

Assignment 2 Joshua Beer

by Joshua Beer

FILE	12940_JOSHUA_BEER_ASSIGNMENT_2_JOSHUA_BEER_145116_1532953815.DOCX		
TIME SUBMITTED	27-MAY-2018 08:38PM (UTC+0100)	WORD COUNT	4878
SUBMISSION ID	89977399	CHARACTER COUNT	29823



DC Motor Controller

Configuration and Programming of Embedded System.

Joshua Beer

PSDE5013

27/05/2018

Contents

.....	1
1. Introduction.....	6
2. System Hardware	7
Selection of Microcontroller.....	7
Closed Loop Feedback	8
Keypad Interface	9
Start/Stop Pushbuttons	10
Development Board	10
LCD screen	11
DC motor.....	11
3. Microcontroller Configuration	13
Timer 0.....	13
OPTION_REG	15
INTCON_REG	16
Timer 1.....	17
PWM	21
Ports	21
Other Registers Used	22
4. Software Design and Coding	30
5. Testing and Analysis of Results.....	42
PWM test	46
Simulation Testing	47
Practical Testing	52
Analysis of Testing.....	56
6. Conclusion.....	57
7. References	59

Works Cited.....	59
8. Appendix.....	60

Figure 1 – System Block Diagram.....	6
Figure 2 - PIC16F877	7
Figure 3 - Incremental Encoder Diagram	8
Figure 4 - Keypad	9
Figure 5 - Matrix HP488 Development Board	10
Figure 6 - Motor MOSFET Setup	12
Figure 7 – Timer 0 Architecture	13
Figure 8 - Equation for Timer Preload Utilising Internal Clock	14
Figure 9 – OPTION_REG	15
Figure 10 - INTCON_REG Description	16
Figure 11 - Timer 1 Architecture	17
Figure 12 - T1CON Description	18
Figure 13 - Timer 2 Architecture	19
Figure 14 - Timer 2 Register	20
Figure 15 - PWM Diagram	21
Figure 16 – ADCON1 Register	22
Figure 17 – PIE1 Register	23
Figure 18 - PIE2 Register	24
Figure 19 - PIR1 Register	25
Figure 20 - PIR2 Register	26
Figure 21 - T1CON Register	27
Figure 22- Timer 2 Associated Registers	28
Figure 23 - CCP1CON Register	29
Figure 24 - PWM and Timer 2 Associated Registers	29
Figure 25 – Functions and Global Variables	30
Figure 26 - Interrupt ISR Function	31
Figure 27 - Main Program	33
Figure 28 - While Loop	35
Figure 29 - Displaying the Speed Code	38
Figure 30 - Display Values on LCD	39
Figure 31 - Scan Keypad Code	40
Figure 32 - Testing Frequency (1)	42
Figure 33 - Testing Frequency (2)	42
Figure 34 - Testing Frequency (3)	43

Figure 35 - Keypad Test (1)	44
Figure 36 - Keypad Test (2)	44
Figure 37 - Keypad Test (3)	45
Figure 38 - PWM Digital Oscilloscope 200 Test	47
Figure 39 – PWM Proteus 200 Test	48
Figure 40 - PWM Digital Oscilloscope 350 Test	49
Figure 41 - PWM Proteus 350 Test	49
Figure 42 - PWM Digital Oscilloscope 100 Test	50
Figure 43 - PWM Proteus 100 Test	50
Figure 44 - Testing Table	51
Figure 45 - PWM at 200 RPM	52
Figure 46 - Duty Cycle at 48	52
Figure 47 - PWM at 1000 RPM	53
Figure 48 - Duty Cycle at 133	54
Figure 49 - PWM at 2000 RPM	54
Figure 50 - Duty Cycle at 391	55
Figure 51 - PWM at 2300 RPM	55
Figure 52 - Duty Cycle at 744	56
Figure 53 - Thyristor	60

1. Introduction

The aims and objectives of this assignment will be to use a microcontroller to interface with the following: DC motor, incremental encoder, analogue sensing circuit, keypad, LCD display and two pushbuttons.

Using the embedded system from the on-chip peripherals, a system must be designed and implemented to provide PWM speed control for the motor with the information being fed back.

A speed set-point must be entered using a keypad and speed sensing feedback must be applied via the encoder and fed back using an analogue value of armature current.

The current feedback was done in the prototype stage and left out of the final design due to space on the LCD.

The LCD display must show the speed set-point, actual speed and armature current.

The system must also be started and stopped using pushbuttons placed on the system.

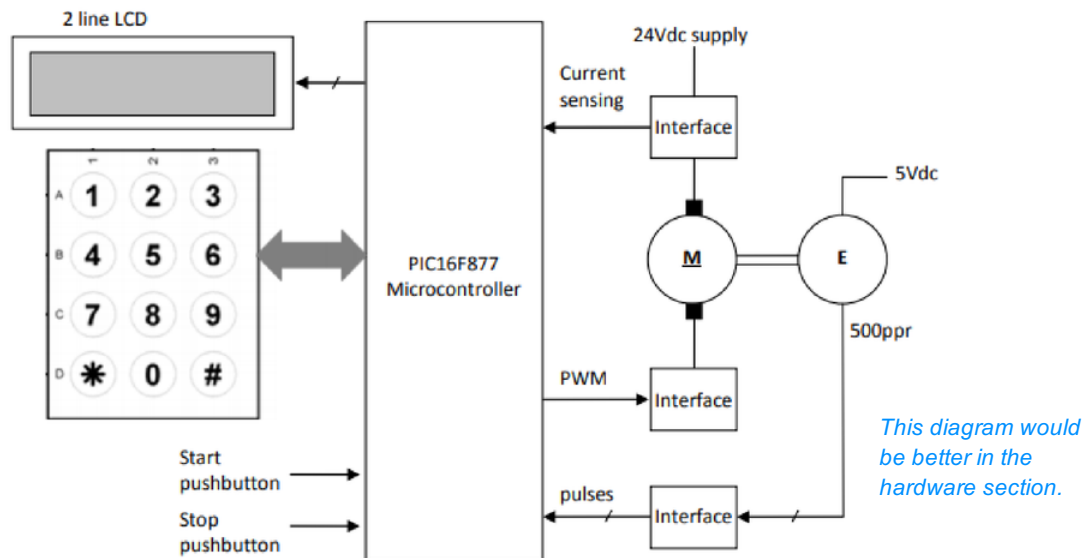


Figure 1 – System Block Diagram

This does give some idea of what the task is about.

2. System Hardware

Selection of Microcontroller

The specific microcontroller chosen for the assignment was the PIC16F877. The PIC chip being used includes 40 different inputs or outputs; it also has PWM Capabilities and all the other on-chip peripherals to meets the criteria for the assignment.

40-Pin PDIP

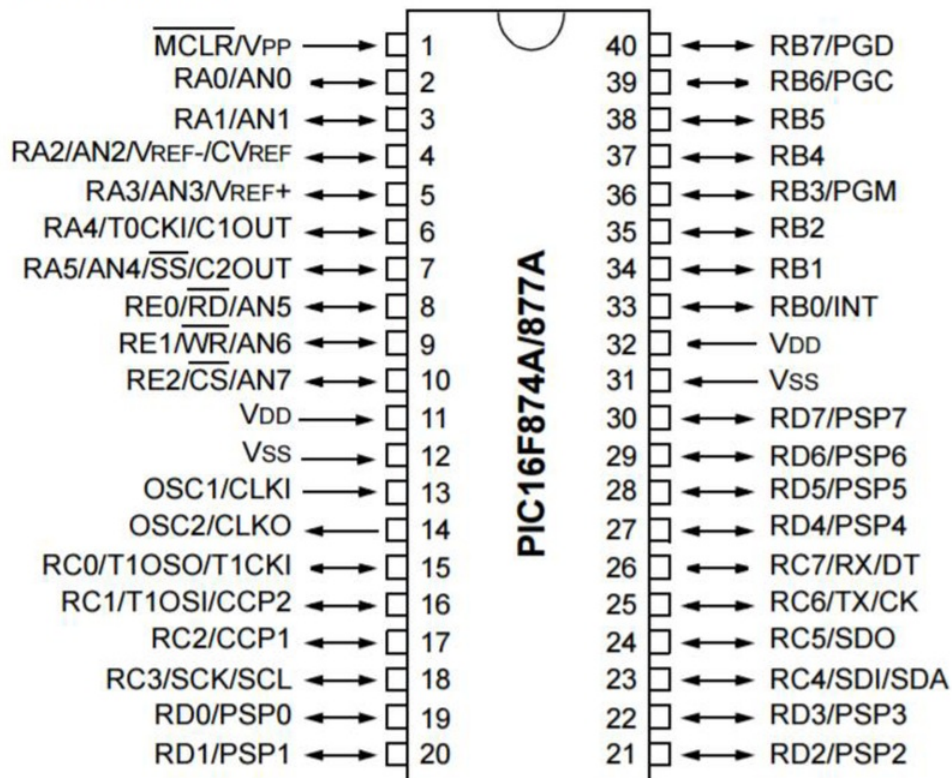


Figure 2 - PIC16F877

(Microchip, 2003)

Error! Reference source not found. above shows the pin out view of the PIC16F877. It shows the 40 different I/O's available to be configured.

Closed Loop Feedback

For the assignment, the speed from the controller needed to be accurately measured and fed back into the system; this is why a closed loop PI controller was used.

Because the system is a closed loop feedback circuit, the speed needs controlling and first has to be measured before the data is sent to the controller itself.

For this particular system (as set out in the aims and objectives taken from the assignment brief), an incremental encoder with an output of 500 PPR (pulses per revolution) was chosen.

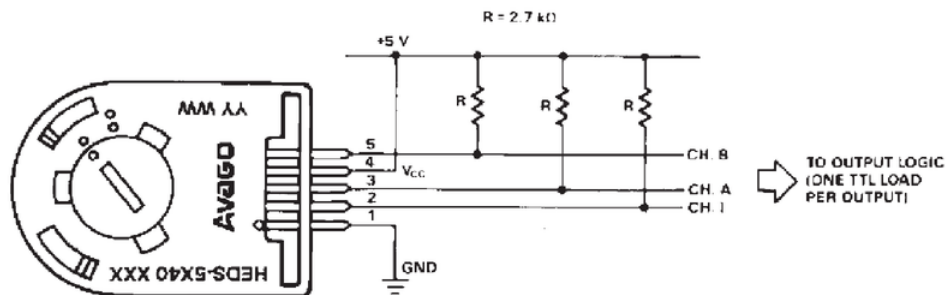


Figure 3 - Incremental Encoder Diagram

(AVAGO technologies, 2012)

Error! Reference source not found. above shows a diagram of the encoder used in the assignment. It shows the pin-out for the encoder, as well as the connections for the device, which are:

- Ground
- V_{CC}
- Output channels

Suitable pullup resistors will be used to connect pins 5, 3 and 2 to the V_{CC} for the system.

Keypad Interface

The assignment brief specifies that the user must be able to enter the speed set point using a keypad; this is set out in the aims and objectives which means that the system must include a keypad.

In the system a 12-button keypad will be used, using a suitable interface to allow the microcontroller to read and use the keypad buttons correctly to control the speed according to the set point entered.

The keypad in question is a 3x4 row input device. This means multiple buttons share the same pin so the device has to be identified through the use of a function to decide which particular button the user has pressed.

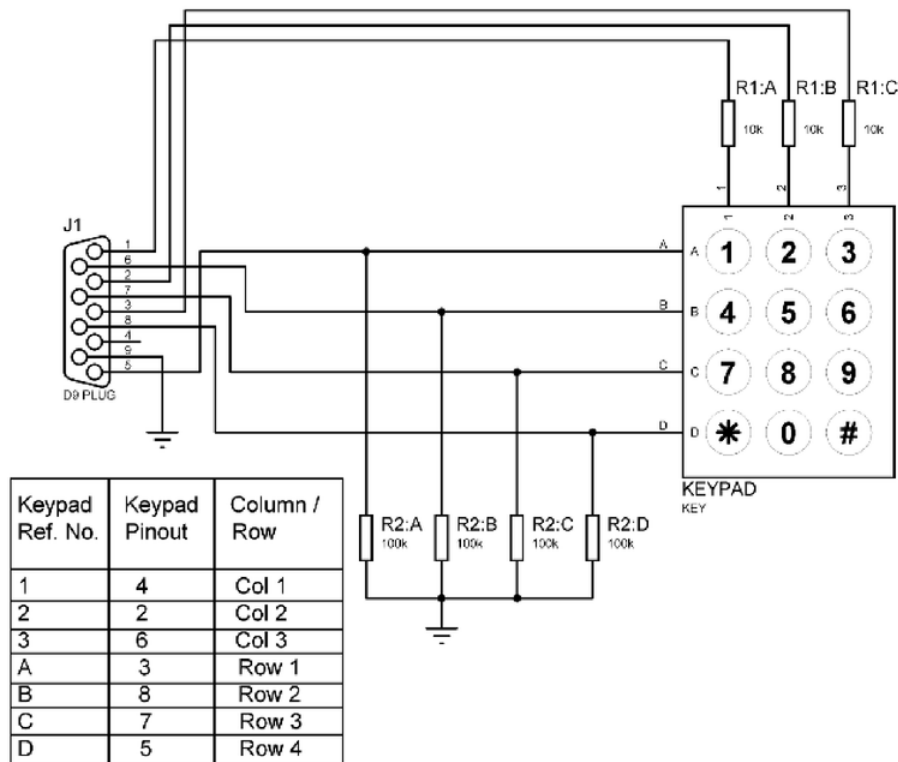


Figure 4 - Keypad

(Matrix, Unspecified)

The diagram in **Error! Reference source not found.** above shows the electrical connections within the keypad device being used in the assignment which is taken from a source from a manufacturer of a similar keypad.

Start/Stop Pushbuttons

The development board that is being used in this assignment includes a number of pushbuttons, two of which are used from PORTA.

One will be used to start the system and one used to stop it.

Development Board

The development board used in this assignment is the HP488.

A picture of the board, including important component labels is shown in figure 5 below, sourced from the manufacturer's data sheet.

Board layout

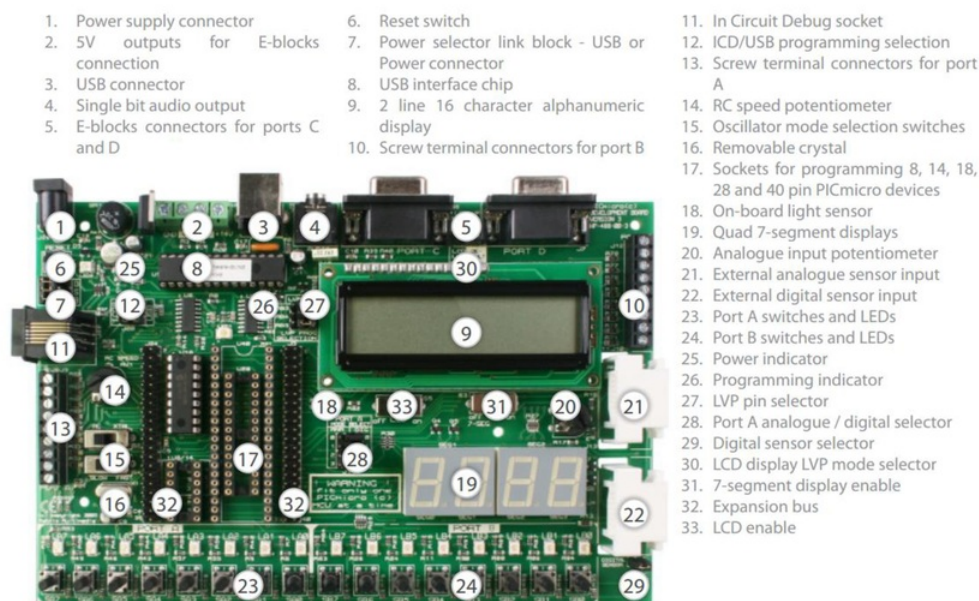


Figure 5 - Matrix HP488 Development Board

(Matrix, Unspecified)

LCD screen

The development has a built-in LCD screen that will be used to achieve the aims and objectives of displaying the speed set-point, actual speed and armature current.

The LCD is connected on the PCB to PORT B using RB0 - RB5 on the PIC chip itself.

Although to allow the use of the LCD screen, the switch labelled 33 in **Error! Reference source not found.** has to be set to the ON position. As this is the LCD enable part of the development board as shown in the figure.

The LCD screen has 2 lines which can be utilised to show values on containing 16 characters per line, which can easily display all necessary values and data described above.

DC motor

The DC motor used within the project has an operating voltage of 24V DC, with the microcontroller having an output voltage of 5V DC.

To allow the safe interaction between the devices a junction will have to be used to isolate the 2 separate voltages from each other.

A MOSFET transistor will be able to switch the motor at the required PWM with the microcontroller activating the MOSFET's gate. (See appendix for how MOSFET's (thyristors work).

With this setup, the microcontroller will output its PWM wave to the transistor, and the transistor will switch the 24V to the DC motor, keeping the 5V and 24V isolated from each other.

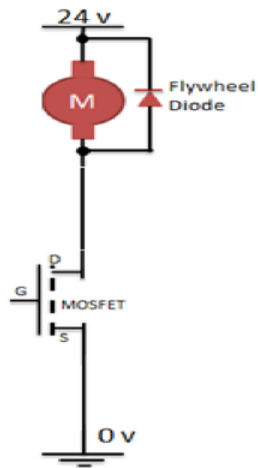


Figure 6 - Motor MOSFET Setup

Figure 6 above shows the motor connections, as well as the placement of a flywheel diode to dissipate any back EMF.

*Hardware has
been explained
fairly well.*

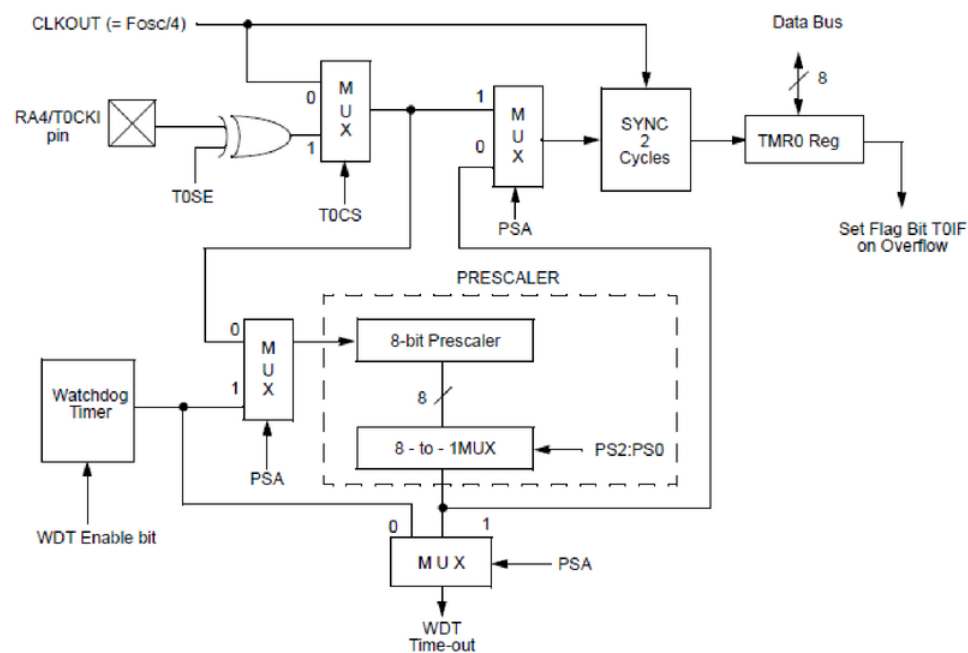
3. Microcontroller Configuration

Timer 0

Figure 7 below shows the internal architecture of timer 0 on the PIC16F877 chip, sourced from the manufactures datasheet.

Before timer 0 can be used, a few registers have to be set, these are the:

- OPTIONS_REG
- INTCON_REG
- TIOF overflow value



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

Figure 7 – Timer 0 Architecture

(Microchip, 2003)

To give accurate timing the microcontroller was configured to use timer 0 with an interrupt interval of 10mS.

The clock source F_{osc} of the PIC16F877 is 19.6608MHz.

$$\text{Countable clock speed} = F_{osc}/4 = 4915200\text{Hz}$$

$$(1/4915200)*256 = 52\mu\text{S}$$

$$\text{MAX PRESCALER} = 256$$

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

$$\text{Period} = 1/19200 \text{ secs}$$

$$\text{Number of counts every } 1/100 \text{ secs} = 192 \text{ (to give 10mS interrupt)}$$

$$\text{Preload timer 0 with } 256 - 192 = 64 \text{ to interrupt every 10mS}$$

Figure 8 - Equation for Timer Preload Utilising Internal Clock

OPTION_REG

The OPTION_REG is shown in 9, detailing what each bit does.

To achieve the aims of the assignment the OPTION_REG is set to 0b10000111.

With bit 7 furthest left and bit 0 furthest right. Therefore the TMR0 rate is selected at 1:256 as it's 111 for bits 0-2. The other bit selected is the RBPU as represented by bit 7.

REGISTER 5-1: OPTION_REG REGISTER

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	
	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7					bit 0			
bit 7	RBPU							
bit 6	INTEDG							
bit 5	T0CS: TMR0 Clock Source Select bit							
	1 = Transition on T0CKI pin							
	0 = Internal instruction cycle clock (CLKOUT)							
bit 4	T0SE: TMR0 Source Edge Select bit							
	1 = Increment on high-to-low transition on T0CKI pin							
	0 = Increment on low-to-high transition on 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					

Figure 9 – OPTION_REG

(Microchip, 2003)

INTCON_REG

The INTCON_REG has to be set in the same way as the OPTION_REG, but the effect of changing the bits has completely different effects.

These effects are detailed below using the register in figure 10.

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	T0IE	INTE	RBIE	T0IF	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	T0IE: 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	T0IF: 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						

Figure 10 - INTCON_REG Description

(Microchip, 2003)

The bits set for INTCON_REG have a binary value of 0b11100000.

This means that the bits to be enabled in the INTCON_REG are the global interrupts to enable the CPU, peripheral interrupts and TMR0 overflow; these have been enabled by setting the bit to a high logic of 1.

Timer 1

Figure 11 below shows the internal architecture of timer 1 on the PIC16F877 chip, sourced from the manufacturers datasheet.

Timer 1 is a 16-bit timer with a maximum count value of 65535.

As it's a 16-bit timer, Timer 1 uses two 8-bit registers TMR1H (H for high) and TMR1L (L for low), storing the high byte and low byte of the 16bit count respectively.

The maximum no load speed of the DC motor is 2300 RPM. The encoder feedback takes its value from the input pin RC0 on the microcontroller. .

Encoder outputs to RC0 input of MCU.

TIMER1 BLOCK DIAGRAM

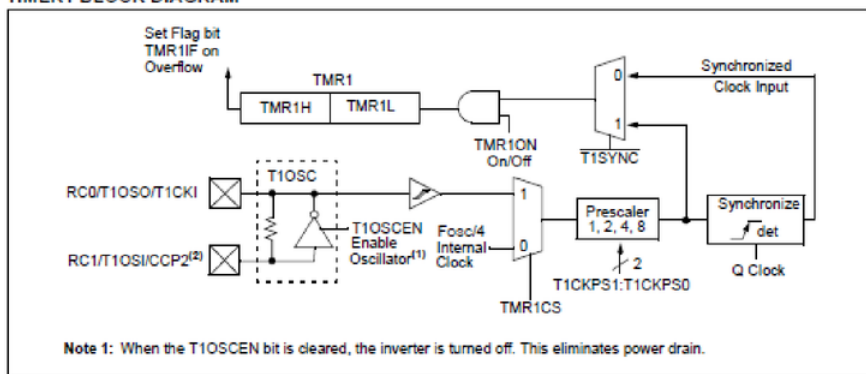


Figure 11 - Timer 1 Architecture

(Microchip, 2003)

Timer 1 uses a separate register than timer 0 to configure the device, T1CON.

The description of T1CON is shown in **Figure 122** below.

	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							
	<u>When TMR1CS = 1:</u>							
	1 = Do not synchronize external clock input							
	0 = Synchronize external clock input							
	<u>When TMR1CS = 0:</u>							
	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 12 - T1CON Description

(Microchip, 2003)

To allow the device to operate as per the specification the T1CON register has to be set at the following binary value 0b00000111. This enables timer 1, enables the external clock from pin RC0/T1OSO/T1CKI (on rising edge) and does not synchronise with the external clock input.

Some other registers have to be set as well to enable the timer 1 overflow. The PIE1 and PIR1 registers have to be set to the following binary values: PIE1 = 0b00000001 and PIR1 = 0b00000000.

The register needs to be able to overflow to use both the high and low timers, TMRH and TRML.

Timer 2

Figure 13 below shows the internal architecture of timer 2 on the PIC16F877 chip, sourced from the manufactures datasheet.

Timer 2's primary function is to interface PWM waves to the DC motor.

Timer 2 is an 8-bit period register giving a maximum count of 256 (as previously stated). The registers that control Timer 2 have to also be set up, which are separate registers than the ones controlling timer 0 and 1.

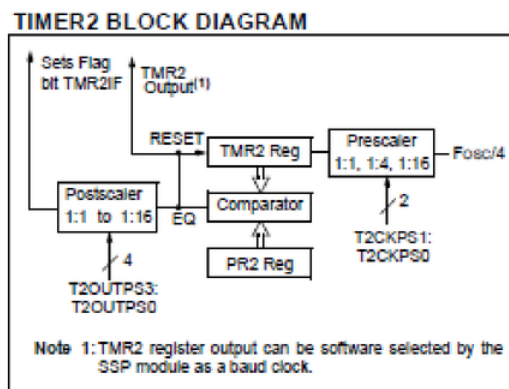


Figure 13 - Timer 2 Architecture

(Microchip, 2003)

	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	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 14 - Timer 2 Register

(Microchip, 2003)

The timer 2 register has to be set to the following binary value: 0b00000101.

This sets the prescaler to 4 and turns TMR2 on.

To enable PWM the CCP1CON register has to be set to a binary value of: 0b00001100.

PWM

Figure 15 shows an 8-bit register is used to store CCPR1L which can be extended to use 10 bits using CCP1CON<5:4>.

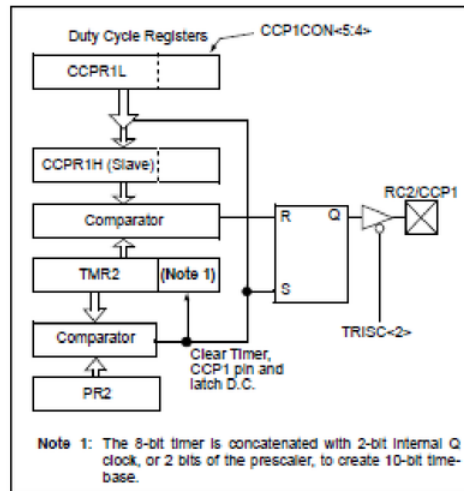


Figure 15 - PWM Diagram

(Microchip, 2003)

The way the chip calculates the PWM duty cycle is by comparing RC2 to PR2, TMR2 then CCPR1L.

Two other registers have to be set to enable the PWM to work correctly, these are:

- PIE2 to 0b00000000
- PIR2 to 0b00000000

Ports

To adhere to the assignment brief and aims and objectives the ports are set as follows:

- Port C has one output that is used for PWM output RC2 - RC0, all other ports in Port C are inputs. Therefore, the TRISB was set to 0b11111011.
- Port D is used to interface the microcontroller with the 12-button keypad. This means RD0 - RD3 need to be inputs and RD4 - RD7 need to be outputs. This means that TRISD needs to be set to 0b11110000.

RC0 is the input for encoder feedback.

Inputs for rows and outputs for columns.

Other Registers Used

All screenshots of registers taken from manufactures datasheet.

REGISTER 11-2: ADCON1 REGISTER (ADDRESS 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7				bit 0			

- bit 7 **ADFM:** A/D Result Format Select bit
 1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
 0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.
- bit 6-4 **Unimplemented:** Read as '0'
- bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7 ⁽¹⁾ RE2	AN6 ⁽¹⁾ RE1	AN5 ⁽¹⁾ RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs ⁽²⁾
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

- Note 1:** These channels are not available on PIC16F873/876 devices.
Note 2: This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.

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

The ADRESH:ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D result register pair, the GO/DONE bit (ADCON0<2>) is cleared and the A/D interrupt flag bit ADIF is set. The block diagram of the A/D module is shown in Figure 11-1.

After the A/D module has been configured as desired, the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as inputs.

To determine sample time, see Section 11.1. After this acquisition time has elapsed, the A/D conversion can be started.

Figure 16 – ADCON1 Register

PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EEIE	ADIE	RCIE	C2IE	C1IE	OSFIE	TXIE	TMR1IE
bit 7							bit 0

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

bit 7	EEIE: EE Write Complete Interrupt Enable bit 1 = Enables the EE write complete interrupt 0 = Disables the EE write complete interrupt
bit 6	ADIE: A/D Converter (ADC) Interrupt Enable bit 1 = Enables the ADC interrupt 0 = Disables the ADC interrupt
bit 5	RCIE: EUSART Receive Interrupt Enable bit 1 = Enables the EUSART receive interrupt 0 = Disables the EUSART receive interrupt
bit 4	C2IE: Comparator 2 Interrupt Enable bit 1 = Enables the Comparator C2 interrupt 0 = Disables the Comparator C2 interrupt
bit 3	C1IE: Comparator 1 Interrupt Enable bit 1 = Enables the Comparator C1 interrupt 0 = Disables the Comparator C1 interrupt
bit 2	OSFIE: Oscillator Fail Interrupt Enable bit 1 = Enables the oscillator fail interrupt 0 = Disables the oscillator fail interrupt
bit 1	TXIE: EUSART Transmit Interrupt Enable bit 1 = Enables the EUSART transmit interrupt 0 = Disables the EUSART transmit interrupt
bit 0	TMR1IE: Timer1 Overflow Interrupt Enable bit 1 = Enables the Timer1 overflow interrupt 0 = Disables the Timer1 overflow interrupt

PIE1 register bits are to enable the peripheral interrupts

Figure 17 – PIE1 Register

PIC16F882/883/884/886/887

2.2.2.5 PIE2 Register

The PIE2 register contains the interrupt enable bits, as shown in Register 2-5.

Note: Bit PEIE of the INTCON register must be set to enable any peripheral interrupt.

REGISTER 2-5: PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
OSPIE	C2IE	C1IE	EEIE	BCIE	ULPWUE	—	CCP2IE
bit 7							bit 0

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

- bit 7 **OSPIE:** Oscillator Fail Interrupt Enable bit
 - 1 = Enables oscillator fail interrupt
 - 0 = Disables oscillator fail interrupt
- bit 6 **C2IE:** Comparator C2 Interrupt Enable bit
 - 1 = Enables Comparator C2 interrupt
 - 0 = Disables Comparator C2 interrupt
- bit 5 **C1IE:** Comparator C1 Interrupt Enable bit
 - 1 = Enables Comparator C1 interrupt
 - 0 = Disables Comparator C1 interrupt
- bit 4 **EEIE:** EEPROM Write Operation Interrupt Enable bit
 - 1 = Enables EEPROM write operation interrupt
 - 0 = Disables EEPROM write operation interrupt
- bit 3 **BCIE:** Bus Collision Interrupt Enable bit
 - 1 = Enables Bus Collision interrupt
 - 0 = Disables Bus Collision interrupt
- bit 2 **ULPWUE:** Ultra Low-Power Wake-up Interrupt Enable bit
 - 1 = Enables Ultra Low-Power Wake-up interrupt
 - 0 = Disables Ultra Low-Power Wake-up interrupt
- bit 1 **Unimplemented:** Read as '0'
- bit 0 **CCP2IE:** CCP2 Interrupt Enable bit
 - 1 = Enables CCP2 interrupt
 - 0 = Disables CCP2 interrupt

Figure 18 - PIE2 Register

PIR1: PERIPHERAL INTERRUPT REQUEST REGISTER 1

R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0
EEIF	ADIF	RCIF	C2IF	C1IF	OSFIF	TXIF	TMR1IF
bit 7							bit 0

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

bit 7	EEIF: EEPROM Write Operation Interrupt Flag bit 1 = The write operation completed (must be cleared in software) 0 = The write operation has not completed or has not been started
bit 6	ADIF: A/D Converter Interrupt Flag bit 1 = A/D conversion complete (must be cleared in software) 0 = A/D conversion has not completed or has not been started
bit 5	RCIF: EUSART Receive Interrupt Flag bit 1 = The EUSART receive buffer is full (cleared by reading RCREG) 0 = The EUSART receive buffer is not full
bit 4	C2IF: Comparator C2 Interrupt Flag bit 1 = Comparator output (C2OUT bit) has changed (must be cleared in software) 0 = Comparator output (C2OUT bit) has not changed
bit 3	C1IF: Comparator C1 Interrupt Flag bit 1 = Comparator output (C1OUT bit) has changed (must be cleared in software) 0 = Comparator output (C1OUT bit) has not changed
bit 2	OSFIF: Oscillator Fail Interrupt Flag bit 1 = System oscillator failed, clock input has changed to INTOSC (must be cleared in software) 0 = System clock operating
bit 1	TXIF: EUSART Transmit Interrupt Flag bit 1 = The EUSART transmit buffer is empty (cleared by writing to TXREG) 0 = The EUSART transmit buffer is full
bit 0	TMR1IF: Timer1 Overflow Interrupt Flag bit 1 = The TMR1 register overflowed (must be cleared in software) 0 = The TMR1 register did not overflow

Figure 19 - PIR1 Register

PIR2 Register

The PIR2 register contains the interrupt flag bits.



Fig. 2-18 PIR2 register

OSFIF - Oscillator Fail Interrupt Flag bit.

- 1 - System oscillator failed and clock input has changed to internal oscillator INTOSC. This bit must be cleared in software.
- 0 - System oscillator operates normally.

C2IF - Comparator C2 Interrupt Flag bit.

- 1 - Comparator C2 output has changed (bit C2OUT). This bit must be cleared in software.
- 0 - Comparator C2 output has not changed.

C1IF - Comparator C1 Interrupt Flag bit.

- 1 - Comparator C1 output has changed (bit C1OUT). This bit must be cleared in software.
- 0 - Comparator C1 output has not changed.

EEIF - EE Write Operation Interrupt Flag bit.

- 1 - EEPROM write completed. This bit must be cleared in software.
- 0 - EEPROM write is not completed or has not started.

BCLIF - Bus Collision Interrupt Flag bit.

- 1 - A bus collision has occurred in the MSSP when configured for I2C Master mode. This bit must be cleared in software.
- 0 - No bus collision has occurred.

ULPWUIF - Ultra Low-power Wake-up Interrupt Flag bit.

- 1 - Wake-up condition has occurred. This bit must be cleared in software.
- 0 - No Wake-up condition has occurred.

CCP2IF - CCP2 Interrupt Flag bit.

- 1 - CCP2 interrupt condition has occurred (unit for capturing, comparing and generating PWM signal). Depending on operating mode, capture or compare match has occurred. In both cases, the bit must be cleared in software. This bit is not used in PWM mode.
- 0 - No CCP2 interrupt condition has occurred.

Figure 20 - PIR2 Register

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T1GINV ⁽¹⁾	TMR1GE ⁽²⁾	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

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

- bit 7 **T1GINV:** Timer1 Gate Invert bit⁽¹⁾
1 = Timer1 gate is active-high (Timer1 counts when gate is high)
0 = Timer1 gate is active-low (Timer1 counts when gate is low)
- bit 6 **TMR1GE:** Timer1 Gate Enable bit⁽²⁾
If TMR1ON = 0:
This bit is ignored
If TMR1ON = 1:
1 = Timer1 is on if Timer1 gate is not active
0 = Timer1 is on
- bit 5-4 **T1CKPS<1:0>:** 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:** LP Oscillator Enable Control bit
If INTOSC without CLKOUT oscillator is active:
1 = LP oscillator is enabled for Timer1 clock
0 = LP oscillator is off
Else:
This bit is ignored. LP oscillator is disabled.
- bit 2 **T1SYNC:** Timer1 External Clock Input Synchronization Control bit
TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock
- bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from T1CKI pin (on the rising edge)
0 = Internal clock (Fosc/4)
- bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Note 1: T1GINV bit inverts the Timer1 gate logic, regardless of source.

2: TMR1GE bit must be set to use either T1G pin or COUT, as selected by the T1GSS bit of the CMCON1 register, as a Timer1 gate source.

Figure 21 - T1CON Register

REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
92h	PR2	Timer2 Period Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.

Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

The operation of timer2 is controlled by T2CON register. In below find the bit pattern of T2CON register.

T2CON: Timer2 Control Register

U-0	R/W-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
•
•
•
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 22- Timer 2 Associated Registers

REGISTER 10-1: CCP1CON REGISTER (ADDRESS 17h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		
—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0		
bit7								bit0	

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7-6: **Unimplemented:** Read as '0'

bit 5-4: **CCP1X:CCP1Y:** PWM Least Significant bits
Capture Mode: Unused
Compare Mode: Unused
PWM Mode: These bits are the two LSBs of the PWM duty cycle. The eight MSBs are found in CCPRxL.

bit 3-0: **CCP1M3:CCP1M0:** CCPx Mode Select bits
0000 = Capture/Compare/PWM off (resets CCP1 module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCP1IF bit is set)
1001 = Compare mode, clear output on match (CCP1IF bit is set)
1010 = Compare mode, generate software interrupt on match (CCP1IF bit is set, CCP1 pin is unaffected)
1011 = Compare mode, trigger special event (CCP1IF bit is set; CCP1 resets TMR1)
11xx = PWM mode

Figure 23 - CCP1CON Register

REGISTERS ASSOCIATED WITH PWM AND TIMER2

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- --0	---- --0
8Ch	PIE1	PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- --0	---- --0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
92h	PR2	Timer2 Module's Period Register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
18h	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PWM and Timer2.

Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

Figure 24 - PWM and Timer 2 Associated Registers

4. Software Design and Coding

To be able to implement the system and control the speed via the set point and also monitor the speed and feedback into the system while also measuring the analogue value for the armature current, the software design and code must utilise the system hardware and microcontroller functions detailed above.

This was done by designing the code below.

```
#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_accu = 0 ,Kp = 0.5, Ki = 0.1;
```

The current monitoring was done only in the prototype stage and was not included due to space on the LCD. If the current monitoring is to be mentioned then the design must also be included.

Figure 25 – Functions and Global Variables

Figure 25 shows the pic.h and LCD.h which are pre-set code called into the main program from MPLAB itself.

There are then 4 functions declared to be used in the program, followed by the global variables for the program which can be changed and used within the functions.

These start with fixed values, some are always int (integer) values which means they will always remain at whole numbers. Whilst others will have float (floating) values which means they can use decimals to display as value, this is shown as Kp = 0.5 and Ki = 0.1 as their set global variable values.

These values were used for the simulation of the program while the values for the practical were Kp = 0.01 and Ki = 0.02.

Good points made as the CAD and dev board systems were slightly different.

```

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==1000) && (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 input
encoder
and PWM
rely on the
ISR.*

Figure 26 - Interrupt ISR Function

Three sources exist within the ISR:

- TOIF
- TMR1IF
- TMR2IF

All three are bits that represent the interrupt flag for; timer 0, timer 1 and timer 2.

If the interrupt flag is set for timer 1, the fault bit is set to indicate a problem with the system.

If timer 0 overflows and the flag is set, the following instructions are carried out:

- The flag bit is first reset.
- The TRM0 is pre-loaded with a value of 64 which means the count is already at 192 ($256 - 64 = 192$).
- The variable milliseconds is incremented by 10. The if statement checks to see if the value of milliseconds is equal to 1000, this results in the containing code running once every 1000ms.
- speed_low_byte and speed_high_byte are both set to the values held in TMR1L and TMR1H respectively.
- TMR1L, TMR1H and milliseconds are then all set to 0.

Timer 1 is in event mode counting incoming pulses so if timer 1 overflows and the bit is set to 1 (as shown in the code), the system isn't functioning correctly and will show a fault as mentioned above.

If timer 2 overflows and the overflow bit is set to 1, the code clears the interrupt flag bit.

*Good
explanation*

```

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;

    T1CON = 0b00000111;    //setup for TIMER 1 in event mode from RC0
    TMR1L = 0;
    TMR1H = 0;

    PR2 = 255;
    CCP1L = 0;             //0% DUTY CYCLE
    T2CON = 0b00000101;    //set up for Timer 2 in PWM mode using RC2
    CCP1CON = 0b00001100;  //set up for Timer 2 in PWM mode using RC2
                          //sets pwm mode and the extra 2 LSB's for PWM
                          //this is the alternative to 1023 / 4 for max PR2 = 255
                          //PWM is now (PR2 = 255)8 bits + CCP1CON 2 bits
                          //this gives you the 10 bits PWM

```

Figure 27 - Main Program

Figure 27 shows the main program with the first line of code setting the enable_keypad set to a value 1, and key is set to no specified value.

TRISA is set to 11111111 which is all inputs.

ADCON is set to 00001110 which sets the specific inputs and outputs using the ADCON register already stated in the assignment.

TRISB is used to the LCD and is set to 00000000 which means all outputs so the LCD can display a character on each space and line of the LCD.

TRISC is set to 00000001 which means RC0 is used for counting with TMR0 and is an input (1) and RC2 used for PWM with TMR2 set as an output (0).

The output pins on PORTB are then set to logic 0.

The LCD_Startup function is one of the pre-written functions that were included in the LCD.h file.

PIE1, PIE2, PIR1 & PIR2 are registers that control/configure the peripheral interrupts for TMR1 and TMR2. (These are set according to spec and manufacturer datasheet which is detailed above already)

All other registers that have been set in the rest of the code in **Error! Reference source not found.** have all been explained in detail in the previous section of the report and further detailed in the comments next to the code.

```

while(1)
{
    speed = (int)((float)((speed_low_byte + speed_high_byte*256)*6)/50);
    // speed = (int)((float)(speed_low_byte + speed_high_byte*256));
    //speed = speed * 3.1;
    //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>2300) setpoint = 2300;
    if(setpoint<0) setpoint = 0;

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

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

    CCP1L = (int)output/4;

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

```

Figure 28 - While Loop

Figure 28 shows the while loop which will continuously repeat itself until the user ends the loop or the program is terminated.

The first part of the while loop calculates the current speed of the motor, based on values gained from the ISR detailed above into the: speed_low_byte and speed_high_byte.

The next if statements are the push buttons to start and stop the system. The first statement is the start button and the next is for the stop button.

The next part of the code is setting the keypad and entering the speed set point values.

To set the digits on the screen to what the user types, first it checks to see if a key has been pressed, if it has then the keypad will be enabled.

If the previous statement is true then each keypress will set one digit at a time, for example:

To enter the number 1055:

1 is pressed, setpoint = $(0*10) + 1 = 1$

0 is pressed, setpoint = $(1*10) + 0 = 10$

5 is pressed, setpoint = $(10*10) + 5 = 105$

5 is pressed, setpoint = $(105*10) + 5 = 1055$

Good explanation

Pressing the hash key [#] will reset the setpoint to 0.

The code once again checks to see if the keypad has been pressed and allows the user to enter another value.

The next part of the code checks to see if the setpoint value is greater than 2300, if it is the setpoint maximum will be 2300 automatically. If the setpoint value is less than 0 the setpoint will be 0 automatically.

The next part of the code will now turn on the PI controller by entering it as a high logic value of 1.

The error is first calculated by taking speed away from the value setpoint.

The proportional part for the pi controller is the error multiplied by the Kp variable.

The integral accumulator part of the PI controller stores the value of error multiplied by Ki accumulatively.

The PI controller output is equal to the proportional part + integral accumulator.

The next part of the code states that if the run variable does not equal 1, the previously mentioned parts of the PI controller are set to 0, as well as setting RC2 to 0 (which sets the port pin RC2 to logic 0).

The next two lines of code are the caps which sets the speed value to a lower limit of 0 and upper limit of 2300 (as stated above).

CCPR1L is the duty cycle register for the PWM, which is calculated based on the value of the PI output.

The DisplayValues function is then called twice to populate the LCD screen with data. Once on each line stated by 0 or 1, this is described above in the LCD outputs part of the initial code.

```

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

```

Figure 29 - Displaying the Speed Code

Figure 29 shows the code which will display the speed of the controller on the LCD line 1.

It takes the 'value' stored already in the code as value (set point entered) and then works out the thousands, hundredths, tenths and units values by dividing by 1000, 100, 10 and 1 respectively.

These values are then taken and +48 is added to display the ASCII equivalent which can be read by the LCD and then entered into the correct part of the string to display the set point followed by a space and RPM with the last LCD bit displaying 0.

The set point mentioned above would display as 1055 RPM0 on the LCD line 1.

```

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
}

```

Figure 30 - Display Values on LCD

Figure 30 shows the first part of the DisplayValues function. This function operates on exactly the same principle as the previous function, so to avoid repetition the differences will be explained.

In the main program, the DisplayValues function displays the setpoint, speed and error on the top line of the LCD while the prop_part, int_accum and output values on the bottom line of the LCD.

The other differences between this function and the previously explained function is the addition of an if statement that checks whether the values are positive or negative, which after checking will add + or – respectively.

Good explanation.

```
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 31 - Scan Keypad Code

The ScanKeypad function checks which key the user has pressed on the 12-button keypad.

The keypad diagram of which this uses has been described above in figure 4. The output for a column is set to a logic 1 and if statements check each row for a logic 1 in turn.

This then sees which exact key on the row was pressed and knows which column it is currently looking at to deduct the exact key pressed.

5. Testing and Analysis of Results

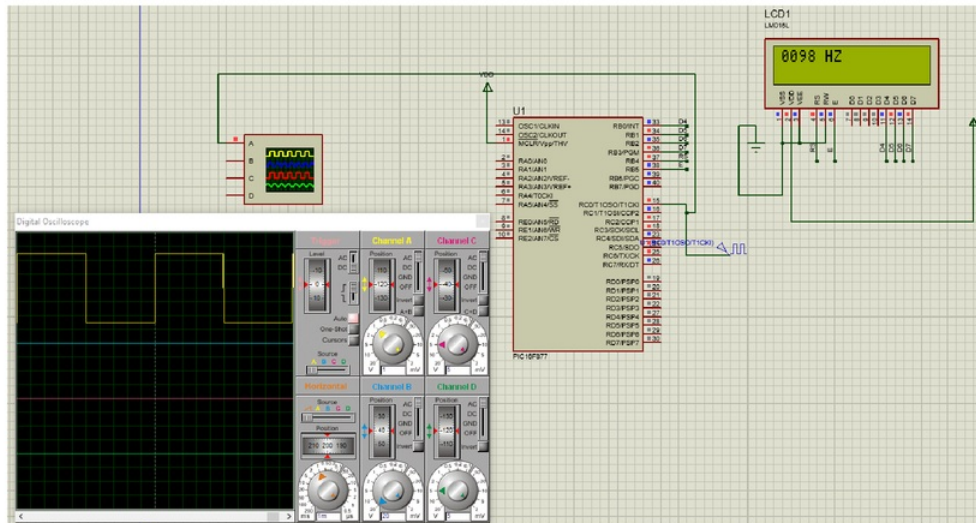


Figure 32 - Testing Frequency (1)

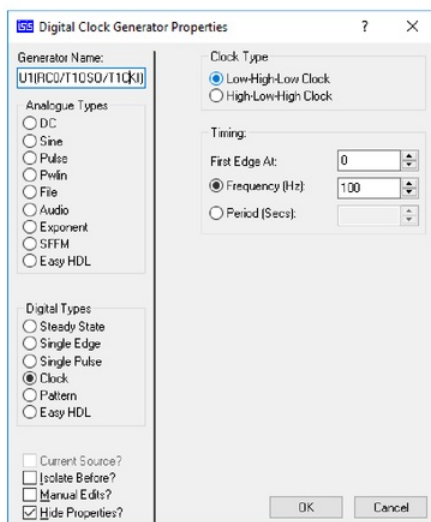


Figure 33 - Testing Frequency (2)

100 Hz = 10 blocks per pulse

$1/10 \times 10^3 = 100$

With the Hz on the screen showing 98 with a 2 Hz error.

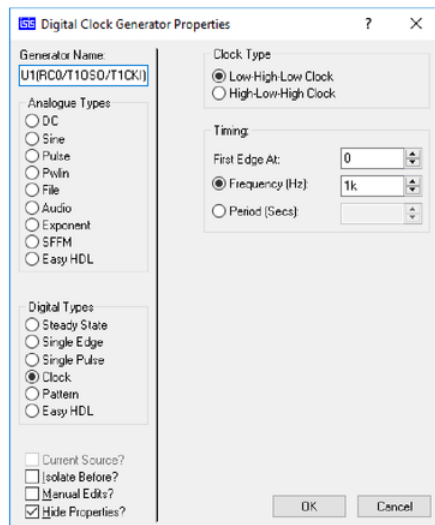


Figure 34 - Testing Frequency (3)

1000 Hz test = 1 block per pulse (10 times smaller)

$$1/1 \cdot 3 = 1000$$

Keypad encoder

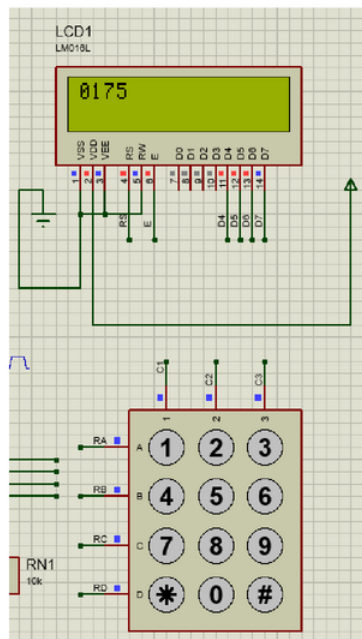


Figure 35 - Keypad Test (1)

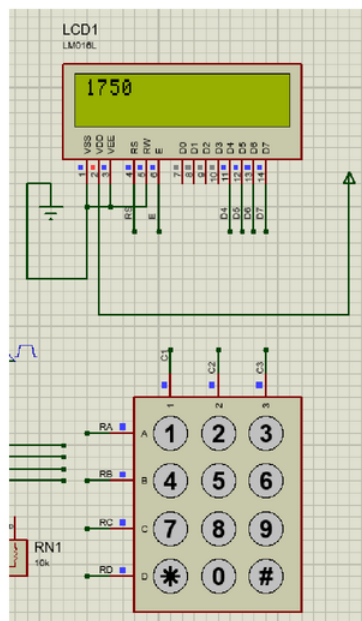


Figure 36 - Keypad Test (2)

Figures 35 and 36 shows the code is shifting the value to the left (entered 175 then added a zero).

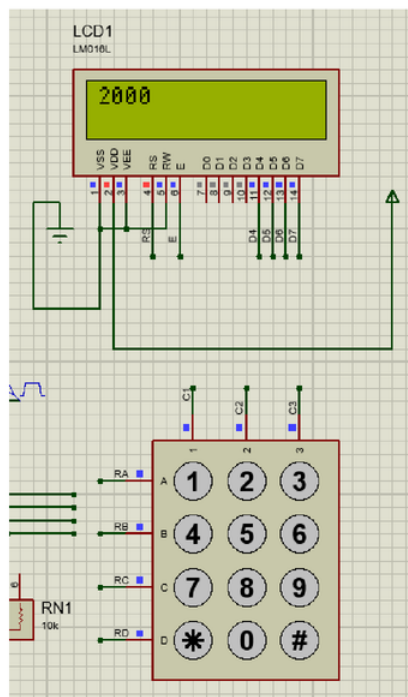
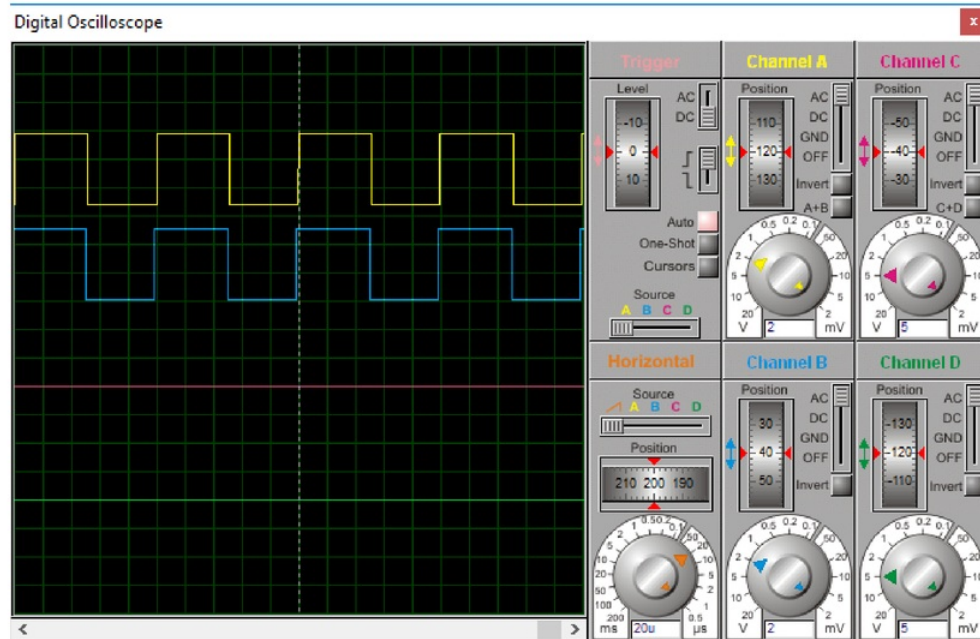


Figure 37 - Keypad Test (3)

Any value above this becomes set to a limit of 2000 as default (maximum pulses in Hz for the motor).

This was a 2000 limit before the final implementation of the code was completed.

PWM test



This is showing the modulation from the code which is incrementing and then decrementing (0-100%) control. Which is 0-127 (8 bits) and back down to zero again. So 00000000 to 11111111

These are the tests carried out from the prototyped designs. The test are good and therefore this prototype is fit for implementation into final design.

Simulation Testing

The PWM wave had to be tested in Proteus to ensure that the program gave the correct output at a given set point. The following test was carried out on set point values 100, 200 and 350 (with 360 being max speed in proteus):

$$\text{RPM} = \text{Frequency} \times (60 / \text{PPR})$$

$$\text{PPR (Pulses per revolution)} = 100$$

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

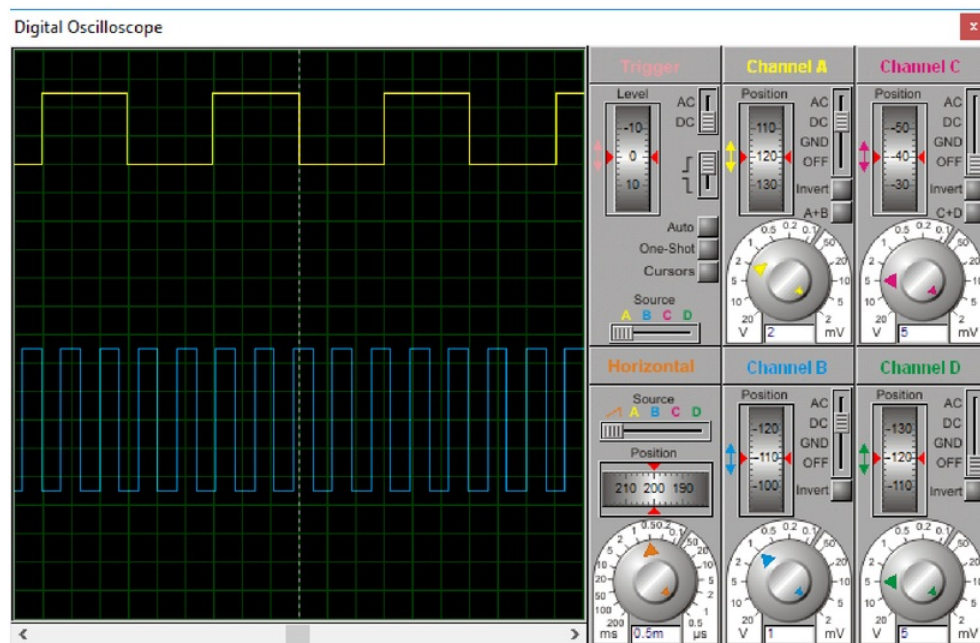


Figure 38 - PWM Digital Oscilloscope 200 Test

Yellow is the set point and blue is the rpm.

*PWM
prototype
tested good*

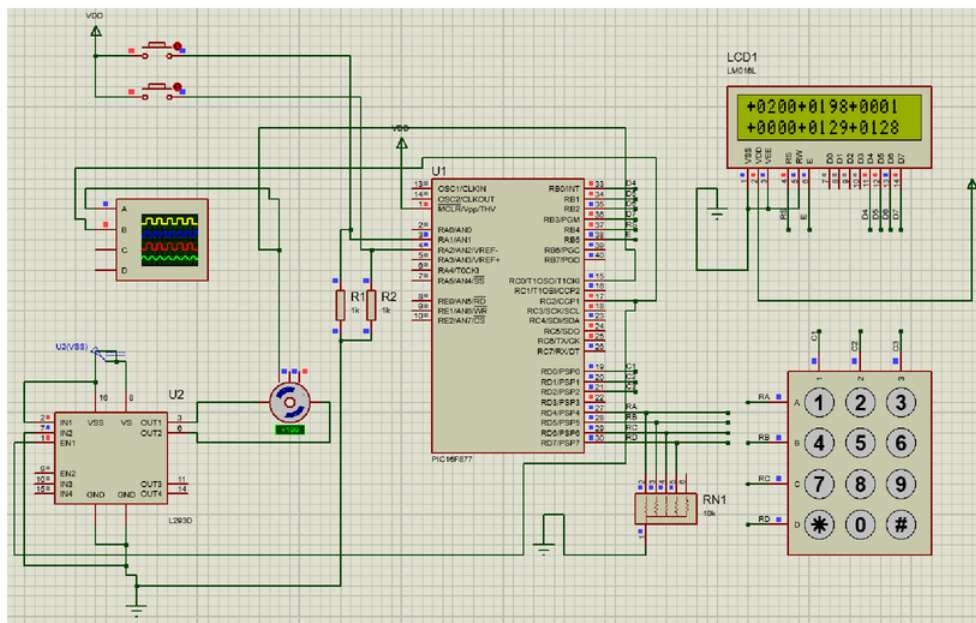


Figure 39 – PWM Proteus 200 Test

The motor shows 199 rpm with the set point as 200 and then it shows the feedback of 198 and the error of 1.

RA1 starts the motor and RA2 stops it as shown in figure 39.

*Resultant fully
functioning
system using
Proteus CAD.*

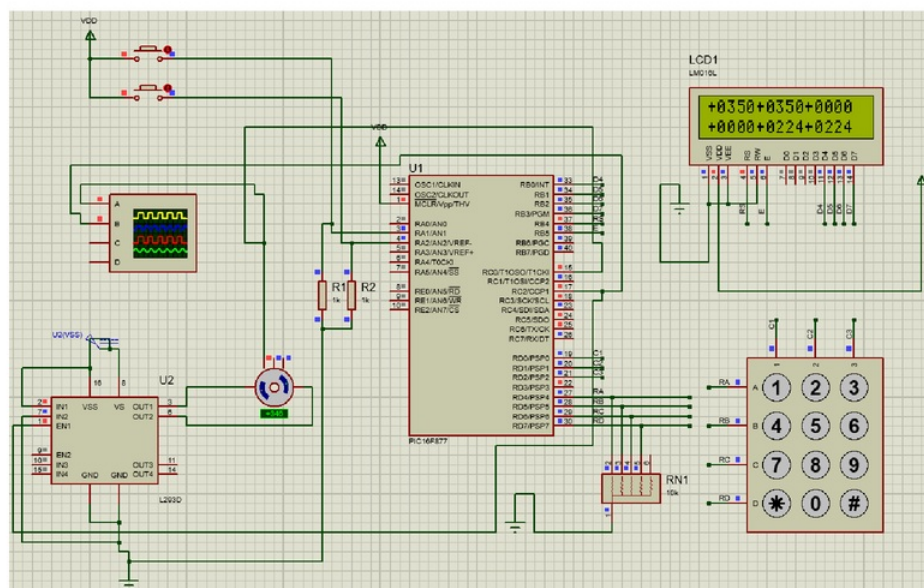
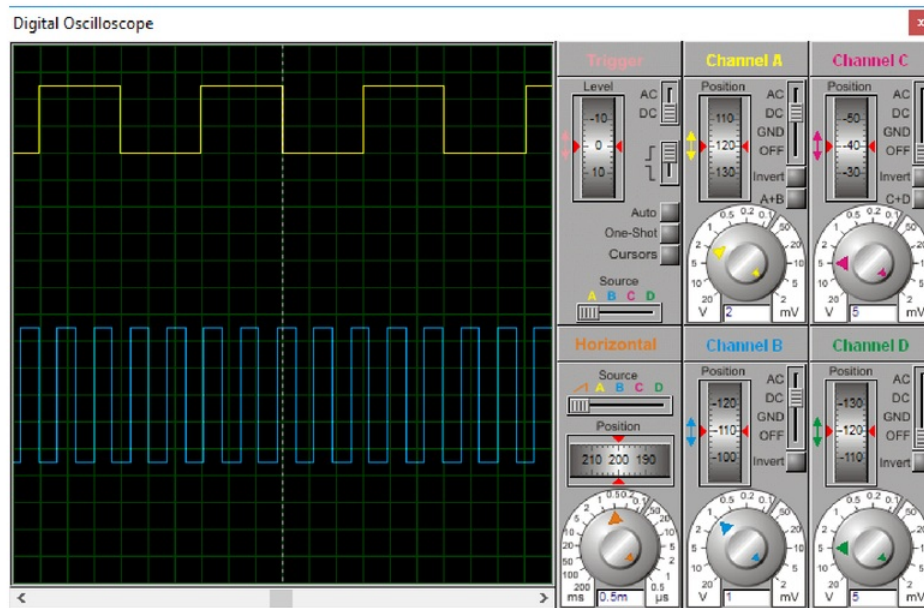


Figure 41 - PWM Proteus 350 Test

The motor shows 350 rpm with the set point as 350 and then it shows the feedback of 350 and the error of 0.

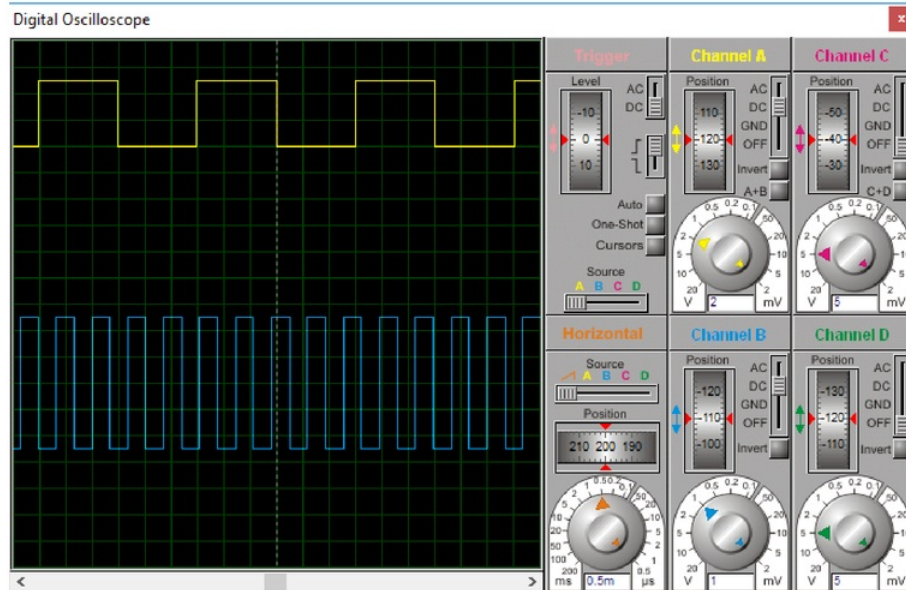


Figure 42 - PWM Digital Oscilloscope 100 Test

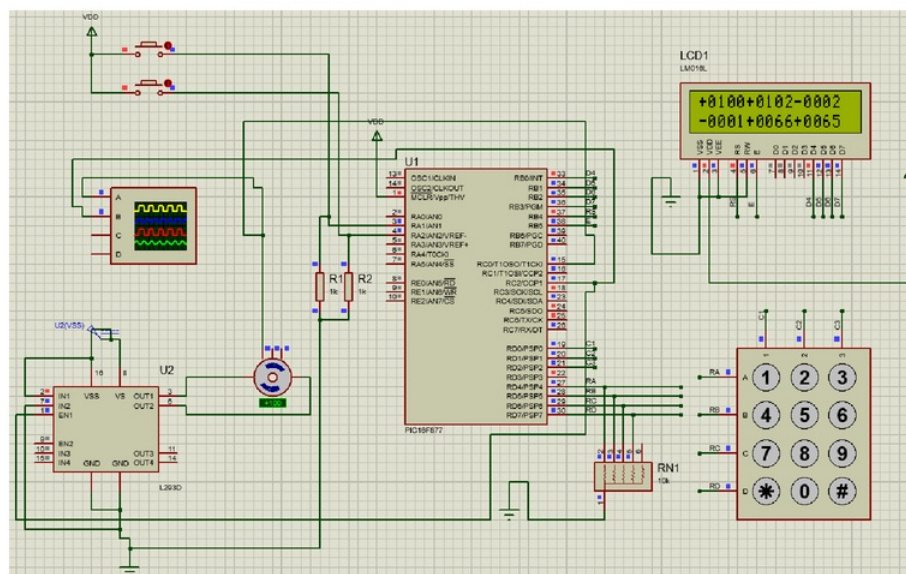


Figure 43 - PWM Proteus 100 Test

The motor shows 102 rpm with the set point as 100 and then it shows the feedback of 102 and the error of -2.

The simulation testing confirms that the system operates correctly. The practical build was then setup & program to allow further tests to be carried out.

The following tests were carried out that test the full functionality of the system.

Please observe the following testing table:

Test	Expected result	Actual result
Enter set point	System should speed up and keep motor at set point speed	Matched expected results
Lightly load motor after set point is entered	System should compensate and keep motor at set point speed	Matched expected results
Does the PWM adjust correctly to the set point	Evidence provided throughout	Evidence seen above
LCD display	Should display all parameters mentioned in the design section	Matched expected results
Press # key	Sets the set point to 0	Matched expected results

Figure 44 - Testing Table

Tests look good and well presented

Practical Testing

Setpoint for figure 45 is 200 RPM

The development board should be explained here with an image to show the configuration.

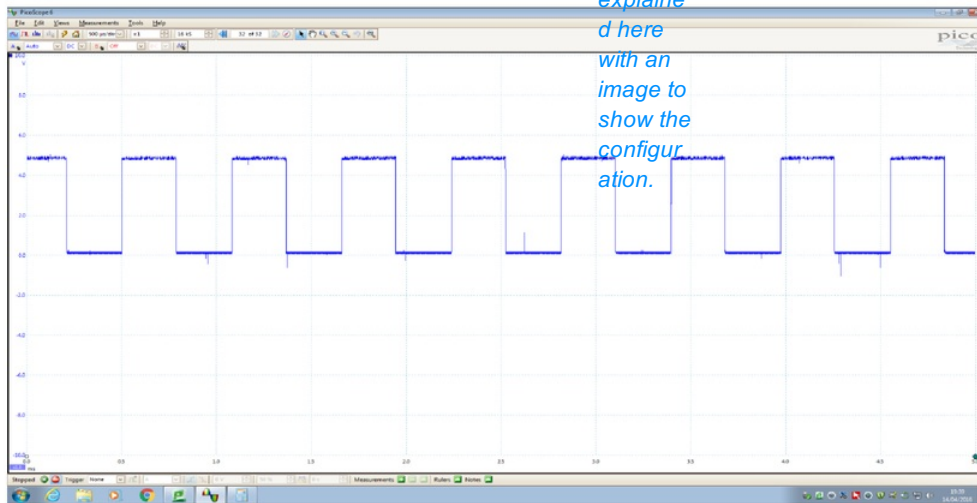


Figure 45 - PWM at 200 RPM

1.2 divisions at 0.5ms per div = 0.0006

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

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

PWM at 200 RPM

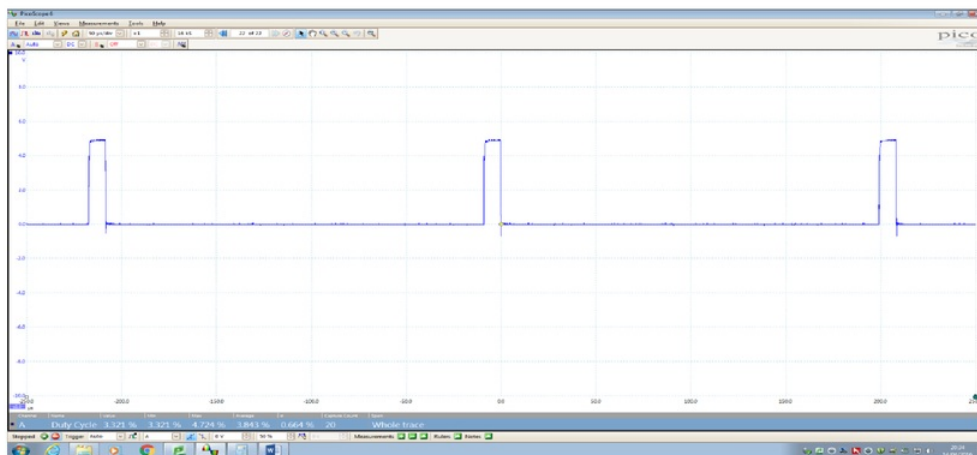


Figure 46 - Duty Cycle at 48

As seen in figure 46 Duty cycle = 48 (actual speed PWM) * 100 / 1023 (Duty at 100%)

Duty cycle = 4.7%

Set point for figure 47 is 1000 RPM

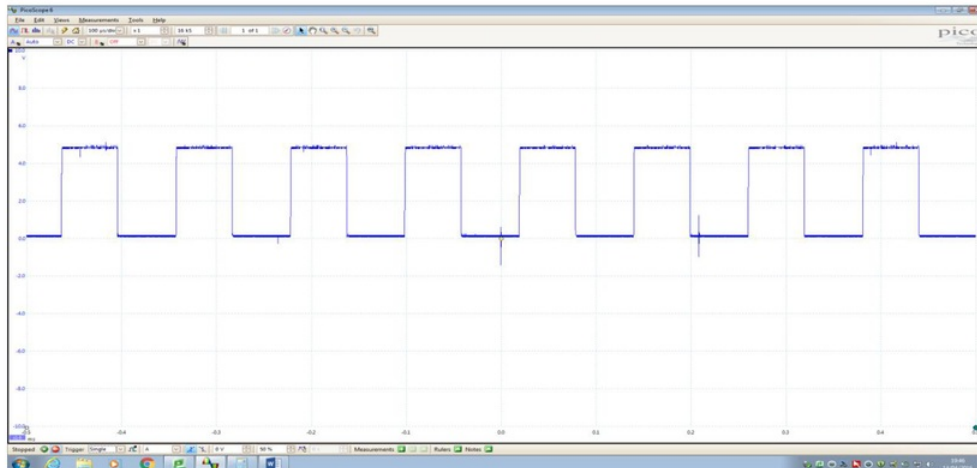


Figure 47 - PWM at 1000 RPM

1.2 divisions at 0.1ms per div = 0.00012

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

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

PWM at 1000 RPM



Figure 48 - Duty Cycle at 133

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

Duty cycle = 13.3%

Set point for figure 49 is 2000 RPM

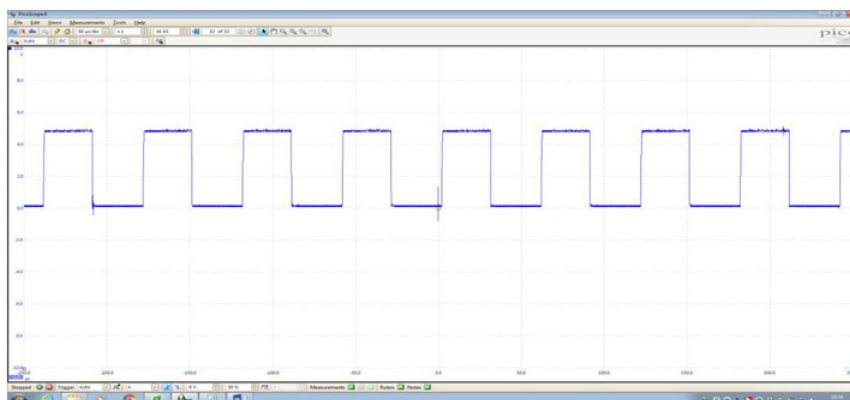


Figure 49 - PWM at 2000 RPM

$1.2 \text{ divs at } 0.05\text{ms per div} = 0.00006$

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

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

PWM at 2000 RPM

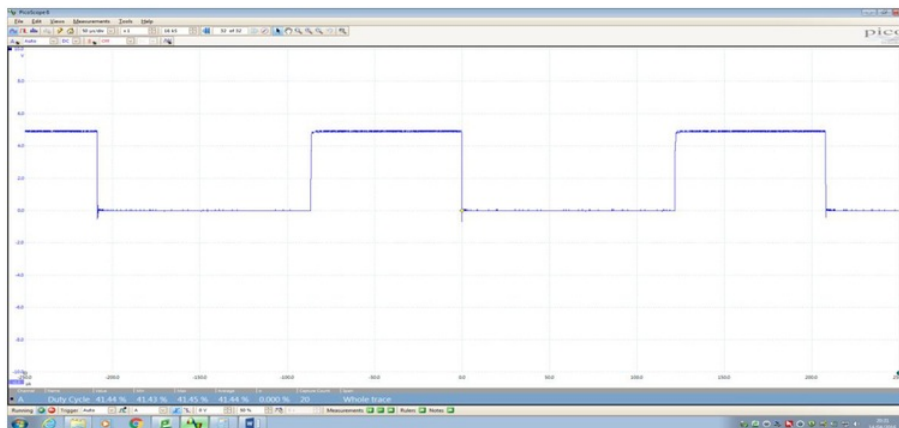


Figure 50 - Duty Cycle at 391

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

Duty cycle = 41.4%

Set point for figure 51 at 2300 RPM

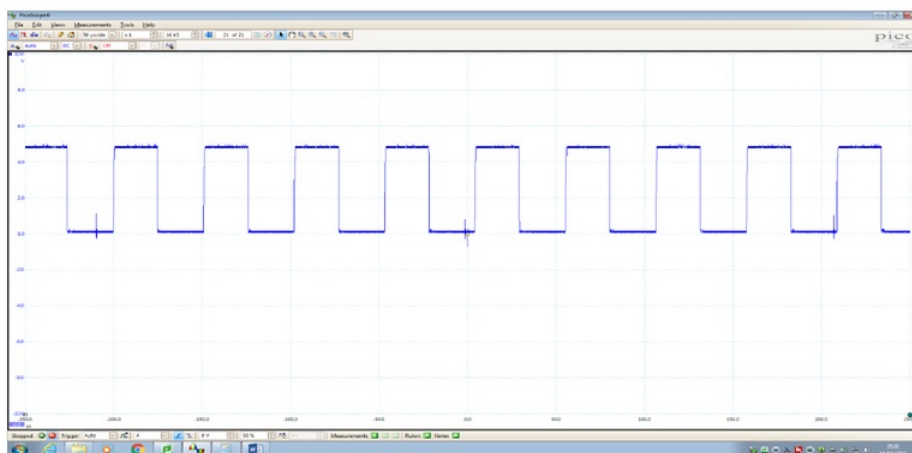


Figure 51 - PWM at 2300 RPM

1.0 div at 0.05ms per div = 0.000055

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

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

PWM at 2300 RPM

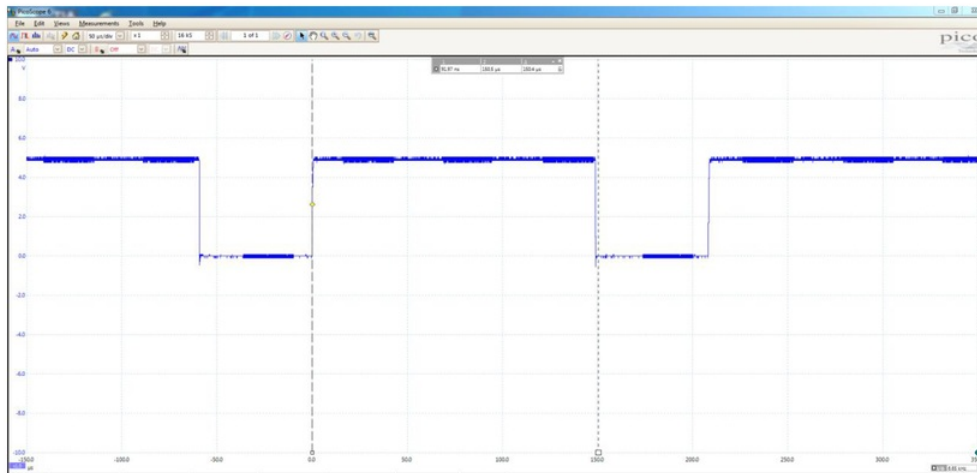


Figure 52 - Duty Cycle at 744

Duty cycle = 744 (actual speed PWM) * 100 / 1023 (Duty at 100%)

Duty cycle = 72.72%

Analysis of Testing

By undertaking simulation and practical testing it can be seen that the set point entered compared to the actual speed is very accurate and has been achieved by developing the code. This has been proved by comparing the results and using the testing table.

The tests look good but there is nothing to say what system they represent. There should be an image and description of the setup for testing the dev board system.

6. Conclusion

To be able to conclude the aims and the objectives of the assignment must be referenced to see if they have been met and achieved.

- The aims and objectives of this assignment were to use a microcontroller to interface with the following: DC motor, incremental encoder, analogue sensing circuit, keypad, LCD display and two pushbuttons.

This has been achieved through developing the code and using the physical microcontroller which was interfaced using the knowledge from the development part of the assignment.

- Using the embedded system from the on-chip peripherals, a system must be designed and implemented to provide PWM speed control for the motor with the information being fed back.

This has been achieved by developing the code for the program and evidence of this provided by the simulation and practical results of the assignment.

- A speed set-point must be entered using a keypad and speed sensing feedback must be applied via the encoder and fed back using an analogue value of armature current.

This has been proved to be successful by the simulation and practical results of the assignment by implementing the code described.

- The LCD display must show the: speed set-point, actual speed and armature current.

This is shown on the simulation of the test, whilst evidence was not gained for the practical, this was noted while testing took place as evidence was recorded correctly and proved.

- The system must also be started and stopped using pushbuttons placed on the system.

This was proven to work correctly.

In conclusion the assignment was a success as all the aims and objectives were met successfully.

*The system was tested
using PROTEUS CAD and
the dev board. This should
be evaluated in this section.*

7. References

Works Cited

- AVAGO technologies. (2012, 3 22). *Data sheet*. Retrieved 4 2, 2017, from <http://www.farnell.com/datasheets/1728690.pdf>
- <https://www.electronics-tutorials.ws/power/thyristor.html>. (2018, April). *httpower/thyristor.html*. Retrieved from <https://www.electronics-tutorials.ws:https://www.electronics-tutorials.ws/power/thyristor.html>
- Lab, E. (n.d.). *blog/wp-content/uploads/2010/11/OPTION_REG.png*. Retrieved 1 2018, from <http://embedded-lab.com>: http://embedded-lab.com/blog/wp-content/uploads/2010/11/OPTION_REG.png
- Matrix. (Unspecified). *Eblocks keypad board datasheet*. Retrieved 4 4, 2017, from <http://www.matrixsl.com/resources/files/datasheets/EB014-30-1.pdf>
- Matrix. (Unspecified). *PICmicro MCU development datasheet*. Retrieved 4 5, 2017, from <http://www.matrixsl.com/resources/files/datasheets/HP488-30-3.pdf>
- Microchip. (2003). *PIC16F87XA Datasheet*. Retrieved 4 1, 2017, from <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>
- Romux. (2010). *Interrupts*. Retrieved 1 31, 2017, from <http://www.romux.com/tutorials/pic-tutorial/interrupts>
- Woodford, C. (2012). *how-thyristors-work.html*. Retrieved from <http://www.explainthatstuff.com: http://www.explainthatstuff.com/how-thyristors-work.html>

8. Appendix

Thyristors are multi-layered semi-conductor devices and are use current flow to control the voltage.

(<https://www.electronics-tutorials.ws/power/thyristor.html>, 2018) States that “a thyristor is a four layer (P-N-P-N) semiconductor device that contains three PN junctions in series.” By using thyristors in a circuit they can be used to control phase, AC motors and lamps. This is because the thyristor consists of two P-N junctions back to back containing an anode, cathode and gate. The P-N junctions can be switched on and off at an extremely fast rate which is why the power to the load can be controlled. Figure 1 shows how the thyristor is made up.

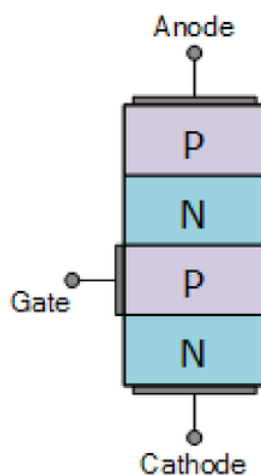


Figure 53 - Thyristor

(<https://www.electronics-tutorials.ws/power/thyristor.html>, 2018)

The gate detects small changes in current which allows it to switch on and off and therefore allows larger currents to flow from the anode to cathode, therefore controlling the flow. This will then keep flowing even if the smaller current at the gate is lost as it “latches” the signal as such to allow the larger current to continue to flow from anode to cathode. (Woodford, 2012)

Assignment 2 Joshua Beer

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

62/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

Text Comment. The current feedback was done in the prototype stage and left out of the final design due to space on the LCD.

Text Comment. This diagram would be better in the hardware section.

Text Comment. This does give some idea of what the task is about. This should however point out the development process using prototyping.

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

Text Comment. Hardware has been explained fairly well.

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

Text Comment. Encoder outputs to RC0 input of MCU.

PAGE 18

Text Comment. Timer 1 is set to event mode with RC0 and overflow every complete ISR count which is 100ms in this case.

PAGE 19

PAGE 20

PAGE 21

Text Comment. Good explanation of why it is 10 bit PWM.

Text Comment. RC0 is the input for encoder feedback.

Text Comment. Inputs for rows and outputs for columns.

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

Text Comment. Configuration is covered fairly well and must be correct because the demo works for the CAD and dev board systems.

PAGE 30

Text Comment. The current monitoring was done only in the prototype stage and was not included due to space on the LCD. If the current monitoring is to be mentioned then the design must also be included.

Text Comment. Good points made as the CAD and dev board systems were slightly different.

Text Comment. The input encoder and PWM rely on the ISR .

Text Comment. Good explanation

Text Comment. Good explanation

Text Comment. Good explanation.

Text Comment. These are the tests carried out from the prototyped designs. The test are good and therefore this prototype is fit for implementation into final design.

Text Comment. PWM prototype tested good

Text Comment. Resultant fully functioning system using Proteus CAD.

PAGE 49

PAGE 50

PAGE 51

Text Comment. Tests look good and well presented.

PAGE 52

Text Comment. The development board should be explained here with an image to show the configuration.

PAGE 53

PAGE 54

PAGE 55

PAGE 56

Text Comment. The tests look good but there is nothing to say what system they represent. There should be an image and description of the setup for testing the dev board system.

PAGE 57

PAGE 58

Text Comment. The system was tested using PROTEUS CAD and the dev board. This should be evaluated in this section.

PAGE 59

PAGE 60

GRADING FORM: CSG-RUBIC

JOSHUA BEER

0

RESEARCH-INFORMED LITERATURE

 n/a

KNOWLEDGE AND UNDERSTANDING

 The introduction has some good points about tasks. The development method using prototyping should be included. 2.5/5 Hardware description is good and covers what is needed. 3/5 The design of the system is explained fairly well throughout the report. The dev board should have been explained with diagrams especially for test referencing. The configuration is explained well. 13/20 The software has been commented and explanations are accurate and make sense. 6.5/10

25/40

ANALYSIS

 Test and analysis is good for both CAD and dev board. The dev board setup should have been included as a reference for the tests. The test explanations are good. 7/10 Conclusion has a point but should include the dev board evaluation. 5/10

12/20

PRACTICAL APPLICATION AND DEP.

 The software design and explanation is good. The dev board setup is missing but all the relevant tests are there. 5.5/10 Quality of work is good overall. 6.5/10

12/20

SKILLS FOR PROFESSIONAL PRAC.

 The report is good apart from the dev board layout which is missing. 6.5/10 Team working is good. 6.5

13/20

