UNIT - II

3 PIC I/O Ports and Timer

Syllabus

I/O Port : *I/O Port structure with programming: I/O Port structure, I/O Port programming, I/O Bit manipulation Programming.*

Timer/Counter : *Registers used for Timer/Counter operation, Delay calculations, Programming of Timers using Embedded C.*

Contents

3.1 I/O Port Structure with Programming

SPPU : Dec.-14, May-16

- In the PIC18 family, the number of I/O ports supported by a particular microcontroller varies depending on the family member and the number of I/O pins the device has.
- Table 3.1.1 shows Ports in PIC18 family members.

Pins	18/20 pins	28 - pin	40 - pin	64/68 - pin	80/84 - pin
Chip	PIC18F1220	PIC18F2220	PIC18F458	PIC18F6525	PIC18F8525
Port A	✓	✓	✓	✓	✓
Port B	✓	✓	✓	✓	✓
Port C		✓	✓	✓	✓
Port D			✓	✓	✓
Port E			✓	✓	✓
Port F				✓	✓
Port G				✓	✓
Port H					✓
Port J					✓

Table 3.1.1 Ports in PIC18 family members

- The PIC18F458 has five I/O ports :
  - **Port A** (7 - Bit) : RA0 - RA6
  - **Port B** (8 - Bit) : RB0 - RB7
  - **Port C** (8 - Bit) : RC0 - RC7
  - **Port D** (8 - Bit) : RD0 - RD7
  - **Port E** (3 - Bit) : RE0 - RE2
- Some pins of the I/O ports are multiplexed with an alternate function from the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.
- Each port has three Special function registers (SFRs) associated with it :
  - TRIS register (Data Direction register)
  - PORT register (reads the levels on the pins of the device)
  - LAT register (output latch)

- The data latch (LAT register) is useful for read-modify write operations on the value that the I/O pins are driving.

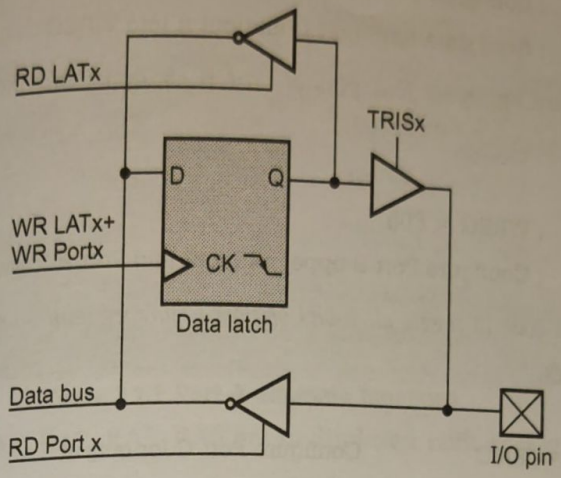

Fig. 3.1.1 Simplified block Diagram of Port/LAT/TRIS operation

- Each of the Ports A-E in the PIC18F458 can be used for input or output. These ports are bi-directional. The data direction register is TRISx (x stands for A - E) is used to set the direction either input or output.
- It is important to note that Data direction needs to be set before the I/O operation.
- Setting a TRISx bit (= 1) will make the corresponding PORTx pin an input (i.e., put the corresponding output driver in a high-impedance mode).
- Clearing a TRISx bit (= 0) will make the corresponding PORTx pin an output (i.e., put the contents of the output latch on the selected pin).
- Reading the PORTx register reads the status of the pins, whereas writing to it will write to the port latch.
- The Data Latch register (LATx) is also memory mapped. Read-modify-write operations on the LATx register, read and write the latched output value for PORTx.

Example 3.1.1 *Write an instruction sequence to output the hex value 0x35 to port B.*

Solution : The Port B should be configured for output before data is written to it.

```
CLRF    TRISB   ; Configure Port B for output
MOVLW   0X35    ; WREG = 35h
MOVWF   PORTB   ; Send 35h to Port B
```

Example 3.1.2 *Write an instruction sequence to read the current value of port D into WREG.*

Solution :

```
SETF TRISD        ; Configure port D for input
MOVF PORTD, W     ; Read data form port D and put it into WREG
```

Example 3.1.3 *Configure the upper four pins of Port B for input and the lower four pins for output.*

Solution :

```
MOVLW 0XF0        ; WREG = F0h
MOVWF TRISB       ; Configure Port B upper as input and Port B lower as output
```

Example 3.1.4 *Write a code to toggle all 8-bits of Port C forever with some time delay between ON and OFF states.*

Solution :

```
        CLRF   TRISC      ; Configure Port C for output
BACK :  MOVLW  0X55       ; WREG = 55h
        MOVWF  PORTC      ; Send 55h to Port C
        CALL   DELAY      ; Wait for some time
        MOVLW  0XAA       ; WREG = AAh
        MOVWF  PORTC      ; Send AAh to Port C
        CALL   DELAY      ; Wait for some time
        GOTO   BACK       ; Do forever
```

Example 3.1.5 *Write a code to toggle all 8-bits of Port B forever with some time delay between ON and OFF states using read-modify-write.*

Solution :

```
        CLRF   TRISB      ; Configure Port B for output
        MOVLW  0X55       ; WREG = 55h
        MOVWF  PORTB      ; Send 55h to Port B
BACK :  COMF   PORTB, F   ; Complement bits of Port B
        CALL   DELAY      ; Wait for some time
        GOTO   BACK       ; Do forever
```

Note Upon reset, TRIS registers of all ports are loaded with value FFH, i.e. 11111111_2. As a result all ports act as input ports upon reset.

3.1.1 Port A

- PORTA is a 7-bit wide (RA0-RA6), bidirectional port. The corresponding Data Direction register is TRISA and Latch register is LATA.

- Table 3.1.2 shows Port A alternate functions.

Bit	Alternate Function
RA0	AN0/CVREF
RA1	AN1
RA2	AN2/VREF-
RA3	AN3/VREF+
RA4	T0CK1
RA5	AN4/SS/LVDIN
RA6	OSC2/CLKO

Table 3.1.2 Port A alternate functions

- The pins RA0, RA1, RA2, RA3, RA5 are multiplexed with analog inputs.
- The pins RA0, RA2 and RA3 are also multiplexed with the analog CVREF, VREF+ and VREF- inputs, respectively.
- The RA4 pin is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open-drain output.
- The RA5 pin is also multiplexed slave select input (SS) and low-voltage detect input (LVDIN)
- The RA6 pin is only enabled as a general I/O pin in ECIO and RCIO Oscillator modes.
- The operation of each pin is selected by clearing/setting the control bits in the ADCON1 register (A/D Control Register 1).
- All other RA port pins have TTL input levels and full CMOS output drivers.
- On a Power-on Reset, RA5 and RA3 : RA0 are configured as **analog inputs** and read as '0'. RA6 and RA4 are configured as **digital inputs**.

3.1.2 Port B

- PORTB is a 8-bit wide (RB0-RB7), bidirectional port. The corresponding Data Direction register is TRISB and Latch register is LATB.
- Read-modify-write operations on the LATB register, read and write the latched output value for PORTB.

Initializing Port B

```
CLRF PORTB      ; Initialize PORTB by clearing output data latches
CLRF LATB       ; Alternate method to clear output data latches
```

Example 3.1.6 *Write an instruction sequence to set RB3:RB0 as outputs, RB5:RB4 as inputs and RB7:RB6 as outputs.*

Solution :

```
MOVLW 30h       ; Value (00110000 in binary) used to initialize data direction
MOVWF TRISC     ; Set RB3:RB0 as outputs, RB5:RB4 as inputs and RB7:RB6 as outputs.
```

- Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit $\overline{\text{RBPU}}$ (INTCON2 register). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.
- Table 3.1.3 shows Port B alternate functions.

Bit	Alternate Function
RB0	INT0
RB1	INT 1
RB2	INT 2/CANTX
RB3	CANRX
RB4	
RB5	PGM
RB6	PGC
RB7	PGD

Table 3.1.3 Port B alternate functions

- The pins RB0, RB1 and RB2 are multiplexed with the external interrupt pins INT0, INT1 and INT2, respectively.
- The pin RB2 is also multiplexed with Transmit signal for CAN bus.
- The pin RB3 is multiplexed with Receive signal for CAN bus.
- Four of the PORTB pins (RB7 : RB4) have an interrupt on-change feature. This "interrupt on change" is triggered when any of the RB7 : RB4 pins, configured as an input, changes level.
- The input pins (of RB7 : RB4) are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7 : RB4 are ORed together to generate the RB Port Change Interrupt with Flag bit RBIF (INTCON register). This interrupt can wake the device from Sleep.

3.1.3 Port C

- PORTC is an 8-bit wide, bidirectional port.
- The corresponding Data Direction register is TRISC and Latch register is LATC.
- Read-modify-write operations on the LATC register, read and write the latched output value for PORTC.
- Table 3.1.4 shows Port C alternate functions.

Bit	Alternate function	Description
RC0	T1OSO/T1CKI	Input/output port pin, Timer1 oscillator output or Timer1/Timer3 clock input.
RC1	T1OSI	Input/output port pin or Timer1 oscillator input.
RC2	CCP1	Input/output port pin or Capture 1 input/Compare 1 output/PWM1 output.
RC3	SCK/SCL	Input/output port pin or synchronous serial clock for SPI™/I^2C™.
RC4	SDI/SDA	Input/output port pin or SPI data in (SPI mode) or data I/O (I^2C mode).
RC5	SDO	Input/output port pin or synchronous serial port data output.
RC6	TX/CK	Input/output port pin, addressable USART asynchronous transmit or addressable USART synchronous clock.
RC7	RX/DT	Input/output port pin, addressable USART asynchronous receive or addressable USART synchronous data.

Table 3.1.4 Port C alternate functions

- PORTC uses Schmitt Trigger input buffers

Initializing Port C

```
CLRF PORTC      ; Initialize PORTC by clearing output data latches
CLRF LATC       ; Alternate method to clear output data latches
```

Example 3.1.7 *Write an instruction sequence to set RC3 : RC0 as inputs RC5 : RC4 as outputs RC7 : RC6 as inputs.*

Solution :

```
MOVLW 0CFh      ; Value (11001111 in binary) used to initialize data direction
MOVWF TRISC     ; Set RC3:RC0 as inputs, RC5:RC4 as outputs and RC7:RC6 as inputs
```

3.1.4 Port D

- PORTD is an 8-bit wide, bidirectional port. The corresponding Data Direction register is TRISD and Latch register is LATD.
- Read-modify-write operations on the LATD register, read and write the latched output value for PORTD.
- Table 3.1.5 shows Port D alternate functions.

Bit	Alternate function	Description
RD0	PSP0/C1IN+	Input/output port pin, Parallel Slave Port bit 0 or C1IN+ comparator input.
RD1	PSP1/C1IN-	Input/output port pin, Parallel Slave Port bit 1 or C1IN- comparator input.
RD2	PSP2/C2IN+	Input/output port pin, Parallel Slave Port bit 2 or C2IN+ comparator input.
RD3	PSP3/C2IN-	Input/output port pin, Parallel Slave Port bit 3 or C2IN- comparator input.
RD4	PSP4/ECCP1/P1A	Input/output port pin, Parallel Slave Port bit 4 or ECCP1/P1A pin.
RD5	PSP5/P1B	Input/output port pin, Parallel Slave Port bit 5 or P1B pin.
RD6	PSP6/P1C	Input/output port pin, Parallel Slave Port bit 6 or P1C pin.
RD7	PSP7/P1D	Input/output port pin, Parallel Slave Port bit 7 or P1D pin.

Table 3.1.5 Port D alternate functions

- PORTD uses Schmitt Trigger input buffers.

Initializing Port D

```
CLRF PORTD      ; Initialize PORTD by clearing output data latches
CLRF LATD       ; Alternate method to clear output data latches
MOVLW 07h       ; Comparator off
MOVWF CMCON     ; Comparator Control Register bits (CM2:CM0) = 111 to make them off.
```

Example 3.1.8 *Write an instruction sequence to set RD3 : RD0 as inputs and RD7 : RD4 as outputs.*

Solution :

```
MOVLW 0Fh       ; Value (00001111 in binary) used to initialize data direction
MOVWF TRISD     ; Set RD3 : RD0 as inputs and RD7 : RD4 as outputs.
```

3.1.5 Port E

- PORTD is an 3-bit wide, bidirectional port. The corresponding Data Direction register is TRISE and Latch register is LATE.
- Read-modify-write operations on the LATE register, read and write the latched output value for PORTE.
- Table 3.1.6 shows Port E alternate functions.

Bit	Alternate function	Description
RE0	AN5/RD	Input/output port pin, analog input or read control input in Parallel Slave Port mode.
RE1	AN6/WR/C1OUT	Input/output port pin, analog input, write control input in Parallel Slave Port mode or Comparator 1 output.
RE2	AN7/CS/C2OUT	Input/output port pin, analog input, chip select control input in Parallel Slave Port mode or Comparator 2 output.

Table 3.1.6 Port E alternate functions

- PORTE uses Schmitt Trigger input buffers.

Initializing Port D

```
CLRF PORTE      ; Initialize PORTE by clearing output data latches
CLRF LATE       ; Alternate method to clear output data latches
```

Example 3.1.9 *Write an instruction sequence to set RE1 : RE0 as inputs and RE2 as output.*

Solution :

```
MOVLW 03h       ; Value (00000011 in binary) used to initialize data direction
MOVWF TRISE     ; Set RE1 : RE0 as inputs and RE2 as output.
```

- The TRISE register also controls the operation of the Parallel Slave Port through the control bits in the upper half of the register, as shown in the Fig. 3.1.2.

IBF	OBF	IBOV	PSPMODE	–	TRISE2	TRISE1	TRISE0

bit 7 bit 0

bit 7 **IBF** : Input Buffer Full status bit

1 = A word has been received and waiting to be read by the CPU

0 = No word has been received

bit 6 **OBF** : Input Buffer Full status bit

1 = The output buffer still holds a previously written word

0 = The output buffer has been read

bit 5 **IBOV** : Input Buffer Overflow Detect bit (in Microprocessor mode)

1 = A write occurred when a previously input word has not been read (must be cleared in software)

0 = No overflow occurred

bit 4 **PSPMODE** : Parallel Slave Port Mode Select bit

1 = Parallel Slave Port mode

0 = General Purpose I/O mode

bit 3 **Uniplemented** : Read as '0'

bit 2 **TRISE2** : RE2 Direction Control bit

1 = Input

0 = Output

bit 1 **TRISE1** : RE1 Direction Control bit

1 = Input

0 = Output

bit 0 **TRISE0** : RE0 Direction Control bit

1 = Input

0 = Output

Fig. 3.1.2 Bit pattern of TRISE register

- When the Parallel Slave Port is active, the PORTE pins function as its control inputs.

Review Questions

1. *Explain all ports (A-E) of PIC18FXXX.* **SPPU : Dec.-14, Marks 8**
2. *Explain functions of Ports in PIC18FXXX.* **SPPU : May-16, Marks 8**
3. *Explain the alternate functions of Port A in PIC18FXXX.*
4. *Explain the alternate functions of Port B and Port C in PIC18FXXX.*
5. *Explain the alternate functions of Port D and Port E in PIC18FXXX.*
6. *Explain the bit pattern of TRISE register.*

3.2 I/O Bit Manipulation Programming

- Many times it is necessary to access only 1 or 2 bits of the port instead of all 8-bits.

- The PIC18 I/O ports allow to access individual bits of the ports without altering the rest of the bits in that port.
- Table 3.2.1 gives the single bit instructions for the PIC18.

Instruction		Function
BCF	f, b, a	Bit clear f
BSF	f, b, a	Bit Set f
BTFSC	f, b, a	Bit Test f, Skip if Clears
BTFSS	f, b, a	Bit Test f, Skit Set
BTG	f, d, a	Bit Toggle f

Table 3.2.1 Single bit instructions for the PIC18

- Bit operations allow us to move, set and clear single bits in registers or numbers that we specify.

BCF

- This instruction will clear a bit that we specify in a given File register. The syntax is :

```
BCF   <register>, <bit_num>
```

- **Example :**

```
BCF   TRISC, 0 ; Clear bit 0 of TRISC register
```

BSF

- This instruction will set a bit that we specify in a given File register. The syntax is :

```
BSF   <register>, <bit_num>
```

- **Example :**

```
BSF   PORTA, 4      ; Set bit 4 of Port A
```

BTFSC

- This instruction will test the bit that we specify in a given File register. If the bit is a 0, the instruction will tell the PIC to skip the next instruction. The syntax is :

```
BTFSC   <register>, <bit_num>
```

- **Example 1 :**

```
BTFSC PORTB, 2 ; Check bit 2 of Port B, if it is zero code skips next instruction
```

- **Example 2 :**

```
LOOP :    BTFSC  PORTA, 3   ; Monitor PA3; repeats instructions in the LOOP until
                            PA3 is 0.
          BRA LOOP
```

BTFSS

- This instruction will test the bit that we specify in a given File register. If the bit is a 1, the instruction will tell the PIC to skip the next instruction. The syntax is :

  BTFSS <register>, <bit_num>

- **Example 1 :**

```
BTFSS  PORTC, 5 ; Check bit 5 of Port C, if it is 1 code skips next instruction
```

- **Example 2 :**

```
LOOP :   BTFSS  PORTB, 3   ; Monitor PB3; repeat instructions in the LOOP until
                           PB3 is 1.

         BRA LOOP
```

BTG

- This instruction will toggles a bit that we specify in a given File register. The syntax is:

  BTG <register>, <bit_num>

- **Example :** Code segment to generate square wave on pin PC2

```
        BCF     TRISC, 2     ; Clear bit 2 of TRISC register to make RC2 an output pin
BACK :  BTG     PORTC, 2     ; Toggle PC2
        CALL    DELAY        ; Wait for some time
        BRA     BACK         ; Repeat forever
```

Review Questions

1. *What is bit addressability ? State its advantage.*
2. *Explain any four bit addressable instructions of PIC18F.*

3.3 Programming Examples

Example 3.3.1 *Write the following programs. Create a square wave of 50% duty cycle on bit 0 of PORTA.*

Solution : The 50% duty cycle means that $T_{ON} = T_{OFF}$. Therefore, we toggle RA0 with a time delay in between each state.

Program 1 : Using BCF and BSF instructions

```
        BCF     TRISA, 0     ; Clear bit 0 of TRISA register to make RA0 an output
                               pin
AGAIN:  BSF     PORTA, 0     ; Set bit 0 of Port A
        CALL    DELAY        ; Wait for Some time
        BCF     PORTA, 0     ; Clear bit 0 of Port A
```

```
        CALL    DELAY       ; Wait for Some time
        BRA     AGAIN       ; Repeat forever
```

Program 2 : Using BTG instruction

```
        BCF     TRISA, 0    ; Clear bit 0 of TRISA register to make RA0 an output
                              pin
AGAIN:  BTG     PORTA, 0    ; Toggle bit 0 of Port A
        CALL    DELAY       ; Wait for Some time
        BRA     AGAIN       ; Repeat forever
```

Example 3.3.2 *Write the following programs. Create a square wave of 66% duty cycle on bit 2 of PORTC.*

Solution : The 66% duty cycle means that $2T_{ON} = T_{OFF}$.

```
        BCF     TRISC, 2    ; Clear bit 2 of TRISC register to make RC2 an output pin
AGAIN : BSF     PORTC, 2    ; Set bit 2 of Port C
        CALL    DELAY       ; Wait for some time twice for 66%
        CALL    DELAY
        BCF     PORTC, 2    ; Clear bit 2 of Port C
        CALL    DELAY       ; Wait for Some time
        BRA     AGAIN       ; Repeat forever
```

Example 3.3.3 *Write a program to perform the following :*
(a) Keep monitoring the RC3 bit until it becomes high
(b) When RC3 becomes high, write value 34H to Port B
(c) Send a high-to-low (H-to-L) pulse to RD2

Solution :

```
        BSF     TRISC, 3    ; Set bit 3 of TRISC register to make RC3 an input pin
        CLRF    TRISB       ; Make Port B an output port
        BCF     TRISD, 2    ; Clear bit 2 of TRISD register to make RD2 an output pin
        MOVLW   0x34        ; WREG = 34H
AGAIN:  BTFSS   PORTC, 3    ; Test RC3, if RC3 = 1, skip next instruction
        BRA     AGAIN       ; Repeat if RC3 = 0
        MOVWF   PORTB       ; Send contents of WREG (34H) to Port B
        BSF     PORTD, 2    ; Set bit 2 of Port D
        BCF     PORTD, 2    ; Clear bit 2 of Port D
```

Example 3.3.4 *Assume that bit RC3 is an input and represents the condition of an oven. If it goes low, it means that the food is ready. Monitor the bit RC3 continuously. Whenever it goes low, send a high-to-low pulse to port PB5 turn on a buzzer.*

Solution :

```
        BSF     TRISC, 3    ; Set bit 3 of TRISC register to make RC3 an input pin
        BCF     TRISB, 5    ; Clear bit 5 of TRISB register to make RB5 an output pin
AGAIN:  BTFSC   PORTC, 3    ; Test RC3, if RC3 = 0, skip next instruction
        BRA     AGAIN       ; Repeat if RC3 = 1
        BSF     PORTB, 2    ; Set bit 2 of Port B
        BCF     PORTB, 2    ; Clear bit 2 of Port B
        BRA     AGAIN       ; Repeat forever
```

Example 3.3.5 *Assume that switch (SW) is connected to pin RB1. Write a program to check the status of SW and perform the following :*
(a) If SW=0, send character 'N' to PORTC
(b) If SW=1, send character 'Y' to PORTC

Solution :

```
        BSF     TRISB, 1    ; Set bit 1 of TRISB register to make RB1 an input pin
        CLRF    TRISC       ; Make Port C an output port
AGAIN:  BTFSC   PORTB, 1    ; Test RB1, if RB1 = 0, skip next instruction
        BRA     NEXT        ; if RB1 = 1, go to NEXT
        MOVLW   A 'N'       ; Load ASCII of character N in WREG
        MOVWF   PORTC       ; Send it to Port C
        BRA     AGAIN       ; Repeat forever
NEXT :  MOVLW   A 'Y'       ; Load ASCII of character Y in WREG
        MOVWF   PORTC       ; Send it to Port C
        BRA     AGAIN       ; Repeat forever
```

Example 3.3.6 *Assume that a switch (SW) is connected to pin RA3 and an LED to pin RB6. Write a program to get the status of the SW and send it to the LED.*

Solution :

```
        BSF     TRISA, 3    ; Set bit 3 of TRISA register to make RA3 an input pin
        BCF     TRISB, 6    ; Clear bit 6 of TRISB register to make RB6 an output pin
AGAIN:  BTFSS   PORTA, 3    ; Test RA3, if RA3 = 1, skip next instruction
        BRA     NEXT        ; if RA3 = 0, go to NEXT
        BSF     PORTB, 6    ; Set RB6 to make LED ON
        BRA     AGAIN       ; Repeat forever
NEXT:   BCF     PORTB, 6    ; Clear RB6 to make LED OFF
        BRA     AGAIN       ; Repeat forever
```

3.4 Input Pin Vs. LATx Port

- Writing to PORTx or LATx has the same effect, but reading PORTx register read the corresponding input pin. Reading LATx, reads the status of the corresponding internal port latch and not the status of the pin.
- Some instructions read the contents of an internal port latch instead of reading the status of an extern pin.
- When such instructions are executed following sequence of actions take place -
  1. The instruction read the internal latch of the LATx and brings that data into the CPU.
  2. The data is processed as per instruction.
  3. The result of the operation is rewritten back to the LATx latch.
  4. The data on the pins are changed only if the port bits are configured as an output pins i.e. TRISx bits are cleared to 0.
- For example, COMF PORTC instruction will read PORTC, complement all the data bits, then write the result back to PORTC.

Review Question

1. *Explain input pin Vs. LATx Port of PIC18F.*

3.5 PIC Programming In C

SPPU : May-12

- C is a high level programming language that is portable across many hardware architectures. This means that architecture specific features such as register definitions, initialization and start up code must be made available to our program via the use of libraries and include files.
- For PIC18F458 microcontroller we need to include the file PIC18F458.h. This file contains all the definitions of PIC18F458 registers. With this information C compiler produces hex file that can be downloaded into the ROM of the microcontroller. It is important to note that the size of the hex file produced by the assembly language is much smaller than the hex file produced by the C compiler. Apart from this fact, there are many reasons for writing programs in C instead of assembly :

1. It is much easier and less time consuming to write programs in C than assembly.
2. C is more flexible; it is easier to modify and update.
3. Programming in C allows to use code available in function libraries.

4. Program written in C for one microcontroller is portable to other microcontrollers with little or no modifications.

3.5.1 Data Types in PIC18 C

- Table 3.5.1 lists the data types that are available in typical PIC18 compiler. Table 3.5.1 also gives the information about the size of the data variable and its value range.

Data type	Bits	Value range
bit	1	0 to 1
char	8	– 128 to + 127
unsigned char	8	0 to 255
short	16	– 32768 to + 32767
unsigned short	16	0 to 65535
int	16	– 32768 to + 32767
unsigned int	16	0 to 65535
short long	24	– 8388608 to 8388607
unsigned short long	24	0 to 16777215
long	32	– 2147483648 to 2147483647
unsigned long	32	0 to 4294967295

Table 3.5.1 Data types widely used by PIC18

3.5.2 PIC Programming Examples in C

Example 3.5.1 *Write PIC18 C program to send binary counter values from 00-FFH to port C.*

Solution :

```
#include < PIC18F458.h >
void main(void)
{
    unsigned char c;
    TRISC= 0;          // Make Port C as an output
    for(c=0; c<=255; c++)
    PORTC=c;
}
```

Example 3.5.2 *Write PIC18 C program to send hex values for ASCII characters of 0, 1, 2, 3, A, B, C, X, Y and Z to Port B.*

Solution :

```
#include < PIC18F458.h >
void main (void)
{
   unsigned char mychar[]="0123ABCXYZ";
   unsigned char c;
   TRISB= 0;              // Make Port B as an output
   for(c=0; c<10; c++)
      PORTB = mychar [c];

}
```

Comment : Program displays values 30H, 31H, 32H, 33H, 41H, 42H, 43H, 58H, 59H and 5AH on Port B.

Example 3.5.3 *Write an PIC18 C program to toggle all the bits of Port B and Port C continuously such that when Port B = FFH, Port C = 00 and vice-versa.*

Solution :

```
#include < PIC18F458.h >
void main(void)
{
   TRISB= 0;            // Make Port B as an output
   TRISC= 0;            // Make Port C as an output
   for(;;)          // repeat continuously
   {
      PORTB = 0x00 ;
      PORTC =0xFF ;
      PORTB = 0xFF ;
      PORTC =0x00 ;
   }
}
```

Example 3.5.4 *Write PIC18 C program to send the sum of values – 12 and 15 to Port B. (Use of signed numbers).*

Solution :

```
#include < PIC18F458.h >
void main(void)
{
```

```
    signed char i, j;
    TRISB= 0;      // Make Port B as an output
    i = -12;
    j = 15;
    PORTB = i + j;
}
```

Example 3.5.5 *Write PIC18 C program to toggle bit 0 of the Port B (RB0) 20,000 times.*

Solution :

```
#include < PIC18F458.h >
#define PORTBIT PORTBbits.RB0        // Single bit declaration
void main(void)
{
    unsigned int i;
    TRISBbits.TRISB0= 0;             // Make Port RB0 as an output
    for(i = 0; i<=20000; i++)
    {
    PORTBIT = 0;                     // Make RB0 = 0
    PORTBIT = 1;                     // Make RB0 = 1
    }
}
```

Example 3.5.6 *Write PIC18 C program to find number of positive and negative data among five byte of array. Send number of positive data to Port B and negative data to Port C.*

Solution :

```
# include < PIC18F458.h >
void main (void)
{
        unsigned char data [ ] = { 0x18, 0x 5A, 0x F0, 0x 80, 0x12 }
        unsigned char i, val ;
        TRISB= 0;          // Make Port B as an output
        TRISC= 0;          // Make Port C as an output
        for (i = 0; i<5; i++)
        {
            val = data [i];
            val = val & 0x80;
            if (val == 0)
```

```
            PORTB = data [i];
        else
            PORTC = data [i];
    }
}
```

Example 3.5.7 *Write PIC18 C program to monitor bit RC5. If it is LOW, send data 55H to Port B; otherwise send AAH to Port D.*

Solution :

```
# include < PIC18F458.h >
#define PORTBIT PORTCbits.RC5 // Single bit declaration
void main (void)
{
    TRISCbits.TRISC5= 1;   // Make Port RC5 as an input
    TRISB= 0;              // Make Port B as an output
    TRISD= 0;              // Make Port C as an output
    while(1)
    {
        if (PORTBIT == 0)
            PORTB = 0x55;
        else
            PORTD = 0xAA;
    }
}
```

Example 3.5.8 *Write PIC18 C program to get a byte data from Port C. If it is less than 1000, send it to Port B; otherwise send it to Port D.* **SPPU : May-12, Marks 8**

Solution :

```
# include < PIC18F458.h >
void main (void)
{
    Unsigned char i;
    TRISC= 0xFFH;     // Make Port C as an input
    TRISB= 0;         // Make Port B as an output
    TRISD= 0;         // Make Port C as an output
    while(1)
    {
        i = PORTC;    // Read data from Port C
        if (i < 1000)
```

```
                PORTB = i;
            else
                PORTD = i;
        }
}
```

Review Questions

1. *State the reasons for writing PIC18F458 program in C instead of assembly.*
2. *Comment on size of PIC18F458 C program.*

SPPU : Dec.-12, May-13, 14

3.6 Time Delays in PIC18 C

- We can create time delay in PIC18 C by two ways :
  - Using for loop statement
  - Using the PIC18 timers
- We cannot get the exact delays using simple for loops because of following reasons :
  1. The instruction execution speed varies according to the number of clock periods per machine cycle.

     The different variants of PIC18 microcontroller use different clock periods per machine cycle.
  2. The crystal frequency connected to the OSC1-OSC2 input pins. The duration of the clock period for the machine cycle is a function of this crystal frequency.
  3. In case of C programs, it is the C compiler that converts the C statements and functions to assembly language instructions. As a result, different compilers produce different code and hence instructions executed in a loop may vary with different compilers.

Example 3.6.1 *Write PIC18 C program to toggle all bits of Port B continuously with a 250 ms delay. Assume that PIC18F458 crystal frequency = 10 MHz.*

Solution :

```
# include < PIC18F458.h >
void DELAY (unsigned int);
void main (void)
{
    TRISB= 0;     // Make Port B as an output
    while(1)
```

```
    {
        PORTB = 0x55;
        DELAY (250);
        PORTB = 0xAA;
        DELAY (250);

    }
}
void DELAY (unsigned int t);
{
    unsigned int i, j;
    for (i = 0; i<t; i++)
        for (j = 0; j<165; j++);

}
```

Example 3.6.2 *Write PIC18 C program to blink LEDs connected to Port C of PIC18.* **SPPU : Dec.-12, May-13, Marks 8**

Solution : Assume that T_{ON} and T_{OFF} of LED = 500 ms.

```
# include < PIC18F458.h >
void DELAY (unsigned int);
void main (void)
{
        TRISC= 0;      // Make Port C as an output
        while(1)
        {
            PORTC = 0x00;
            DELAY (500);
            PORTC = 0xFF;
            DELAY (500);
        }
}
void DELAY (unsigned int t);
{
    unsigned int i, j;
    for (i = 0; i<t; i++)
        for (j = 0; j<165; j++);
}
```

Example 3.6.3 *Write PIC18 C program to toggle all bits of Port B, C, D of PIC18 continuously with a 250ms delay.* **SPPU : May-14, Marks 8**

Solution :

```
# include < PIC18F458.h >
void DELAY (unsigned int);
void main (void)
{
        TRISB= 0;     // Make Port B as an output
        TRISC= 0;     // Make Port C as an output
        TRISD= 0;     // Make Port D as an output
    while(1)
    {
            PORTB = 0x55;
            PORTC = 0x55;
            PORTD = 0x55;
            DELAY (250);
            PORTB = 0xAA;
            PORTC = 0xAA;
            PORTD = 0xAA;
            DELAY (250);
    }
}
void DELAY (unsigned int t);
{
    unsigned int i, j;
    for (i = 0; i<t; i++)
        for (j = 0; j<165; j++);
}
```

❑❑❑

UNIT - II

4 PIC Timers

Syllabus

Timer/Counter : Registers used for Timer/Counter operation, Delay calculations, Programming of Timers using Embedded C.

Contents

4.1 Registers used for Timer/Counter Operation

- PIC18 has two to five timers depending on the PIC family it belongs to. They are Timer 0, 1, 3, 4 and 5. PIC18F458 has four timers : Timer 0, Timer 1, Timer 2, and Timer 3.
- They can be used either as timers to generate time delays or as counters to count events.
- In PIC18F458, timers 0, 1 and 3 are 16-bit timers while timer 2 is 8-bit timer. These timers can be used as timer, counters or for PWM generation.

Timer	Size	Control Register	Count Registers
TIMER0	8-bit/16 bit	T0CON	TMR0H,TMR0L
TIMER1	16-bit	T1CON	TMR1H,TMR1L
TIMER2	8-bit	T2CON	TMR2
TIMER3	16-bit	T3CON	TMR3H,TMR3L

Table 4.1.1 Timers in PIC18F458

- Selecting clock source clock source can be internal or external and is controlled by bit TxCS :
  - TxCS = 0 : Clock source is internal and is taken from Fosc/4.
  - TxCS = 1: Clock source is external and is taken from TxCKI pin; in this case TxSE controls the edge of the signal which triggers increment.
  - **Note :** x can be 0, 1, 2 or 4.

Review Questions

1. *How many timers are available in PIC18 microcontroller ?*
2. *How many timers are available in PIC18F458 microcontroller ?*
3. *Lists control and count registers for timers available in PIC18F458 microcontroller.*
4. *State various clock sources available for timers in PIC18F458 microcontroller.*

4.2 Prescaling of PIC18 Timers

SPPU : May-13

- In some cases, the clock coming from the oscillator could be too fast for an application. We can lower it by using the frequency prescaler.
- **Timer 0 :** The prescaler circuit divides the signal frequency by prescale values of 1 : 2, 1 : 4, ..., 1 : 256 according to the status of T0PS2 : T0PS0 : Timer0 Prescaler Select bits. The prescaler is activated by bit PSA :
  - PSA = 0 : Prescaler is selected.

- PSA = 1 : Prescaler is not selected.
- **Timer 1 :** The prescaler circuit divides the signal frequency by prescale values of 1 : 1, 1 : 2, 1 : 4, 1 : 8 according to the status of **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits.
- **Timer 2 :** The prescaler circuit divides the signal frequency by prescale values of 1 : 1, 1 : 4, 1 : 16 according to the status of **T2CKPS1:T2CKPS0:** Timer2 Clock Prescale Select bits.
- **Timer 3 :** The prescaler circuit divides the signal frequency by prescale values of 1 : 1, 1 : 2, 1 : 4, 1 : 8 according to the status of **T3CKPS1:T3CKPS0:** Timer3 Input Clock Prescale Select bits.

Review Question

1. *Describe prescaling.* **SPPU : May-13, Marks 8**

4.3 Timer 0

SPPU : Dec.-14, May-15

4.3.1 Features

- The features of Timer 0 are :
  1. Software selectable as an 8-bit or 16-bit timer/counter.
  2. Readable and writable
  3. Dedicated 8-bit software programmable prescaler
  4. Clock source selectable to be external or internal
  5. Interrupt-on-overflow from FFh to 00h in 8-bit mode and FFFFh to 0000h in 16-bit mode
  6. Edge select for external clock

4.3.2 T0CON : Timer0 Control Register

- Fig. 4.3.1 shows the Timer0 Control register (T0CON). It is an 8-bit, read/write register used to that controls all the aspects of Timer0 including : enable/disable timer0, select the operating mode, select the clock source and its transition (HIGH to LOW or LOW to HIGH), enable/disable prescaler and to select the prescaler value if enabled.

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

bit 7 bit 0

bit 7 **TMR0ON :** Timer0 On/Off Control bit

1 = Enables Timer0

0 = Stops Timer0

bit 6 **T08BIIT :** Timer0 8-bit/16-bit Control bit

1 = Timer0 is configured as an 8-bit timer/counter

0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS :** Timer0 Clock Source Select bit

1 = Transition on T0CKI pin

0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE :** Timer0 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 :** Timer0 Prescaler Assignment bit

1 = Timer0 Prescaler is not assigned. Timer0 clock input bypasses prescaler.

0 = Timer0 prescaler is assigned. Timer0 clock comes from prescaler output.

bit 2-0 **T0PS2 : T0PS0 :** Timer0 Prescaler Select bits

111 = 1 : 256 Prescale value

110 = 1 : 128 Prescale value

101 = 1 : 64 Prescale value

100 = 1 : 32 Prescale value

011 = 1 : 16 Prescale value

010 = 1 : 8 Prescale value

001 = 1 : 4 Prescale value

000 = 1 : 2 Prescale value

Fig. 4.3.1 Bit pattern for timer0 control register

4.3.3 Block Diagram

- Fig. 4.3.2 shows the simplified block diagram of Timer 0 in 8-bit mode.
- Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI maximum prescale.
- Timer0 can operate as a timer or as a counter. Timer mode is selected by clearing the T0CS bit.

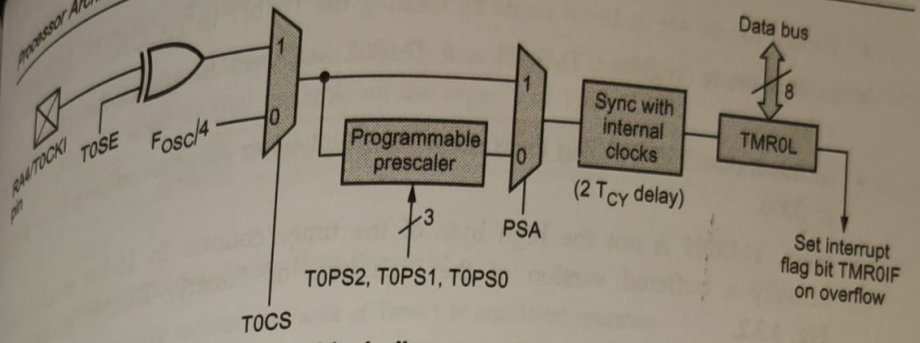

Fig. 4.3.2 Simplified block diagram of Timer 0 in 8-bit mode

- In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler).
- Counter mode is selected by setting the T0CS bit. 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). Clearing the T0SE bit selects the rising edge.
- The prescaler circuit divides the signal frequency by prescale values of 1:2, 1:4, ..., 1:256 according to the status of T0PS2:T0PS0: Timer0 Prescaler Select bits. The prescaler is activated by bit PSA :
  - PSA = 0 : Prescaler is selected.
  - PSA = 1 : Prescaler is not selected.
- An overflow occurs when a timer register (TMR0L) has already counted the maximum value it can count (255). At overflow the counter value become 0 again. An overflow triggers an interrupt and set TMR0IF flag bit.
- Fig. 4.3.3 shows the simplified block diagram of Timer 0 in 16-bit mode.

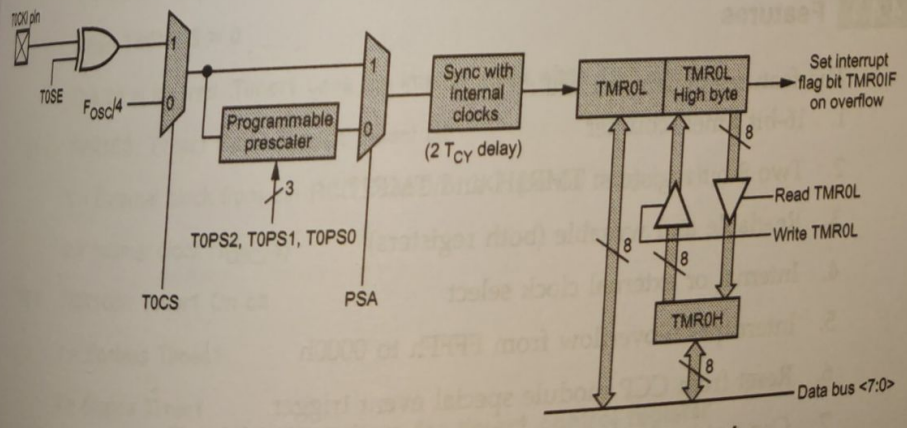

Fig. 4.3.3 Simplified block diagram of Timer 0 in 16-bit mode

- Timer0 can be set in 16-bit mode by clearing the T08BIT in T0CON.
- In this mode, registers TMR0H and TMR0L are used to access the 16-bit timer value.
- In this mode, TMR0IF flag bit is set when 16-bit timer value overflows from FFFFh to 0000.
- Here, TMR0H is not the high byte of the timer/counter in 16-bit mode, but is actually a buffered version of the high byte of Timer0. This is illustrated in Fig. 4.3.2.

4.3.4 Applications

- Applications of Timer0 are :
  - Generate wide range of accurate time delays by selecting the appropriate prescaler options.
  - Measure frequency of an unknown signal.

Review Questions

1. *List the features of Timer0.*
2. *Draw and explain the bit pattern for Timer0 Control Register*
3. *Explain timer 0 mode of PIC18XX and its applications in detail.*

SPPU : Dec.-14, May-15 Marks 8

4. *Draw and explain the simplified block diagram for Timer0 in 8-bit mode.*
5. *Draw and explain the simplified block diagram for Timer0 in 16-bit mode.*

4.4 Timer 1

SPPU : Dec.-14, May-15

4.4.1 Features

- The features of Timer 1 are :
  1. 16-bit timer/counter
  2. Two 8-bit registers: TMR1H and TMR1L
  3. Readable and writable (both registers)
  4. Internal or external clock select
  5. Interrupt-on-overflow from FFFFh to 0000h
  6. Reset from CCP module special event trigger
  7. Can not be disabled by SLEEP instruction

4.4.2 T1CON : Timer1 Control Register

- Fig. 4.4.1 shows the Timer1 Control register (T1CON). It is an 8-bit, read/write register used to that controls all the aspects of Timer1.

RD16		T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit

1 = Enables register read/write of Timer1 in one 16-bit operation

0 = Enables register read/write of Timer1 in one 8-bit operations

bit 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 bit

1 = Timer1 oscillator is enabled

0 = Timer1 oscillator is shut-off

The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit

When TMRIC = 1

1 = Do not synchronize external clock input

0 = Synchronize external clock input

When TMR1CS = 0

This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0

bit 1 **TMR1CS:** Timer1 Clock Source Select bit

1 = External clock from pin RC0/T1OSO/T1CK1 (on the rising edge)

0 = Internal clock ($F_{OSC}/4$)

bit 0 **TMR1ON:** Timer1 On bit

1 = Enables Times1

0 = Slopes Timer1

Fig. 4.4.1 Bit pattern for timer1 control register

4.4.3 Block Diagram

- Fig. 4.4.2 shows the simplified block diagram of Timer 1.

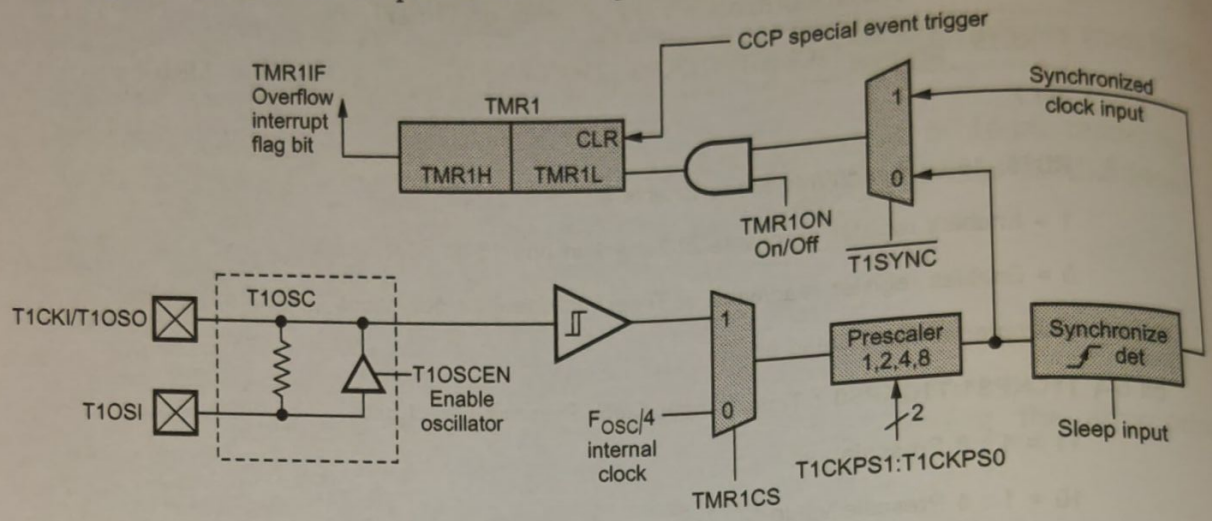

Fig. 4.4.2 Simplified block diagram of Timer1

- Timer1 can be enabled/disabled by setting/clearing control bit, TMR1ON (T1CON register).
- The operating mode for the Timer1 is determined by the clock select bit, TMR1CS (T1CON register). Timer1 can operate in one of these modes :
  - As a timer
  - As a synchronous counter
  - As an asynchronous counter
- When TMR1CS = 0 : Timer1 increments every instruction cycle.
- When TMR1CS = 1 : Timer1 increments on every rising edge of the external clock input or the Timer1 oscillator, if enabled.
- When the Timer1 oscillator is enabled (T1OSCEN is set), the T1OSI and T1OSO/T1CKI pins become inputs. This oscillator is a low power oscillator and it is utilized in the sleep mode. In the sleep mode, the Timer1 is not disabled and hence it can be used to implement the-chip real time clock (RTC).
- Timer1 also has an internal "Reset input". This Reset can be generated by the CCP module.
- The prescaler circuit divides the signal frequency by prescale values of 1:1, 1:2, 1:4, 1:8 according to the status of T1CKPS1:T1CKPS0: Timer1 Input Clock Prescale Select bits.

- Here, TMR1IF flag bit is set when 16-bit timer value in TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h.

4.4.4 Applications

- Applications of Timer1 are :
  - Generate accurate time delays by selecting the appropriate prescaler options.
  - Measure frequency of an unknown signal.
  - Implement Real Time Clock (RTC)
  - Generate special event trigger in compare mode of CCP module.

Review Questions

1. *List the features of Timer1.*
2. *Draw and explain the bit pattern for Timer1 Control Register.* **SPPU : Dec.-15, Marks 4**
3. *Explain timer 1 mode of PIC18XX and its applications in detail.* **SPPU : Dec.-14, May-15 Marks 8**
4. *Draw and explain the simplified block diagram for Timer1.* **SPPU : Dec.-14, 15, May-15**

4.5 Timer 2

4.5.1 Features

- The features of Timer 2 are :
  1. 8-bit timer (TMR2 register)
  2. 8-bit period register (PR2)
  3. Readable and writable (both registers)
  4. Software programmable prescaler (1 : 1, 1 : 4, 1 : 16)
  5. Software programmable postscaler (1 : 1 to 1 : 16)
  6. Interrupt on TMR2 match of PR2
  7. SSP module optional use of TMR2 output to generate clock shift

4.5.2 T2CON: Timer2 Control Register

- Fig. 4.5.1 shows the Timer1 Control register (T2CON). It is an 8-bit, read/write register used to that controls all the aspects of Timer2.

	TOUTPS3	TOUTPS2	TOUTPS2	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

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

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

0000 = 1 : 1 Postscale

0001 = 1 : 2 Postscale

•

•

•

1111 = 1 : 16 Postscale

bit 2 **TMR2ON** : Timer2 On bit

1 = Timer2 is on

0 = Timer2 is off

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

00 = Prescaler is 1

01 = Prescaler is 4

1x = Prescaler is 16

Fig. 4.5.1 Bit pattern for timer2 control register

- Timer2 can be shut-off by clearing control bit TMR2ON (T2CON register) to minimize power consumption.

4.5.3 Block Diagram

- Fig. 4.5.2 shows the simplified block diagram of Timer 2.

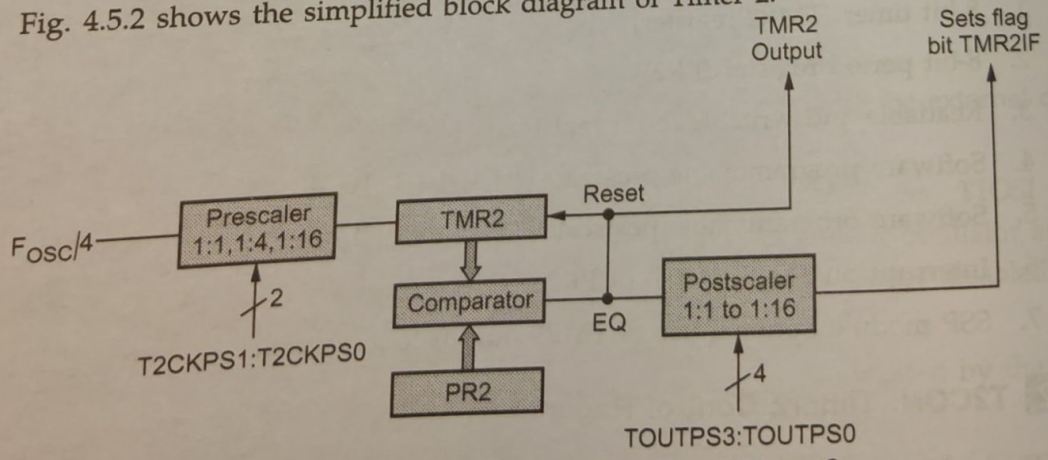

Fig. 4.5.2 Simplified block diagram of Timer 2

- Timer2 is disabled on power-on-reset.
- It consists of two 8-bit, readable/writable registers : TMR2 and PR2, comparator, and prescaler and postscaler circuits.

- The clock source for TMR2 is Fosc/4 for both prescaler and Postscaler options.
- There is no external clock source for Timer2 and hence can't be used as a counter.
- The prescale option of 1:1, 1:4 or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON register).
- The 4-bit postscaler gives a 1:1 to 1:16 scaling options to generate a TMR2 interrupt.
- TMR2 register increments from 00 to the value equal to PR2. Comparator compares values from TMR2 and PR2 registers. When these values are equal, TMR2IF flag is set in PIR1 register and TMR2 reset to 00.
- Timer2 can be used as the PWM time base for the PWM mode of the CCP module.
- The prescaler and postscaler counters are cleared when any of the following occurs :
  1. A write to the TMR2 register
  2. A write to the T2CON register
  3. Any device Reset (Power-on Reset, $\overline{MCLR}$ Reset, Watchdog Timer Reset or Brown-out Reset)
- TMR2 is not cleared when T2CON is written.

4.5.4 Applications

- Applications of Timer 2 are :
  1. Generate accurate time delays by selecting the appropriate prescaler options.
  2. Acts as an input for SSP module to generate clock shift.
  3. Used as PWM time base for the PWM mode of the CCP module.
  4. Generate periodic interrupts.

Review Questions

1. *List the features of Timer2.*
2. *Draw and explain the bit pattern for Timer2 Control Register.* **SPPU : Dec.-15, Marks 4**
3. *Explain timer2 mode of PIC18XX and its applications in detail.* **SPPU : Dec.-14, May-15, Marks 8**
4. *Draw and explain the simplified block diagram for Timer2.*

4.6 Timer 3

SPPU : Dec.-14, May-15

4.6.1 Features

- The features of Timer 3 are :
  1. 16-bit timer/counter
  2. Two 8-bit registers: TMR3H and TMR3L)

3. Readable and writable (both registers)
4. Internal or external clock select
5. Interrupt-on-overflow from FFFFh to 0000h
6. Reset from CCP1/ECCP1 module trigger

4.6.2 T3CON : Timer3 Control Register

- Fig. 4.6.1 shows the Timer3 Control register (T3CON). It is an 8-bit, read/write register used to that controls all the aspects of Timer3.

RD16	T3ECCP1	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

bit 7 **RD16 :** 16-bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operation

bit 6,3 **T3ECCP1 : T3CCP1 :** Timer3 and Timer1 to CCP1/ECCP1 Enable bits
1x = Timer3 is the clock source for compare/capture CCP1 and ECCP1 modules
01 = Timer3 is the clock source for compare/capture of ECCP1
Timer1 is the clock source for compare/capture of CCP1
00 = Timer1 is the clock source for compare/capture of CCP1 and ECCP1 modules

bit 5-4 **T3CKPS1 : T3CKPS0 :** Timer3 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 2 $\overline{\text{T3SYNC}}$ **:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the system clock comes from Timer1/Timer3)
When TMR3CS = 1
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0;
This bit is ignored. Timer3 uses the
internal clock when TMR3CS = 0.

bit 1 **TMR3CS :** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T1CK
(on the rising edge after the first falling edge)

bit 0 **TMR3ON :** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

Fig. 4.6.1 Bit pattern for Timer3 control register

4.6.3 Block Diagram

- Fig. 4.6.2 shows the simplified block diagram of Timer3.

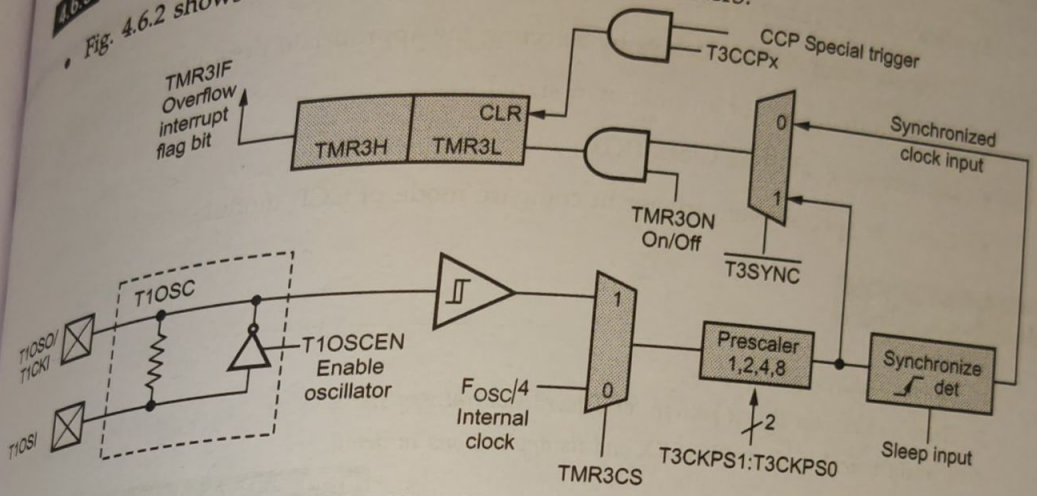

Fig. 4.6.2 Simplified block diagram of Timer3

- Timer3 can be enabled/ disabled by setting/clearing control bit, TMR3ON (T3CON register).
- The operating mode for the Timer3 is determined by the clock select bit, TMR3CS (T3CON register). Timer3 can operate in one of these modes :
  - As a timer
  - As a synchronous counter
  - As an asynchronous counter
- When TMR3CS = 0, Timer3 increments every instruction cycle.
- When TMR3CS = 1, Timer3 increments on every rising edge of the Timer1 external clock input or the Timer1 oscillator, if enabled.
- The Timer1 oscillator may be used as the clock source for Timer3. When the Timer1 oscillator is enabled (T1OSCEN is set), the T1OSI and T1OSO/T1CKI pins become inputs.
- Timer3 also has an internal "Reset input". This Reset can be generated by the CCP module.
- The prescaler circuit divides the signal frequency by prescale values of 1 : 1, 1 : 2, 1 : 4, 1 : 8 according to the status of T3CKPS1:T3CKPS0: Timer3 Input Clock Prescale Select bits.
- Here, TMR3IF flag bit is set when 16-bit timer value in TMR3 register pair (TMR3H:TMR3L) increments from 0000h to FFFFh and rolls over to 0000h.

4.6.4 Applications

- Applications of Timer 3 are :
  - Generate accurate time delays by selecting the appropriate prescaler options.
  - Measure frequency of an unknown signal.
  - Implement Real Time Clock (RTC)
  - Generate special event trigger in compare mode of CCP module.

Review Questions

1. *List the features of Timer3.*
2. *Draw and explain the bit pattern for Timer3 Control Register.*
3. *Explain timer3 mode of PIC18XX and its applications in detail.*

SPPU : Dec.-14, May-15, Marks 8

4. *Draw and explain the simplified block diagram for Timer3.*

4.7 Delay Calculations

- Let's take an example. Suppose we have to generate a delay of 1 ms having 10 MHZ oscillator frequency of PIC18F458.

Calculate timer input frequency

Given : F_{OSC} = 10 MHz. Let us take prescaler = 1 : 4

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio}$$

$$= \frac{10}{4} \times \frac{1}{4} = 625 \text{ kHz}$$

Find the period which will be taken by the timer to increment the count

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{625\,\text{kHz}} = 1.6\ \mu\text{s}$$

- Therefore, after each 1.6 µs, the timer value will be incremented by 1.

Find the count needed for a delay of 1 ms

$$\text{Count} = \frac{1\,\text{ms}}{1.6\,\mu\text{s}} = 625$$

- Since this count is greater than 256 we need use 16 - bit timer.

Calculate value to be loaded in Timer register for a delay of 1 ms

- The value to be loaded in 16-bit Timer register is calculated as :
  65536 – Count = 65536 – 625 = 64911 = FD8FH
- Load this hex value in the 16-bit timer register to produce desired delay.

Using 8-bit Timer

- By selecting larger prescaler ratio we can generate same delay with 8-bit counter.
  Let us select prescaler 1 : 64.

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio}$$

$$= \frac{10}{4} \times \frac{1}{64} = 39.0625 \text{ kHz}$$

Find the period which will be taken by the timer to increment the count

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{39.0625 \text{ kHz}} = 25.6 \ \mu s$$

- Therefore, after each 25.6 , μs the timer value will be incremented by 1.

Find the count needed for a delay of 1 ms

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{1 \text{ ms}}{25.6 \mu s}$$

$$= 39.0625 \approx 39$$

- Since this count is less than 256 we need use 8- bit timer.
- The value to be loaded in 8-bit Timer register is calculated as :
  256 - Count = 256 – 39 = 217 = D9H
- Load this hex value in the 8-bit timer register to produce desired delay.

Review Question

1. *Explain the delay calculations for timers in PIC18.*

4.8 Programming of Timers using Embedded C

SPPU : Dec.-16

- Steps for Programming PIC18F458 Timer
1. Configure the Timer Control Register.
2. Clear Timer interrupt flag.
3. Load the count in Timer register.
4. Set Timer ON bit to start the Timer operation.

5. Wait for Timer interrupt flag to become 1. Timer interrupt flag bit = 1, indicates the occurrence of the timer overflow.

Example 4.8.1 *Write a C18 program to toggle all bits of Port B continuously with delay of 20 ms using Timer 0 , 16 bit mode and 1: 8 prescaler. Assume XTAL = 10 MHz.*

Solution : F_{OSC} = 10 MHZ, Prescaler = 1 : 8

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio}$$

$$= \frac{10}{4} \times \frac{1}{8} = 312.5 \text{ kHz}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{312.5 \text{ kHz}} = 3.2 \ \mu s$$

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{20 \text{ ms}}{3.2 \mu s} = 6250$$

The value to be loaded in 16-bit Timer register is :

65536 – Count = 65536 – 6250 = 59286 = E796H

T0CON Configuration

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	
0	0	0	0	0	0	1	0	= 02 H

```
#include <P18F458.h>
void T0Delay(void);
void main(void)
{
  TRISB=0;              // configure Port B as output
  while(1)
  {
    PORTB= 0x55;        // Load bit patterns
    T0Delay ( );
    PORTB= 0xAA;
    T0Delay ( );
  }
}
void T0Delay ( )
```

```
{
  T0CON=0x02;                          // Timer0, 16 bit mode, 1:8 prescaler
  TMR0H=0xE7;                          // Load Higher byte in TMR0H
  TMR0L= 0x96;                         // Load Lower byte in TMR0L
  T0CONbits.TMR0ON=1;                  // Start the Timer0
  while(INTCONbits.TMR0IF==0);         // Check for overflow
  T0CONbits.TMR0ON=0;                  // Turn off Timer0
  INTCONbits.TMR0IF==0;                // Clear the Timer0 interrupt flag
}
```

Example 4.8.2 *Write a C18 program to toggle only the RC5 bit continuously every 50 ms using Timer 0 , 16 bit mode and 1: 4 prescaler. Assume XTAL = 10 MHz.*

Solution : Given : F_{OSC} = 10 MHZ, Prescaler = 1 : 4

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio} = \frac{10}{4} \times \frac{1}{4} = 625 \text{ kHz}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{625 \text{ kHz}} = 1.6 \ \mu\text{s}$$

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{50 \text{ ms}}{1.6 \mu\text{s}} = 31250$$

The value to be loaded in 16-bit Timer register is :

65536 - Count = 65536 – 31250 = 34286 = 85EEH

T0CON Configuration

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	
0	0	0	0	0	0	0	1	= 01 H

```
#include <P18F458.h>
void T0Delay(void);
#define  PORTBit  PORTCbits.RC5
void main(void)
{
 TRISCbits.TRISC5=0;  // configure pin 5 of Port C as output
 while(1)
 {
  PORTBit ^= 1;       // Toggle the bit RC5
  T0Delay ( );        // Wait for 50 ms
```

```
    }
}
void T0Delay ( )
{
    T0CON=0x01;                          // Timer0, 16 bit mode, 1:4 prescaler
    TMR0H=0x85;                          // Load Higher byte in TMR0H
    TMR0L= 0xEE;                         // Load Lower byte in TMR0L
    T0CONbits.TMR0ON=1;                  // Start the Timer0
    while(INTCONbits.TMR0IF==0);         // Check for overflow
    T0CONbits.TMR0ON=0;                  // Turn off Timer0
    INTCONbits.TMR0IF==0;                // Clear the Timer0 interrupt flag
}
```

Example 4.8.3 *Write a C18 program to toggle only the RB4 bit continuously every 10 ms using Timer0, 8-bit mode. Assume XTAL = 10 MHz.*

Solution :

Given : F_{OSC} = 10 MHZ, Assume : Prescaler = 1 : 1

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio} = \frac{10}{4} \times \frac{1}{1} = 2.5 \text{ MHz}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{2.5\,\text{MHz}} = 0.4\,\mu\text{s}$$

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{10\,\text{ms}}{0.4\,\mu\text{s}} = 25000$$

For 8-bit timer, prescaler ratio >= 25000/256 = 97.66

$\therefore$ Prescaler = 1 : 128

Count (with Prescaler = 1 : 128) = 25000/128 = 195.3125 $\approx$ 195

The value to be loaded in 8-bit Timer register is :

256 – Count = 256 – 195 = 61 = 3DH

T0CON Configuration

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	
0	1	0	0	0	1	1	0	= 46 H

```
#include <P18F458.h>
void T0Delay(void);
#define  PORTBit  PORTBbits.RB4
```

```
void main(void)
{
  TRISBbits.TRISB4=0;                    // configure pin 4 of Port B as output
  while(1)
  {
    PORTBit ^= 1;                        //  Toggle the bit RB4
    T0Delay ( );                         // Wait for 10 ms
  }
}
void T0Delay ( )
{
  T0CON=0x46;                            // Timer0, 8 bit mode, 1:128 prescaler
  TMR0L= 0x3D;                           // Load byte in TMR0L
  T0CONbits.TMR0ON=1;                    // Start the Timer0
  while(INTCONbits.TMR0IF==0);           // Check for overflow
  T0CONbits.TMR0ON=0;                    // Turn off Timer0
  INTCONbits.TMR0IF==0;                  // Clear the Timer0 interrupt flag
}
```

Note : When maximum prescalar count is not sufficient to produce desired delay we can call Delay subroutine multiple times. See following example.

Example 4.8.4 *Write a C18 program to toggle only the RB4 bit continuously every 100 ms using Timer0, 8-bit mode. Assume XTAL = 10 MHz.*

Solution : Given : F_{OSC} = 10 MHZ, Assume : Prescaler = 1 : 1

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio} = \frac{10}{4} \times \frac{1}{1} = 2.5 \text{ MHz}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{2.5 \text{ MHz}} = 0.4 \ \mu\text{s}$$

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{100 \text{ ms}}{0.4 \ \mu\text{s}} = 250000$$

For 8-bit timer, prescaler ratio >= 250000/256 = 970.66

Here, we choose maximum prescaler

∴ Prescaler = 1 : 256

Thus, the delay subroutine must be called 4 times so that $4 \times 256 > 970.66$.

Count (with Prescaler = 1 : 256) = $250000/(256 \times 4) = 244.14 \approx 244$

The value to be loaded in 8-bit Timer register is :

256 – Count = 256 – 244 = 12 = 0CH

T0CON Configuration

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	
0	1	0	0	0	1	1	1	= 47 H

```
#include <P18F458.h>
void T0Delay(void);
#define  PORTBit  PORTBbits.RB4
void main(void)
{
  TRISBbits.TRISB4=0;        // configure pin 4 of Port B as output
  while(1)
  {
    PORTBit ^= 1;            //  Toggle the bit RB4
    for (i = 0; i < 4; i++)
        T0Delay ( );         // Wait for 25 ms
  }
}
void T0Delay ( )
{
    T0CON=0x47;                          // Timer0, 8 bit mode, 1:256 prescaler
    TMR0L= 0x0C;                         // Load byte in TMR0L
    T0CONbits.TMR0ON=1;                  // Start the Timer0
    while(INTCONbits.TMR0IF==0);         // Check for overflow
    T0CONbits.TMR0ON=0;                  // Turn off Timer0
    INTCONbits.TMR0IF==0;                // Clear the Timer0 interrupt flag
}
```

Actual Delay :

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio} = \frac{10}{4} \times \frac{1}{256} = 9.765625 \text{ kHz}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{9.745625 \text{ kHz}} = 102.4 \text{ µs}$$

Delay subroutine execution time = 244 × 102.4 µs = 24.9856 ms

Since delay subroutine is called 4 times we get delay of 99.9424 ms

Example 4.8.5 *Write a C18 program to create frequency of 5KHz on pin RC5 using Timer1. Assume XTAL = 10 MHz.*

Solution : Given : F_{OSC} = 10 MHz, No Prescaler

$$F_{TIMER} = \frac{F_{OSC}}{4} = \frac{10}{4} = 2.5 \text{ MHz}$$

$$Period = \frac{1}{F_{TIMER}} = \frac{1}{2.5 \text{ MHz}} = 0.4 \ \mu s$$

$$T = \frac{1}{5 \text{ kHz}} = 200 \ \mu s$$

$$T_{ON} = T_{OFF} = \frac{T}{2} = 100 \ \mu s$$

$$Count = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{100 \mu s}{0.4 \mu s} = 250$$

The value to be loaded in 16-bit Timer register is :

65536 - Count = 65536 - 250 = 65286 = FF06H

```
#include <P18F458.h>
void T1Delay(void);
#define  PORTBit  PORTCbits.RC5
void main(void)
{
 TRISCbits.TRISC5=0;                  // configure pin 5 of Port C as output
 while(1)
 {
  PORTBit ^= 1;                       //  Toggle the bit RC5
  T1Delay ( );                        // Wait for 50 ms
 }
}
void T1Delay ( )
{
  T1CON=0x00;                         // Timer1, 16 bit mode, no prescaler
  TMR1H=0xFF;                         // Load Higher byte in TMR1H
  TMR1L= 0x06;                        // Load Lower byte in TMR1L
  T1CONbits.TMR1ON=1;                 // Start the Timer1
  while(PIR1bits.TMR1IF==0);          // Check for overflow
  T1CONbits.TMR1ON=0;                 // Turn off Timer1
  PIR1bits.TMR1IF==0;                 // Clear the Timer1 interrupt flag
}
```

Example 4.8.6 *Write a program to generate 100 msec delay using Timer1. What are the values to be loaded in T1CON, TMR1H, and TMR1L. Assume XTAL = 8 MHz.*

SPPU : Dec.-16, Marks 8

Solution : Given : F_{OSC} = 8 MHZ, Assume : Prescaler = 1 : 4

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio}$$

$$= \frac{8}{4} \times \frac{1}{4} = 500 \text{ kHz}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{500 \text{ kHz}} = 2 \mu s$$

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{100 \text{ ms}}{2 \mu s} = 50000$$

The value to be loaded in 16-bit Timer register is :

65536 - Count = 65536 - 50000 = 15536= 3CB0H

```
#include <P18F458.h>
void T1Delay(void);
void T1Delay ( )
{
    T1CON=0x20;                  // Timer1, 16 bit mode, 1 : 4 prescaler
    TMR1H=0x3C;                  // Load Higher byte in TMR1H
    TMR1L= 0xB0;                 // Load Lower byte in TMR1L
    T1CONbits.TMR1ON=1;          // Start the Timer1
    while(PIR1bits.TMR1IF==0); // Check for overflow
    T1CONbits.TMR1ON=0;          // Turn off Timer1
    PIR1bits.TMR1IF==0;          // Clear the Timer1 interrupt flag
}
```

Example 4.8.7 *Write a program to turn on RB4 pin when THR2 reaches value 100 decimal. Assume XTAL = 10 MHz.*

Solution :

```
#include <P18F458.h>
#define  PORTBit  PORTBbits.RB4
void main(void)
{
    TRISBbits.TRISB4=0;          // configure pin 4 of Port B as output
```

```
T2CON=0x00;                    // Timer2,  no prescaler, no post scaler
TMR2=0x00;                     // TMR2 = 0
PORTBit = 0;                   // PB4 = 0
PR2 = 100;                     // PR2 = 100
T2CONbits.TMR2ON=1;            // Start the Timer2
while(PIR1bits.TMR2IF==0);     // Check for overflow
PORTBit = 1;                   // PB4 = 1
T2CONbits.TMR2ON=0;            // Turn off Timer2
PIR1bits.TMR2IF==0;            // Clear the Timer2 interrupt flag
while(1);                      // Wait forever
}
```

Example 4.8.8 *Using prescaler and postscaler, find the largest time delay that can be generated using Timer2 . Assume XTAL = 10 MHz.*

Solution : F_{OSC} = 10 MHZ. For Timer 2, Maximum prescaler = 16 and maximum postscaler = 16

$$F_{TIMER} = \frac{F_{OSC}}{4} \times \text{Prescaler ratio} = \frac{10}{4} \times \frac{1}{16} = 156.25 \text{ kHz}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{156.25 \text{ kHz}} = 6.4 \ \mu s$$

Using Timer2, the longest time delay can be generated making PR2 = 255.

$\therefore$ Longest delay = 255 × 6.4 µs × Postscale = 255 × 6.4 µs × 16 = **26.112 ms**

Example 4.8.9 *Write a program to generate a square wave of 25 Hz frequency on RB4 using Timer3, 16-bit mode with no prescaler. Assume XTAL = 10 MHz.*

Solution : F_{OSC} = 10 MHz. No Prescaler

$$F_{TIMER} = \frac{F_{OSC}}{4} = \frac{10}{4} = 2.5 \text{ MHZ}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{2.5 \text{ MHz}} = 0.4 \ \mu s$$

$$T = \frac{1}{25 \text{Hz}} = 0.04 \text{ s}$$

$$T_{ON} = T_{OFF} = \frac{T}{2} = 0.02 \text{ s}$$

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Period}} = \frac{0.0\ 2s}{0.4 \ \mu s} = 50000$$

The value to be loaded in 16-bit Timer register is :

65536 – Count = 65536 – 50000 = 15536 = 3CB0H

```
#include <P18F458.h>
void T3Delay(void);
#define  PORTBit  PORTBbits.RB4
void main(void)
{
  TRISBbits.TRISB4=0;              // configure pin 4 of Port B as output
  while(1)
 {
    PORTBit = ~PORTBit;            //  Toggle the bit RB4
    T3Delay ( );                   // Wait for 0.02 s
  }
}
void T3Delay ( )
{
     T3CON=0x00;                   // Timer3, 16 bit mode, no prescaler
     TMR3H=0x3C;                   // Load Higher byte in TMR3H
     TMR3L= 0xB0;                  // Load Lower byte in TMR3L
     T3CONbits.TMR3ON=1;           // Start the Timer3
     while(PIR2bits.TMR3IF==0); // Check for overflow
     T3CONbits.TMR3ON=0;           // Turn off Timer3
     PIR2bits.TMR3IF==0;           // Clear the Timer3 interrupt flag
}
```

❑❑❑