

EXAMINATION INTERNAL VERIFICATION FORM (2017-2018)

MODULE :	PSDE5013 Configuration and Programming of Embedded Systems	LEVEL:	5
PROGRAMME :	HNC Electrical and Electronic Engineering	TERM OF EXAM: (SPRING OR SUMMER)	Summer
LEARNING OUTCOME(S) BEING ASSESSED:	MILO1, MILO2	EXAM PAPER:	Main
		EXAMINATION AUTHOR :	R. Jenkins

VERSION:	DATE SUBMITTED FOR VERIFICATION:	INTERNAL VERIFIER COMMENTS (Approval or detail of amendments required):	DATE:
1.	26-03-18	Meets the assessment pattern and tests the module outcomes appropriately	26/03/18
2.			

INTERNAL VERIFIER PLEASE TICK TO INDICATE THAT THE EXAMINATION:	
✓	Conforms to the assessment pattern described in the module syllabus
✓	Tests the module learning outcome(s) appropriately

SIGNATURES OF FINAL APPROVAL:	SIGNATURE	DATE
INTERNAL VERIFIER: I confirm that this version of the exam paper is ready to be sent to the external examiner	<i>R. Davies</i>	27/03/18
EXAM AUTHOR: I confirm that internal verifier and external examiner approval have been received and that this version of the exam paper is ready for printing.		

**UNIVERSITY OF WALES:
TRINITY SAINT DAVID**



**PRIFYSGOL CYMRU:
Y DRINDOD DEWI SANT**

LOCATION: COLEG SIR GAR

LLEOLIAD: COLEG SIR GAR

LEVEL 5

LEFEL 5

**EXAMINATION PERIOD
MAY 2018**

**CYFNOD ARHOLIADAU
MAI 2018**

BSc ELECTRICAL AND ELECTRONIC ENGINEERING

MODULE CODE: PSDE5013

CONFIGURATION AND PROGRAMMING OF EMBEDDED SYSTEMS

TIME: 2 HOURS

ANSWER THREE QUESTIONS OUT OF FOUR

TOTAL MARKS = 102

NUMBER OF PAGES = 6

THIS PAPER INCLUDES AN APPENDIX OF 15 PAGES

Equipment or materials required – Calculators are permitted

Adnoddau neu ddefnyddiau sydd eu hangen – Caniateir cyfrifiannell

MAIN PAPER

Question 1

- (a) Configure the microcontroller for an interrupt routine which will interrupt the processor every 10ms when a 10MHZ crystal clock is used.

Give the necessary configuration details for the relevant function registers using the data sheets in the appendix. [8]

- (b) Using the configuration from part (a), create an interrupt routine (ISR) in C code that will count the minutes and hours for a digital clock with a 10MHZ internal clock.

Use in – line comments to explain the code given the following global variables have already been declared and initialised: -

[6]

```
int milli = 0, secs = 0, mins = 0, hrs = 10;
```

- (c) Create a typical state diagram that explains the alarm functions of a digital alarm clock.

This will include: -

[8]

- (i) Alarm time display
- (ii) Alarm mins increment and decrement
- (iii) Alarm hrs increment and decrement
- (iv) Alarm set and cancel
- (v) Alarm output on / off

- (d) Create a C program that implements the state diagram from part (c). Include in the code the necessary inputs, outputs, function prototypes and global variables given.

[12]

```
void LCD_Startup();
void DisplayTime (int line);
void DisplayAlarmTime (int line);
void DisplaySetAlarmTime (int line);
```

```
int alarm, state, alarm_mins, alarm_hrs;
```

[34 marks]

Question 2

- (a) A PIC microcontroller has many on – board peripherals, one of them is an ADC (Analogue to Digital Converter). Use the data sheets provided in the appendix to configure the special function registers (SFR's) for the ADC to read four continuous analogue signals from various sensors connected to analogue channels AN0, AN1, AN6, AN7. The ADC is to be configured for 10 bit resolution with a conversion clock rate of $F_{osc}/32$. Use positive and negative reference voltages connected to RA2 and RA3 in the configuration. [9]

- (b) The measurements taken at the ADC channel AN0 are recorded in the table shown in Figure Q2B. The temperature ranges between 0° and 12° C and the raw value 50 from the ADC must be converted into 0° . Explain why 52 elements must be used in the array for the look up table and explain the look up table function. [6]

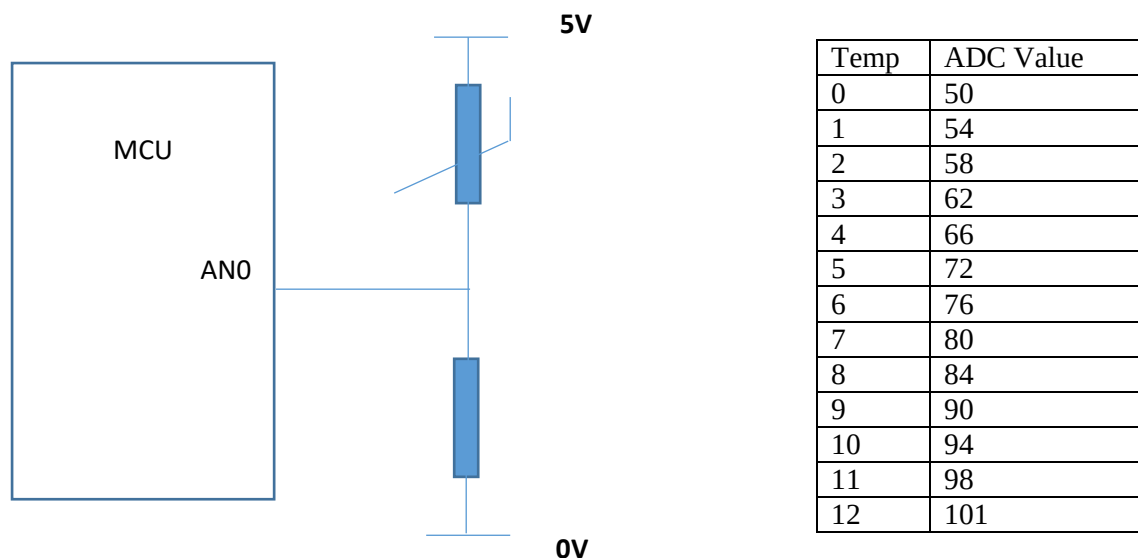


Figure Q2B

- (c) Create an array in C code using the look up table in part (b). [6]
- (d) If the look up table in part (c) has been declared as a global variable, write the main function in C code that samples the ADC and displays the temperature. Use the following function prototypes given below and include all configuration. [13]

```
void LCD_Startup();
void Display_LCD_Value (int line, int value);
void Delay (int t);
```

[34 marks]

Question 3

- (a) Using the system shown below in Figure Q3A as reference, explain how the MCU reads the encoder pulses and how can this be tested.
Include all configurations and code snippets. [14]

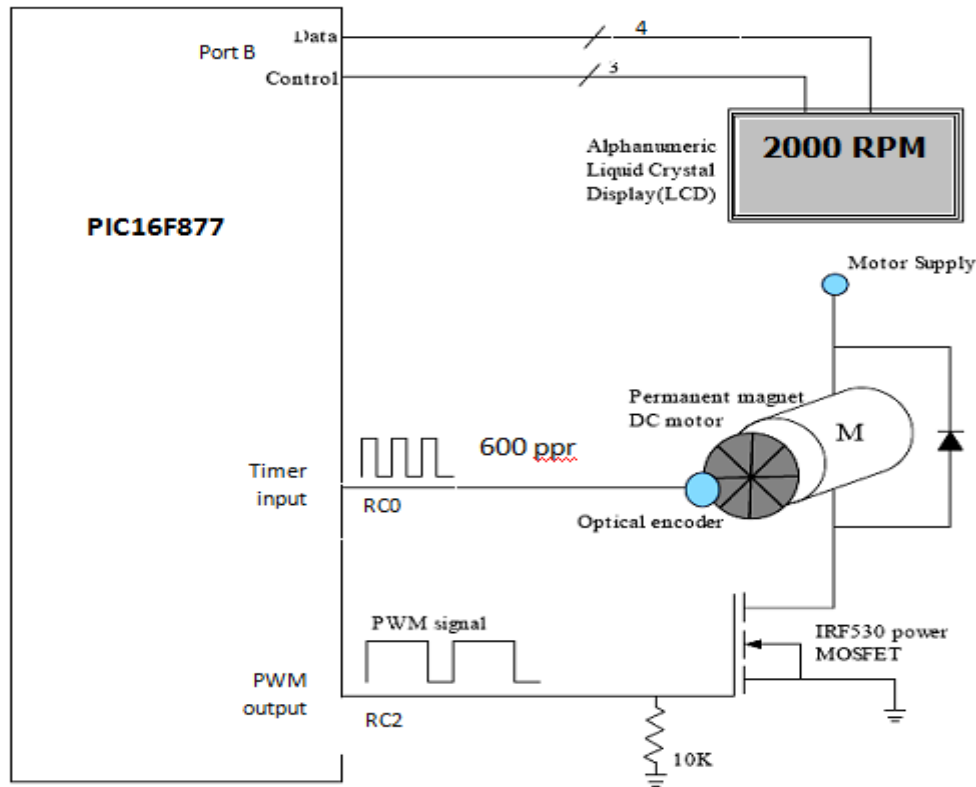


Figure Q3A

- (b) Create the C code for an optical encoder input at RC0 and display the measured value on the LCD in RPM (Revolutions Per Minute).
The encoder gives 600ppr and the display should be updated every 100ms.
Use the function prototypes given and include all configurations. [12]

```
DisplaySpeed (int line, int value);
LCD_Startup();
```

- (c) Using the circuit in Figure Q3A, create a C program that will drive the DC motor using a pulsed signal. Show all configurations and set duty cycle at 75% in 10-bit resolution mode of operation. [8]

[34 marks]

Question 4

- (a) With reference to the circuit shown in Figure Q4A of a PIC microcontroller port D, explain using a diagram and snippets of C code how a 3 x 4 keypad encoder can be implemented.

[14]

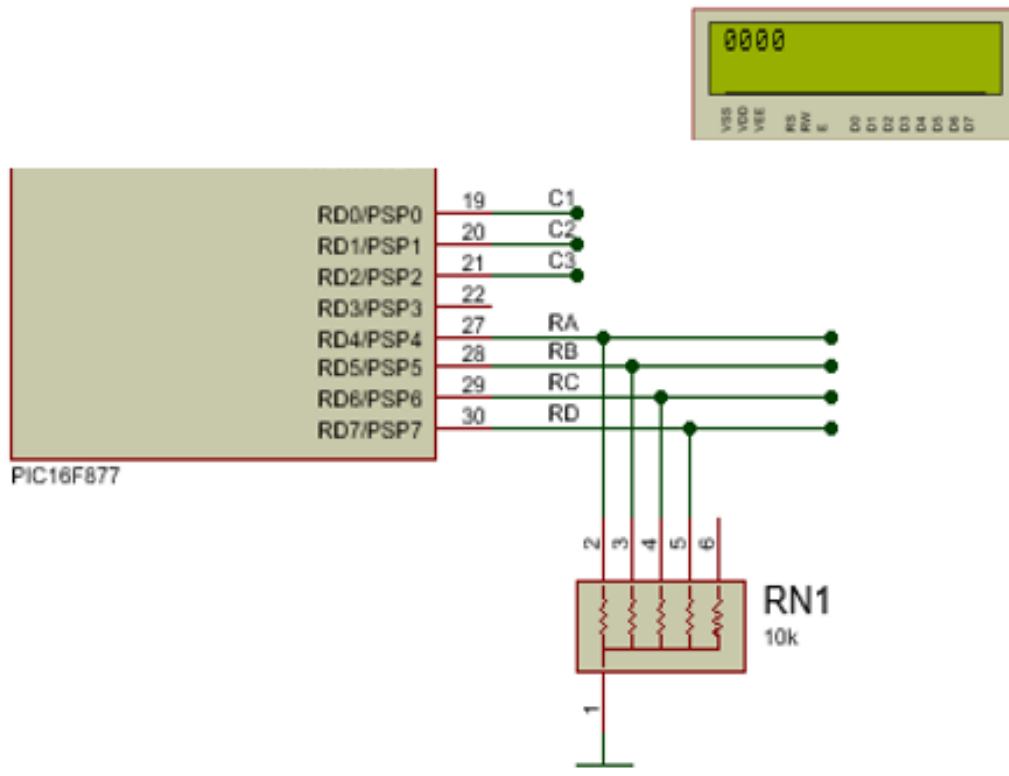


Figure Q4A

- (b) The keypad encoder function is to input a set value which can be displayed on the LCD display. This value is returned from a scan function which detects key press events.

[12]

Create a C program that handles this returned value and implement a constraint and a reset for the value entered. Use in line comments to explain the code.

Use the following scan function that returns a key press value:

```
int ScanKeypad ();
```

- (c) If the keypad encoder has four key entries to be displayed on the LCD in units, tens, hundreds and thousands units order, create a function that will handle the four inputs and order the display. Use in line comments to explain the C code.

Use the function prototypes given:-

[8]

```
void DisplayValue(int line, int value);  
void DisplayString(int line, char string);
```

[34 marks]

PIC16F87X

2.2.2.1 STATUS Register

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bits for data memory.

The STATUS register can be the destination for any instruction, as with any other register. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to the device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable, therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, CLRF STATUS will clear the upper three bits and set the Z bit. This leaves the STATUS register as 000u u1uu (where u = unchanged).

It is recommended, therefore, that only BCF, BSF, SWAPF and MOVWF instructions are used to alter the STATUS register, because these instructions do not affect the Z, C or DC bits from the STATUS register. For other instructions not affecting any status bits, see the "Instruction Set Summary."

Note: The C and DC bits operate as a borrow and digit borrow bit, respectively, in subtraction. See the SUBLW and SUBWF instructions for examples.

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

	R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
	bit 7							bit 0
bit 7	IRP: Register Bank Select bit (used for indirect addressing) 1 = Bank 2, 3 (100h - 1FFh) 0 = Bank 0, 1 (00h - FFh)							
bit 6-5	RP1:RP0: Register Bank Select bits (used for direct addressing) 11 = Bank 3 (180h - 1FFh) 10 = Bank 2 (100h - 17Fh) 01 = Bank 1 (80h - FFh) 00 = Bank 0 (00h - 7Fh) Each bank is 128 bytes							
bit 4	$\overline{TO}$: Time-out bit 1 = After power-up, CLRWD instruction, or SLEEP instruction 0 = A WDT time-out occurred							
bit 3	$\overline{PD}$: Power-down bit 1 = After power-up or by the CLRWD instruction 0 = By execution of the SLEEP instruction							
bit 2	Z: Zero bit 1 = The result of an arithmetic or logic operation is zero 0 = The result of an arithmetic or logic operation is not zero							
bit 1	DC: Digit carry/borrow bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for borrow, the polarity is reversed) 1 = A carry-out from the 4th low order bit of the result occurred 0 = No carry-out from the 4th low order bit of the result							
bit 0	C: Carry/borrow bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) 1 = A carry-out from the Most Significant bit of the result occurred 0 = No carry-out from the Most Significant bit of the result occurred Note: For borrow, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.							

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

PIC16F87X

2.2.2.2 OPTION_REG Register

The OPTION_REG Register is a readable and writable register, which contains various control bits to configure the TMR0 prescaler/WDT postscaler (single assignable register known also as the prescaler), the External INT Interrupt, TMR0 and the weak pull-ups on PORTB.

Note: To achieve a 1:1 prescaler assignment for the TMR0 register, assign the prescaler to the Watchdog Timer.

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

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0	
bit 7	RBPU: PORTB Pull-up Enable bit 1 = PORTB pull-ups are disabled 0 = PORTB pull-ups are enabled by individual port latch values							
bit 6	INTEDG: Interrupt Edge Select bit 1 = Interrupt on rising edge of RB0/INT pin 0 = Interrupt on falling edge of RB0/INT pin							
bit 5	T0CS: TMR0 Clock Source Select bit 1 = Transition on RA4/T0CKI pin 0 = Internal instruction cycle clock (CLKOUT)							
bit 4	T0SE: TMR0 Source Edge Select bit 1 = Increment on high-to-low transition on RA4/T0CKI pin 0 = Increment on low-to-high transition on RA4/T0CKI pin							
bit 3	PSA: Prescaler Assignment bit 1 = Prescaler is assigned to the WDT 0 = Prescaler is assigned to the Timer0 module							
bit 2-0	PS2:PS0: Prescaler Rate Select bits							
	Bit Value	TMR0 Rate	WDT Rate					
	000	1 : 2	1 : 1					
	001	1 : 4	1 : 2					
	010	1 : 8	1 : 4					
	011	1 : 16	1 : 8					
	100	1 : 32	1 : 16					
	101	1 : 64	1 : 32					
	110	1 : 128	1 : 64					
	111	1 : 256	1 : 128					

Legend:

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

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

PIC16F87X

2.2.2.3 INTCON Register

The INTCON Register is a readable and writable register, which contains various enable and flag bits for the TMR0 register overflow, RB Port change and External RB0/INT pin interrupts.

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt flag bits are clear prior to enabling an 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-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							

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

5.0 TIMER0 MODULE

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter
- Readable and writable
- 8-bit software programmable prescaler
- Internal or external clock select
- Interrupt on overflow from FFh to 00h
- Edge select for external clock

Figure 5-1 is a block diagram of the Timer0 module and the prescaler shared with the WDT.

Additional information on the Timer0 module is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

Timer mode is selected by clearing bit T0CS (OPTION_REG<5>). In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

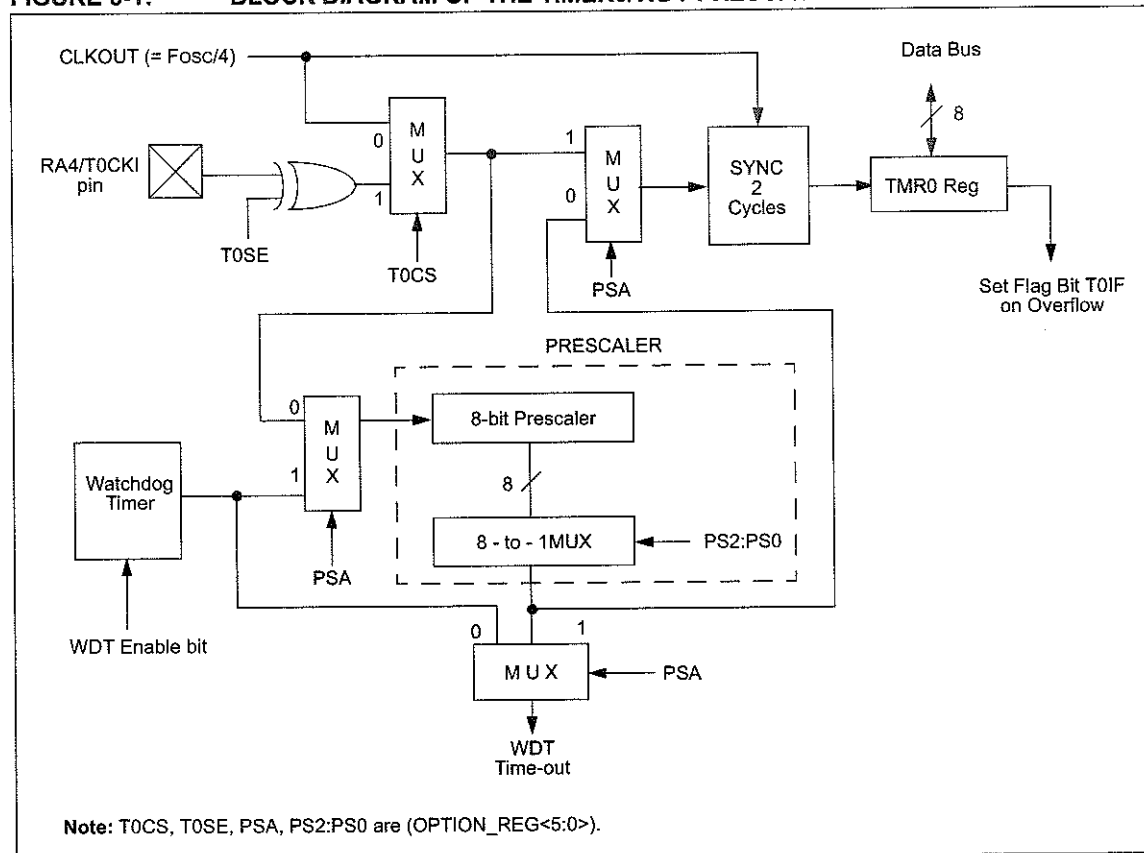
Counter mode is selected by setting bit T0CS (OPTION_REG<5>). In Counter mode, Timer0 will increment either on every rising, or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (OPTION_REG<4>). Clearing bit T0SE selects the rising edge. Restrictions on the external clock input are discussed in detail in Section 5.2.

The prescaler is mutually exclusively shared between the Timer0 module and the Watchdog Timer. The prescaler is not readable or writable. Section 5.3 details the operation of the prescaler.

5.1 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h. This overflow sets bit T0IF (INTCON<2>). The interrupt can be masked by clearing bit T0IE (INTCON<5>). Bit T0IF must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from SLEEP, since the timer is shut-off during SLEEP.

FIGURE 5-1: BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER



PIC16F87X

5.2 Using Timer0 with an External Clock

When no prescaler is used, the external clock input is the same as the prescaler output. The synchronization of T0CKI with the internal phase clocks is accomplished by sampling the prescaler output on the Q2 and Q4 cycles of the internal phase clocks. Therefore, it is necessary for T0CKI to be high for at least 2Tosc (and a small RC delay of 20 ns) and low for at least 2Tosc (and a small RC delay of 20 ns). Refer to the electrical specification of the desired device.

5.3 Prescaler

There is only one prescaler available, which is mutually exclusively shared between the Timer0 module and the Watchdog Timer. A prescaler assignment for the

Timer0 module means that there is no prescaler for the Watchdog Timer, and vice-versa. This prescaler is not readable or writable (see Figure 5-1).

The PSA and PS2:PS0 bits (OPTION_REG<3:0>) determine the prescaler assignment and prescale ratio.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g. CLRF 1, MOVWF 1, BSF 1, x...etc.) will clear the prescaler. When assigned to WDT, a CLRWDI instruction will clear the prescaler along with the Watchdog Timer. The prescaler is not readable or writable.

Note: Writing to TMR0, when the prescaler is assigned to Timer0, will clear the prescaler count, but will not change the prescaler assignment.

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
	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					

Legend:

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

Note: To avoid an unintended device RESET, the instruction sequence shown in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023) must be executed when changing the prescaler assignment from Timer0 to the WDT. This sequence must be followed even if the WDT is disabled.

PIC16F87X

TABLE 5-1: REGISTERS ASSOCIATED WITH TIMER0

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
01h,101h	TMR0	Timer0 Module's Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
81h,181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'.
Shaded cells are not used by Timer0.

11.0 ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the other devices.

The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. The A/D conversion of the analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, Vss, RA2, or RA3.

The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D's internal RC oscillator.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register, shown in Register 11-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 11-2, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference), or as digital I/O.

Additional information on using the A/D module can be found in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

REGISTER 11-1: ADCON0 REGISTER (ADDRESS: 1Fh)

	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							
	<u>If ADON = 1:</u>							
	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							

Note 1: These channels are not available on PIC16F873/876 devices.

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

PIC16F87X

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.
- 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.

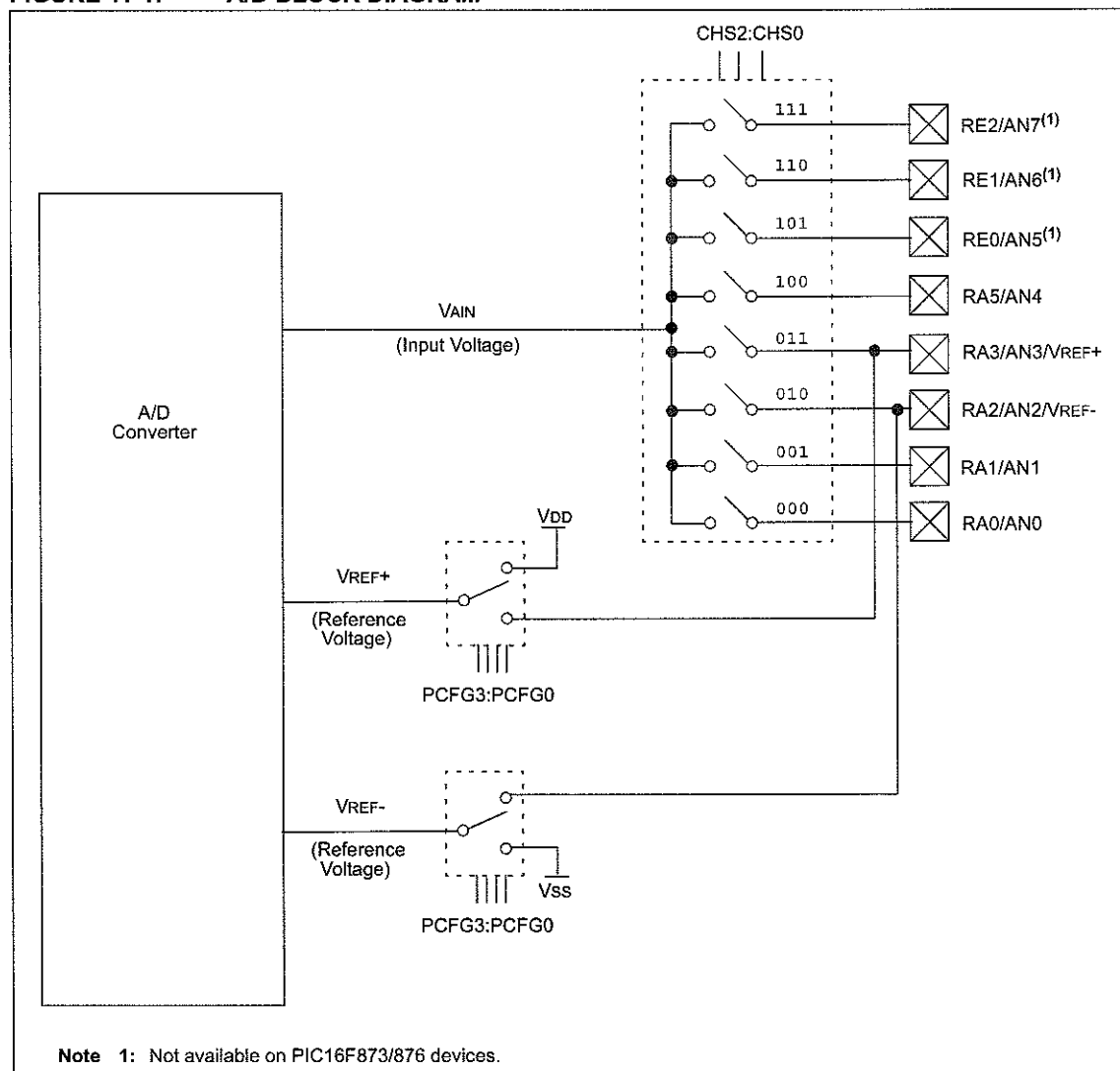
To determine sample time, see Section 11.1. After this acquisition time has elapsed, the A/D conversion can be started.

PIC16F87X

These steps should be followed for doing an A/D Conversion:

1. Configure the A/D module:
 - Configure analog pins/voltage reference and digital I/O (ADCON1)
 - Select A/D input channel (ADCON0)
 - Select A/D conversion clock (ADCON0)
 - Turn on A/D module (ADCON0)
2. Configure A/D interrupt (if desired):
 - Clear ADIF bit
 - Set ADIE bit
 - Set PEIE bit
 - Set GIE bit
3. Wait the required acquisition time.
4. Start conversion:
 - Set GO/DONE bit (ADCON0)
5. Wait for A/D conversion to complete, by either:
 - Polling for the GO/DONE bit to be cleared (with interrupts enabled); OR
 - Waiting for the A/D interrupt
6. Read A/D result register pair (ADRESH:ADRESL), clear bit ADIF if required.
7. For the next conversion, go to step 1 or step 2, as required. The A/D conversion time per bit is defined as TAD. A minimum wait of 2TAD is required before the next acquisition starts.

FIGURE 11-1: A/D BLOCK DIAGRAM



6.0 TIMER1 MODULE

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L), which are readable and writable. The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 Interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a timer
- As a counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle. In Counter mode, it increments on every rising edge of the external clock input.

Timer1 can be enabled/disabled by setting/clearing control bit TMR1ON (T1CON<0>).

Timer1 also has an internal "RESET input". This RESET can be generated by either of the two CCP modules (Section 8.0). Register 6-1 shows the Timer1 control register.

When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI/CCP2 and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored, and these pins read as '0'.

Additional information on timer modules is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

	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							

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

PIC16F87X

6.1 Timer1 Operation in Timer Mode

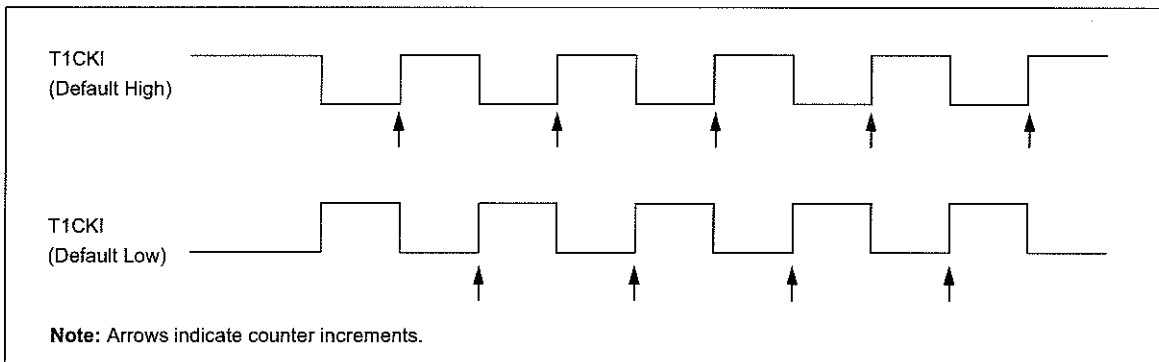
Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is $F_{osc}/4$. The synchronize control bit T1SYNC (T1CON<2>) has no effect, since the internal clock is always in sync.

6.2 Timer1 Counter Operation

Timer1 may operate in either a Synchronous, or an Asynchronous mode, depending on the setting of the TMR1CS bit.

When Timer1 is being incremented via an external source, increments occur on a rising edge. After Timer1 is enabled in Counter mode, the module must first have a falling edge before the counter begins to increment.

FIGURE 6-1: TIMER1 INCREMENTING EDGE



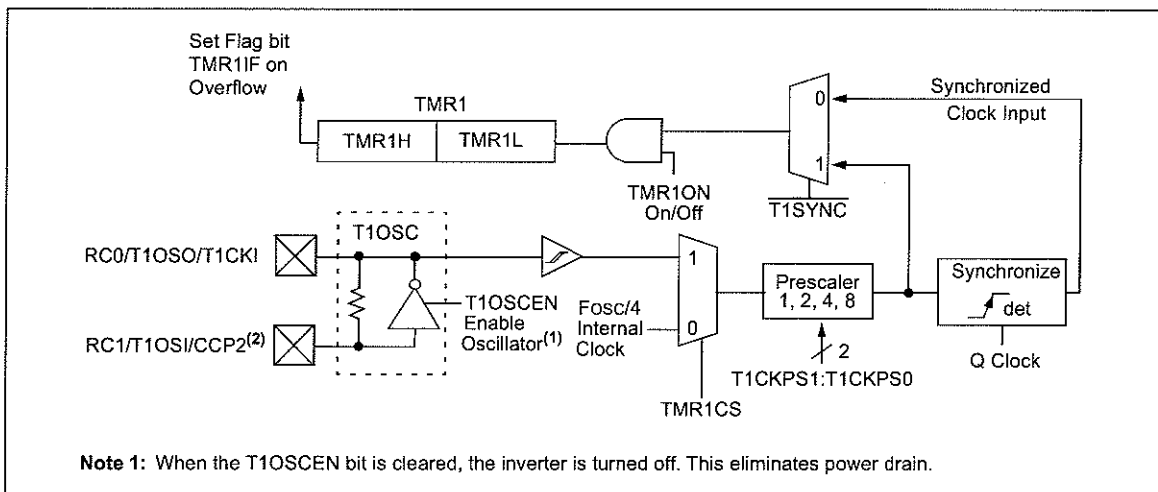
6.3 Timer1 Operation in Synchronized Counter Mode

Counter mode is selected by setting bit TMR1CS. In this mode, the timer increments on every rising edge of clock input on pin RC1/T1OSI/CCP2, when bit T1OSCEN is set, or on pin RC0/T1OSO/T1CKI, when bit T1OSCEN is cleared.

If T1SYNC is cleared, then the external clock input is synchronized with internal phase clocks. The synchronization is done after the prescaler stage. The prescaler stage is an asynchronous ripple-counter.

In this configuration, during SLEEP mode, Timer1 will not increment even if the external clock is present, since the synchronization circuit is shut-off. The prescaler, however, will continue to increment.

FIGURE 6-2: TIMER1 BLOCK DIAGRAM



6.4 Timer1 Operation in Asynchronous Counter Mode

If control bit $\overline{T1SYNC}$ ($T1CON<2>$) is set, the external clock input is not synchronized. The timer continues to increment asynchronous to the internal phase clocks. The timer will continue to run during SLEEP and can generate an interrupt-on-overflow, which will wake-up the processor. However, special precautions in software are needed to read/write the timer (Section 6.4.1).

In Asynchronous Counter mode, Timer1 cannot be used as a time-base for capture or compare operations.

6.4.1 READING AND WRITING TIMER1 IN ASYNCHRONOUS COUNTER MODE

Reading TMR1H or TMR1L while the timer is running from an external asynchronous clock, will guarantee a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values itself, poses certain problems, since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers, while the register is incrementing. This may produce an unpredictable value in the timer register.

Reading the 16-bit value requires some care. Examples 12-2 and 12-3 in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023) show how to read and write Timer1 when it is running in Asynchronous mode.

6.5 Timer1 Oscillator

A crystal oscillator circuit is built-in between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting control bit T1OSCEN ($T1CON<3>$). The oscillator is a low power oscillator, rated up to 200 kHz. It will continue to run during SLEEP. It is primarily intended for use with a 32 kHz crystal. Table 6-1 shows the capacitor selection for the Timer1 oscillator.

The Timer1 oscillator is identical to the LP oscillator. The user must provide a software time delay to ensure proper oscillator start-up.

TABLE 6-1: CAPACITOR SELECTION FOR THE TIMER1 OSCILLATOR

Osc Type	Freq.	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF

These values are for design guidance only.

Crystals Tested:

32.768 kHz	Epson C-001R32.768K-A	± 20 PPM
100 kHz	Epson C-2 100.00 KC-P	± 20 PPM
200 kHz	STD XTL 200.000 kHz	± 20 PPM

Note 1: Higher capacitance increases the stability of oscillator, but also increases the start-up time.

2: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

6.6 Resetting Timer1 using a CCP Trigger Output

If the CCP1 or CCP2 module is configured in Compare mode to generate a "special event trigger" ($CCP1M3:CCP1M0 = 1011$), this signal will reset Timer1.

Note: The special event triggers from the CCP1 and CCP2 modules will not set interrupt flag bit TMR1IF ($PIR1<0>$).

Timer1 must be configured for either Timer or Synchronized Counter mode to take advantage of this feature. If Timer1 is running in Asynchronous Counter mode, this RESET operation may not work.

In the event that a write to Timer1 coincides with a special event trigger from CCP1 or CCP2, the write will take precedence.

In this mode of operation, the CCPRxH:CCPRxL register pair effectively becomes the period register for Timer1.

PIC16F87X

6.7 Resetting of Timer1 Register Pair (TMR1H, TMR1L)

TMR1H and TMR1L registers are not reset to 00h on a POR, or any other RESET, except by the CCP1 and CCP2 special event triggers.

T1CON register is reset to 00h on a Power-on Reset, or a Brown-out Reset, which shuts off the timer and leaves a 1:1 prescale. In all other RESETS, the register is unaffected.

6.8 Timer1 Prescaler

The prescaler counter is cleared on writes to the TMR1H or TMR1L registers.

TABLE 6-2: REGISTERS ASSOCIATED WITH TIMER1 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	TOIE	INTE	RBIE	TOIF	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
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

Note 1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.

7.0 TIMER2 MODULE

Timer2 is an 8-bit timer with a prescaler and a postscaler. It can be used as the PWM time-base for the PWM mode of the CCP module(s). The TMR2 register is readable and writable, and is cleared on any device RESET.

The input clock ($F_{osc}/4$) has a prescale option of 1:1, 1:4, or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>).

The Timer2 module has an 8-bit period register, PR2. Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register. The PR2 register is initialized to FFh upon RESET.

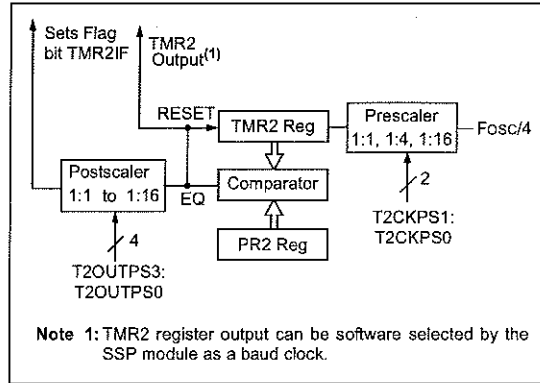
The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a TMR2 interrupt (latched in flag bit TMR2IF, (PIR1<1>)).

Timer2 can be shut-off by clearing control bit TMR2ON (T2CON<2>), to minimize power consumption.

Register 7-1 shows the Timer2 control register.

Additional information on timer modules is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

FIGURE 7-1: TIMER2 BLOCK DIAGRAM



REGISTER 7-1: T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

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

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

PIC16F87X

7.1 Timer2 Prescaler and Postscaler

The prescaler and postscaler counters are cleared when any of the following occurs:

- a write to the TMR2 register
- a write to the T2CON register
- any device RESET (POR, MCLR Reset, WDT Reset, or BOR)

TMR2 is not cleared when T2CON is written.

7.2 Output of TMR2

The output of TMR2 (before the postscaler) is fed to the SSP module, which optionally uses it to generate shift clock.

TABLE 7-1: 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	TOIE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSP1F ⁽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 the PIC16F873/876; always maintain these bits clear.

**UNIVERSITY OF WALES:
TRINITY SAINT DAVID**



**PRIFYSGOL CYMRU:
Y DRINDOD DEWI SANT**

LOCATION: COLEG SIR GAR

LLEOLIAD: COLEG SIR GAR

LEVEL 5

LEFEL 5

**EXAMINATION PERIOD
MAY 2018**

**CYFNOD ARHOLIADAU
MAI 2018**

MODULE CODE: PSDE5013
Configuration and programming of Embedded Systems

TIME: 2 HOURS

ANSWER THREE QUESTIONS

Equipment or materials required – Calculators are permitted

Adnoddau neu ddefnyddiau sydd eu hangen – Caniateir cyfrifiannell

Note: The appendix contains selected datasheet pages.

Main paper solutions

Question 1

(a)

[8]

Internal clk = 10 MHz

$F_{osc} = 10\text{MHz} / 4 = 2500000\text{Hz}$ per instruction

Time period = $1 / \text{instruction cycle time} = 1 / 2500000 = 0.4\mu\text{s}$

Timer increment = $0.4\mu\text{s} * 256 [\text{prescaler}] = 102.4\mu\text{s}$

For 10ms = $10\text{ms} / 102.4\mu\text{s} = 98$

$98 - 2$ for errors = 96.

Preload counter TMR0 = $256 - 96 = 160$

TMR0 = 160; //in code for 10ms delay

Configuration:

INTCON = 10100000

Bit 1 and 2 – unused

Bit 3 – clear TOIF

Bit 4 - unused

Bit 6 – disable peripheral interrupts

Bit 8 – global interrupt enable

OPTION_REG = 00000111

Bit 1 – 3 – prescaler 1:256 maximum

Bit 4 – use prescaler enabled

Bit 5 – not used

Bit 6 – TOSC select internal clock

Bit 7 and 8 – not used

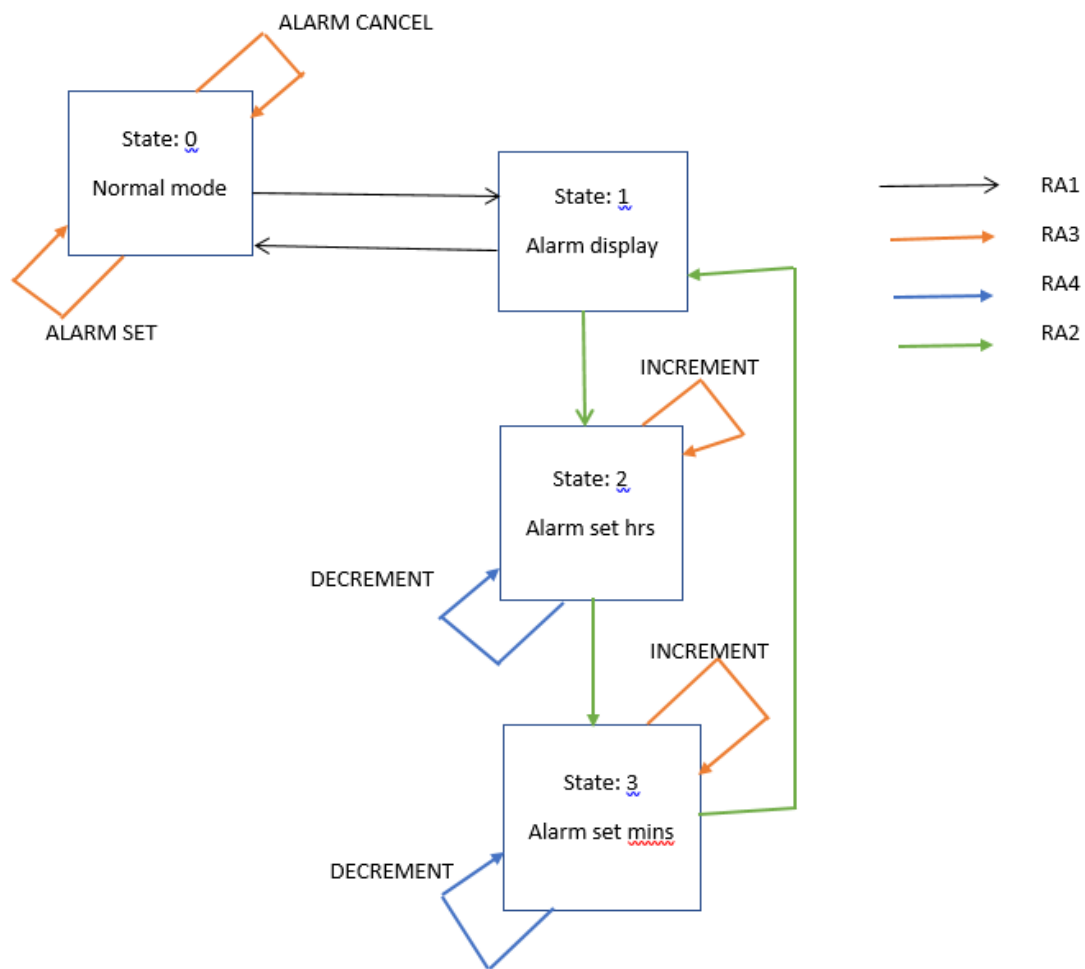
(b)

[6]

```
Void interrupt ISR (void){           //no return value from function
if (TOIF == 1){                     //check if TMR0 has interrupted
    TOIF = 0;                       //reset interrupt
    TMR0 = 160;                     //preload register for 96 counts
    milli ++;                       //increment milliseconds
    if (milli == 1000){              //check for 1 second
        milli = 0;                  //initialise milli
        secs++;                      //increment seconds
        if (secs == 60){             //check for 1 minute
            secs = 0;                //initialise seconds
            mins++;                   //increment minutes
            if (mins == 60){          //check for 1 hour
                mins = 0;             //initialise minutes
                hrs++;                //increment hours
                if (hrs == 24){       //check for 1 day
                    hrs = 0;          //initialise hours
                }//end if hrs
            }//end if mins
        }//end if secs
    }//end if milli
} //end if TOIF
} //end ISR
```

(c)

[8]



(d)

[12]

```
Void main (void){                                     //start of main program
TRISA = 0B00011111;                                  //output for alarm and inputs for switches
TRISB = 0b00000000;                                  //LCD output
INTCON = 0b10100000;                                  //set up interrupt
OPTION_REG = 0b00000111;                              //set up prescaler 1:256
TMR0 = 160;                                           //preload TMR0 with 160 and count for 96
LCD_Startup();                                       //power up LCD and initialise

While (1){                                           //start super loop
Case 0: DisplayTime (0);                             //display normal time
    if (RA1) state = 1;                               //go to state 1
    if (RA3 == 1){
        if (alarm == 0){                             //if alarm off
            alarm = 1;                                 //set alarm
        }
        else{
            alarm = 0;                                 //alarm cancel
        }
    }
    break;

Case 1: DisplayAlarmTime (0);                         //alarm time line 0
    if (RA1) state = 0;                               //normal mode
    if (RA2) state = 2;                               //go to state 2
    break;

Case 2: DisplaySetAlarmTime (0); //Display set alarm time on line 0
    if (RA1) state = 1;                               //check alarm time
    if (RA2) state = 3;                               //go to state 3
    if (RA3){
        alarm_hrs ++;                                //increment alarm hours
    }
    if (RA4){
        alarm_hrs --;                                //decrement alarm hours
    }
    break;

Case 3: DisplaySetAlarmTime (0); //Display set alarm time on line 0
    if (RA1) state = 1;                               //check alar time
    if (RA2) state = 0;                               //normal mode
    if (RA3){
        alarm_mins ++;                                //increment alarm mins
    }
    if (RA4){
        alarm_mins --;                                //decrement alarm mins
    }
    break;
} //end while
} //end main
```

[34 marks]

Question 2

(a)

[9]

AN0 = RA0

AN1 = RA1

AN6 = RE1

AN7 = RE2

TRISA = 00000011

//RA0 and RA1 set as inputs

TRISE = 00000110

//RE1 and RE2 set as inputs

ADCON0 = 10000001

//Fosc/32 set, chan 0 selected and ADC enabled

ADCON1 = 10001000

//Bit 7 right justified, 6 channels enabled

//RA2 and RA3 VREF –ve and +ve

ADCON0 channel select bits are 5:3 CHS2, CHS1, CHS0

Select channel AN0 – 000

AN1 – 001

AN6 – 110

AN7 – 111

(b)

[6]

Total number of raw values to be stored within LUT = $((101 - 50) + 1) = 52$ elements

The look up table LUT enables the values to be manipulated or set against the raw values from the ADC. This means that the device can be made to look more linear if needed.

This is because more than one raw value can represent a unit value.

(c)

[6]

int LUT [52] = {0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7,8,8,8,8,8,8,9,9,9,9,10,10,10,10, 11,11,11,12};

(d)

[13]

```
Void main (void){
int index, temp, ADC_Value;
TRISA = 0b00000011;
TRISE = 0b00000110;
ADCON0 = 0b10000001;
ADCON1 = 0b10001000;

LCD_Startup ();
while (1){
    GODONE = 1;                                //check ADC data
    while (GODONE ==1);
    Delay (100);
    ADC_Value = (int) ADRESL + (int) ADRESH <<8; //2 x 8 bit regs configured
                                                //for 10 bits
                                                //shift 8 places for 10 bit
                                                //resolution
    Index = ADC_Value – 50;                     //take 50 from initial
                                                //value for 0 marker
    temp = LUT [index];                         //temp = element value

    Display_LCD_Value (1, temp);                //display value on line 1
} //end while
} //end main
```

[34 marks]

Question 3

(a)

[14]

INTCON1 = 0b11100000 – Bit 8 is global interrupt enable GIE
Bit 7 is PEIE peripheral Int enable where 1 = all enabled
Bit 6 is TOIE – timer 0 overflow Int bit
Bit 2 is TOIF – timer 0 overflow Int flag
1 – TMR0 has overflowed
0 – TMR0 did not overflow

PEIE = 0b00000011 – bit 0 timer 1 enabled
Bit 1 timer 2 enable (not used here)

OPTION_REG = 0b10000111 – bit 2:0 prescaler bits for TMR0 rate
111 is set for 1:256

T1CON = 0b00000111 - this sets up the TMR1 in event mode with TMR0 Int
Bit 2 – do not sync with internal clk
Bit 1 – use external clk from RC0 pin on rising edge
Bit 0 – enable timer 1

The Int is set up for 1 second by configuring TMR0 for 10ms by pre-loading the timer with a pre-calculated value. This value ensures the overflow occurs every 10ms.
This is extended using C code to make a 1 second Int.

```
milliseconds += 10;  
if (milliseconds == 1000)    - 100 x 10ms = 1 sec
```

The number of pulses at RC0 input are counted using the event of 1s on TMR0. The value is stored 2 x 8 bit registers TMR1L and TMR1H. These 2 registers make up a 16 bit register by using a shift right configuration.

The value is then saved to a variable.
A calculation is created in the C program to convert the number of pulses per second i.e. frequency to RPM for motor speed.

```
Speed = (int) ((TMR1L + TMR1H * 256) * 60) / 600;
```

Where the conversion = frequency * 60 (mins) / 600(ppr)

This can be tested using frequency inputs from a generator at RC0 on MCU and using variable frequencies.

i/p HZ	600 ppr to RPM
100	10
500	50
1000	100
5000	500
10000	1000
20000	2000

(b)

[12]

```

Void Interrupt ISR (void){
    if (TOIF == 1){
        TOIF = 0;
        TMR0 160;                //Int every 10ms
        milliseconds += 10;
        if (milliseconds == 100){ //extend interrupt to 100ms

            speed_low_byte = TMR1L;
            speed_high_byet = TMR1H;
            TMR1L = 0;
            TMR1H = 0;
            milliseconds = 0;
        }
    }
    if (TMR1F == 1){             //Timer 1 flag is set
        TMR1F = 0;               //clear flag
    }
} //end ISR

Void main (void){
    TRISA = 0xff;                //all inputs
    TRISB = 0x00;                //all outputs for LCD
    TRISC = 0x01;                //input RC0 used for count on TMR1

    LCD_Startup ();

    PIE1 = 0b00000011;          //peripheral interrupts enable for TMR1 and
                                //TMR2
    INTCON1 = 0b11100000;        //setup TMR0
    OPTIONS_REG = 0b10000111;
    T1CON = 0b00000111;          //set up timer 1in event mode from RC0

    While (1){
        Speed = (int) ((speed_low_byte + speed_high_byte + 256) * 6) / 60);
        DisplaySpeed (0, speed);
    } //end while
} //end main

```

(c)

[8]

```
Void Interrupt ISR (void){
    if (TMR2IF == 1){
        TMR2IF = 0;                //clear Int flag of timer 2for PWM
    }
} //end ISR

Void main (void){
    TRISC = 0x01;                  //output RC2 used for PWM with TMR2          PR2 = 255;
                                   //max for PWM time period register
    CCPR1L = 0;                    //0% duty cycle initialise
    T2CON = 0b00000111;           //set up TMR2 in PWM mode
                                   //TMR2 enable and prescale = 16
    CCP1CON = 0b00001100;         ///set the extra 2 LSB bits for PWM 10 bit mode

    While (1){
        CCPR1L = (1024 / 100) * 75; //75% duty cyclePWM
    } //end while
} //end main
```

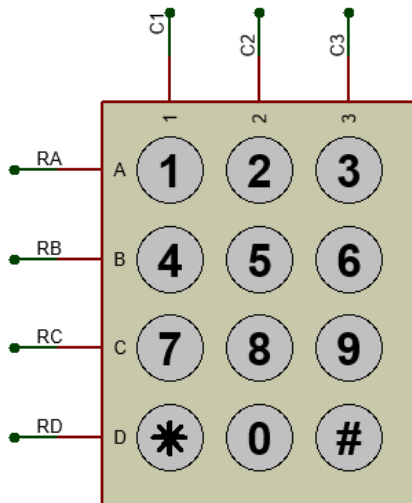
[34 marks]

Question 4

(a)

[14]

The keypad encoder show below is connected to the microcontroller using port D. C1 to C3 are the columns and are connected to RD0 to RD2. The rows RA to RD are connected to RD4 to RD7. This encoder uses 7 of the 8 bits of the port.



The columns are energised in turn and the program scans the rows for a key press. When a key is pressed the junction with the corresponding column goes high. The identity of the key pressed is recognised by the program and the function returns the value as an integer.

The C code below shows a function prototype that performs this process by energising the columns and scanning each column in turn. The delay provides stability. As can be seen the 3 columns RD0 to RD3 are energised and turned of sequentially as part of the scanning process. The rows are detected by the program during the scan.

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

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

    RD1 = 1;                                     //scan column 2
    Delay(1);
    if(RD4 == 1)    key = 2;                     //wait for key = 2
    if(RD5 == 1)    key = 5;                     //wait for key = 5
```

```

if(RD6 == 1)    key = 8;           //wait for key = 8
if(RD7 == 1)    key = 0;           //wait for 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 = '#';        //wait for key = # which is used for reset

RD2 = 0;        //disable column 3

return key;     //return the value of the key pressed to
                //the main program function call
}

```

(b)

[12]

```
#include <pic.h>

int ScanKeypad (void);
int key_value=0;                                //0 for first key press for position 1 which is units

void main(void)
{
    int enable_keypad=1, key;

    TRISD = 0b11110000;                        //keypad rows input and columns output on
                                                //PORT D

    while(1)                                    //super loop
    {
        key = ScanKeypad();                    //scan keypad function call

        if((key!=-1) && (enable_keypad==1))      //cannot use <0 and >0 because we
                                                // need zero
                                                //therefore we have to use != -1 and == -1
                                                //enable_keypad stops the debounce count
        effect
                                                //and latches keypress

        {
            Key_value = key_value * 10 + key;    //key entries move over 1 space (x10) each key
                                                // press for 10's, 100's, 1000's and key_value is 0 at first press for
            units

            enable_keypad=0;                    //latch key press to distinguish from the next press
        }

        if(key=='#')                          //# key will reset the value to 0
        {
            Key_value = 0;
        }

        if(key==-1)    enable_keypad=1;        //keeps the keypad active
                                                // and resets the latch
                                                // enable for next press

        if(key_value>2000)    key_value = 2000;    //key_value = 2000 if
                                                // greater than 2000

        if(key_value<0)    key_value = 0;        //key_value < 0 then key_value = 0

    }//end while
}
```

(c)

[8]

```
void DisplayValues (int line, int value)           //function start
{
    int units, tens, hunds, thous;                //local variable declaration
    unsigned char string[4];                       //4 element array declared locally

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

    string[0] = thous + 48;                         //store the digit values in the
    string[1] = hunds + 48;                         //array elements 0 – 3 followed
    string[2] = tens + 48;                          //by asci 0 to end each line
    string[3] = units + 48;

    DisplayString(line, string); //send string data to function in LCD.main to print
                                //on LCD line 0 or 1
} //end DisplayValue function
```

[34 marks]