

PSDE 5013 Assn 2 Nathan Smith

by Nathan Smith

FILE	15460_NATHAN_SMITH_PSDE_5013_ASSN_2_NATHAN_SMITH_145116_614609388.PDF		
TIME SUBMITTED	26-MAY-2018 02:56PM (UTC+0100)	WORD COUNT	5421
SUBMISSION ID	89961353	CHARACTER COUNT	29193



HND ELECTRICAL & ELECTRONIC ENGINEERING

PSDE 5013 CONFIGURATION & PROGRAMMING OF EMBEDDED SYSTEMS

ASSIGNMENT 2 DC MOTOR CONTROLLER

Nathan Smith Word Count = 5338

Contents

1. Introduction.....	3
2. System Hardware	5
3. Microcontroller Configuration.....	12
4. Software Design and Coding	22
5. Testing and Analysis of results	30
6. Conclusions and Discussions	39
7. References and Appendices.....	40
Works Cited.....	40
Figure 1: Block Diagram Incremental Encoder	6
Figure 2: HEDS-5X40 incremental encoder pull up resistors.....	6
Figure 3: PIC16F877 data sheet diagram (MICROCHIP, 2013)	7
Figure 4: Microcontroller overview (MICROCHIP, 2013)	8
Figure 5: Start Stop logic	8
Figure 6: keypad used in the assignment	9
Figure 7: HP888 Development board (MICROCHIP, 2013).....	10
Figure 8: Data sheet diagram for the 24vdc motor	11
Figure 9: MOSFET with motor	11
Figure 10: ADCON1 Register	12
Figure 11: ADCON1.....	12
Figure 12: ADCON0 Table.....	13
Figure 13: ADCON0 Register	13
Figure 14: Timer 0 block diagram	14
Figure 15: Timer 0 settings.....	14
Figure 16: Option Register data (R-B, 2010)	15
Figure 17: Option_Reg table.....	15
Figure 18: INTCON table	16
Figure 19: INTCON Register	16
Figure 20: Timer 1 block diagram (MICROCHIP, 2013)	17

Figure 21: T1CON configuration	17
Figure 22: PIE1	18
Figure 23: T1CON	18
Figure 24: PIR1 overflow	18
Figure 25: Block diagram Timer 2.....	19
Figure 26: TIMER 2 architecture.....	19
Figure 27: PWM.....	20
Figure 28: PIE2.....	20
Figure 29: pir2	20
Figure 30: First section code	22
Figure 31: Second section code	23
Figure 32: Third section code	24
Figure 33: 4th code section	25
Figure 34: Fifth section of code	27
Figure 35: 6th code section	28
Figure 36: Last code section.....	29
Figure 37: Setting clock to 100hz	30
Figure 38: 100HZ clock setup.....	30
Figure 39: 1000hz clock setup.....	31
Figure 40: 1000hz clock display	31
Figure 41: Testing LCD.....	32
Figure 42: 100rpm	33
Figure 43: 300RPM.....	33
Figure 44: 360rpm	34

1. Introduction

This assignment will look at interfacing a microcontroller with a DC motor, incremental encoder, analogue sensing circuit, keypad, LCD display and two pushbuttons. A design will be created and code written to provide a practical set of results to be tested. The on chip peripherals will be utilised to provide PWM speed control on the motor with feedback. The speed set points will be set via the keypad and the feedback provided from the encoder with correct sensing coming in the form of an analogue voltage which is proportional to the armature current. The LCD will show the speed set point, speed and armature current and the motor must be stopped and started from the pushbuttons. The diagram below shows the block diagram.

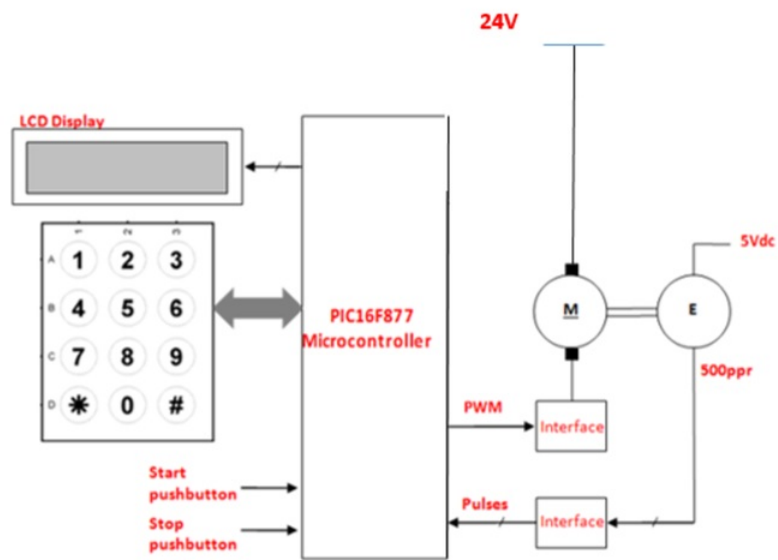
The first part of the assignment will look at research into the system hardware of the microcontroller and will include research on the individual pieces of hardware state in the design brief. The research will look at a written explanation and the use of images and data sheets to explain the features.

The next part will be the Microcontroller configuration and this will be a detailed explanation and look at different registers used with their set up detailed. The appendix will provide further reading for this section.

The next part will be the software design and the testing of the system, these parts will give explanations on the code for the system along with simulated designs build and explained. The testing will test that the features designed work together to produce the output as per the brief.

The assignment will be concluded and any recommendations will be included at the end of the assignment.

The introduction should mention the design and test process using prototypes for the stages of the design. These were the current measurement was in the prototype being the keypad and left out due to encoder, optical space on the LCD. Encoder feedback and PWM driver.



This diagram should be the hardware description.

2. System Hardware

In order to achieve the system hardware which has been outlined in the assignment brief a selection of hardware is to be chosen to operate alongside the software programs and produce the outcomes and systems for the assignment. The assignment was required as stated above to run a DC motor, this would require a control system to be operated which is the first part of this section.

The control system to be used will be a closed loop control system with feedback. These control systems are made up of three main areas to complete a control loop, the first section is the control section. The control section included the controller which in the case of this system would be an PI controller, this will receive feedback from the loop and control the output depending on the desired values. The second part is the plant section, this section is the correction element and the process element which is the actual mechanical process that is being controlled or regulated. The third section is the instrumentation section and consists of the sensing element and the signal conditioning which is the physical measuring device and the transfer of the feedback data to the control system.

This assignment requirements uses an incremental encoder device which is used as a controller system and the desired output was required to be at 500 pulse per revolutions. An encoder is a mechanical/ electrical device which will convert a rotary or linear movement into a signal which can either be digital or a pulse form. The incremental encoder will generate a pulse for every step which is incremented in the rotation, this encoder has a high-resolution output making it a good choice for the system. (PC Control Learning Zone, 2008).

Figure 1 below shows the block diagram of the incremental encoder which has been selected to be used for the assignment, the encoder is a 'HEWLETT PACKARD' encoder and the HEDS-5x40 encoder was selected.

*This
could
have
been
clearer.*

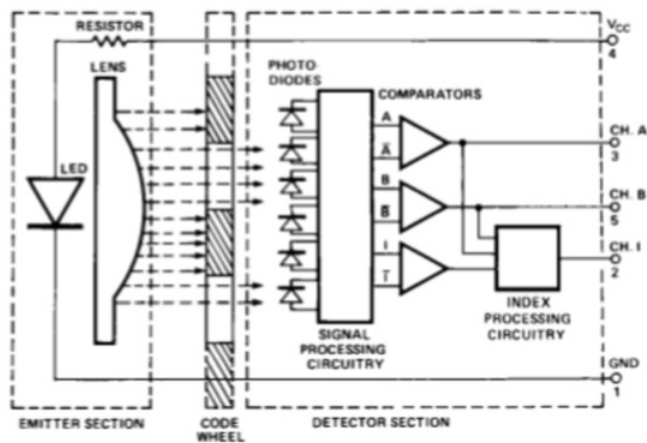


Figure 1: Block Diagram Incremental Encoder

Figure 2 below also shows the pull up resistors on the HEDS-5X40 encoder for pins 2,3 and 5 along with the other pin labels.

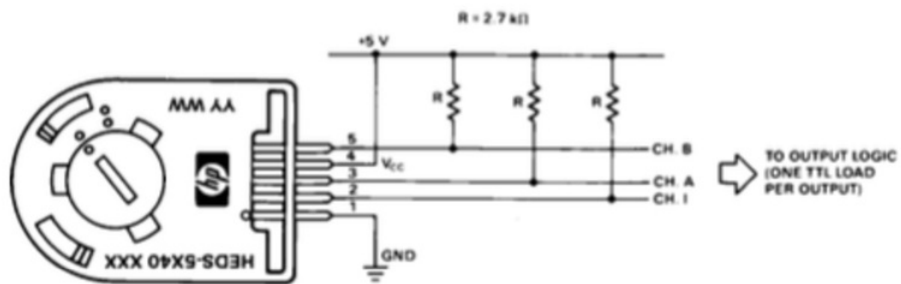


Figure 2: HEDS-5X40 incremental encoder pull up resistors.

The next price of system hardware is the microcontroller which is to be interfaced to the DC motor. The microcontroller which was discussed and selected was the 'PIC16F877' microcontroller as it has the ability to provide a wide range of input and output capabilities as well as being able to work with the PCM capabilities and provide support for the rest of the assignment criteria. Figure 3 below shows the data sheet snapshot of the microcontroller stating all the pins number and descriptions.

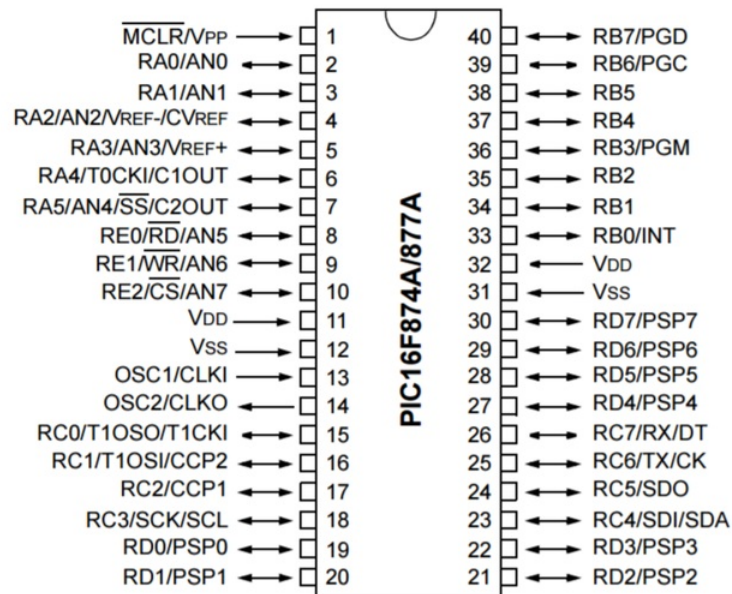


Figure 3: PIC16F877 data sheet diagram (MICROCHIP, 2013)

The microcontroller above is figure 3 is a 40 pin PDIP controller and figure 4 below shows the features and design of the microcontroller.

The next piece of hardware is the keypad, the keypad is used to act as an input to the system, the keypad will come into the instrumentation control loop as explained above as a control part of the loop. The user can select the described value on the 12-button keypad and the system can then respond to this output to produce an output. The design of the program shown below in the report gave the microcontroller the correct interface to work with a keypad.

The used for the assignment was discussed and chosen to be a 3 x 4 matrix keypad which gives the function of using a number of different buttons connecting to a single pin on the controller, the function will be programmed to the controller to identify which button selection has been pressed at each pin. Figure 6 below shows the keypad used in the assignment build and is a 3x4 keypad as described above.

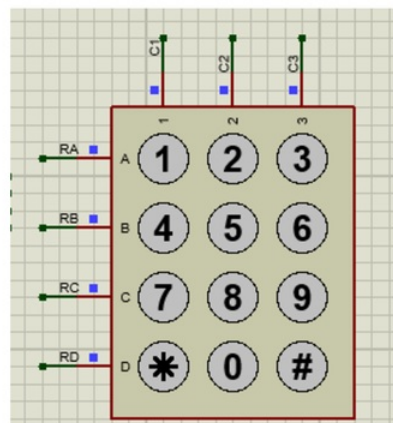


Figure 6: keypad used in the assignment

The part is the development board used for the assignment, the board used was the HP488 board which has been used in previous subjects. The board is a multifunctioning board and has the correct inputs and outputs as well as the correct hardware included to operate to the brief specifications. Figure 7 below shows the board taken from the manufacturers data sheet.

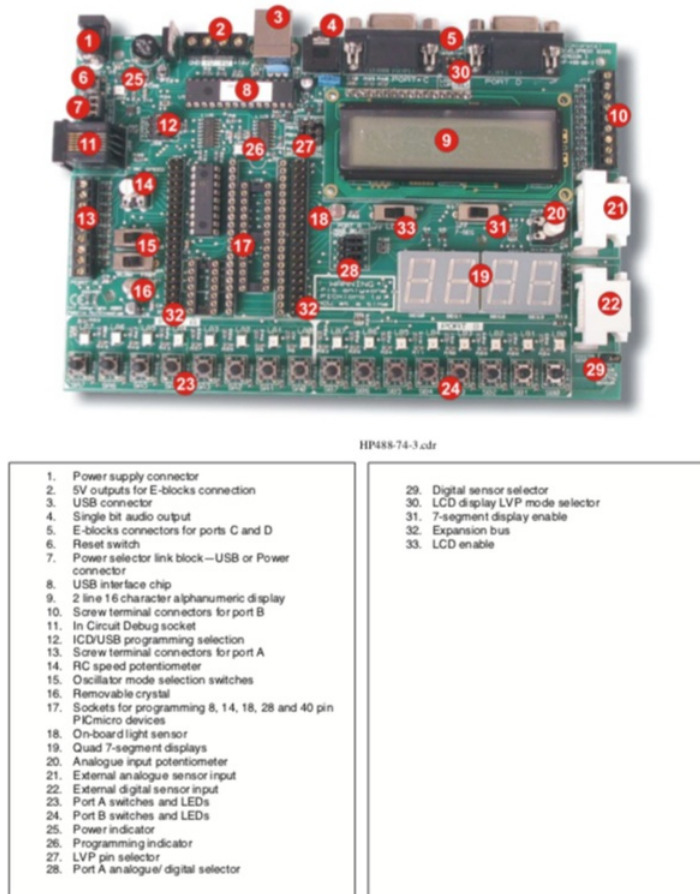


Figure 7: HP888 Development board (MICROCHIP, 2013)

Figure 7 above includes all the relevant labels for each part of the development board. (matrixtsl, 2005). The data sheet can be found in the appendix for reference.

The DC motor is the next part of the hardware, the dc motor that was discussed and selected for the assignment was a 24V direct current motor. The motor is taken from RS components and utilised a high performing low inertia servo motor which has a integral dual track encoder which as above will provide some control to the system. The output from the microcontroller is set at 5 volts so a junction is to be used in the system to provide isolation to the separate voltages and provide a safe interface between each device.

Figure 8 below shows the data sheet and diagram for the DC motor used for this assignment.

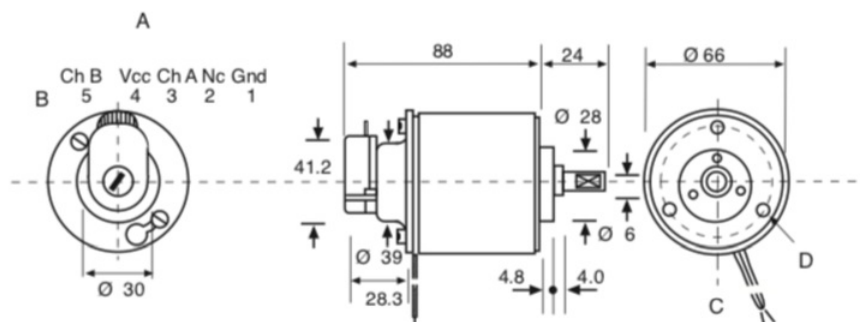


Figure 8: Data sheet diagram for the 24vdc motor

The last part of hardware in the system is the MOSFET transistor. This transistor is required in the system to provide a switch in the motor when the required PWM is met, this will switch with the microcontroller to allow access at the mosfet's gate. When this is set up the microcontroller can then provide a PWM wave as an output to the transistors. This separation of the 24v and 5v can be done then via a switching from the transistors to send 24v to the motor, this is to keep the system running safe and correct. Figure 9 below shows the set-up of a MOSFET with a motor.

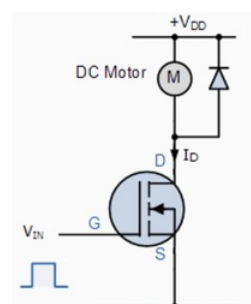


Figure 9: MOSFET with motor

3. Microcontroller Configuration

To achieve the requirements that the report has set out for the microcontroller and control circuit to achieve, a variety of different elements have to be set up and configured which will be set out and detailed below.

The first configuration is the ADCON1 register configuration, this is the register that will define which PORTA pins are used for digital input and output, along with what is used by the A/D module.

The **ADCON1** register had a configuration set to = 10001110. The specification require the use of two digital inputs to be used for the start and stop features. The table below in figure 10 shows the configuration for the register ADCON1 which was required to meet the specification required by the brief and to operate the system.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
1	0	0	0	1	1	1	0

Figure 10: ADCON1 Register

Figure 11 below shows the ADCON1 register with the settings for bit 6-4 being set to 0 and the setting of bit 7 set at 1 for the right hand justified. The PCFG3-PCFG0 have their settings set at 1110 with the setting for the PCFG are set at 1110 making the RA1-RA5 set to a digital form to enable the correct settings to be produced which will meet the specifications for the system.

	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	A	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

Figure 11: ADCON1

The next register configuration is for the **ADCON0**, the table below in figure 12 shows the settings in which the register has been set to. The diagram of the configuration settings in figure 13 shows that the ADCON0 pins for 6-7 are set at a value of 10 for the FOSC/32, bits 3-5 are set to a value of 000 because ADCON1 requires the register to set the RA0 channel. Bit 2 has a setting of 1 because the A/D convertor will need to be active for the system to operate. Bit 1 is then set to 1 due to the module convertor being active and operational. All these values are saved and stored to the ADRESL and ADRESH locations at a 10 bit result.

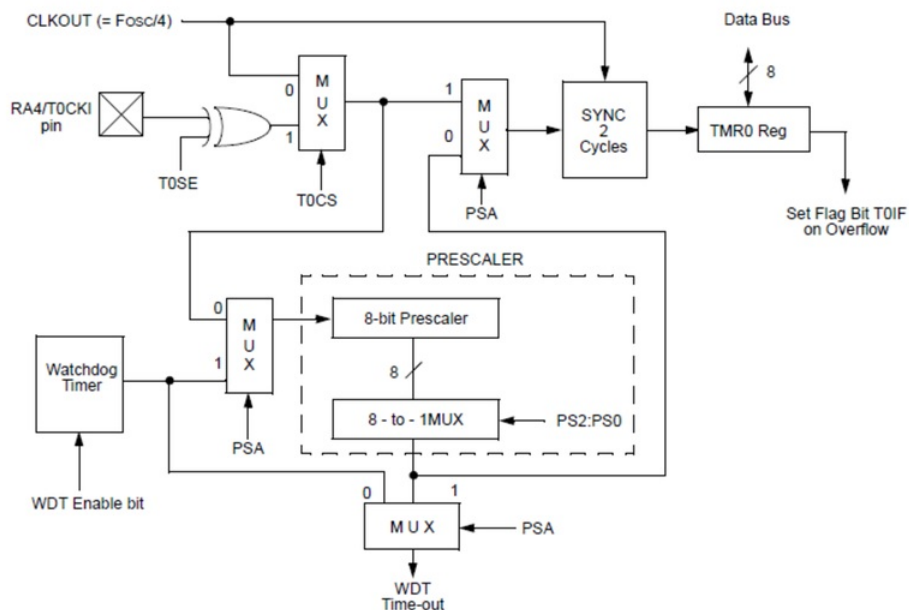
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON
1	0	0	0	0	0	0	1

Figure 12: ADCON0 Table

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7								bit 0
bit 7-6	ADCS1:ADCS0: A/D Conversion Clock Select bits 00 = Fosc/2 01 = Fosc/8 10 = Fosc/32 11 = Frc (clock derived from the internal A/D module RC oscillator)							
bit 5-3	CHS2:CHS0: Analog Channel Select bits 000 = channel 0, (RA0/AN0) 001 = channel 1, (RA1/AN1) 010 = channel 2, (RA2/AN2) 011 = channel 3, (RA3/AN3) 100 = channel 4, (RA5/AN4) 101 = channel 5, (RE0/AN5) ⁽¹⁾ 110 = channel 6, (RE1/AN6) ⁽¹⁾ 111 = channel 7, (RE2/AN7) ⁽¹⁾							
bit 2	GO/DONE: A/D Conversion Status bit If ADON = 1: 1 = A/D conversion in progress (setting this bit starts the A/D conversion) 0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)							
bit 1	Unimplemented: Read as '0'							
bit 0	ADON: A/D On bit 1 = A/D converter module is operating 0 = A/D converter module is shut-off and consumes no operating current							

Figure 13: ADCON0 Register

The **Timer 0** register configuration is the next register to be explained, this is shown in a block diagram form below in figure 14.



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

Figure 14: Timer 0 block diagram

To enable the timer0 to be used in the system, the OPTION_REG needs to be set up and configured along with the TOIF having its overflow calculated. These register configurations are detailed below in this report section. Figure 15 below shows some details of the setup of the Timer 0 register.

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.

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

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

MAX PRESCALER = 256

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

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

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

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

Figure 15: Timer 0 settings

The next configuration is the **OPTION_REGISTER**, the table below in figure 17 sets out the settings for the Option register. An Option register is a writable and readable register which will contain a variety of control bits used to configure other devices. (Pictutorials , 2017).

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
1	0	0	0	0	1	1	1

Figure 17: Option_Reg table

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	R/W-1
	RBP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7								bit 0
bit 7	RBP							
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 16: Option Register data (R-B, 2010)

The next register to be set up is the **INTCON_REGISTER**, this is the register which will contain the status for the high importance interrupts. Figure 19 below shows the INTCON register data. This register is required to enable the interrupts to the CPU, the table below is figure 18 shows the table for the configuration of the INTCON register.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
1	1	1	0	0	0	0	0

Figure 18: INTCON table

The global Interrupt enable bit is then set to 1 in this register which will allow the unmasked interrupts to be enabled or activated. The PEIE bit is set to the value of 1 and will then activate the unmasked interrupts peripherals. The TMR0 overflow bit enable has a setting of 1 which is to allow the TMR0 interrupt.

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-0	R/W-x
GIE	PEIE	TOIE	INTE	RBIE	TOIF	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 **TOIE:** 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 **TOIF:** 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 19: INTCON Register

The timer 1 is the next register to be set up, this register is the second available timer on the chip. The timer 1 is made up of a 16 bit ($2^{16} = 65535$) counter and will consist of two 8 bit registers which are the tmr1l and the tmr1h registers. The inputs for the encoder feedback in the timer 1 is set at RC0, and the no load speed maximum value is 2300 RPM. Figure 20 below shows the block diagram for the timer 1. (Microchip, 2003)

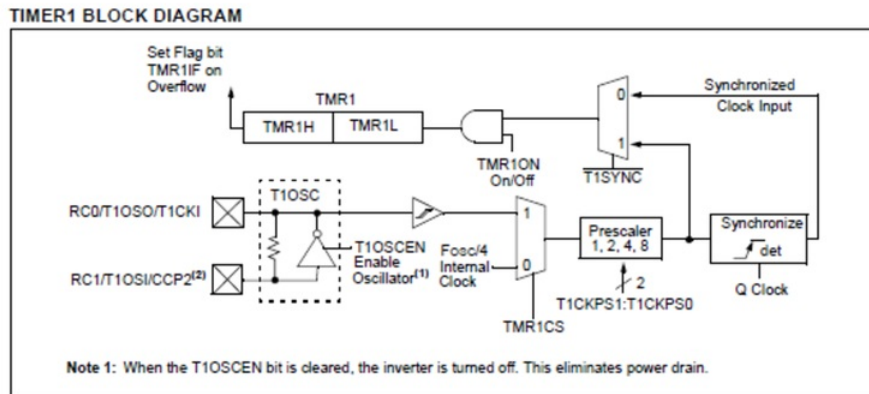


Figure 20: Timer 1 block diagram (MICROCHIP, 2013)

The device will not operate as required unless the next register is set up correctly, thus register is the T1CON, this has to be set to the value of 0B00000111 in binary form. The table in figure 21 below shows the configuration of the T1CON register.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
0	0	0	0	0	1	1	1

Figure 21: T1CON configuration

The diagram in figure 22 below shows the settings of each bit and are as follows, bit 7-6 set at 0, bit 5-4 set at 00 due to the prescaler value not being needed and 1 pulse is = to 1 rpm. Bit 3 is set to 0 due shut off of the oscillator and bit 2 is set to 1. Bit 1 is also set at 1 and bit 0 is set to 1 to enable the timer 1.

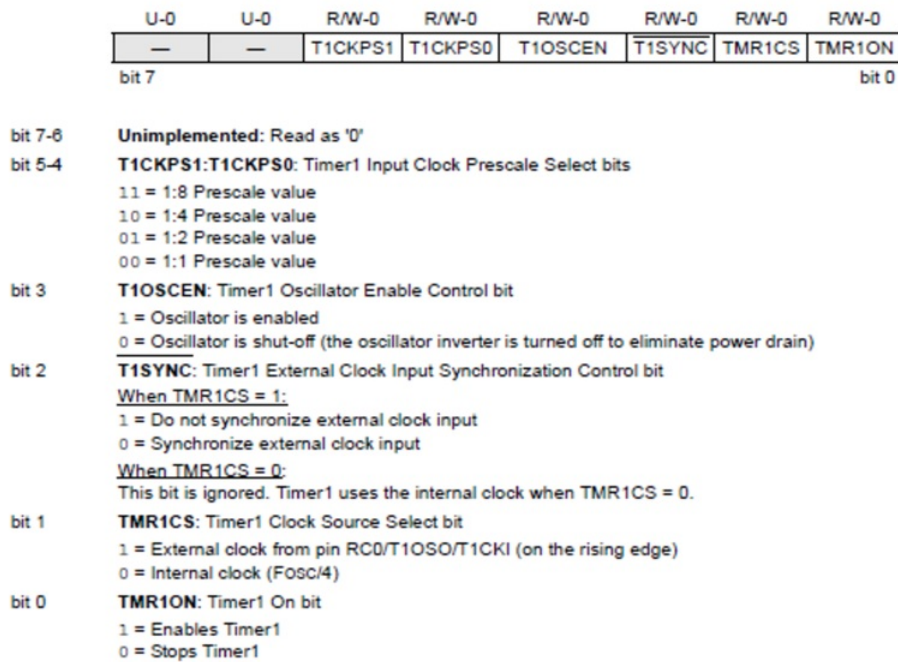


Figure 23: T1CON

The system will also need the PIE1 to be configured which will configure different bits which are required to get enabled. This configuration that is set up is shown in figure 23 below.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
0	0	0	0	0	0	0	1

Figure 22: PIE1

As seen all are set to the value of 0 with the exception of the TMR1IE which has a setting of 1 as it's needed to enable the overflow on TMR1. To flag the overflow from timer 1 then PIR1 register can be configured and set at zero as shown below in figure 23. The overflow code for the program can then become – 0B00000001 and PIR1 – 0B00000000.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
0	0	0	0	0	0	0	0

Figure 24: PIR1 overflow

The next register is the **TIMER 2**, this register has the main usage to interface the PWM waves with a direct current motor. The timer 2 includes a 8bit period register with a maximum output count of 256, there are different registers from timer 1 and 0 which are set up to provide control to the Timer 2, the diagram below in figure 25 shows the **TIMER 2** register internal structure.

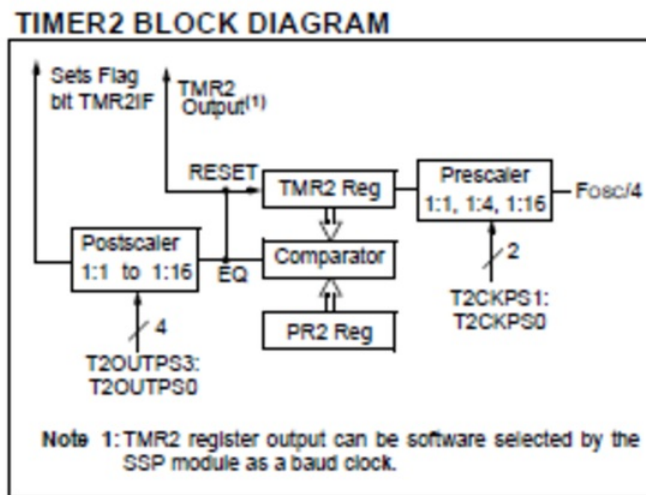


Figure 25: Block diagram Timer 2

The setting for the **TIMER 2** is set at `0B00000101` and a setting of `0B00001100` is required to enable PWM on the **CCP1CON** register. The **TIMER 1** architecture is shown below in figure 26.

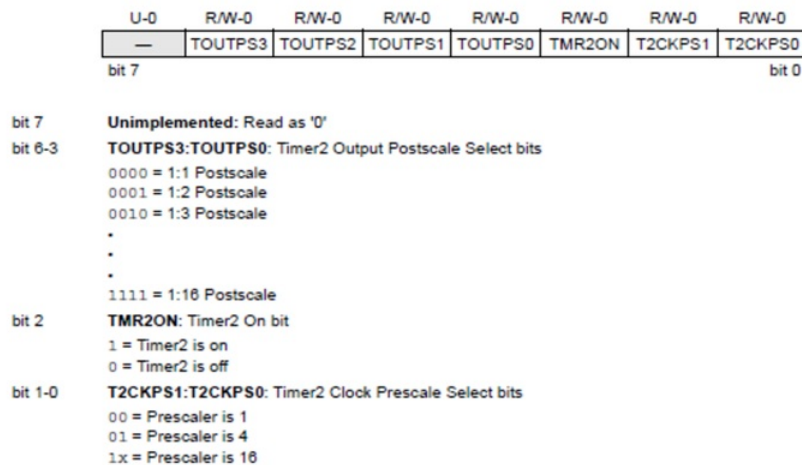


Figure 26: **TIMER 2** architecture

The last configuration to be talked about is the [PWM](#), this can be seen in the block diagram form below in figure 27. The CCPR1L is set up to be a 8 bit register in normal operating modes, but does have to ability to be set as a 10 bit register if it is required by CCP1CON <5:4>. Comparing PR2 TO TMR2 and onto CCPR1L provides a duty cycle to RC2, this is how the period and duty cycle are calculated.

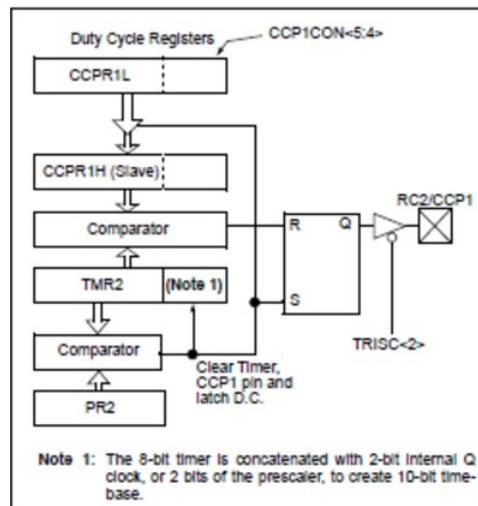


Figure 27: PWM

The registers PIR2 and PIE2 are set to 0B00000000, and are shown in figures 28 and 29 below.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	Reserved	-	EEIE	BCLIE	-	-	CCP2IE
0	0	0	0	0	0	0	0

Figure 28: PIE2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	Reserved	-	EEIF	BCLIF	-	-	CCP2IF
0	0	0	0	0	0	0	0

Figure 29: pir2

The configuration is correct. Some of the explanations are a little vague.

The PORTC has a configuration which is set to one output which is RC2 and set to 0, these provide PWM outputs with the remanding being set as inputs. To receive the count the RCO is set to an input. PORTD is the allocation for the keypad with a setting for RD0-RD3 and all outputs set to all inputs.

PORTC (TRISC) = 0b11111011

PORTD (TRISD) = 0b11110000

*T
h
i
s
i
s
c
o
r
r
e
c
t
.*

4. Software Design and Coding

To enable the system to work as required in the brief, a c programme or code was required to be written to enable the system to operate. The code was written and built up over time by understanding and building the individual section of the program and adding them together to create the final program.

The first section of the code can be seen below in figure 30. The first line outlines the 'include pic.h' statement which is required to be added when using a pic chip for the programme. The next line is the 'include lcd.h' which is adding the LCD files which have been pre-build in another location ready to be added into this programme, this include line makes these pre-build programmes now included in this code, this is done to enable the program to write to the LCD more efficiently and correctly. It is not convenient to include the whole pic files into this file so adding is done in most code work.

The next section of the code is where 4 functions are declared to the system, these are declared in standard c programming format and are the 'void DisplaySpeed, void DisplayValues, Int ScanKeypad and void Delay' functions. Next float type variables are declared for the high priority variables along with some other variables which are used in the program. Setpoints are chosen and inserted by the operator with error being introduced in-between the setpoint and the speed current, and the output from the system. The PI controller will store values using the prop_part = 0 to the ki = 0.02; line in the code. The proportional and integral become the Ki and Kp and re explained above in the reperch theory section to what they do. As stated above figure 30 below shows this section of code set for ideal values.

```
#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_accur = 0 ,Kp = 0.01, Ki = 0.02; //The
```

Figure 30: First section code

```

////////////////////////////////// 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;
    }
}
////////////////////////////////// main program ////////////////////////////////////

```

Figure 31: Second section code

TOIF, TMR1 and TMR2 should be capitals.

The next section of code which can be seen above in figure 31 is for the interrupt ISR routine, this section of code is built to count pulses in the system accurately. The code includes 3 main statements or sources, these are the toif, the tmr1if and the tmr2if. These three statements are representing the interrupts for the timer 0,1 and 2.

To indicate that the system is problematic or something has gone wrong, the flags for timer 1 will be enabled, timer 1 will also indicate the system is operating correctly if the timer 1 is overflowed with thee bit set as its counting the pulses as they are incoming.

This part does not make sense

If the same happened above with respect to timer 2 and this now overflows, all the code will clear any interrupts flags. This is not the case foe timer 0 if that starts to overflow causing the flag to set which will cause a series of events, these are the flag will firstly be reset, the value of 64 is entered as a pre-load and the millisecond value is increased by ten. Next the 'if' line or statements will check in brackets to see if the value has reached 100 milliseconds, this will equal 1 second and activates the code to run every second or 100 milliseconds. The 'speed low and high _ byte' statements are set to the values which were set in the tmr1l & TMR1H sections which are then both set along with the Ms value back to zero.

The explanation goes some way

```

//////////////////////////////////// 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;

    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 = 0b00000100;

```

Figure 32: Third section code

The above section of code seen in figure 32 is for the first main program loop, this section of code which appears in this main loop before the first while loop is code which has been set up. The first section of the program is initiating the variables for the key and the enable keypad, these are 'int' statements and are set to values which are '1' for the keypad and nothing for the key. Next comes the ADCON1 register, this will tell the programme if the 'TRISB, C and D are either set to outputs or inputs, this is displayed in the format of '0x00' which is hexadecimal format with a binary format following.

Port b follows which states the setting of the output and is set to '0', next is the LCD start-up section which was set in the first section of code as a pre-loaded programme from another location, but too big to physically include. Next the PIE1 to PIE2 section will control the interrupts which are peripheral and are pre-set.

*The explanation
is not clear.*

```

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 * 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_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/4;

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

```

Figure 33: 4th code section

Figure 33 which is included above is for the super loop for the while section, this code will operate at a continuous rate with the first part of coding in the above sections of the main loop only operating once. The speed current from the motor is first worked out in the code which will take its values for the equation from the speed low and high bytes detailed above in the code. Next there is 2 if statements which look to check if the pushbutton has been activated for stop or start settings and following this the run section will be set. The scan keypad will then look for the values entered to the key section which is set by the operator. This value is compared with pre-determined functions for a specific key value set by the operator.

Next the if statement that follows will display the values on the screen depending on what the operator has entered, the first part checks for a pressed key which will activate the keypad. The keypad will work like an old mobile phone as an example, only one button can be pressed at one time meaning to set a value of 1, then the number 1 is pressed and for 12 the digits 1 then 2 will need to be pressed. The hash key will revert the value back to zero.

Next there is an if statement which will check the program to make sure no digits on the keypad are being selected which would make the system available to take another value and change the number. The value then is set between 0 – 2300 and anything which falls under will revert back to zero and if it goes over will be set to the maximum value of 2300.

Next is the proportional and integral section, following a value of 1 to be equal to the system will work out the error which is done by subtracting the value of speed from the value of the setpoint, this moves onto the proportional part which takes the value for the error worked out above and multiplies this with the proportional variable. The I section or integral section will accumulatively store values of error which has been above times with the ki. Proportional/Integral output is the sum of the P section and the I section, and if the value is not 1 as stated above then all values for the proportional integral controller are set back to zero.

This explanation is confusing.

The final part of the code in this section is setting the maximum and minimum values of 0 and 2300 for the speed, the ccr11 part is the register duty cycle for the PWM and is worked out with the proportional integral output values. The very last part then displays the values onto the LCD as required.

This part is explained, but there are parts of the explanation here that don't make a lot of sense. The whole section should have been split into smaller sections for clarity.

```

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;

```

Figure 34: Fifth section of code

The next section of code shown above in figure 34 is for the display speed functions. This code will be activated when the line for display speed is stated in the program, this will then activate this code and is explained as follows. The first line with the 'int line and int value' statements are entered into this function, the code then will declare local variables which are then 'int units, tens, hunds, thous and temp'. the next section of code will then calculate the four-digit value stores int the 'value' area to a number in the local variables stated above.

An array comes next with each of the components being sent to this array with the value of the component being added with 48, this will convert the number into its ASCII equal value. A blank area is then inserted for the display and then followed with the letters for RPM to display onto the screen. The last line will then display the value to the LCD.

This is a clearer explanation.

```

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 1.
}

```

Figure 35: 6th code section

Figure 35 above is showing the display values functions. This function is the same as the function which has been described above with slight differences. The main code or program uses the display values part of the code to as expected display the values onto the LCD, this will display the speed, error and setpoint on the top line with prop part, int accum and outputs of the bottom of the LCD. This part of code can also check if values are negative or positive with the use of some if statements as seen above.

Explanation is reasonable.

```
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 36: Last code section

The last section of code is the scan keypad section, this part looks to see what button the operator has pressed to the physical keypad, outputs for each column have a setting of 1 and there is a series of if statements which check the rows for the logic 1. The key which has been selected by the operator can then be found with a grid fashion of checks to find the pushed button. The last delay section is to delay the programme for the number of cycles at the input with a for loop, this will not affect the ISR as that is the main priority of the system.

Good explanation

5. Testing and Analysis of results

The first part of the tests was to carry out a simulation test of the system, this was done using the proteus program found in the college. The first simulation is shown below in figure 37 and is setting the clock to 100hz.

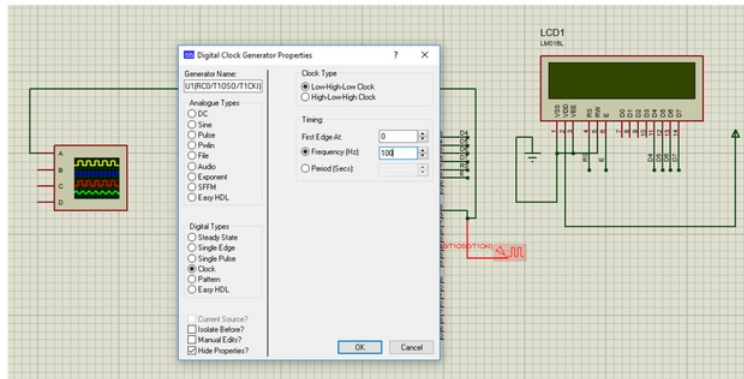


Figure 37: Setting clock to 100hz

Clock set at 100Hz = 10 blocks per pulse at 1ms each, maths = $1\text{ms} / 10\text{ divisions}$.
 $1/10 = 100\text{Hz}$. Shown working below in figure 38.

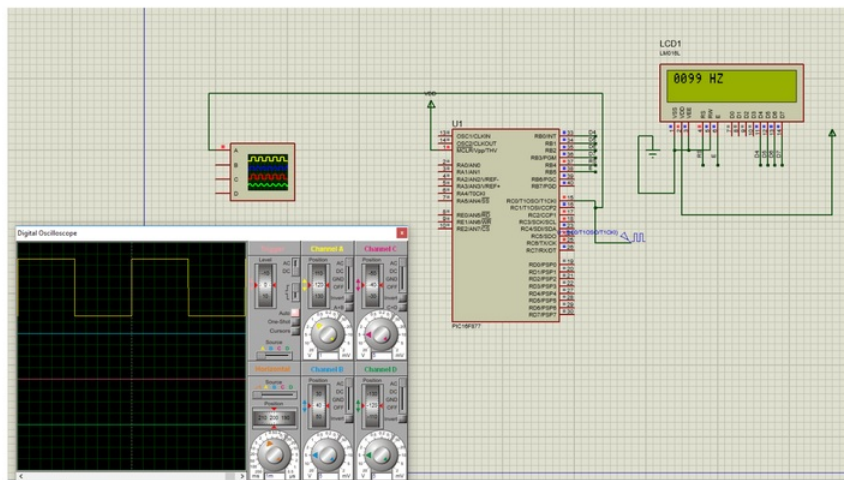


Figure 38: 100HZ clock setup

The two below screenshots 39 and 40 use the same principle and equations above but to work out the value of and display a 1000hz clock.

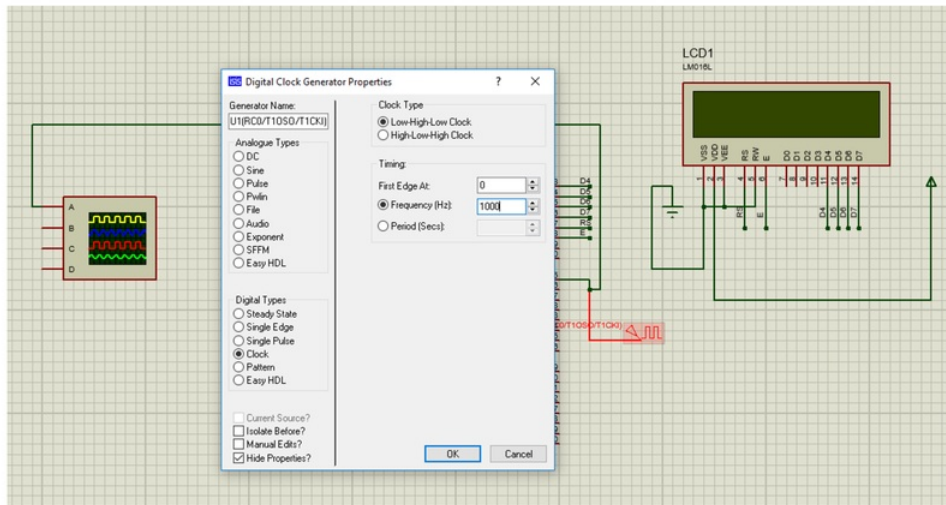


Figure 39: 1000hz clock setup

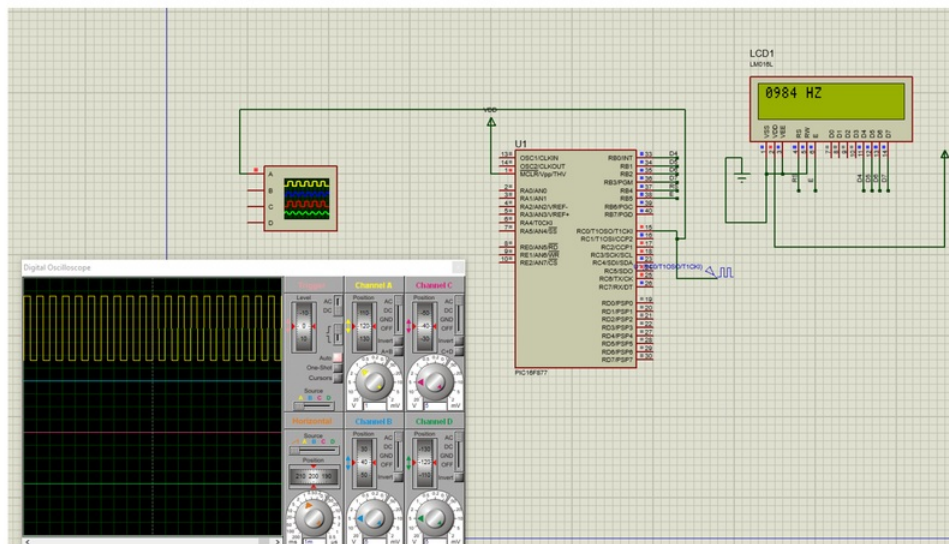


Figure 40: 1000hz clock display

This is the test for the encoder prototype.

The next screenshot shown below in figure 41 shows the value entered into the keypad being displayed on the LCD screen. The value of 1500 was selected and shown below.

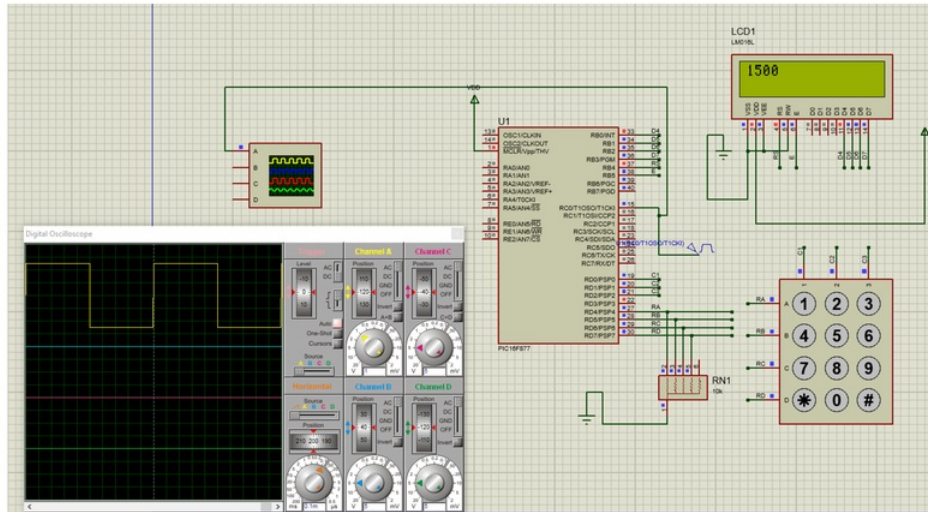


Figure 41: Testing LCD

The keypad encoder prototype test.

The PWM was to be tested to ensure that the setpoint entered into the program gave the correct output on the simulation, this will prove is the system is operating correctly or wrong. Three tests were carried out on the system to ensure operation and the values of 100rpm, 300rpm and the maximum of 360rpm.

The revolutions per minute (RPM) is calculated by – frequency * (60 ÷ PPR).

The Pulses per revolution (PPR) = 100.

Frequency is = 1/(Divisions on scope) *(Time base on scope).

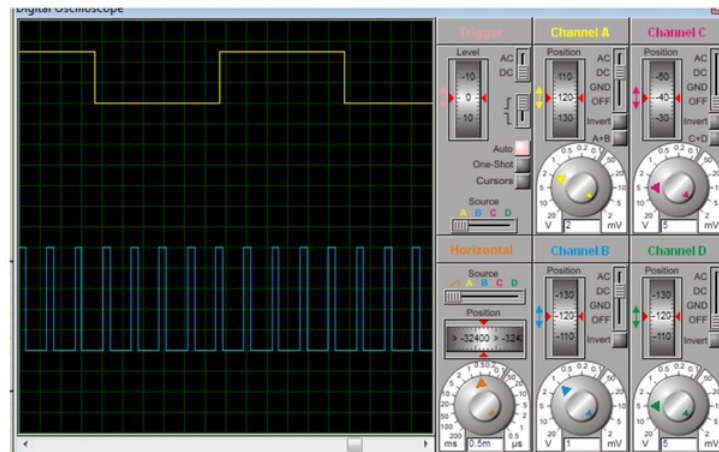


Figure 42: 100rpm

Figure 42 above shows the setting of 100rpm, the actual speed was recorded at 99, and the frequency using the equation above is $1/6 = 166.7$. the calculated RPM is $166.7 * 60/100 = 100$ RPM.

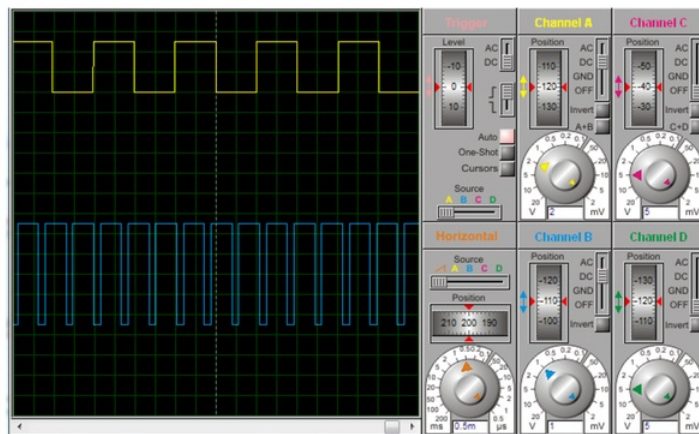


Figure 43: 300RPM

Figure 43 above shows the setting of 300rpm, the actual speed was recorded at 298, and the frequency using the equation above is $1/2 = 500$. the calculated RPM is $500 * 60/100 = 300$ RPM.

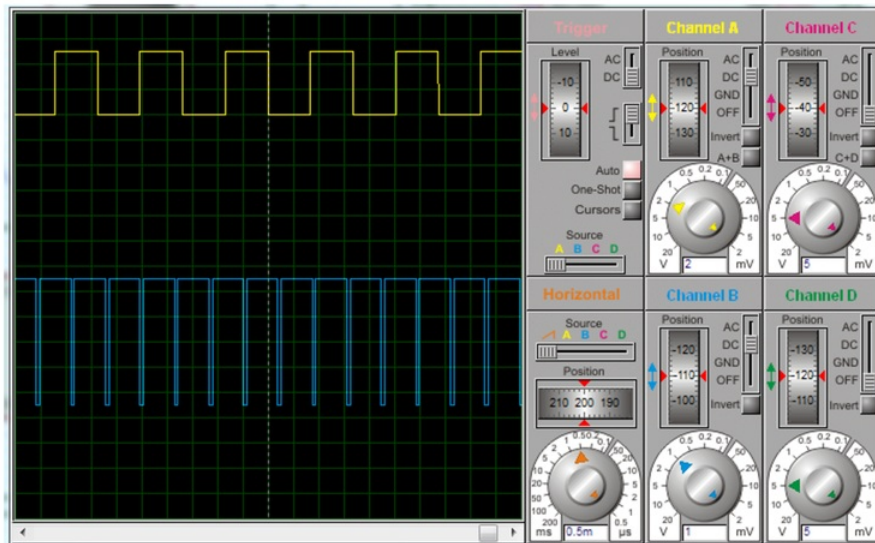


Figure 44: 360rpm

Figure 44 above shows the setting of 360rpm, the actual speed was recorded at 356, and the frequency using the equation above is $1/1.76 = 568.2$. the calculated RPM is $568.2 * 60/100 = 360$ RPM.

The above tests prove that the system was operating correctly and could now move onto the practical build to prove the operation in a practical form.

Five practical tests will be undertaken, they will include;

1. Setpoint enter (Should keep system running at setpoint) – Pass,
2. Load motor small increments (Should keep motor up with setpoint) – Pass,
3. PWM adjusting to setpoint (Shown adjusting of PWM) – Pass,
4. LCD displaying correct (Should show all parameters) – Pass,
5. Hash key press (Should set system back to zero) – Pass.

*This is the PWM
prototype test.*

Practical testing results.

200 rpm Set point. Shown below in figure 45.

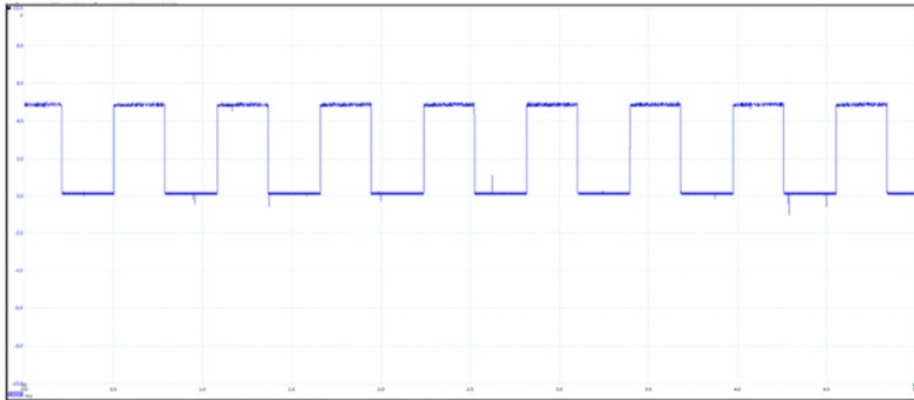


Figure 45: 200rpm practical

1.2 divs at 0.5ms per div = 0.0006. $F = 1 / 0.0006 = 1666\text{HZ}$

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

PWM set to 200 RPM

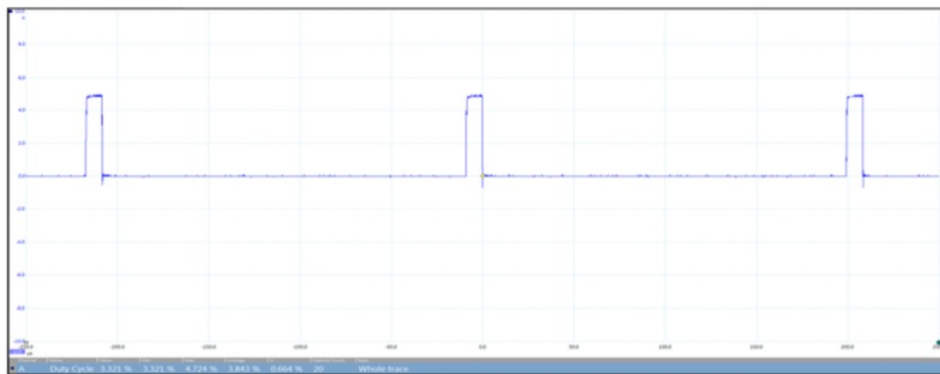


Figure 45: PWM 200rpm

Duty cycle = 48 (actual speed PWM) * 100/1023 (Duty at 100%)

Duty cycle = 4.7%

Setpoint 1000 RPM

1000 RPM Setpoint.

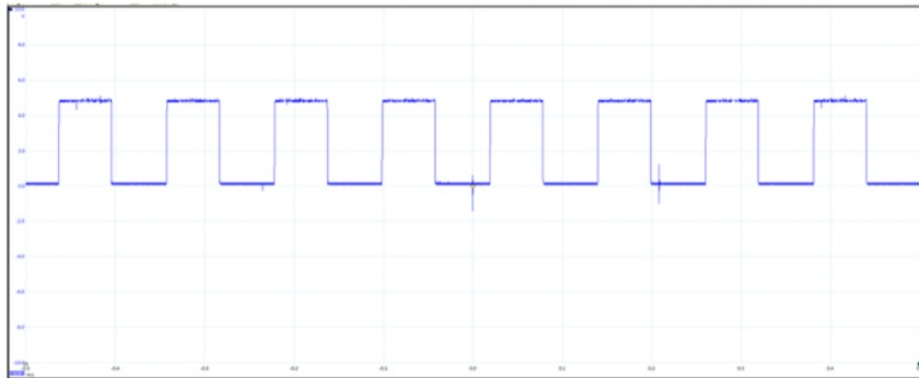


Figure 45: 1000rpm setpoint

1.2 divs 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

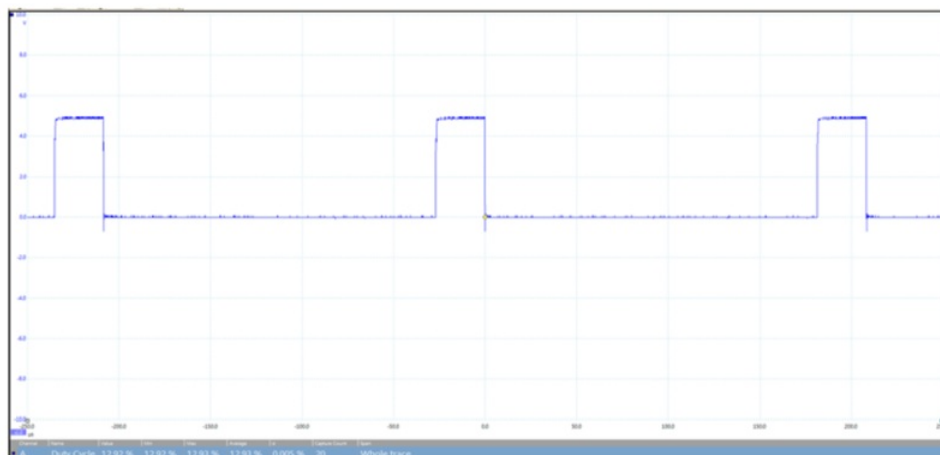


Figure 45: PWM at 1000rpm

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

$$\text{Duty cycle} = 13.3\%$$

Setpoint of 2000 RPM

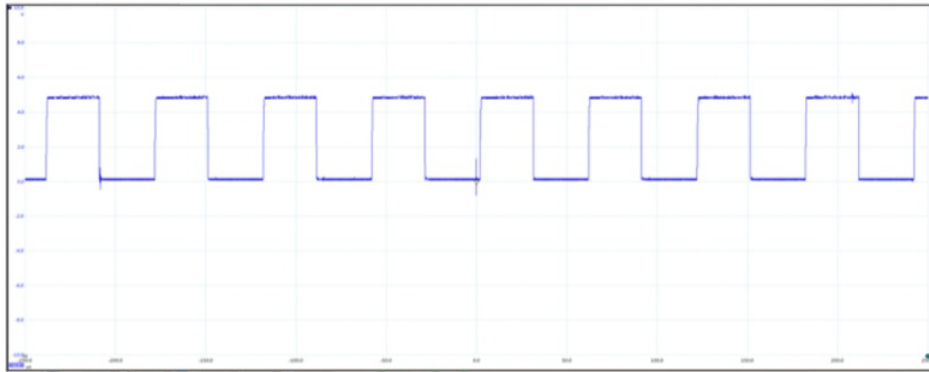


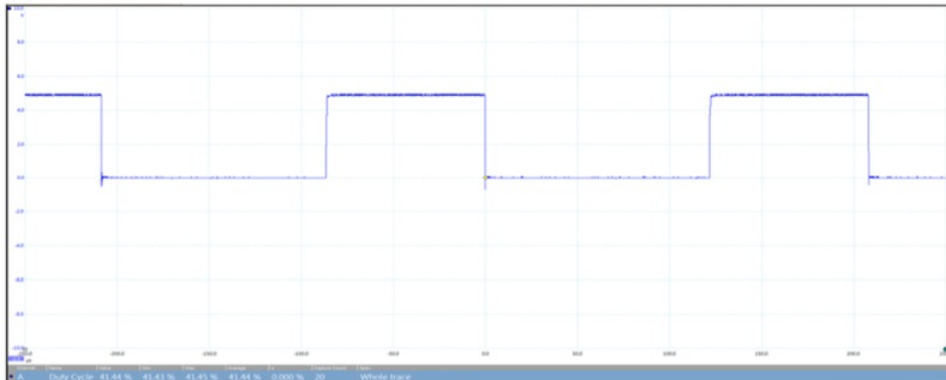
Figure 45: 2000rpm setpoint

1.2 divs at 0.05ms per div = 0.00006

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

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

PWM at 2000 RPM



Duty cycle = 391 (actual speed PWM) * 100/1023 (Duty at 100%)

Duty cycle = 41.4%

Setpoint 2300 RPM

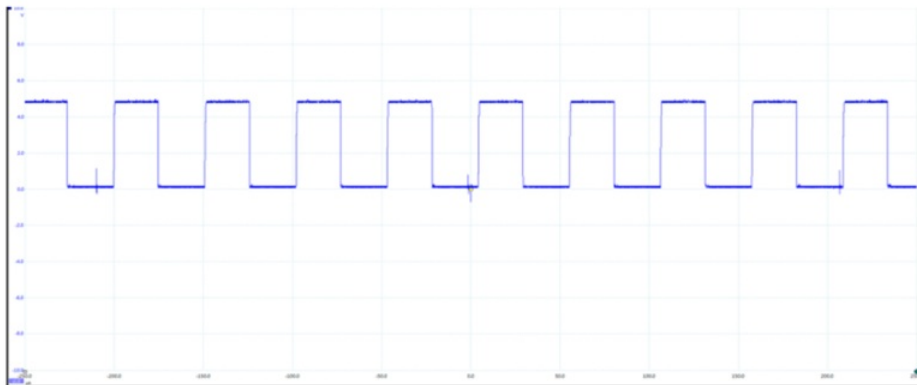


Figure 45: 2300 rpm

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 set to 200 RPM

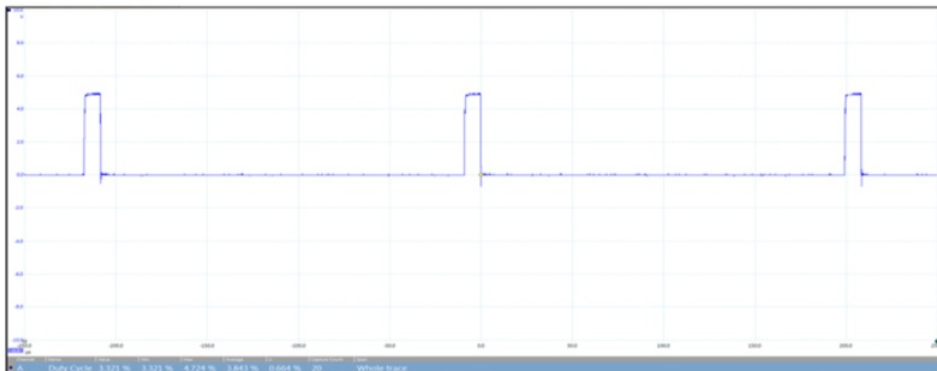


Figure 45: PWM 200rpm

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

$$\text{Duty cycle} = 4.7\%$$

Setpoint 1000 RPM

This should include at least an image of the practical demo with explanation.

6. Conclusions and Discussions

The assignment aimed to make use of a microcontroller and interface it with a DC Motor, incremental encoder, analogue sensing circuit, keypad, LCD display and two pushbuttons. The embedded system was designed and the system was tested as per the assignment brief.

The research or system hardware section was designed to investigate the different pieces of hardware that was used in the assignment and system. The research helped to gain valuable information and knowledge on the microcontroller which was then used to help to design and explain how the system worked. The chip set up also gained good knowledge and helped towards designing the system. The research covered all areas asked and was explained with the use of block text and images and data sheets, this gives the reader a broader understanding and shows the use of theory and application.

The system was designed to meet the required specifications, the user was able to set the keypad to a required input and see the output onto the screen of the system. The design was built with the code and then tested using proteus and a practical build, as shown in the results above the system did as was required and they system functions with no errors. The LCD screen was implemented and displayed the correct values with the correct spacing and locations.

Overall the design testing and simulation was a success, the system work as required and no problems were encountered, a newer microcontroller would provide better use and a recommendations and updated computers and monitors working at a faster speed could prove useful in the future. As shown above in the report the PI controller worked successfully and created a good working control system.

The conclusion says nothing about the final system design. It is also missing in the report. This is very vague and has no point to it.

7. References and Appendices

Works Cited

matrixtsl, 2005. *Version 3 PICmicro MCU development board HP488-00-3 Technical datasheet*. [Online]

Available at: <http://www.matrixtsl.com/datasheets/HP488-30-3.pdf>

[Accessed 20 05 2018].

Microchip, 2003. *PIC16F87XA Datasheet*. [Online]

Available at: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>

MICROCHIP, 2013. *PIC16F87XA*. [Online]

Available at: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582C.pdf>

[Accessed 20 05 2018].

PC Control Learning Zone, 2008. *Incremental encoders*. [Online]

Available at: https://www.pc-control.co.uk/incremental_encoders.htm

[Accessed 20 05 2018].

Pictutorials , 2017. *PIC 16X84 Basics*. [Online]

Available at: https://pictutorials.com/The_Option_Register.htm

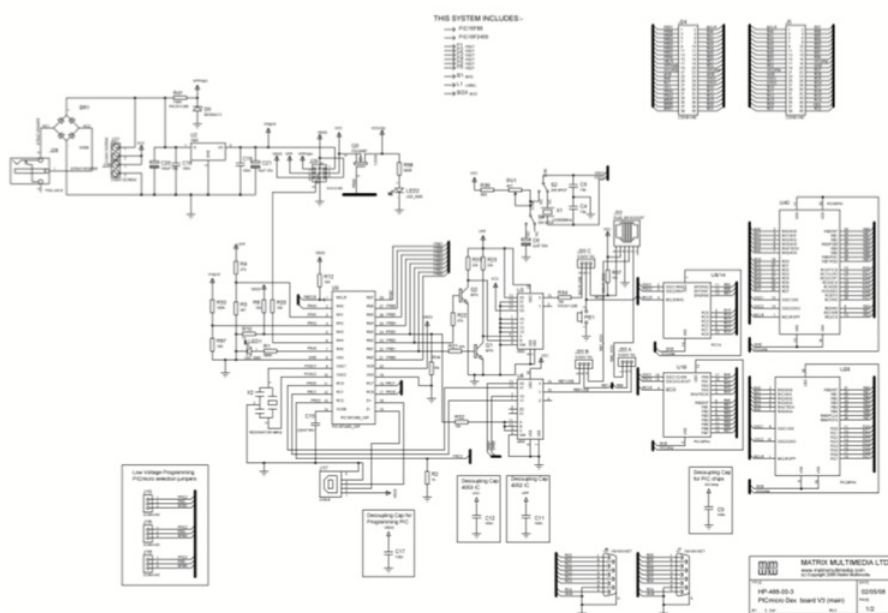
R-B, 2010. *ATC Semitec Thermistors*. [Online]

Available at: <http://embedded-lab.com/blog/lab-7-timers-and-counters-part-1/>

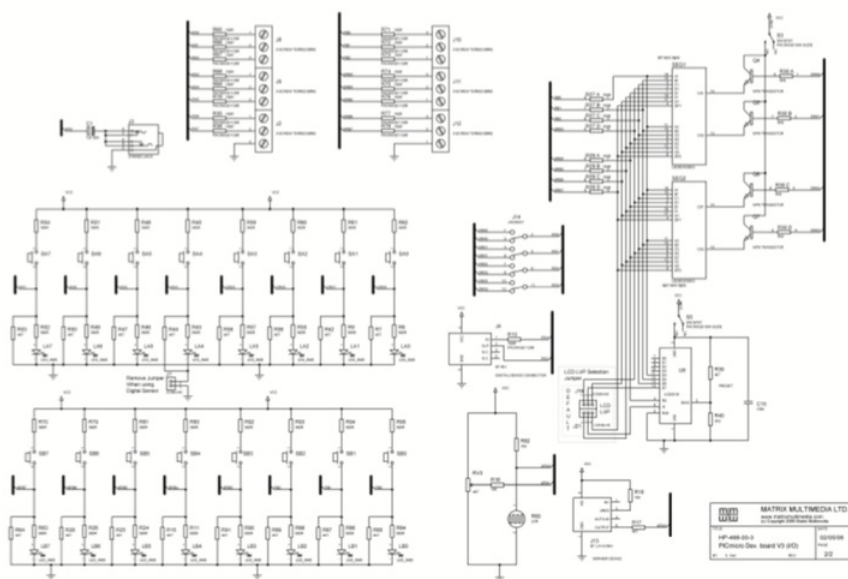
*Good refs but there is no
research section in the report.*

(MICROCHIP, 2013), datasheets below;

Appendix 1 – Circuit diagram

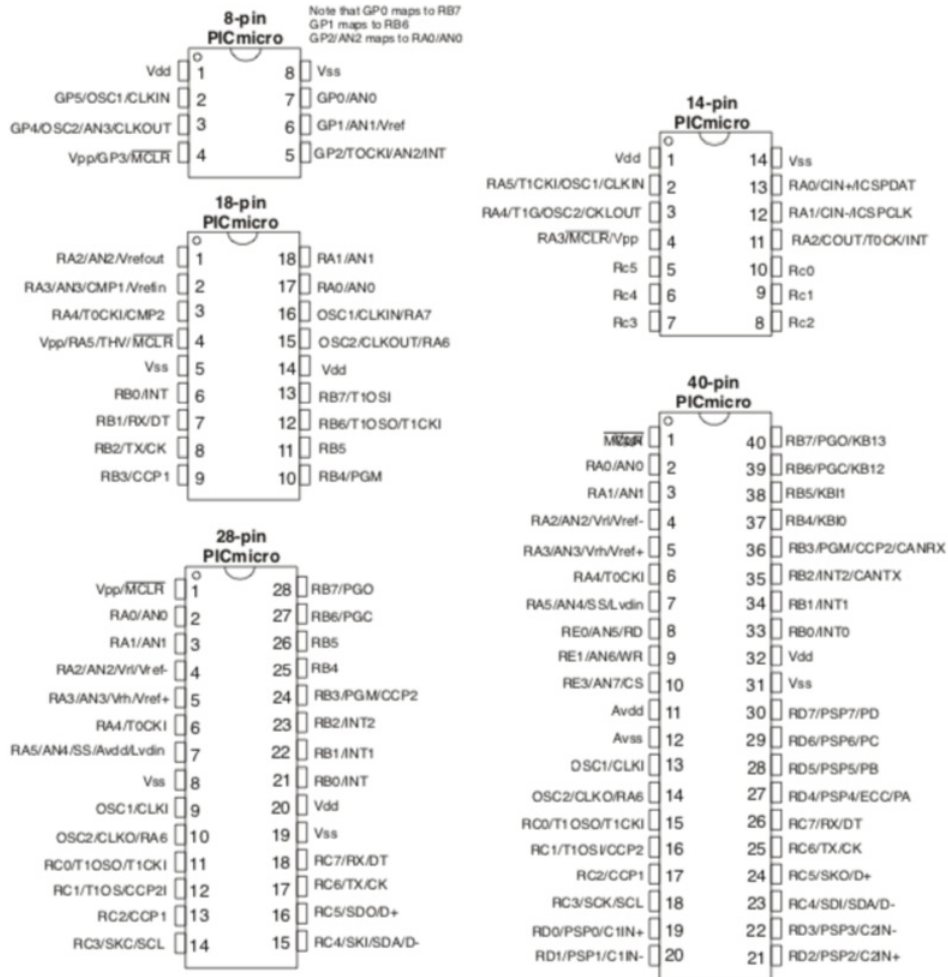


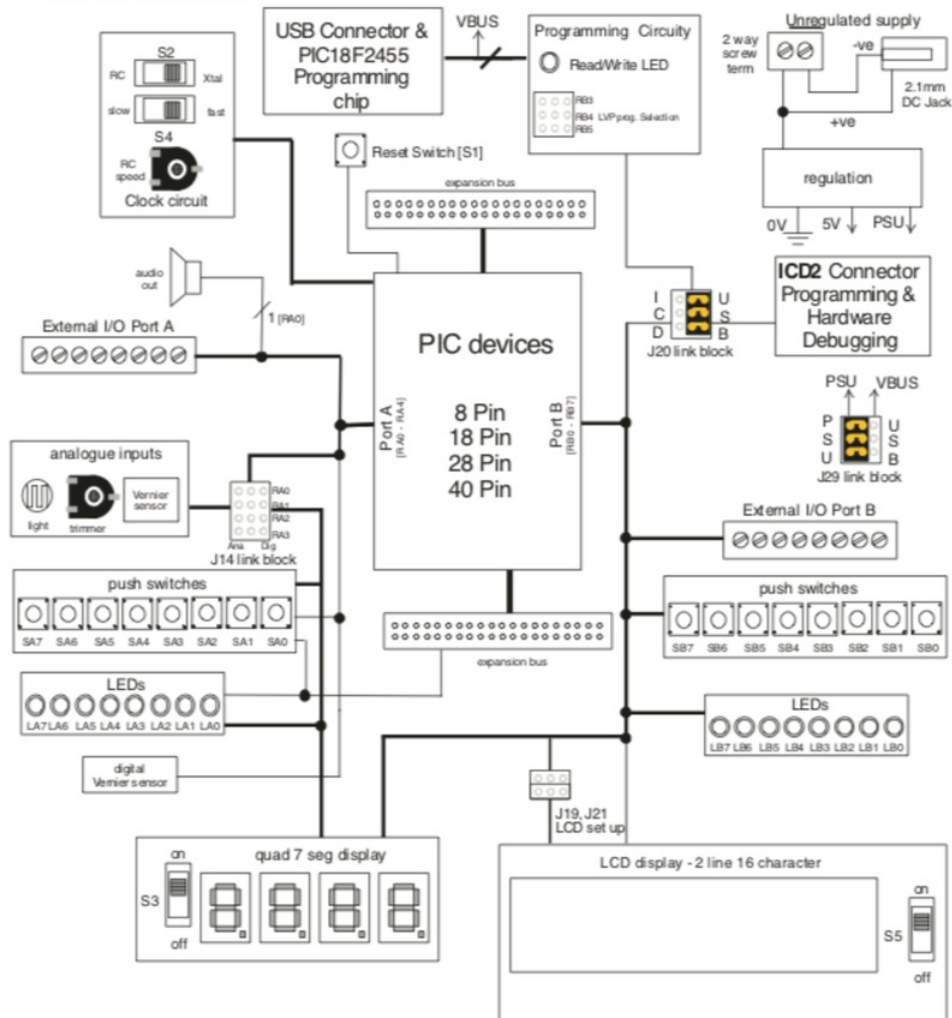
Appendix 1 – Circuit diagram



6. PICmicro microcontroller pin out details

Broadly speaking the ranges of PICmicro devices are designed to be upwards compatible: the pin functions on an 18-pin device are available on a 28-pin device and a 40-pin device. This can be seen from the following excerpt from the Microchip product selector card. The following diagram shows the pin out of the various PICmicro devices:



4. Block schematic

2. General information

1. Description

This flexible development board is an ideal platform for learning and project development. The board will program a range of 8, 14, 18, 28 and 40 pin PICmicro® microcontroller devices from the 12, 16 and 18 series PICmicro range. The board is programmed using the USB port (from which it can also take power) and is supplied with a comprehensive programming utility - PPP. The board is ideal for using in conjunction with one of our 3 PICmicro CD ROM resources to help learn PICmicro programming using flow charts, C or assembly code. The board can program Low Voltage Programmable PICmicro MCUs and deliver a limited amount of power using only the USB supply, but we recommend that an external power supply (product code HPPSU2) is used to take maximum advantage of the board's features. The board is compatible with our range of E-blocks modules which contain circuitry for a wide range of functions - from simple switches and LEDs, through to internet servers, and Bluetooth transceivers. The board is also compatible with Microchip's In Circuit Debugging 2 system.

2. Features

- Makes it easier to teach and learn PICmicro programming
 - Designed for educational use
 - 3 CD ROM based resources in assembly, C, and flow chart programming are available
 - Free download software, PPP, provides seamless send and verify functions
- Makes it easier to develop PICmicro projects
 - Supports low cost Flash-programmable PICmicro devices
 - Fully featured displays - 16 individual LEDs, quad 7-segment display and LCD display
 - USB programmer
 - Supports PICmicro microcontrollers with A/D converters
 - On-board sensors
 - A comprehensive range of external analogue and digital sensors are available for project work
 - Fully protected expansion bus for project work
 - All inputs and outputs available on screw terminal connectors for easy connection
 - E-Blocks compatible with 2 E-Blocks connectors fitted to ports C and D
- Low cost
- Used as a programmer and as a development board
- Programs a wide range of PICmicro MCU devices
- Full suite of programming software available
- RC or Xtal operation
- In-Circuit Debugging via MPLAB® ICD2

3. New features for Version 3

The following are the improvements made on version 3:

- Improved power circuit for either polarity power supply
- Compatible with a wider range of devices in Low Voltage Program mode
- Greater options for LCD connection to port B for Low Voltage Programming
- Support for a wider range of PICmicro devices
- Fast programming - 8k in under 10 seconds

PSDE 5013 Assn 2 Nathan Smith

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

39/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

Text Comment. The introduction should mention the design and test process using prototypes for the stages of the design. These being the keypad encoder, optical encoder feedback and PWM driver.

Text Comment. The current measurement was in the prototype and left out due to space on the LCD.

PAGE 5

Text Comment. This diagram should be the hardware description.

PAGE 6

Text Comment. This could have been clearer.

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

Text Comment. The MOSFET purpose is explained, but it should be noted that the MCU can only handle 5V max. This external device provided the interface capability to switch at 24V.

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

Text Comment. The configuration is correct. Some of the explanations are a little vague.

PAGE 22

Text Comment. This is correct.

PAGE 23

Text Comment. This points out the prototype method used which is significant to the final design.

Text Comment. This explains the inheritance of the LCD functions.

PAGE 24

Text Comment. TOIF, TMR1 and TMR2 should be capitals.

Text Comment. This part does not make sense.

Text Comment. The explanation goes some way to explain the process of the ISR, but there are ,any spelling and grammar errors which corrupt its legibility.

PAGE 25

Text Comment. The explanation is not clear.

PAGE 26

PAGE 27

Text Comment. This explanation is confusing.

Text Comment. This part is explained, but there are parts of the explanation here that don't make a lot of sense. The whole section should have been split into smaller sections for clarity.

PAGE 28

Text Comment. This is a clearer explanation.

PAGE 29

PAGE 30

Text Comment. Explanation is reasonable.

Text Comment. Good explanation

PAGE 31

PAGE 32

Text Comment. This is the test for the encoder prototype.

PAGE 33

Text Comment. The keypad encoder prototype test.

PAGE 34

PAGE 35

Text Comment. This is the PWM prototype test.

PAGE 36

PAGE 37

PAGE 38

PAGE 39

Text Comment. This should include at least an image of the practical demo with explanation.

PAGE 40

Text Comment. The conclusion says nothing about the final system design. It is also missing in the report. This is very vague and has no point to it.

PAGE 41

Text Comment. Good refs but there is no research section in the report.

GRADING FORM: CSG-RUBIC

NATHAN SMITH**0****RESEARCH-INFORMED LITERATURE**

n/a

KNOWLEDGE AND UNDERSTANDING

Introduction should have included the design process using prototypes. This just gives a very brief overview. 2/5 System hardware have been explained, but some part are unclear. 2.5/5 System design is not explained very well and does not include the final design diagram or overall explanation. The configuration does make sense and is correct. 8/20 Software design should have been divided up into smaller sections. The explanation in parts in the report are very vague and confused. 4/10

17/40**ANALYSIS**

The testing covers all parts of the system. There



should be some screen shots to indicate what the tests represent on the LCD. This should cover the CAD and dev board designs. 5/10 Conclusion does not make any point mostly because there are no images for the final design to make comparisons. 2/10

7/20

PRACTICAL APPLICATION AND DEP.



Software design is good but the explanations are poor and some of this is down to poor spelling and grammar. 3/10 Quality of work is poor. 3/10

6/20

SKILLS FOR PROFESSIONAL PRAC.



The report contains many errors and there is work missing. It looks like it was done in a hurry. 3/10 Team working is good. 6/10 Team working

9/20