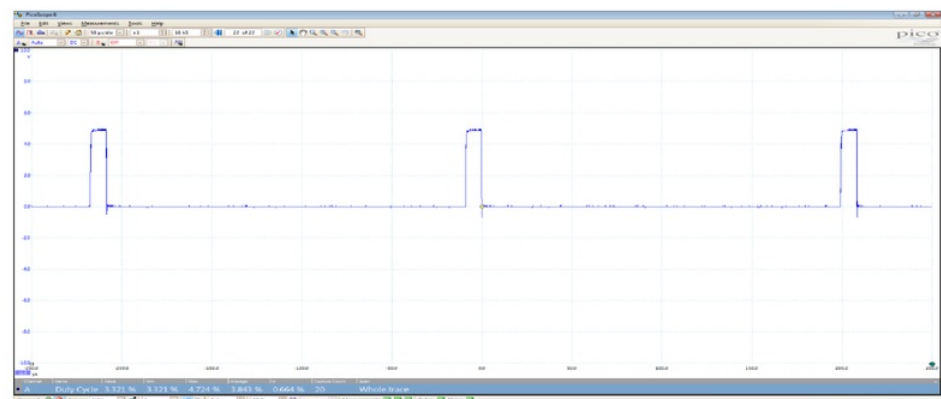


Figure 23 - Practical Result @ 200RPM PWM

$$\text{Duty cycle} = 48 (\text{actual speed PWM}) * 100 / 1023 (\text{Duty at 100\%})$$

$$\text{Duty cycle} = 4.69\%$$

Set-point 1000 RPM

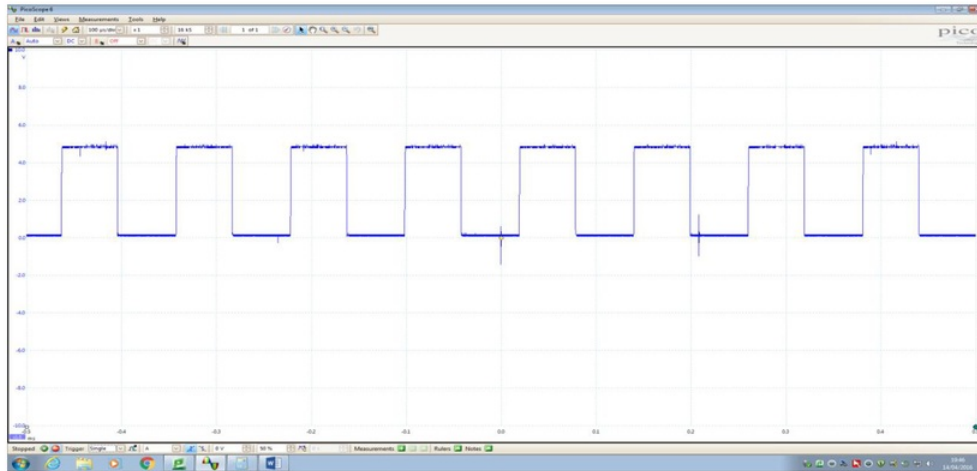


Figure 24 - Practical Result @ Set-Point 1000RPM

1.2 divisions at 0.1ms per division = 0.00012

$$F = 1 / 0.00012 = 8333\text{HZ}$$

$$\text{RPM} = 8333 * (60 / 500) = 999.6 \text{ RPM}$$

PWM at 1000 RPM

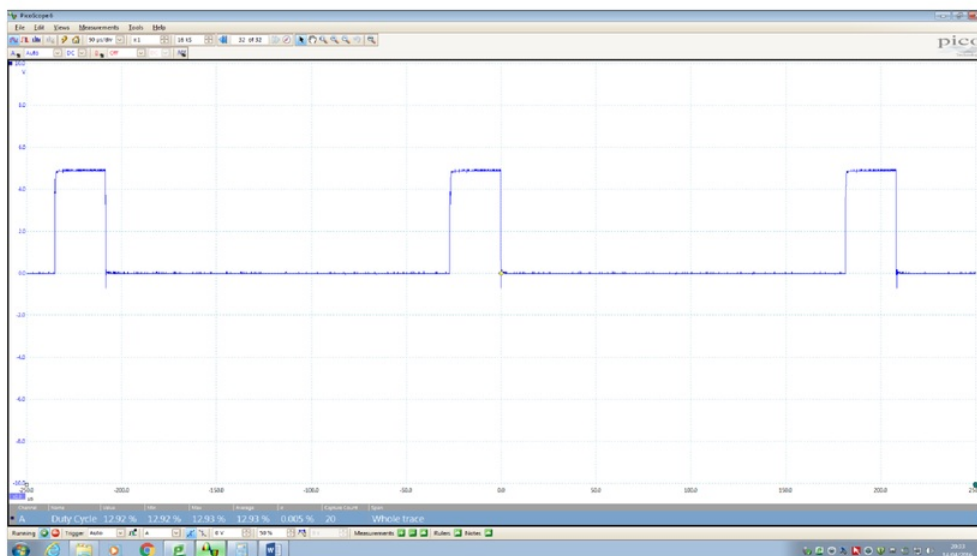


Figure 25 - Practical Result @ 1000RPM PWM

$$\text{Duty cycle} = 133 (\text{actual speed PWM}) * 100 / 1023 (\text{Duty at 100\%})$$

$$\text{Duty cycle} = 13\%$$

Set-point 2000 RPM

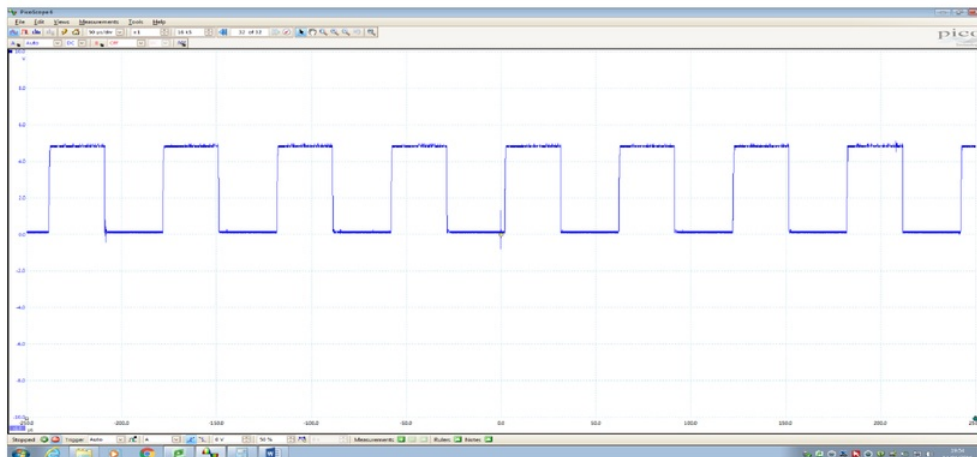


Figure 26 - Practical Result @ Set-Point 2000RPM

1.2 divisions at 0.05ms per division = 0.00006

$$F = 1 / 0.00006 = 16666\text{HZ}$$

$$\text{RPM} = 16666 * (60 / 500) = 1999.2 \text{ RPM}$$

PWM at 2000 RPM

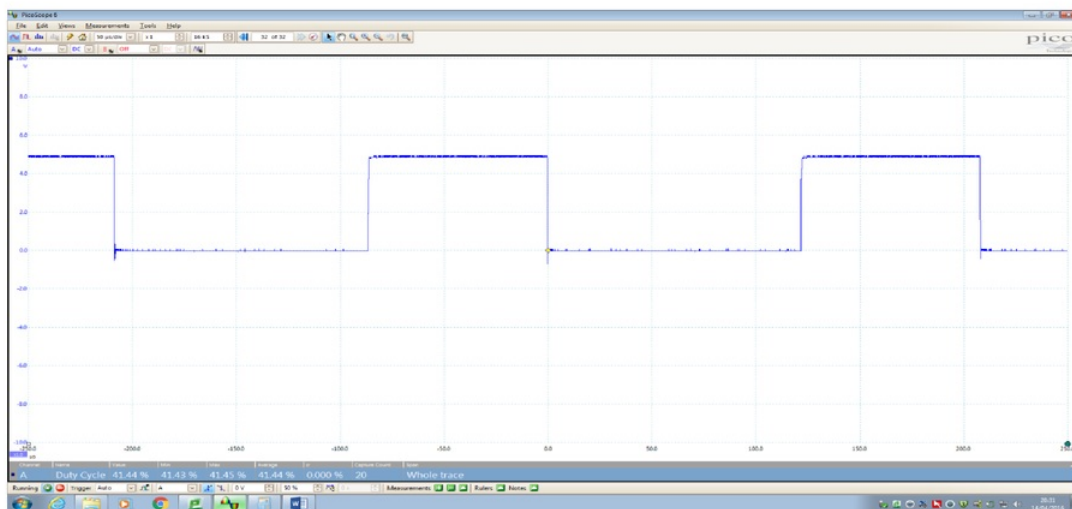


Figure 27 - Practical Result @ 2000RPM PWM

$$\text{Duty cycle} = 391 (\text{actual speed PWM}) * 100 / 1023 (\text{Duty at 100\%})$$

$$\text{Duty cycle} = 38.22\%$$

Set-point 2300 RPM

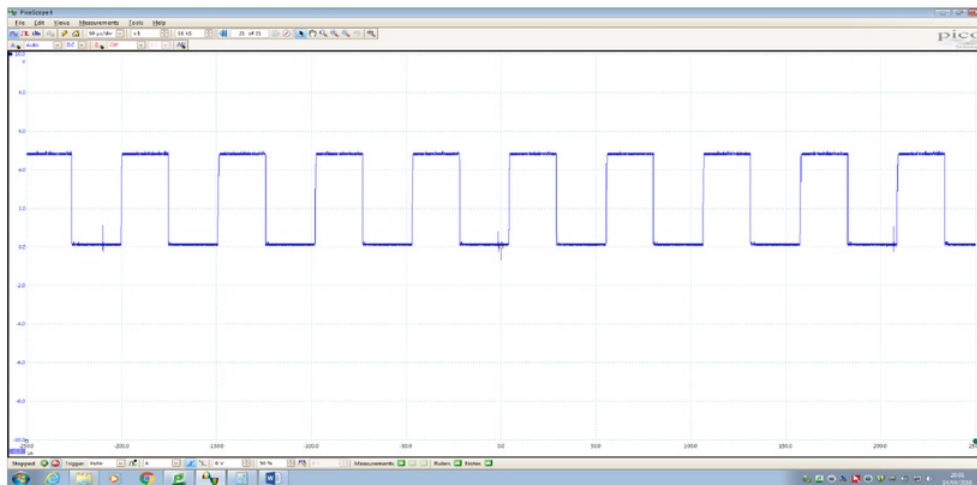


Figure 28 - Practical Result @ Set-Point 2300RPM

1.0 divisions at 0.05ms per division = 0.000055

$$F = 1 / 0.0005 = 19000\text{HZ}$$

$$\text{RPM} = 19000 * (60 / 500) = 2280 \text{ RPM}$$

PWM at 2300 RPM

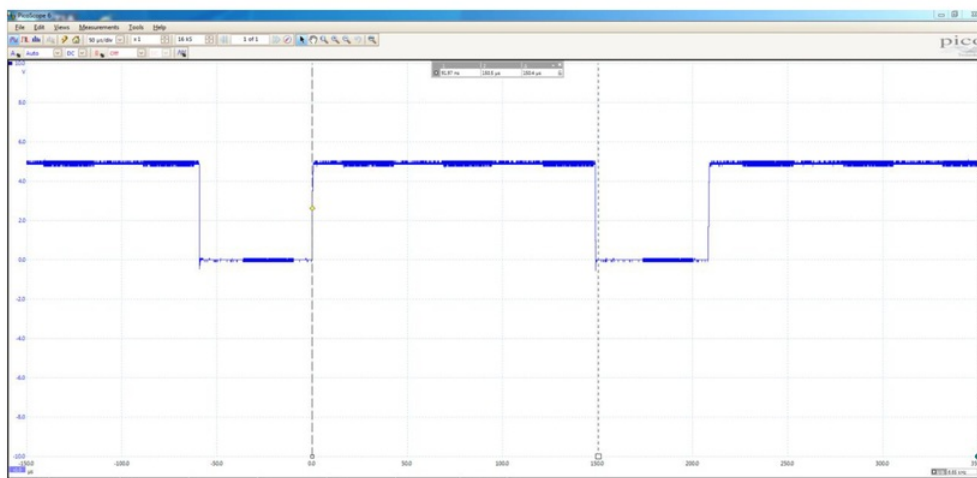


Figure 29 - Practical Result @ 2300RPM PWM

$$\text{Duty cycle} = 744 (\text{actual speed PWM}) * 100 / 1023 (\text{Duty at 100\%})$$

$$\text{Duty cycle} = 72.72\%$$

Two tables were drawn up from the measured practical results – the first table compares the calculated RPM with the actual RPM.

RPM Set-Point	Calculated RPM	Actual RPM
200	206	204
500	500	504
1000	1000	1003
1500	1500	1503
2000	2000	2002
2350	2352	2352

The next table compares the RPM of the system with the duty cycle %, which is how hard the system is working.

RPM Set-Point	Actual RPM	% Duty Cycle
200	204	4.7%
500	504	7.8%
1000	1003	13.3%
1500	1503	21.9%
2000	2001	41.4%
2100	2106	50%
2200	2201	62.5%
2300	2295	87%
2350	2349	99.6%

The practical results were obtained using picoscope software which was attached to the practical circuit.

The tests look good. There is a full range of testing done for comparison.

6. Discussion and Conclusions

The overall aims of the assignment were met – the aims were to interface a microcontroller to a DC motor, incremental encoder, analogue sensing circuit, 12-button keypad, LCD and two pushbuttons. PWM speed control was utilized to control the motor with current feedback, and a speed set-point could be entered via the connected keypad. The LCD display showed the speed set-point, actual speed, the error, the prop part, the int accum and the armature current.

The programming language that was used for the program was C, it was chosen as it is a suitable programming language for embedded systems, is relatively simple, and can perform all of the necessary functions to complete the task at hand. The program that was used to write the code for the system was MPLab due to its intuitiveness, and the program that was used to design the circuits in order to simulate the system was Proteus as it proves an excellent program for real-time circuit simulation.

The majority of the assignment was carried out as an individual, but the physical build for the final design was carried out as a group with the assistance of the lecturer. The report writing, simulation, calculations and conclusion were all carried out as an individual.

*Analog sensing circuit
was tested on prototype
only and omitted in final
system due to LCD
space. Comparisons here about
how the system
performed with
disturbances as tested in
the demonstration.*

7. Appendices & Bibliography

References

Avago. (2012, March 22). *HEDM-55xx/560x & HEDS-55xx/56xx Datasheet*. Retrieved from Avago Technologies: https://www.infineon.com/dgdl/Infineon-Encoder_HEDS-5540-A14-AP-v01_00-EN.pdf?fileId=5546d46147a9c2e40147d3d593970357

eblocks Keypad Board. (n.d.). Retrieved from MATRIX:
<http://www.matrixtsl.com/resources/files/datasheets/EB014-30-1.pdf>

MATRIX. (n.d.). *e-blocks*. Retrieved from MATRIXtsl:
<http://www.matrixtsl.com/resources/files/datasheets/HP488-30-3.pdf>

PIC16F87XA Data Sheet. (2003). Retrieved from Microchip:
<http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>

*The refs are enough for this report
as there is no research section.*

assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

65/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

Text Comment. Current sensing was not used in the final system only in one of the prototypes because of space on the LCD.

Text Comment. Good intro

PAGE 5

PAGE 6

PAGE 7

PAGE 8

Text Comment. The hardware description is reasonable, but the MOSFET interface is missing.

PAGE 9

PAGE 10

PAGE 11

PAGE 12

Text Comment. Explanation of the configuration is good.

PAGE 13

Text Comment. Prototypes have been identified here - good.

Text Comment. Prototype has been designed and tested using CAD software.

PAGE 14

Text Comment. Keypad design and test is good

PAGE 15

Text Comment. Design looks good.

PAGE 16

Text Comment. Good explanation.

PAGE 17

Text Comment. Commented code is good.

Text Comment. Good explanations of the initial conditions and identifying how the 10 bit PWM is configured.

PAGE 18

Text Comment. Good explanation of functions. Shows a good understanding of the process.

PAGE 19

PAGE 20

Text Comment. Explanation is a little brief. There are 3 sets of display on each line of the LCD defined by the arguments of the prototype function.

PAGE 21

Text Comment. This is also a little brief. The columns are enabled and the related row is scanned. This is done in turn by column.

PAGE 22

PAGE 23

PAGE 24

PAGE 25

Text Comment. The image of the development board under test should be included here.

PAGE 26

Text Comment. The tests look good. There is a full range of testing done for comparison.

Text Comment. Analog sensing circuit was tested on prototype only and omitted in final system due to LCD Space.

Text Comment. Should be some comparisons here about how the system performed with disturbances as tested in the demonstration.

Text Comment. The refs are enough for this report as there is no research section.

GRADING FORM: CSG-RUBIC

JOSEFF JONES

0

RESEARCH-INFORMED LITERATURE

KNOWLEDGE AND UNDERSTANDING



Introduction briefly covers the tasks with the aid of a system diagram. 3/5 System hardware is explained well, but there are parts missing such as the MOSFET interface. 2.5/5 The design and configuration has been explained well and the prototype method of development and its importance is pointed out. 15/20 Software design shows a good understanding of how the program

27/40

was put together from the individual prototypes.
The code is explained well. 6.5/10

ANALYSIS



The testing has been thorough and all aspects have been explained with calculations. 6.5/10 Conclusion is reasonable. An improvement would be to include the performance of the system with a disturbance to load the output. 5.5/10

12/20

PRACTICAL APPLICATION AND DEP.



Practical application was good. a good working demo was carried out with the CAD system and development board and results recorded. The prototype design method is tried and tested and worked well with this system. 6.5/10 Quality of work is good. 6.5/10

13/20

SKILLS FOR PROFESSIONAL PRAC.



Report is of a good standard. 6/10 Team working was very good with some leadership qualities demonstrated. 7/10

13/20