Sinhgad Institutes

Microcontrollers[304184]

Dr. D. S. Mantri

Professor

Dept. Of Electronics and Telecommunication Engg,

Sinhgad Institute of Technology, Lonavala

dsmantri.sit@sinhgad.edu

Cell. +91 9922431612

UNIT–IV: PIC Peripheral Support

Contents: Brief summary of Peripheral support, Timers and its Programing(mode 0 &1), Interrupt Structure of PIC18FXXXX with SFR, PORTB change Interrupts, use of timers with interrupts, CCP modes: Capture, Compare and PWM generation, DC Motor speed control with CCP, Block diagram of in-built ADC with Control registers, Sensor interfacing using ADC: All programs in embedded C

Unit Objectives : On completion the students will be able to :

1. Understand the basic concept of Timers used for de;;ay calculations
2. Get the ide about the software and hardware interrupts
3. Get view of timer programing with SFRs used
4. Explains the objective of CCP mode
5. Study of CCP mode of operation
6. Able to design DC motor speed control circuit
7. write embedded C programs to test the performance
8. Understand the support of Peripheral devices

***Unit outcomes* :**

1. write programs of delay
2. Configure PIC in CCP modes

Outcome Mapping:

PEOs:11,2 POs:1,2,3,4,5, 12 COs: 2 PSOs:1

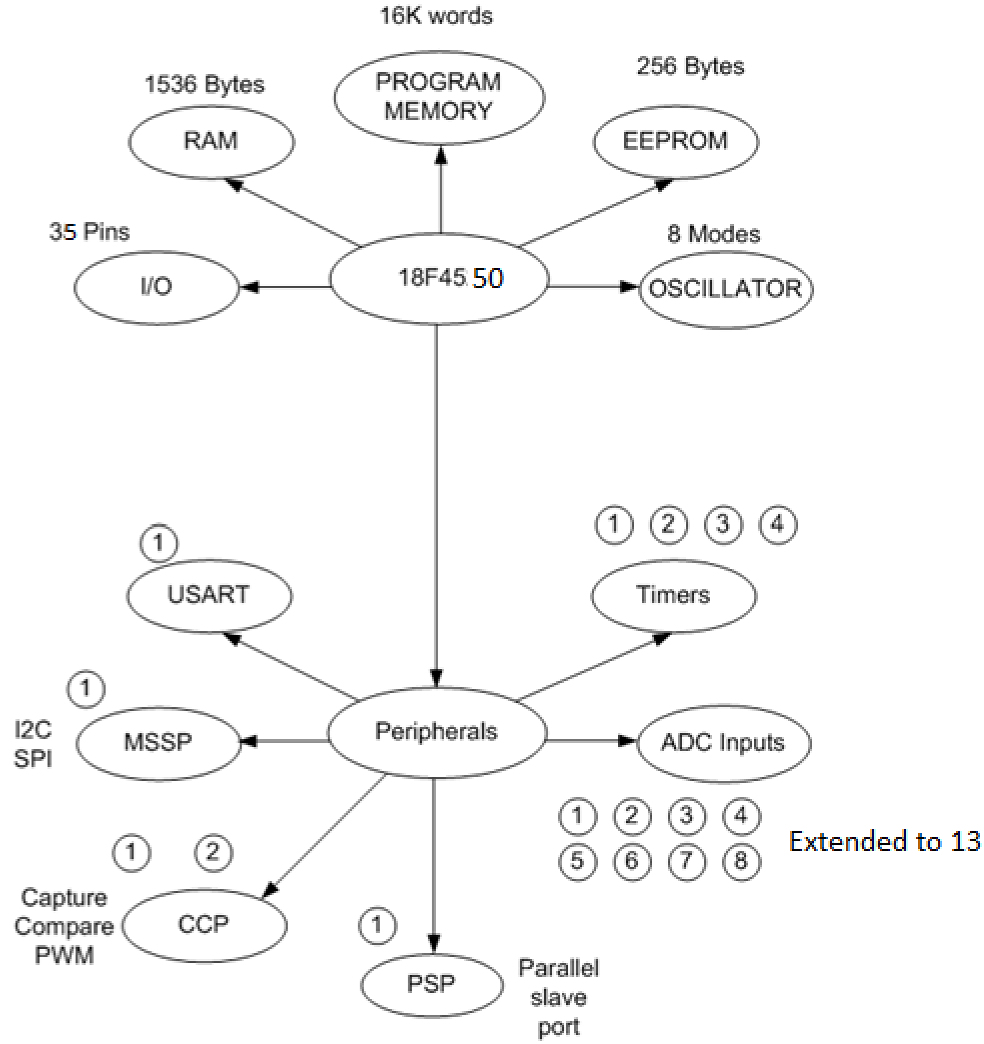
Books :

R3: Peatman, John B, "Design with PIC Microcontroller", Pearson Education PTE,

R4: Data Sheet of PIC 18FXXXX series

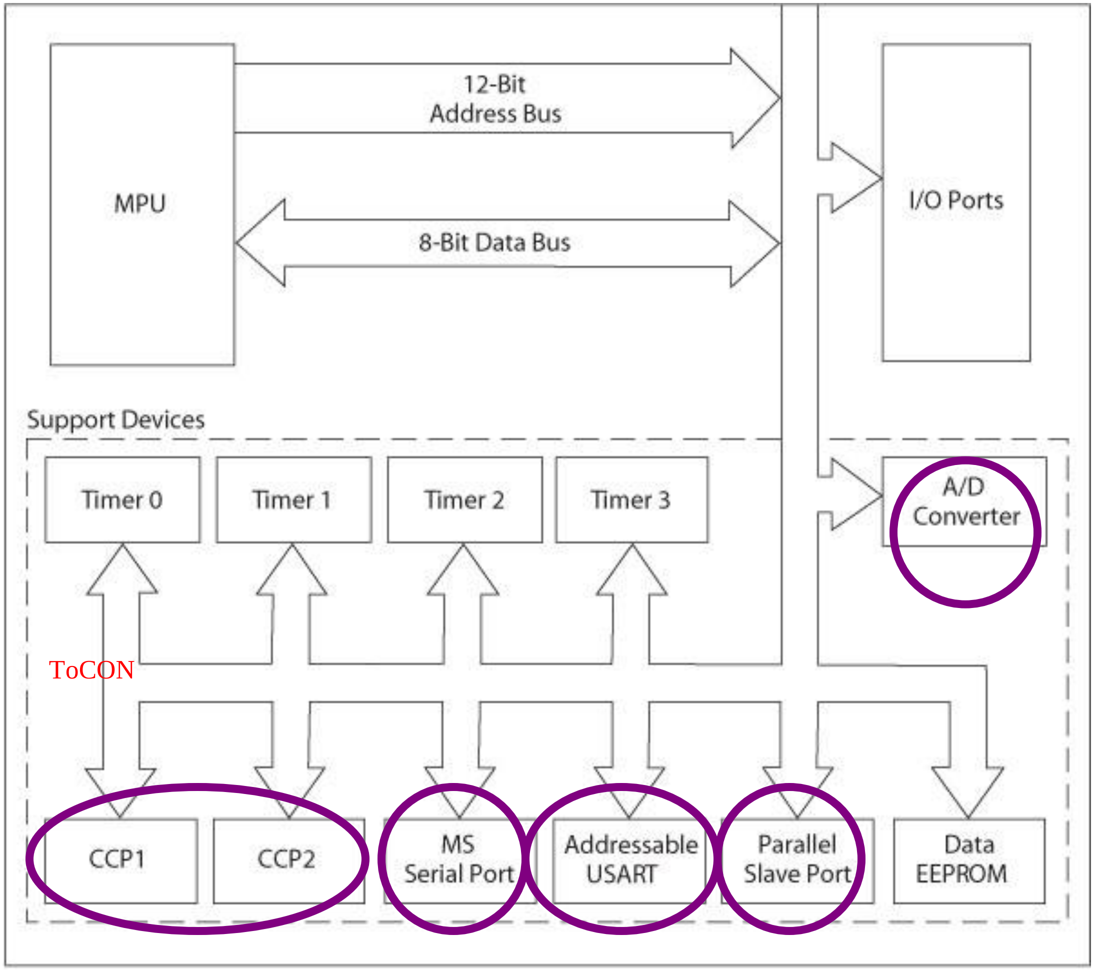
Peripharal Supports- SFRS

- The PIC 18FXXXX has the following peripherals:
  - Data ports:
    - A (7-Bits)
    - B, C and D (8-Bits)
    - E (4- bits)
  - Counter/Timer modules.
    - Modules 0,2 (8-Bits)
    - Modules 1,3 (16-Bits)
  - CCP Modules.
  - I2C/SPI serial port.
  - USART port.
  - ADC 10-bits 13 CH
  - EEPROM 256 Bytes



MCU Support Devices-SFRS

- Timers
  - A value is loaded in the register and continue changing at every clock cycle – time can be calculated
  - Can count on rising or falling edge
  - There are several timers: 8-bit, 16-bit
  - Controlled by SFR
- Master Synchronous Serial Port (MSSP)
  - Serial interface supporting RS232
- Addressable USART
  - Another serial data communication
- A/D converter
- Parallel Slave Port (PSP)
- Capture, Compare and PWM (CCP Module)



Sinhgad Institutes

Peripherals: Timer Module

- The Timer0 module timer/counter which can work as timer/ counter has the following features:
  - 8-bit or 16 bit timer/counter
  - 8-bit software programmable prescaler
  - Internal or external clock
  - select Interrupt on overflow from FFh to 00h
  - Edge select for external clock
- Timer1 is 16 bit timer/ counter and cannot be operated in 8 bit.
- Timer2 is an 8-bit timer with a prescaler. It can be used as the PWM time-base for the PWM mode of the CCP module(s).
- Timer3 is 16 bit timer/ counter and cannot be operated in 8 bit. It also works in CCP mode.

Peripherals: MASTER SYNCHRONOUS SERIAL PORT(MSSP) MODULE

- The Master Synchronous Serial Port (MSSP) module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:
- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I2C)

Peripherals: Enhanced universal synchronous Asynchronous receiver transmitter

The EUSART can be configured in the following modes:

- Asynchronous (full duplex) with:
  - Auto-wake-up on character reception
  - Auto-baud calibration
  - 12-bit Break character transmission
- Synchronous – Master (half duplex) with selectable clock polarity
- Synchronous – Slave (half duplex) with selectable clock polarity

Peripherals: Parallel Slave Port

- In addition to its function as a general I/O port, PORTD can also operate as an 8-bit wide Parallel Slave Port (PSP) or microprocessor port.
- PSP operation is controlled by the 4 upper bits of the TRISE Register.
- Setting control bit, PSPMODE (TRISE<4>), enables PSP operation as long as the Enhanced CCP module is not operating in dual output or quad output PWM mode. In Slave mode, the port is asynchronously readable and writable by the external world.
- The PSP can directly interface to an 8-bit microprocessor data bus. The external microprocessor can read or write the PORTD latch as an 8-bit latch.

Advanced Analog Features

- 10-bit, up to 13-Channel Analog-to-Digital Converter module (A/D) with: -Conversion available during Sleep -Up to 8 channels available
- Analog Comparator module: -Programmable input multiplexing
- Comparator Voltage Reference module
- Programmable Low-Voltage Detection (LVD) module: -Supports interrupt-on-Low-Voltage Detection
- Programmable Brown-out Reset (BOR)

IO port programming in PIC- SFRS

- PIC18 has many ports Depending on the family member and on the number of pins on the chip
- Each port can be configured as input or output. Bidirectional port
- Each port has some other functions Such as timer , ADC, interrupts and serial communication
- Some ports have 8 bits, while others may not
- Each port has three registers for its operation:

  TRIS register (Data Direction register):

  If the corresponding bit is 0 -- Output

  If the corresponding bit is 1 -- Input

- PORT register : (reads the levels on the pins of the device)
- LAT register (output latch):The Data Latch register is usefu
- for read-modify-write operations on the value that the
- IO pins are driving

IMP :Upon reset all ports are configured as input --TRISx register has 0FFh

Pins	Address
PORT A	F80H
PORT B	F81H
PORT C	F82H
PORT D	F83H
PORT E	F84H
LATA	F89H
LATB	F8AH
LATC	F8BH
LATD	F8CH
LATE	F8DH
TRISA	F92H
TRISB	F93H
TRISC	F94H
TRISD	F95H
TRISE	F96H

Peripherals: Compare-Compare-Pulse Width Modulation (CCP)

- The compare mode can cause an event like simply turning on the device when the contents of Timer matches with CCP register.
- In Capture mode, an event at CCP pin will cause contents of timer to be loaded in CCP register.
- Pulse width modulation feature allows to create pulses of variable duty cycle.
- The main difference between Enhanced CCP module and standard CCP is that it allows four pins for implementation of H bridge or half H bridge for DC motor control. -1, 2 or 4 PWM outputs

Timers and its Applications

- PIC18 has two to five timers: Depending on the family number
- All up-counters : 8-bit and 16-bit
  - Timer0 : 8-bit or 16-bit timer
  - Timer1&3 : 16-bit timers
  - Timer2 &4 : 8-bit timer
  - SFRs : T0CON-T2CON

These timers can be used as

- Time delay
- Pulse wave generation
- Pulse width or frequency measurement
- Timer as an event counter

- Up-counter
  - Counter is incremented at every clock cycle
  - When count reaches the maximum count, a flag is set
  - Counter can be reset to zero or to the initial value
- Down-counter
  - Counter is decremented at every clock cycle
  - When count reaches zero, a flag is set
  - Counter can be reset to the maximum or the initial value
- Free-running counter
  - Counter runs continuously and only readable
  - When it reaches the maximum count, a flag is set

TMR0: Timer0- 8 or 16 bit

- 8-bit or 16-bit timer : can be accessed as low and high byte
- Readable and writable
- can be configured as Timer and event counter
- Requires Two SFRS , T0CON and INTCON

Parameters in T0CON register

- Eight pre-scale values (Bit2-Bit0)
- Clock source (Bit5)
  - Internal (instruction cycle) --- Timer
  - External clock connected to pin RA4/T0CK1 -- Counter
    - Rising edge or falling edge (Bit4)
- Generates an interrupt or sets a flag when it overflows
  - TMR0IF, Flag must be cleared to start the timer again

TMR0H								TMR0L							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

T0CON Reg- Timer control Register-8 bit

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

TMR0ON	D7	Timer0 ON and OFF control bit 1 = Enable (start) Timer0 0 = Stop Timer0
T08BIT	D6	Timer0 8-bit/16-bit selector bit 1 = Timer0 is configured as an 8-bit timer/counter. 0 = Timer0 is configured as a 16-bit timer/counter.
T0CS	D5	Timer0 clock source select bit 1 = External clock from RA4/T0CKI pin 0 = Internal clock (Fosc/4 from XTAL oscillator)
T0SE	D4	Timer0 source edge select bit 1 = Increment on H-to-L transition on T0CKI pin 0 = Increment on L-to-H transition on T0CKI pin
PSA	D3	Timer0 prescaler assignment bit 1 = Timer0 clock input bypasses prescaler. 0 = Timer0 clock input comes from prescaler output.
T0PS2:T0PS0	D2 D1 D0	Timer0 prescaler selector
	0 0 0 = 1:2	Prescale value (Fosc / 4 / 2)
	0 0 1 = 1:4	Prescale value (Fosc / 4 / 4)
	0 1 0 = 1:8	Prescale value (Fosc / 4 / 8)
	0 1 1 = 1:16	Prescale value (Fosc / 4 / 16)
	1 0 0 = 1:32	Prescale value (Fosc / 4 / 32)
	1 0 1 = 1:64	Prescale value (Fosc / 4 / 64)
	1 1 0 = 1:128	Prescale value (Fosc / 4 / 128)
	1 1 1 = 1:256	Prescale value (Fosc / 4 / 256)

Timer0

T0CON

B7	B6	B5	B4	B3	B2	B1	B0
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

Prescaler Select Bits

- 1 = Enables Timer0
- 0 = Stops Timer0

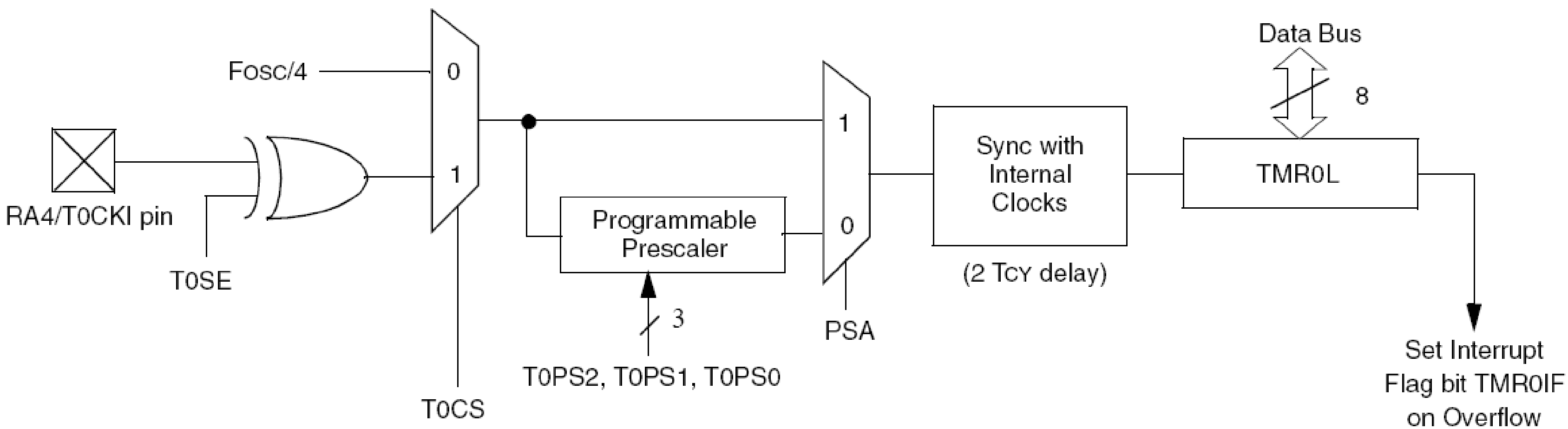
- 1 = 8-bit Timer/Counter
- 0 = 16-bit Timer/Counter

Clock Source
- 1 = TOCK1
- 0 = Instruction Cycle

- 1 = Falling Edge
- 0 = Rising Edge

- 1 = No Prescaler
- 0 = Prescaler Assigned

111 = 1:256	011 = 1:16
110 = 1:128	010 = 1:8
101 = 1:64	001 = 1:4
100 = 1:32	000 = 1:2



TMR0IF flag bit– INTCON-Overflow check

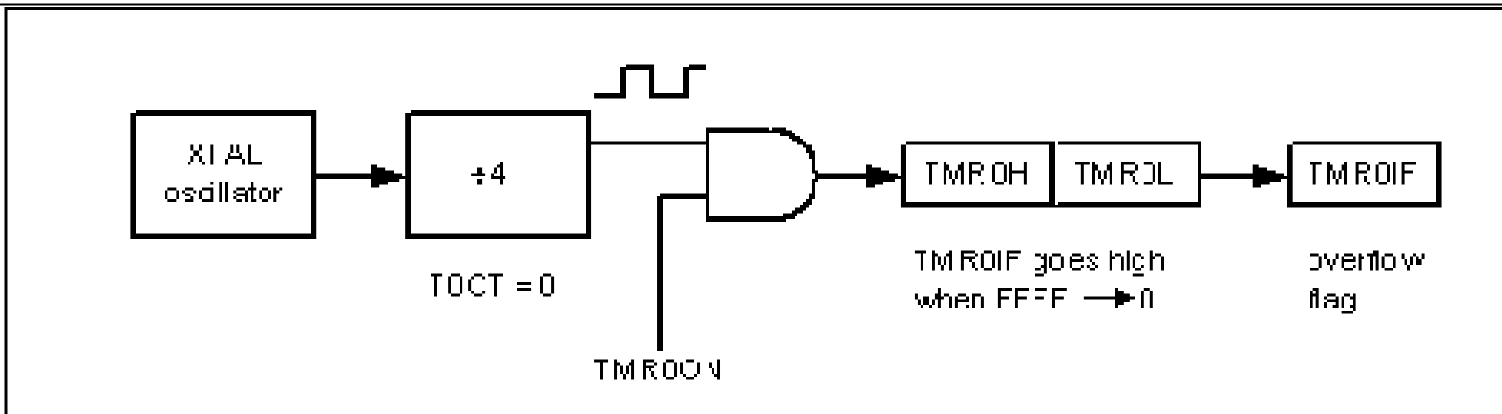
					TMR0IF		

TMR0IF D2 Timer0 interrupt overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (FFFF to 0000, or FF to 00 in 8-bit mode).

The importance of TMR0IF: In 16-bit mode, when TMR0H:TMR0L overflows from FFFF to 0000 this flag is raised. In 8-bit, it is raised when the timer goes from FF to 00. We monitor this flag bit before we reload the TMR0H:TMR0L registers.

The other bits of this register are discussed in Chapter 11.

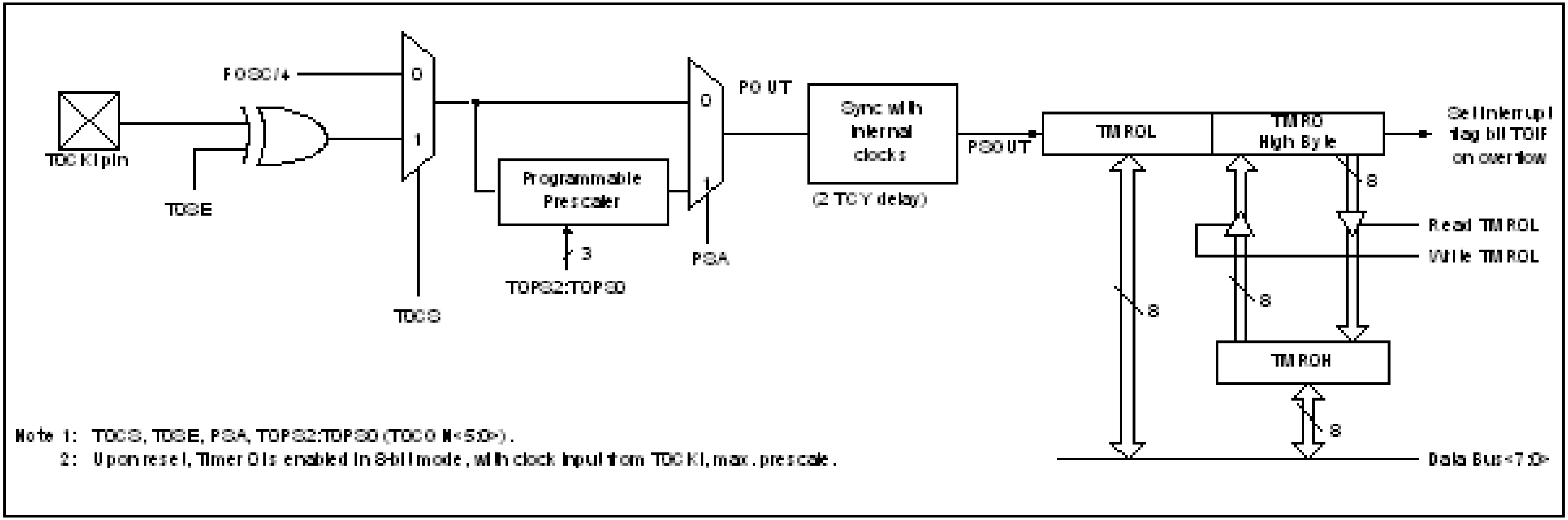
INTCON (Interrupt Control Register) has the TMR0IF Flag



Characteristics and operations of 16-bit mode

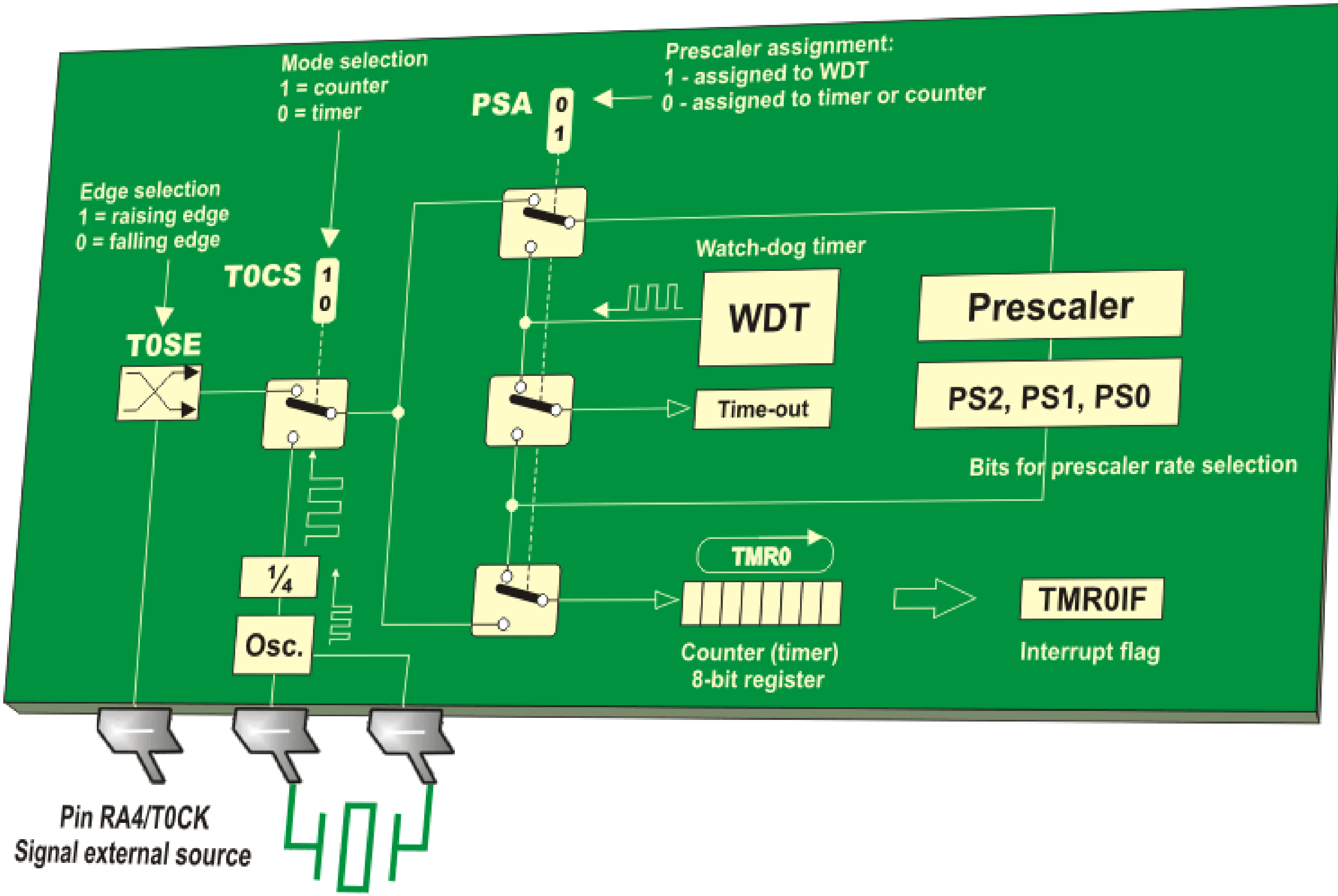
1. 16-bit timer, 0000 to FFFFH.
2. After loading TMR0H and TMR0L, the timer must be started.
3. Count up, till it reaches FFFFH, then it rolls over to 0000 and activate TMR0IF bit.
4. Then TMR0H and TMR0L must be reloaded with the original value and deactivate TMR0IF bit.

Timer0- 16-bit Block Diagram



Load TIMER0H first and then TIMER0L since TIMER0H will be kept in temporary reg. to avoid the errors during counting if TIMER0ON flag is set to High

Timer0- functional Block Diagram



Sinhgad Institutes

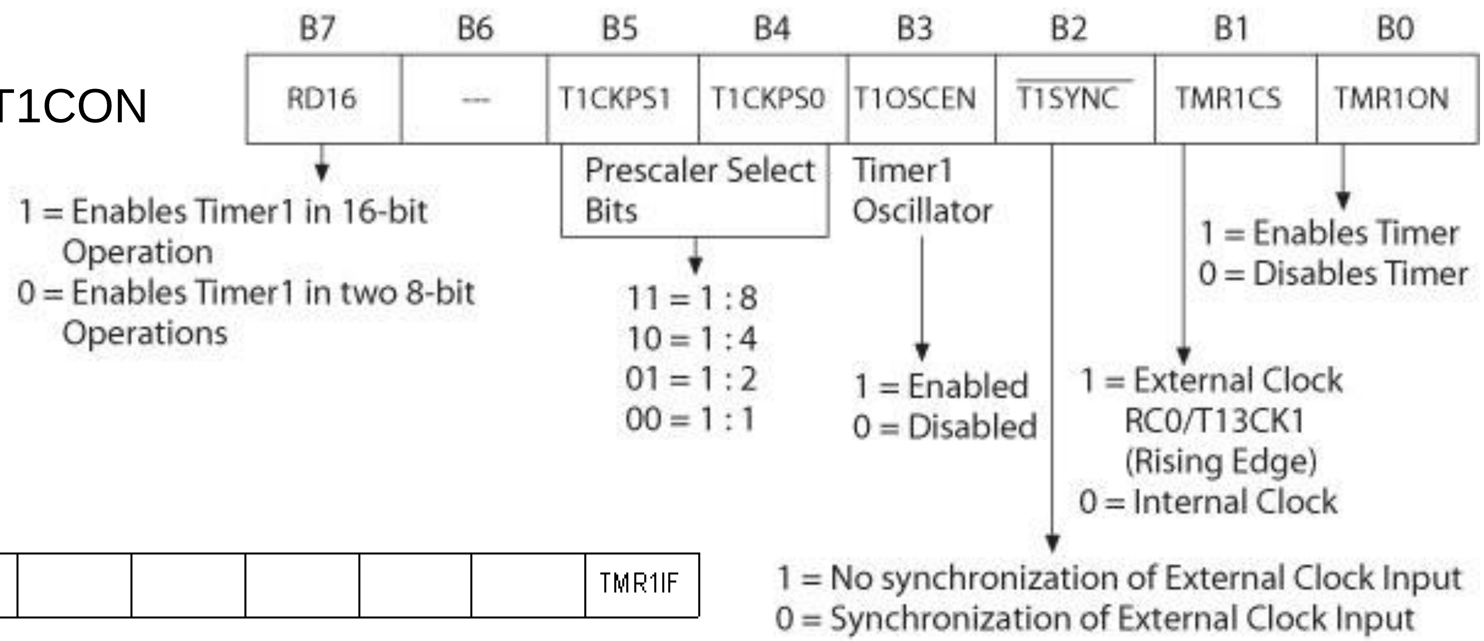
Timer 1- 16 bit

- Programmed in 16-bit mode only and does not support 8-bit mode
- It has 2 bytes named as TMR1L and RMR1H [It can count up 65.535 pulses in a single cycle]
- Has four Prescale values [1:1,1:2,1:4,1:8]
- It has SFR as T1CON and TMR1IF
- The module incorporates its own low-power oscillator to provide an additional clocking option.
- Used as a low-power clock source for the microcontroller in power-managed operation.
- Interrupt
  - Generates an interrupt or sets a flag when it overflows
    - TMR1IF : Flag must be cleared to start the timer again
- Resetting Timer1 using CCP module
  - CCP1 in the Compare mode
  - Timer1 and CCP1compared at every cycle
  - When a match is found, Timer1 is reset

Timer 1- 16 bit- SFRS

- 16-bit counter/timer : Four prescale values (Bit5-Bit4)
  - Clock source (Bit1) : Internal (instruction cycle)
    - External (pin RC0/T13CK1) on rising edge

T1CON



PIR1

							TMR1IF

TMR1IF D1 Timer1 Interrupt overflow flag bit
0 = Timer1 did not overflow.
1 = Timer1 has overflowed (FFFF to 0000).

The importance of TMR1IF: When TMR1H:TMR1L overflows from FFFF to 0000, this flag is raised. We monitor this flag bit before we reload the TMR1H:TMR1L registers.

T1CON (Timer 1 Control) Register

RD16	---	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

RD16 D7 16-bit read/write enable bit
1 = Timer1 16-bit is accessible in one 16-bit operation.
0 = Timer1 16-bit is accessible in two 8-bit operations.

D6 Not used

T1CKPS2:T1CKPS0 D5 D4 Timer1 prescaler selector

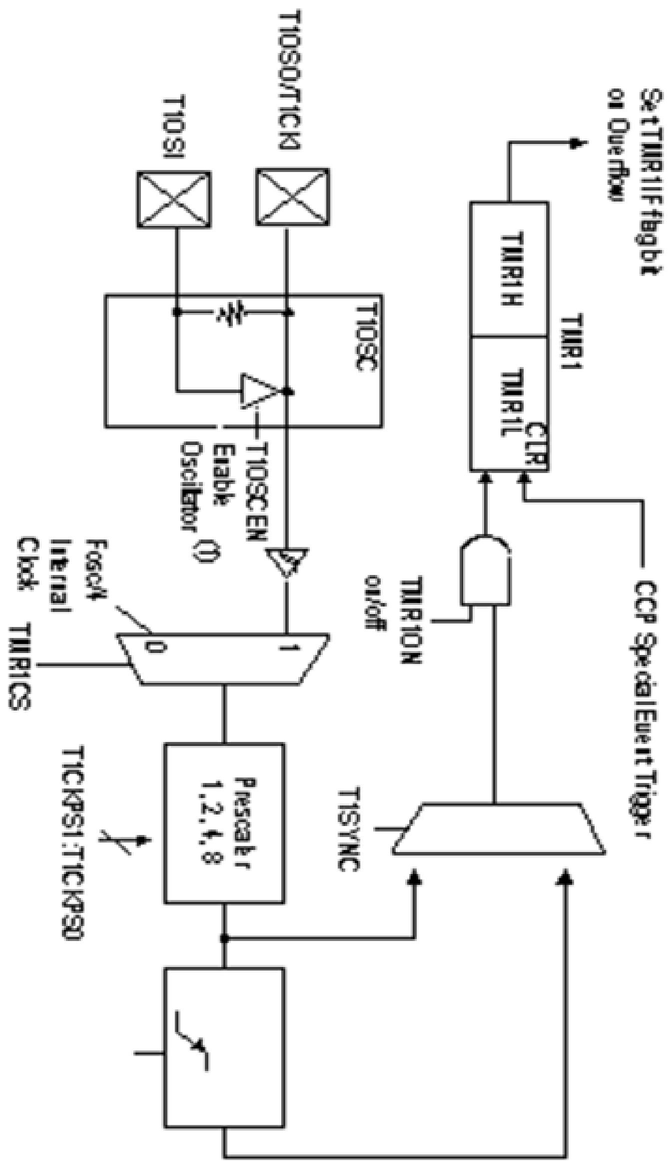
D5	D4		
0	0	= 1:1	Prescale value
0	1	= 1:2	Prescale value
1	0	= 1:4	Prescale value
1	1	= 1:8	Prescale value

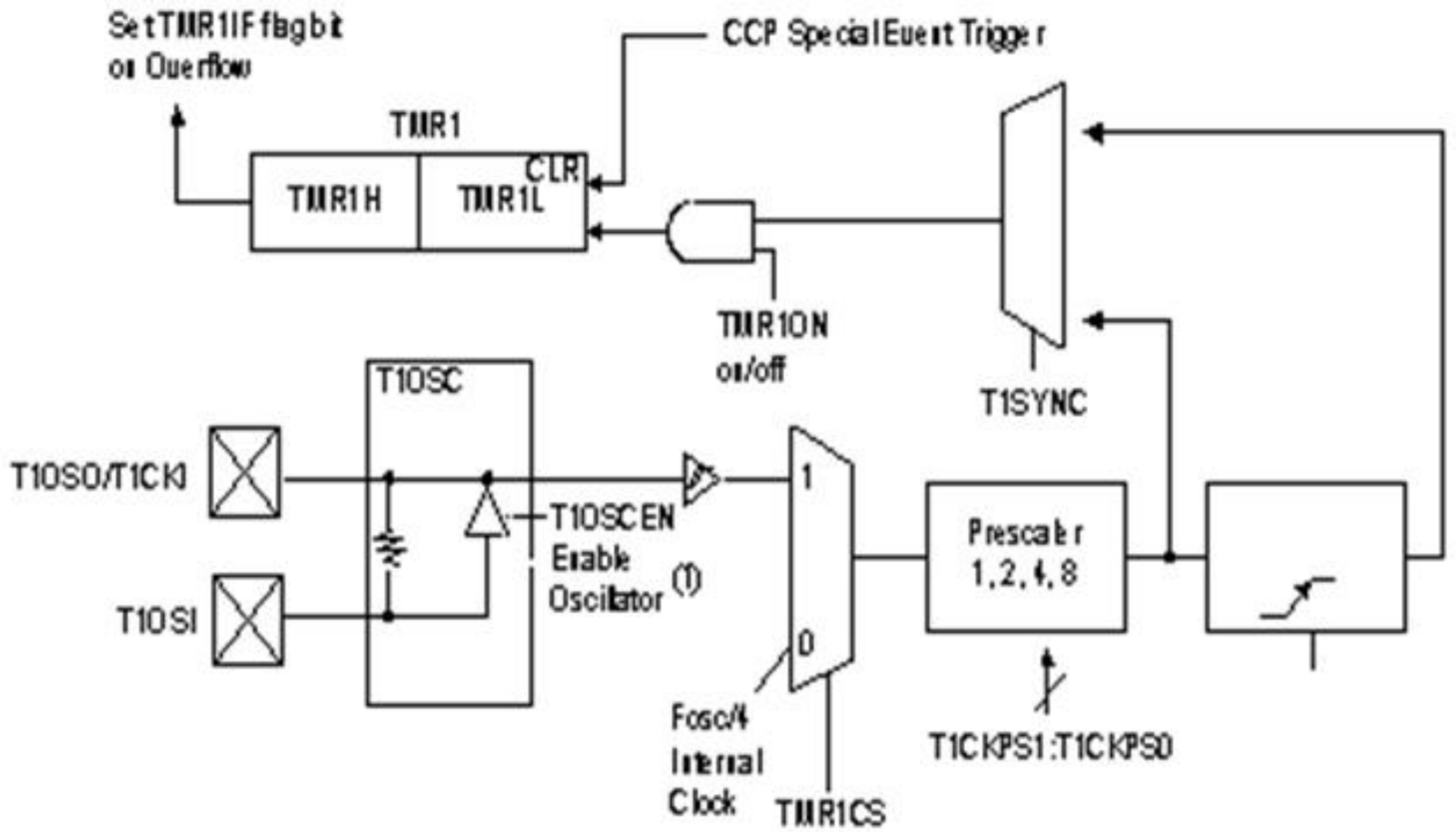
T1OSCEN D3 Timer1 oscillator enable bit
1 = Timer1 oscillator is enabled.
0 = Timer1 oscillator is shutoff

T1SYNC D2 Timer1 synchronization (used only when TMR1CS = 1 for counter mode to synchronize external clock input)
If TMR1CS = 0 this bit is not used.

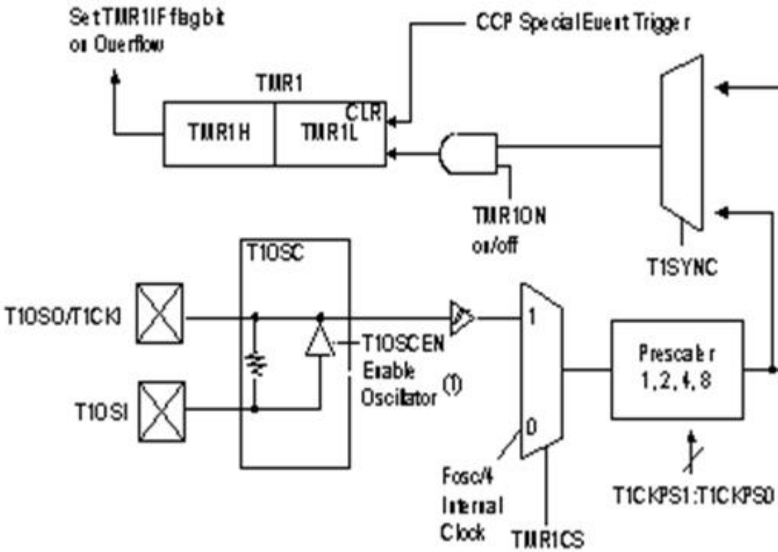
TMR1CS D1 Timer1 clock source select bit
1 = External clock from pin RC0/T1CKI
0 = Internal clock (Fosc/4 from XTAL)

TMR1ON D0 Timer1 ON and OFF control bit
1 = Enable (start) Timer1
0 = Stop Timer1

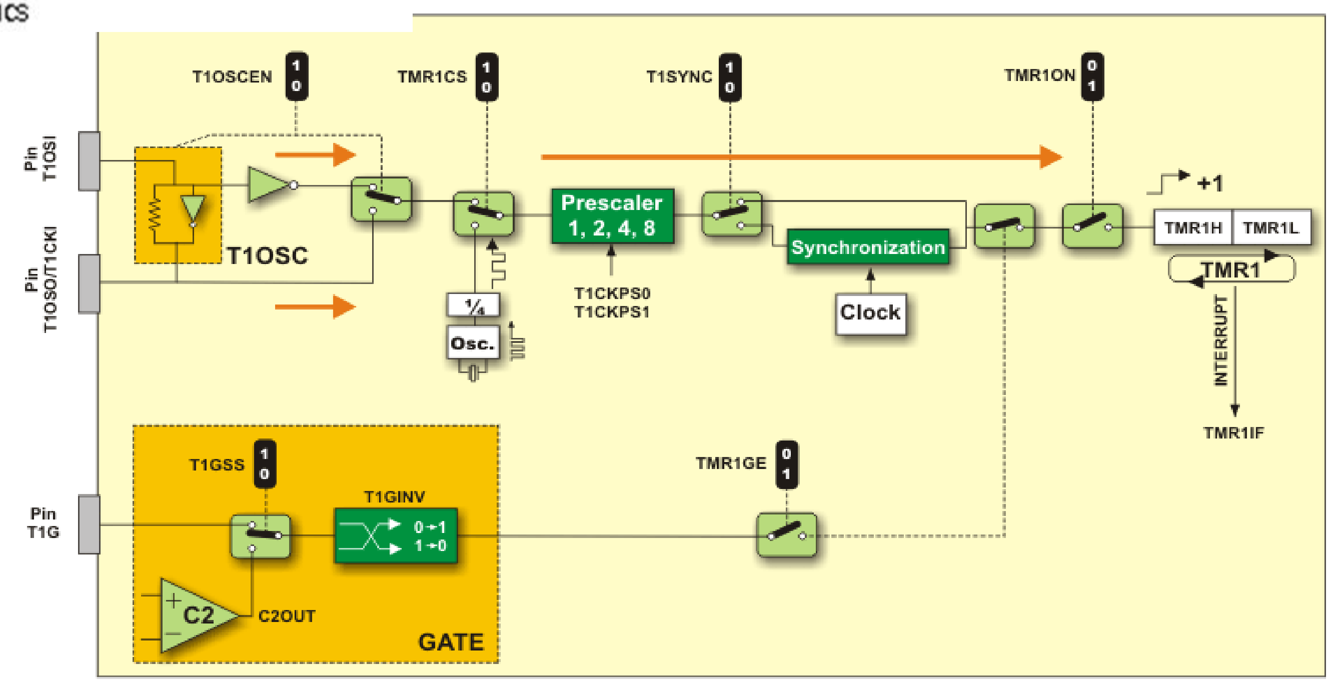



Set TMR1IF flag bit
on Overflow
TMR1
TMR1H
TMR1L
CLR
CCP Special Event Trigger
TMR1ON
on/off
T1SYNC
T1OSC
T1OSO/T1CKI
T1OSI
T1OSCEN
Enable
Oscillator (1)
1
0
Fosc/4
Internal
Clock
TMR1CS
Prescaler
1, 2, 4, 8
T1CKPS1:T1CKPS0

Timer1 Block Diagram



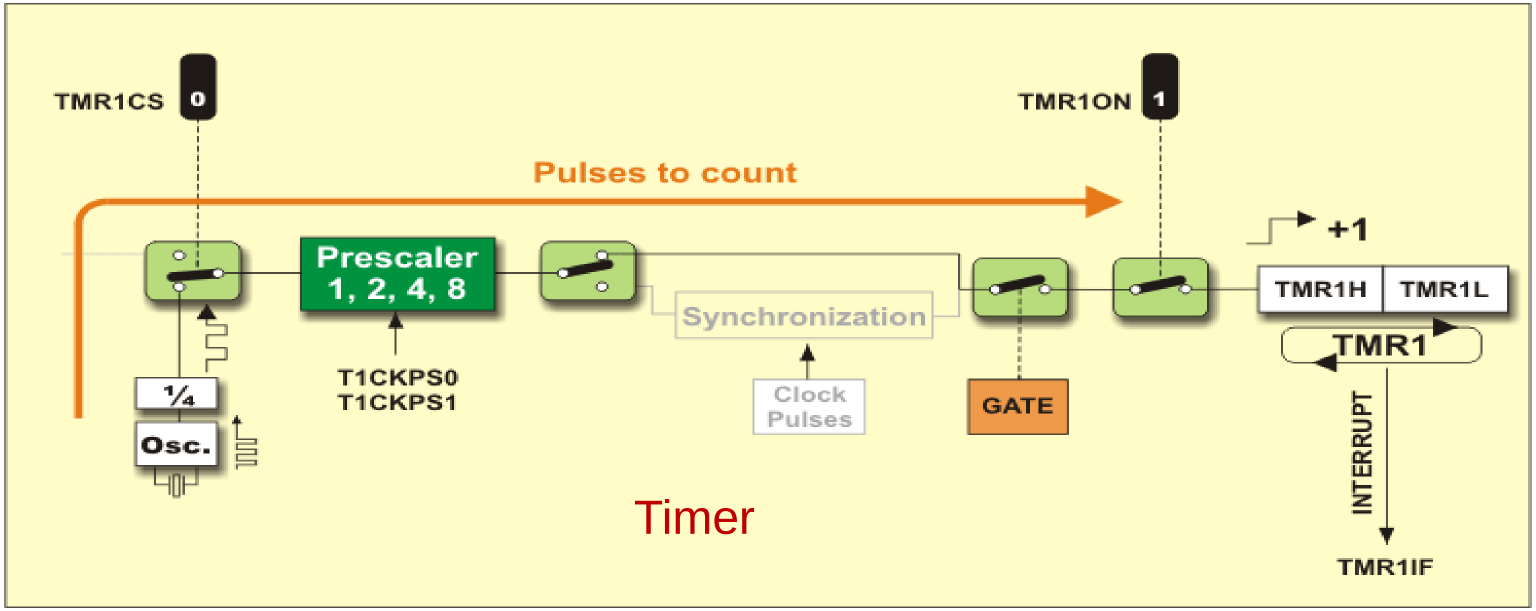
Timer TMR1 has a completely separate prescaler which allows 1, 2, 4 or 8 divisions of the clock input. The prescaler is not directly readable or writable. However, the prescaler counter is automatically cleared upon write to the TMR1H or TMR1L register.

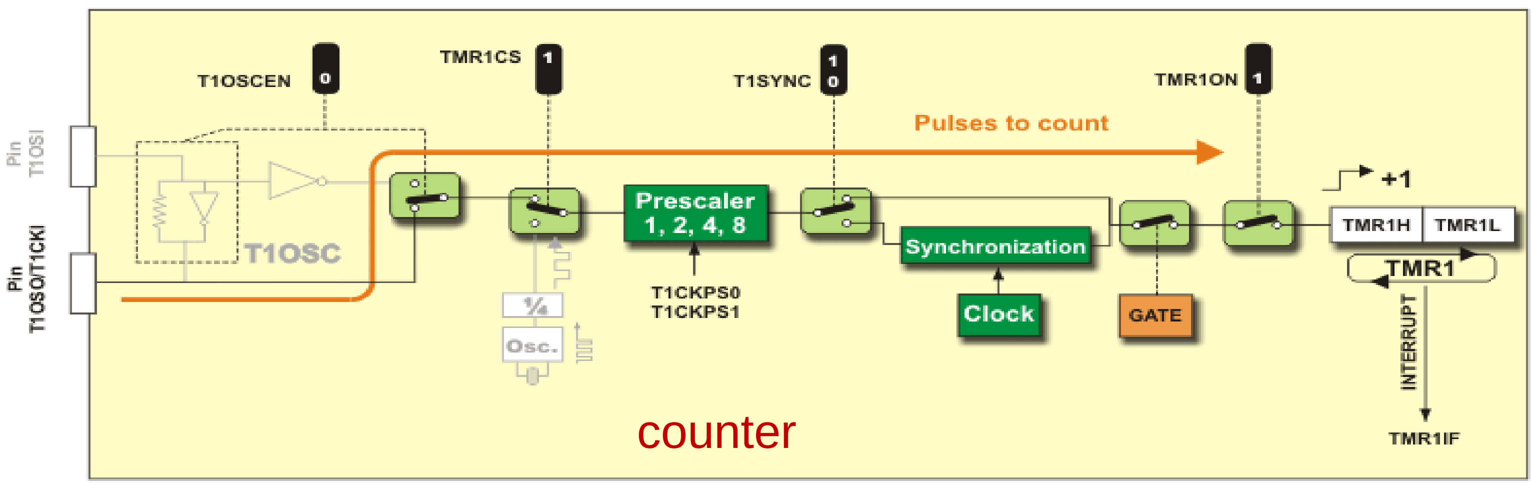


Counter programming

- Used to counts event outside the PIC
  - Increments the TMR0H and TMR0L registers
- T0CS in T0CON reg. determines the clock source,
  - If T0CS = 1, the timer is used as a counter
  - Counts up as pulses are fed from pin RA4 (T0CKI)
  - What does T0CON=0110 1000 mean?
- If TMR1CS=1, the timer 1 counts up as clock pulses are fed into pin RC0

Timer and counter





Sinhgad Institutes

TMR2: Timer2- 8-bit

- 8-bit period register (PR2)- Fixed value
- TMR2 and PR2 are readable and writable
- TMR2 increments from 00 to the value equal to PR2
- TMR2IF flag from PIR1 reg. is raised and TMR2 reset to 00
- The clock source for TMR2 is Fosc/4 for both prescaler and Postscaler options.
- There is no external clock source ,hence cant not used as counter
- Three prescale values (Bit1-Bit0) and 16 postscale values (Bit6-Bit3)
- Flag (TMR2IF) is set when TMR2 matches PR2: Can generate an interrupt

TMR2: Timer2- SFRS

(T2CON)

B7	B6	B5	B4	B3	B2	B1	B0
------	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
	Postscale Select Bits					Prescale Select Bits	

Postscale Select Bits:

1111 = 1: 16
1110 = 1: 15
⋮
0001 = 1:2
0000 = 1: 1

TMR2ON:

1 = EnablesTimer
0 = Disables Timer

Prescale Select Bits:

1X = 1:16
01 = 1: 4
00 = 1: 1

(PIR1)

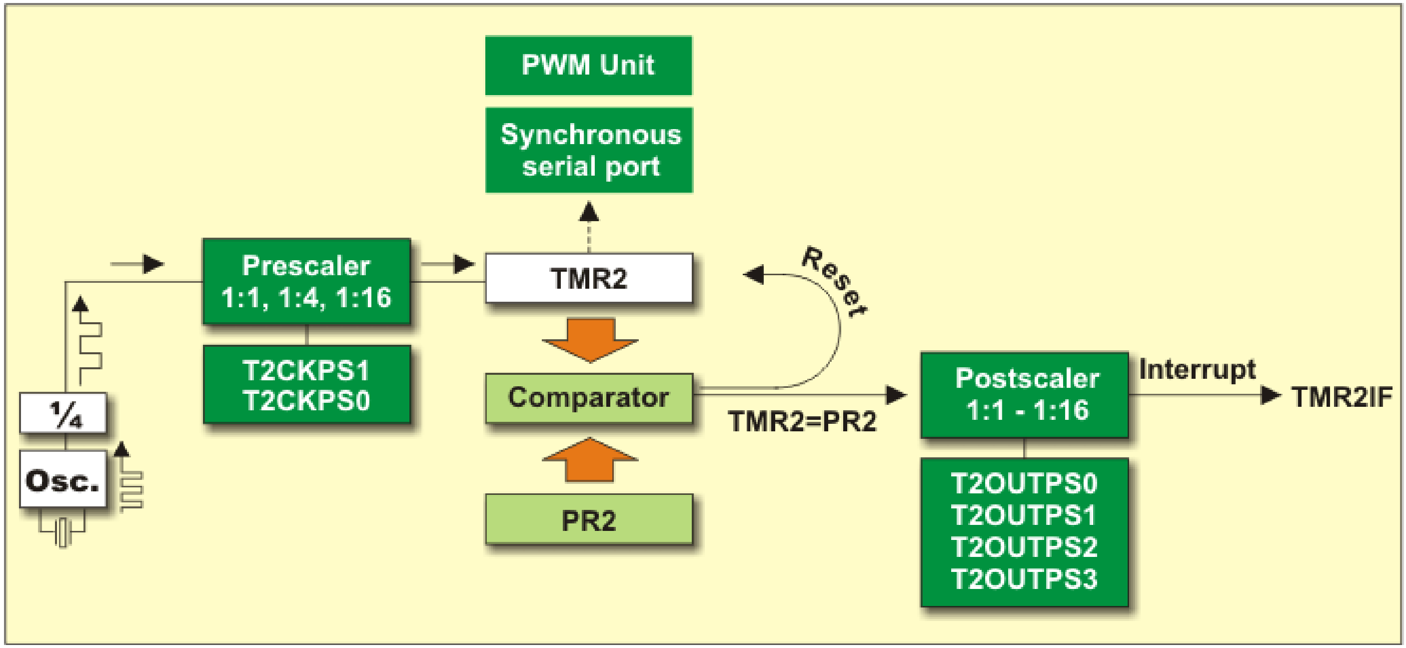
B7	B6	B5	B4	B3	B2	B1	B0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

TMR2IF: Timer2 Interrupt overflow flag Bit

0-TMR2 value is not equal to PR2 register
1- TMR2 value is equal to PR2 register

Timer2- Block Diagram

- Timer2 operation : 8-bit number is loaded in PR2
- When TMR2 and PR2 match: Output pulse is generated and the timer is reset
- Output pulse goes through postscaler: Sets the flag TMR2IF

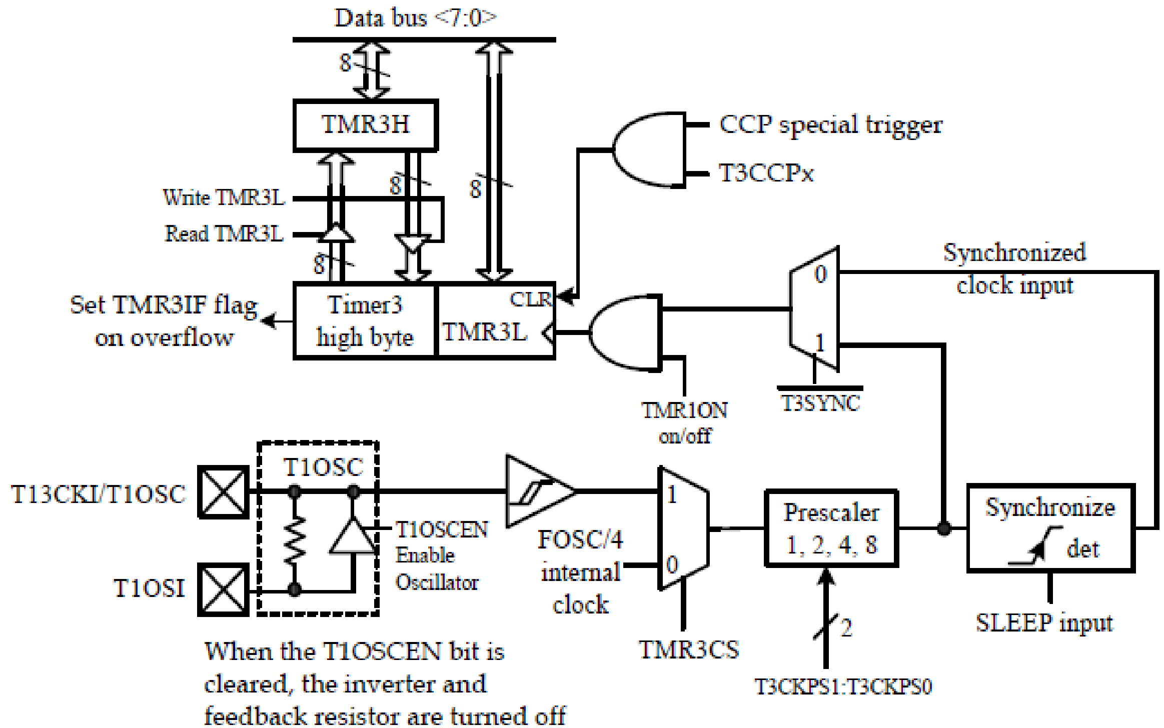


When using the TMR2 timer, :one should know:

- Upon power-on, the PR2 register contains the value FFh;
- Both prescaler and postscaler are cleared by writing to the TMR2 register;
- Both prescaler and postscaler are cleared by writing to the T2CON register; and
- On any reset, both prescaler and postscaler are cleared.

TMR3: Timer3- 16-bit

- Programmed in 16-bit mode only and does not support 8-bit mode
- It has 2 bytes named as TMR3L and RMR3H [It can count up 65.535 pulses in a single cycle]
- Has four Prescale values [1:1,1:2,1:4,1:8]
- It has SFR as T3CON and TMR3IF
- Generates an interrupt or sets a flag when it overflows
- TMR3IF : Flag must be cleared to start the timer again and goes high when TMR3H:TMR3L overflow from FFFF to 0000h occurs. It is part of PIR2 reg.



TMR3: Timer3- 16-bit -SFRS

Sinhgad Institutes

T3CON

	7	6	5	4	3	2	1	0
	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
value after reset	0	0	0	0	0	0	0	0

RD16: 16-bit read/write mode enable bit
- 0 = Enables read/write of Timer3 in two 8-bit operations
- 1 = Enables read/write of Timer3 in 16-bit operation

T3CCP2:T3CCP1: Timer3 and Timer1 to CCPx enable bits
- 00 = Timer1 and Timer2 are the clock sources for CCP1 through CCP5
- 01 = Timer3 and Timer4 are the clock sources for CCP2 through CCP5; Timer1 and Timer2 are the clock sources for CCP1
- 10 = Timer3 and Timer4 are the clock sources for CCP3 through CCP5; Timer1 and Timer2 are the clock sources for CCP1 and CCP2
- 11 = Timer3 and Timer4 are the clock sources for CCP1 through CCP5

T3CKPS1:T3CKPS0: Timer3 input clock prescale select bits
- 00 = 1:1 prescale value
- 01 = 1:2 prescale value
- 10 = 1:4 prescale value
- 11 = 1:8 prescale value

$\overline{\text{T3SYNC}}$: Timer3 external clock input synchronization select bit

When TMR3CS = 1
- 0 = Synchronizes external clock input
- 1 = Do not synchronize external clock input

When TMR3CS = 0
- This bit is ignored.

TMR3CS: Timer3 clock source select bit
- 0 = Instruction cycle clock (FOSC/4)
- 1 = External clock from pin RC0/T1OSO/T13CKI

TMR3ON: Timer3 on bit
- 0 = Stops Timer3
- 1 = Enables Timer3

PIE2 Register

OSCIE	CMIE	---	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE

Sinhgad Institutes

Comparison of Timers

	TIMER 0	TIMERS 1 & 3	TIMERS 2 & 4
SIZE OF REGISTER	8-bits or 16-bits	16-bits	8-bits
CLOCK SOURCE (Internal)	Fosc/4	Fosc/4	Fosc/4
CLOCK SOURCE (External)	T0CKI pin	T13CKI pin or Timer 1 oscillator (T1OSC)	None
CLOCK SCALING AVAILABLE (Resolution)	Prescaler 8-bits (1:2→1:256)	Prescaler 2-bits (1:1, 1:2, 1:4, 1:8)	Prescaler (1:1,1:4,1:16) Postscaler (1:1→1:16)
INTERRUPT EVENT	On overflow FFh→00h	On overflow FFFFh→0000h	TMR REG matches PR2
CAN WAKE PIC FROM SLEEP?	NO	YES	NO

Timer :mode , Time and Count selection

To select mode:

- Timer mode is selected by the T0CS bit of the T0CON register, (T0CS: 0=timer, 1=counter);
- When used, the prescaler should be assigned to the timer/counter by clearing the PSA bit of the T0CON register. The prescaler rate is set by using the PS2-PS0 bits of the same register; and
- When using interrupt, the GIE and TMR0IE bits of the INTCON register should be set.

To measure time:

- Reset the TMR0 register or write some well-known value to it;
- Elapsed time (in microseconds when using quartz 4MHz) is measured by reading the TMR0 register; and
- The flag bit TMR0IF of the INTCON register is automatically set every time the TMR0 register overflows. If enabled, an interrupt occurs.

To count pulses:

- The polarity of pulses are to be counted is selected on the RA4 pin are selected by the TOSE bit of the T0CON register (T0SE: 0=positive, 1=negative pulses); and
- Number of pulses may be read from the TMR0 register. The prescaler and interrupt are used in the same manner as in timer mode.

Timer Delay Calculation

for XTAL = 10 MHz with No Prescaler

- General formula for delay calculation T = 4/(10MHz) = 0.4 usecond
  - Divide the desired Time delay by 0.4 µs
  - Perform 65536-n , N= required dealy/ 0.4µs
  - Convert decimal value to Hex – yyxx
  - Set TIMER0L=xx and TIMER0H =yy

(a) in hex	(b) in decimal
(FFFF – YYXX + 1) × 0.4 µs where YYXX are the TMR0H, TMR0L initial values respectively. Notice that YYXX values are in hex.	Convert YYXX values of the TMR0H, TMR0L register to decimal to get a NNNNN decimal number, then (65536 – NNNNN) × 0.4 µs

Programming timers 0 and 1

- Every timer needs a clock pulse to tick
- Clock source can be
  - **Internal** → 1/4th of the frequency of the crystal oscillator on OSC1 and OSC2 pins (Fosc/4) is fed into timer
  - **External:** pulses are fed through one of the PIC18's pins → Counter
- Timers are 16-bit wide
  - Can be accessed as two separate reg. (TMRxL & TMRxH)
  - Each timer has TCON (timer Control) reg.

REGISTERS ASSOCIATED WITH TIMER0

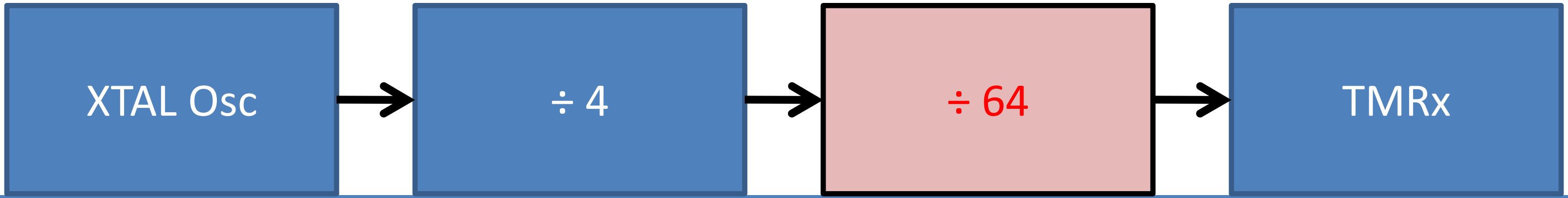
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMR0L	Timer0 Register Low Byte								54
TMR0H	Timer0 Register High Byte								54
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	53
INTCON2	$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	53
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	54
TRISA	—	TRISA6[1]	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	56

RCON Register

IPEN	SBOREN	---	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$

Prescaler and generating larger delay

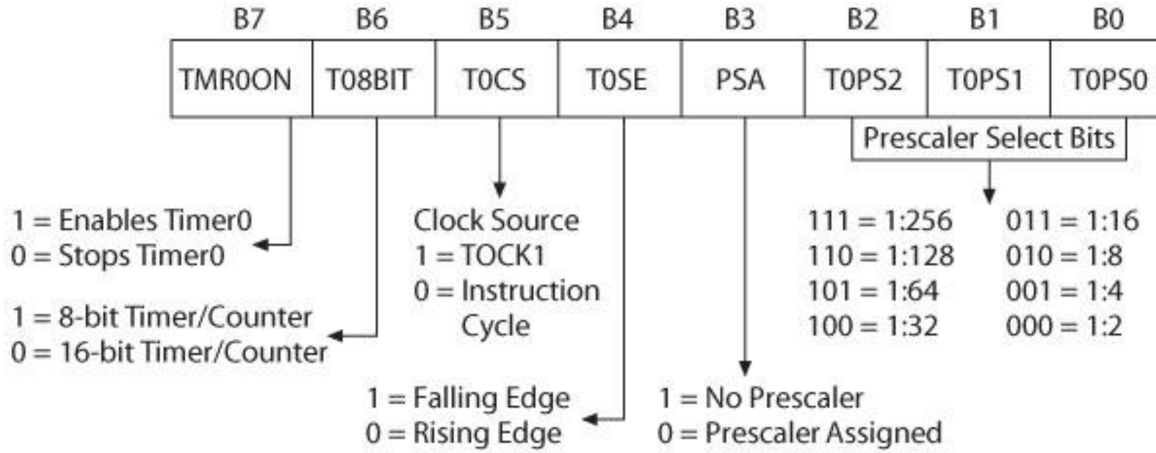
- The size of delay depend on
  - The Crystal frequency
  - The timer's 16-bit register.
- The largest timer happens when TMR0L=TMR0H=00
- Prescaler option is used to duplicate the delay by dividing the clock by a factor of 2,4, 8,16, 32,64 ,128,256
  - If T0CON=0000 0101, then T = 4*64/f



Summary

- The PIC18 can have up to four or more timers/counters. Depending on the family member
- Timers: Generate Time Delays (using Crystal)
- Counters: Event counter (using Pulse outside)
- Timers are accessed as two 8-bit registers, TMRLx and TMRHx
- Can be used either 8-bit or 16-bit
- Each timer has its own Timer Control register

Write a C18 program to toggle all bits of Port B continuously with delay of 10 ms using Timer 0 , 16 bit and no presclar




```
#include <P18FXXXX.h>
 void T0Delay(void);
 void main(void)
 {
    TRISB=0x00;
    While(1)
   {
      PORTB= 0x00;     // Load bit patterns
      T0Delay ( );
      PORTB= 0xFF;
      T0Delay ( );
   }
}
 void T0Delay ( )
{
  T0CON=0x08;                  // 0000 1000 Timer0, 16 bit, no prescaler
  TMR0H=0x9E;                  // load Higher yte in TMR0H
  TMR0L= 0x58;                 // Load Lower byte to TMR0L
  T0CONbits.TMR0ON=1;          // start the timer for upcount  9E58—FFFF, TMR0IF=1
  While(INTCONbits.TMR0IF==0); //  Check for overflow
  T0CONbits.TMR0ON=0;          //Turnoff timer
  INTCONbits.TMR0IF==0;        // clear the Timre0 flag
}
```

- Assume that Crystal frequency = 10 MHz
- Internal time delay = $4/(10*10^6) = 0.4*$ µs
- N= 10ms/0.4 µs = 25000
- Count= 65536-25000= $(40536)_{10}$
- Hex Value to be loaded =$(9E\ 58)_{16}$
- Load TMR0H=9E h and TMR0L=58h

Write a C18 program to generate frequency of 2500 Hz on PORTC.2 continuously using Timer 1 , 16 bit and no pre-scaler .

	R/W - 0	R-0	R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYN	TMR1SC	TMR1ON

```
#include <P18FXXXX.h>
void T1Delay(void);
#define mybit PORTCbits.RC2
void main (void)
{
        TRISCbits.TRISC2=0;
        while(1)
       {
          mybit^=1;
          T1Delay ( );
       }
  }

void T1Delay ( )
{
   T1CON=0x00;                    // Timer1, 16 bit, no pre-scaler
   TMR1H=0xFE;                    // load Higher byte in TMR1H
   TMR1L= 0x06;                   // Load Lower byte to TMR1L
   T1CONbits.TMR1ON=1;       // Start the timer for up count
   while(PIR1bits.TMR1IF==0);//  Check for overflow
   T1CONbits.TMR1ON=0;       // Turn off timer
   PIR1bits.TMR1IF==0;          // Clear the Timer1 flag
}
```

Solution : Calculation of TMR1H and TMR1L values

1. Assume that Crystal frequency = 10 MHz
2. For 2500 Hz frequency, Total time T= 1/ 2500 Hz= 400 µs i.e. $T_{on} = T_{off}$ = 200 µs
3. Internal time delay = 4/(10*106) = 0.4 µs
4. N= 200/0.4 µs = 500
5. Count = 65536 – 550 = $(65036)^{10}$
6. Hex value to be loaded = $(FE\ 0C)^{16}$
7. Load TMR1H = FF H and TMR1L = 06 H

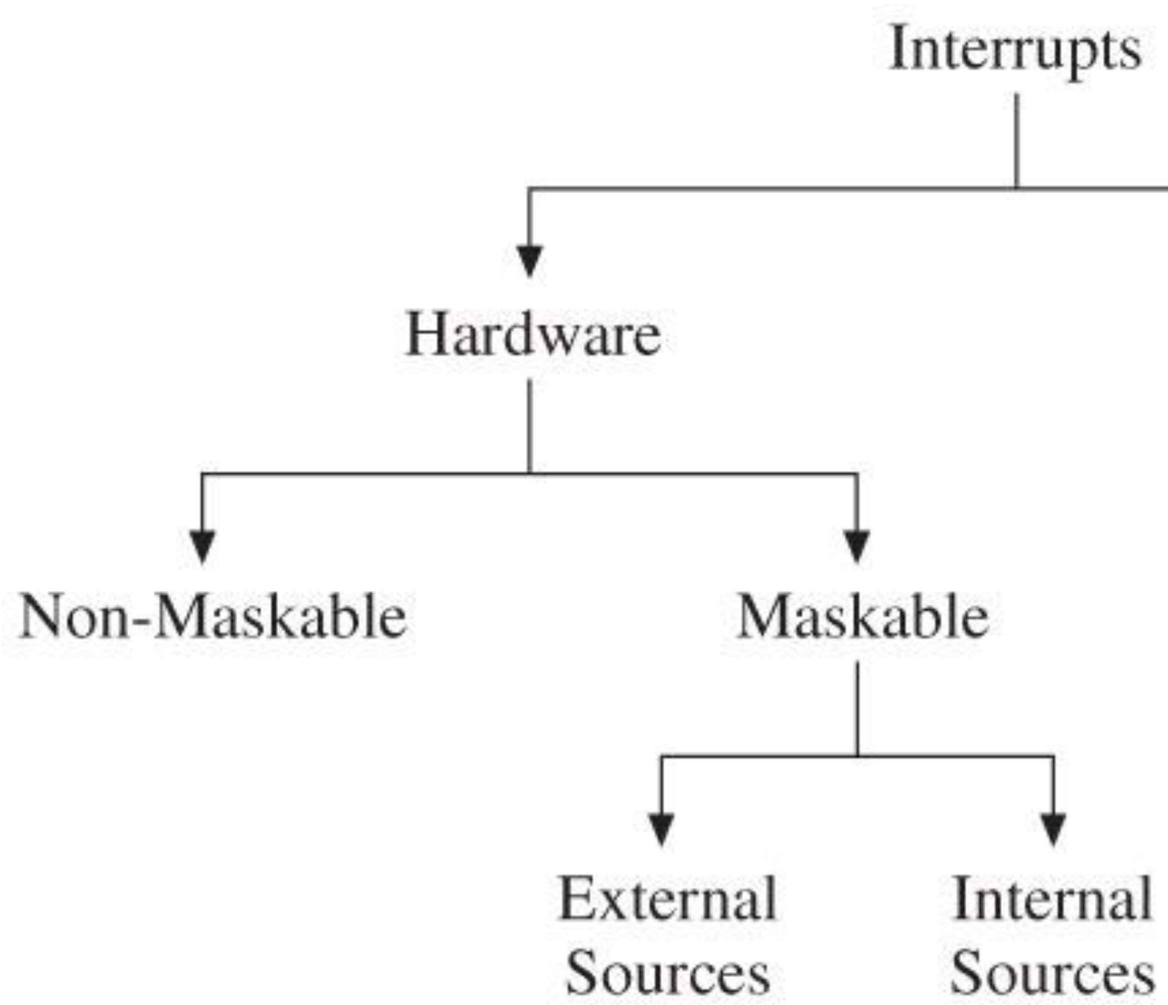
Sinhgad Institutes

Find the largest time delay that can be generated using Timer2, Using pre-scaler and post-scaler

	U - 0	R - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0
T2CON	.....	T2OUTPS3	T2OUTPS3	T2OUTPS3	T2OUTPS3	TMR2ON	T2CKPS1	T2CKPS0

```
#include <P18F4550.h>
#define mybit PORTCbits.RC2
void T1Delay(void);
void main (void)
{
    TRISCbits.TRISC2=0;
    T2CON=0X7B;              // Timer2, pre-scale=post=16
    TMR2=0X00;
    while(1)
    {
      PR2=255;                        // Load PR2 for highest value
      T2CONbits.TMR2ON=1;         // Start the timer
      while(PIR1bits.TMR2IF==0);  // Check for Timer2 flag
      mybit=~mybit;                   // Toggle the bits
      T2CONbits.TMR2ON=0;         // Turn off timer
       PIR1bits.TMR2IF==0;           // Clear the Time1 flag
    }
 }
```

Types of Interrupts



An interrupt VS Polling

PIC18 has two vectors: High and Low

Interrupt vs Polling Methods

Sr.	Interrupts	Polling
01	Peripheral Request Microcontroller service	Microcontroller continuously Monitors the status of device
02	Efficient – ISR	Not efficient
03	Uses Priority Method to serve the request	Uses Round Robine Method to serve the devices
04	Microcontroller can ignore the request	It does not happen in Polling
05	Requires Less time in execution	It wastes the time of Microcontroller
06	Timer interrupts are used to stop the execution	Checks for statistical conditions

PIC18 Interrupt sources

- Timer interrupts , TMR0IF, TMRIF--- etc. Timer Rollover Events-Software
- 3 or 4 External Interrupts (INT0-INT3): Hardware
  - Three pins of PORTB :RB0/INT0, RB1/INT1,and RB2/INT2 Can be used to connect external interrupting sources : Keypads or switches
  - Edge Triggered
  - Rising or Falling edge selected in INTCON2 register
- PORTB Interrupt on Change (RB4-RB7): External hardware
  - ✓ PORTB Interrupt (RBI): Change in logic levels of pins RB4-RB7
- Comparator Output Change
- A/D Conversion Complete
- Communication Channel Events– Receiver and Transmitter - Serial I/O
- CCP and Other Peripheral Events…

Interrupt Sources- A Glimpse

Timer0 interrupt
External interrupt on RB0/INT (programmable edge trigger)
PortB change interrupt (RB7:RB4)
Parallel Slave Port interrupt
Analog to digital interrupt
UART Receive interrupt
UART Transmit Empty interrupt
Synchronous Serial Port interrupt

T0IF	INTF	RBIF	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	EEIF	BCLIF	CCP2IF
Yes	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Compare/Capture/PWM-1 interrupt
Timer2 interrupt
Timer1 interrupt
EEPROM interrupt
Bus Collision(I^2C) interrupt
Compare/Capture/PWM-2 interrupt

RESET	000000
High Priority Interrupt	000008-000017H , GOTO instruction for ISR
Low Priority Interrupt	000018

MPU Response to Interrupts

- When interrupts are enabled
  - MPU checks interrupt request flag at the end of each instruction
- If interrupt request is present, the MPU
  - Resets the interrupt flag
  - Saves the return address on the stack
- MPU redirected to appropriate memory location
  - Interrupt vectors

RESET	000000
High Priority Interrupt	000008
Low Priority Interuupt	000018

- Interrupt service routine (ISR) meets request
- MPU returns to where it was interrupted
  - Specific return instruction

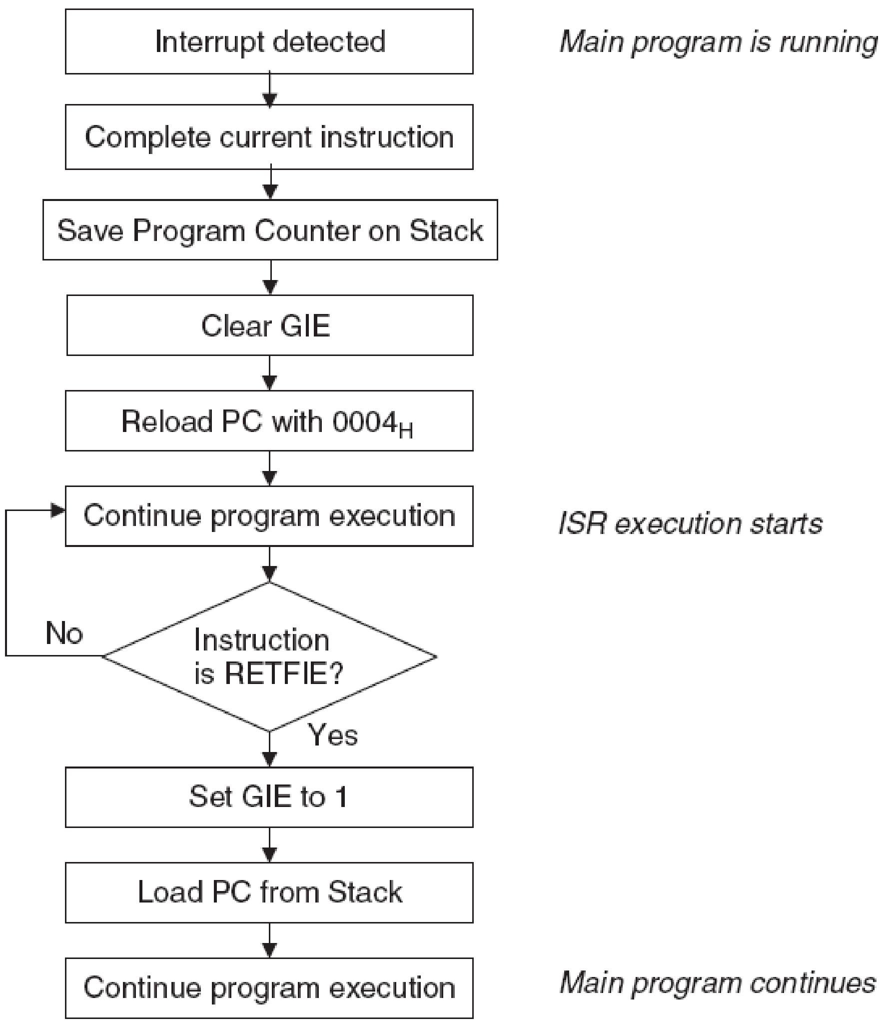
Steps in executing an Interrupts

Sinhgad Institutes

Steps In Execution of Interrupt

1. It finishes the instruction it is executing and saves the address of next instruction on the stack.
2. It jumps to a fixed location in the memory called as interrupt vector table and IVT diverts the Microcontroller to ISR.
3. It Executes the ISR until it reaches to last instruction of the subroutine which is RETFIE Upon executing RETFIE instruction, Microcontroller returns to the place from where it was interrupted.
4. First it gets the PC address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.

PIC Response to an Interrupt



Vector Locations

RESET : 000000

High Priority Interrupt : 000008-000017H. ,

Low Priority Interuupt : 000018

Interrupt Service Routine (ISR)

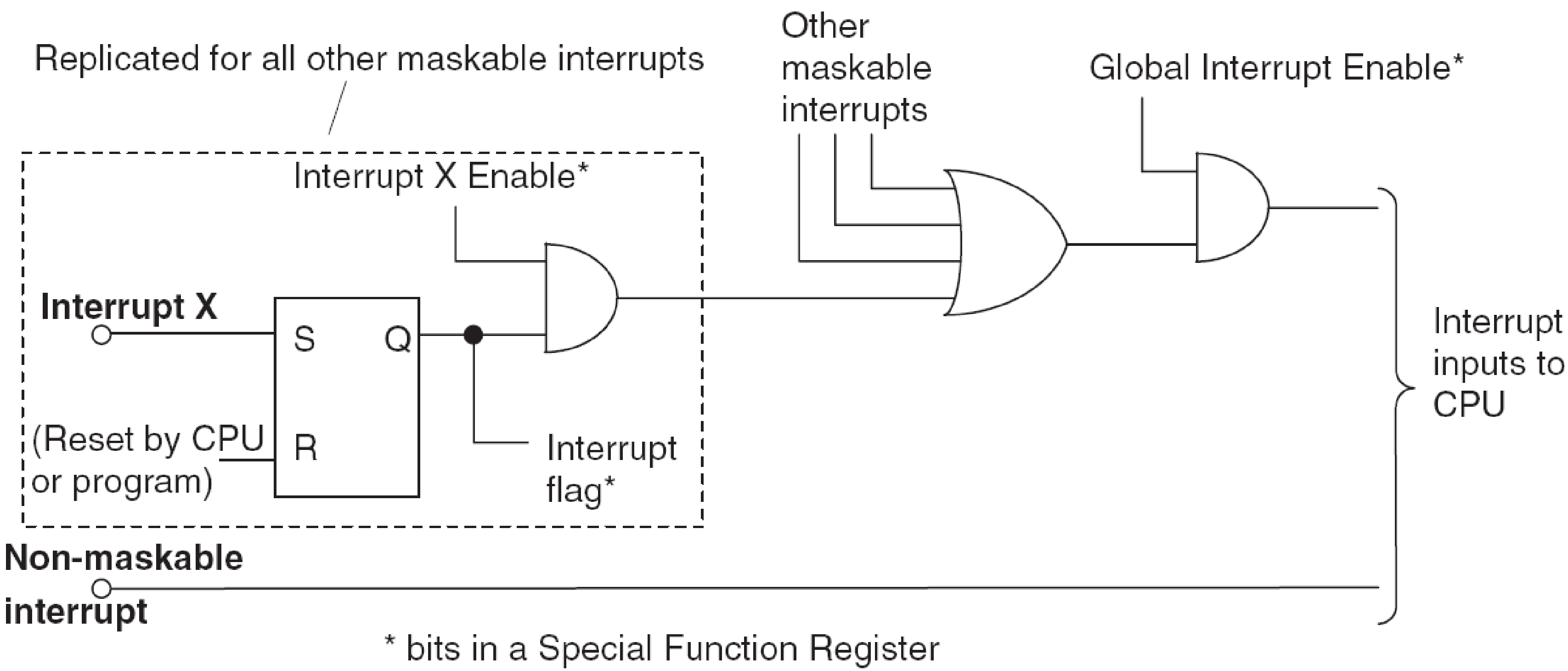
- Similar to a subroutine
- Attends to the request of an interrupting source
  - Clears the interrupt flag
  - Should save register contents that may be affected by the code in the ISR
  - Must be terminated with the instruction RETFIE
- When an interrupt occurs, the MPU:
  - Completes the instruction being executed
  - Disables global interrupt enable
  - Places the return address on the stack

- High-priority interrupts (0x00008) :The contents of W, STATUS, and BSR registers are automatically saved into respective shadow registers.
- Low-priority interrupts(0x00018): These registers must be saved as a part of the ISR, If they are affected
- RETFIE [s] ;Return from interrupt
- RETFIE FAST ;FAST equivalent to s = 1
  - If s =1: MPU also retrieves the contents of W, BSR, and STATUS registers

PIC18 Interrupt Sources - SFRS

- Special Function Registers (SFRs)
  - RCON
    - Priority Enable
  - INTCON
    - External interrupt sources
  - IPR, PIE, and PIR
    - Internal peripheral interrupts
- Valid interrupt
  - Interrupt request bit (flag)(IF)
  - Interrupt enable bit (IE)
  - Priority bit (IP)

General Interrupt Structure



RESET	000000
High Priority Interrupt	000008-000017H , GOTO instruction for ISR
Low Priority Interuupt	000018

Interrupt Priority -Enable

- Interrupt priorities
  - High-priority interrupt vector 000008_H
  - Low-priority interrupt vector 000018_H
  - A high-priority interrupt can interrupt a low-priority interrupt in progress.
  - Interrupt priority enable
    - Bit7 (IPEN) in RCON register

RCON

RCON Register

IPEN	SBOREN	---	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

Reset Bits

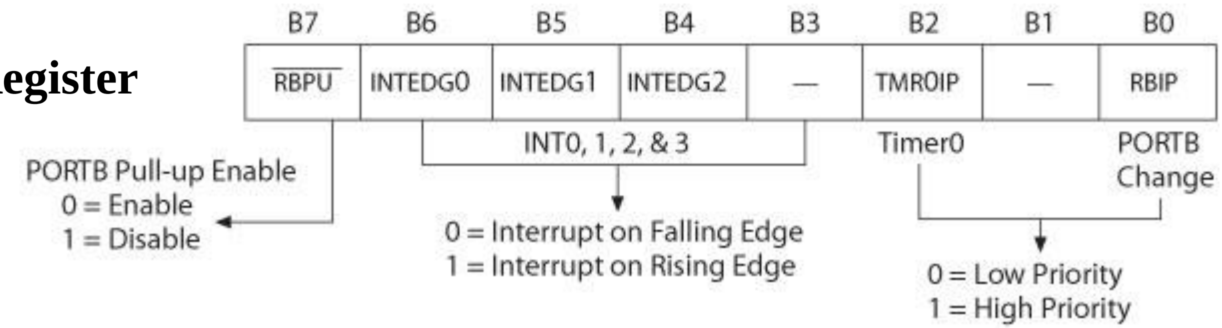
IPEN: Interrupt Priority Enable
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupt

External Interrupts- INT0,INT1,INT2

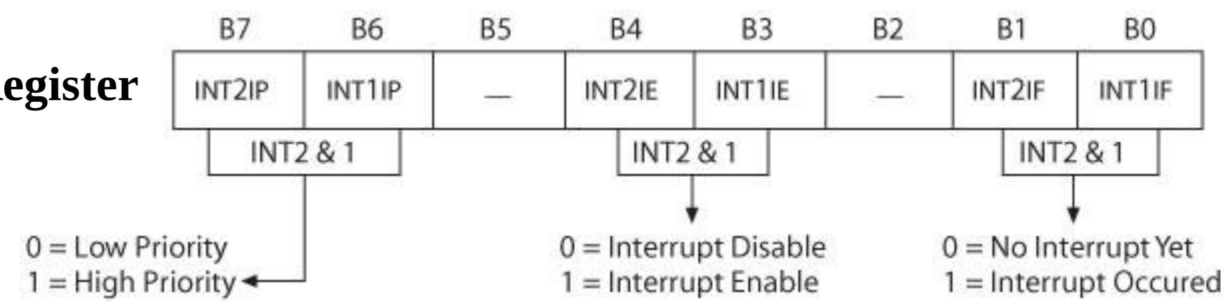
INTCON Register

B7	B6	B5	B4	B3	B2	B1	B0
GIE/GEIH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

INTCON2 Register



INTCON3 Register



Internal Interrupt Registers-Timers, ADC & Serial I/O

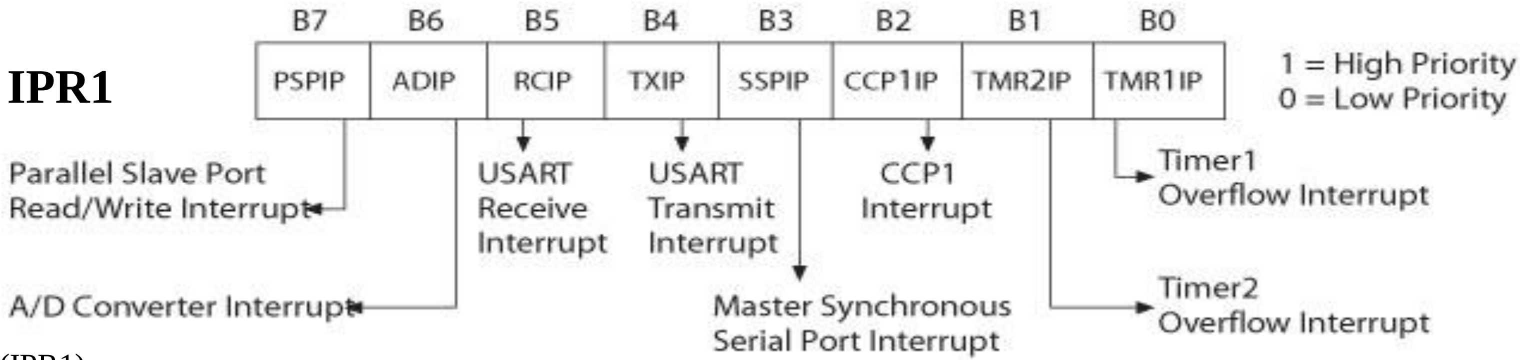
Interrupt Sources

1. Timers
2. ADC
3. Serial I/O

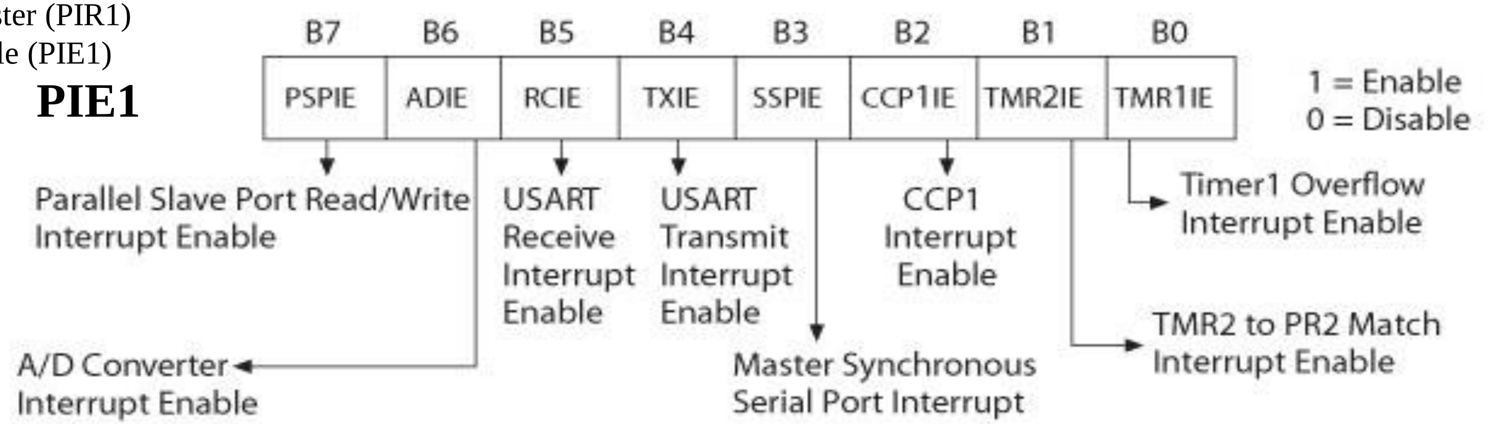
Interrupt Registers

1. Interrupt Priority Register(IPR1)
2. Peripheral Interrupt Register (PIR1)
3. Peripheral Interrupt Enable (PIE1)

IPR1



PIE1



PIR1

B7	B6	B5	B4	B3	B2	B1	B0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

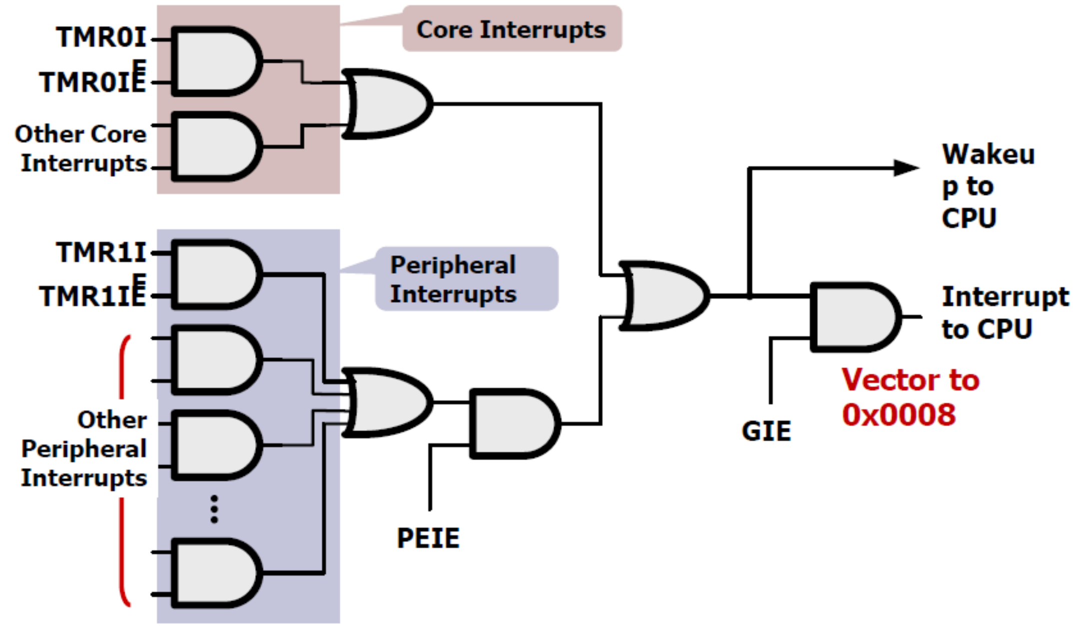
IPR2

OSCIP	CMIP	---	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP

OSCIP: Oscillator Fail Interrupt Priority
CMIP: Comparator Interrupt Priority
----: Unimplemented Bit
EEIP: Data EEPROM/Flash Write Operation Interrupt Priority
BCLIP: Bus Collision Interrupt Priority
HLVDIP: High/Low Voltage Detect Interrupt Priority
TMR3IP: Timer3 Interrupt Priority
CCP2IP: CCP2 Interrupt Priority

Sinhgad Institutes

Interrupt structure (Legacy Mode)internal



1. It is used for wake-up to CPU . Or serve the high priority interrupts
2. It uses the both internal and peripheral interrupts to wake-up CPU .
3. INTCON register is used to enable and disable the Core (TMR0IF,INT0IF --) and Peripheral interrupts
4. CPU will get weak-up call with Core and Peripheral Interrupts when GIE bit of INTCON is high
5. When PEIE and GIE bit of INTCON is high , Peripheral Interrupts cause the CPU to wake-up

Interrupt registers

RCON: Priority Enable

INTCON: External interrupt sources

IPR, PIE, and PIR: Internal peripheral interrupts

Valid interrupt

Interrupt request bit (flag)(IF)

Interrupt enable bit (IE)

Priority bit (IP)

Vector Locations

RESET : 000000

High Priority Interrupt : 000008-000017H. ,

Low Priority Interrupt : 000018

RCON Register

IPEN	SBOREN	---	RI	TO	PD	POR	BOR

INTCON:

B7	B6	B5	B4	B3	B2	B1	B0
GIE/GEIH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

PIR1

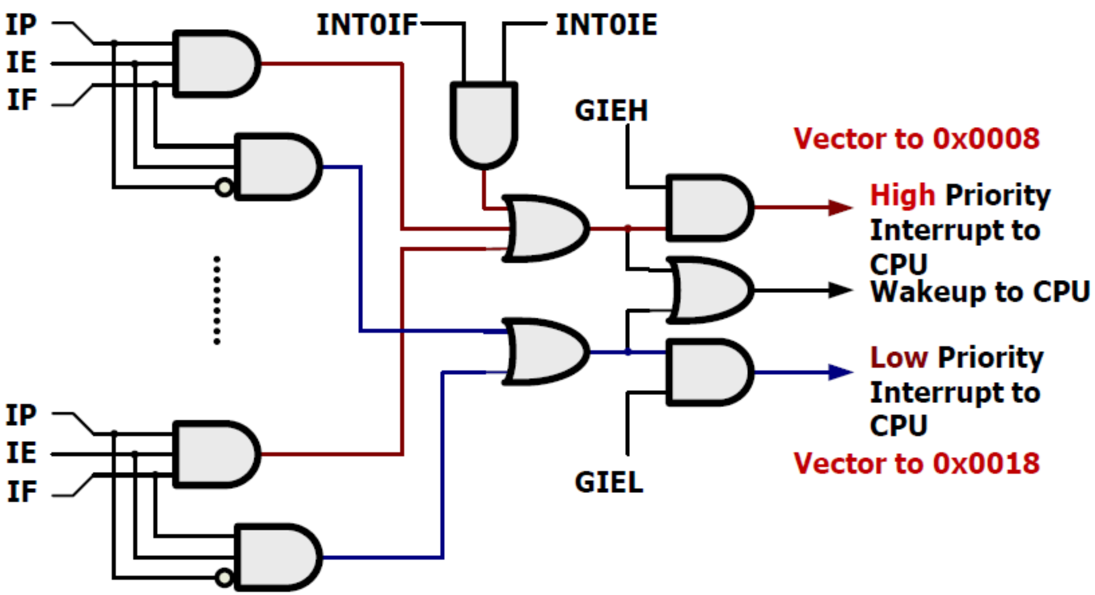
B7	B6	B5	B4	B3	B2	B1	B0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

IPR1

B7	B6	B5	B4	B3	B2	B1	B0
PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP

Sinhgad Institutes

Interrupt structure (Priority Mode)External



1. It is used on the basis of priority for wake up to CPU .
2. Each interrupt has three registers , IF, IP, and IE to check for Valid interrupt, decide priority and enable .
3. INTCON register is used to enable and disable the High and Low priority interrupts
4. CPU will get weak-up call on either low or High priority interrupt is enable
5. When PEIE/GIEL bit of INTCON is high , Low priority interrupts are enable (actually No priority)
6. When GIEH bit of INTCON is high , High priority interrupts are enable (IPR1 register is in active mode)

Interrupt registers

RCON: Priority Enable

INTCON: External interrupt sources

IPR, PIE, and PIR: Internal peripheral interrupts

Valid interrupt

Interrupt request bit (flag)(IF)

Interrupt enable bit (IE)

Priority bit (IP)

Vector Locations

RESET : 000000

High Priority Interrupt : 000008-000017H. ,

Low Priority Interrupt : 000018

RCON Register

IPEN	SBOREN	---	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

INTCON:

B7	B6	B5	B4	B3	B2	B1	B0
GIE/GEIH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

PIR1

B7	B6	B5	B4	B3	B2	B1	B0
PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

IPR1

B7	B6	B5	B4	B3	B2	B1	B0
PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP

Interrupt structure (Legacy Mode)

1. Operates on the internal operations as reset (MCLR), data (RB7) and clock (RB6) signals. MCLR is used for device reset and RB6 for serial clock, RB7 for serial data.
2. Even when the dedicated port is enabled, the ICSP functions remain available through the legacy port. When VIHH is seen on the MCLR/VPP/RE3 pin, the state of the ICRST/ICVPP pin is ignored.
3. The ICPRT Configuration bit can only be programmed through the default ICSP port (MCLR/RB6/RB7).
4. The power-managed Sleep mode in the PIC18F2455/2550/4455/4550 devices is identical to the legacy Sleep mode offered in all other PIC devices.

How to create an ISR-Programming Support

Sinhgad Institutes

1 Write a prototype for your interrupt handler function

```
void HighISR(void);

#pragma code HighVector=0x08
void IntHighVector(void)
{
    _asm goto HighISR _endasm
}


#pragma interrupt HighISR
void HighISR(void)
{
    /* YOUR CODE HERE */
}
```

- This is the prototype for the function that will be called whenever an interrupt occurs
- Give it any name you like
- It is an ordinary function prototype except:
  - The return type must be `void`
  - The parameter list must be `void`

When there are multiple requests, The interrupt source must be identified by checking the interrupt flags

Programmer responsibility: 1. Provide code for interrupt vector,2.Provide Interrupt Service Routine (ISR)

How to create an ISR

2 Start a new code section for the interrupt vector

```
void HighISR(void);

#pragma code HighVector=0x08
void IntHighVector(void)
{
    _asm goto HighISR _endasm
}


#pragma interrupt HighISR
void HighISR(void)
{
    /* YOUR CODE HERE */
}
```

- The section must be named, but the name is arbitrary
- Force the section to be located at the desired interrupt vector address:
  - High Priority = 0x08
  - Low Priority = 0x18

How to create an ISR

3 Populate the interrupt vector

```
void HighISR(void);

#pragma code HighVector=0x08
void IntHighVector(void)
{
    _asm goto HighISR _endasm
}


#pragma interrupt HighISR
void HighISR(void)
{
    /* YOUR CODE HERE */
}
```

- Write a function to be located at the interrupt vector
  - The name is arbitrary, but it must not be the same as the section name
  - The return type and parameter list must be `void`
- Insert one line of inline assembly code to `goto` your interrupt handler function (`HighISR`)

How to create an ISR

4 Tell compiler which function is an interrupt handler

```
void HighISR(void);

#pragma code HighVector=0x08
void IntHighVector(void)
{
     _asm goto HighISR _endasm
}


#pragma interrupt HighISR
void HighISR(void)
{
     /* YOUR CODE HERE */
}
```

- This tells the compiler to treat this function a bit differently from ordinary functions (more soon...)
- For high priority interrupt handler use: `#pragma interrupt`
- For low priority interrupt handler use: `#pragma interruptlow`

Sinhgad Institutes

How to create an ISR

5 Write your interrupt handler function

```
void HighISR(void);

#pragma code HighVector=0x08
void IntHighVector(void)
{
    _asm goto HighISR _endasm
}


#pragma interrupt HighISR
void HighISR(void)
{
    /* YOUR CODE HERE */
}
```

- Just like a normal function except:
  - Return type must be `void`
  - Parameters must be `void`
  - May need to do context save and restore on any global variables you modify
  - If global variable is intentionally changed by interrupt, the variable must be declared as `volatile`
  - You are still responsible for clearing interrupt flags

Programming Timer interrupt

Timer Interrupt Flag bits and registers

Sr. No	Interrupt	Flag Bit	Register	Enable bit	Register
1	Timer0	TMR0IF	INTCON	TMR0IE	INTCON
2	Timer1	TMR1IF	PIR1	TMR1IE	PIE1
3	Timer2	TMR2IF	PIR1	TMR2IE	PIE1
4	Timer3	TMR3IF	PIR3	TMR3IE	PIE2

INTCON

D7	D6	D5	D4	D3	D2	D1	D0
GIE/GIEH	PEIE/GIEL	TMROIE	INT0IE		TMR0IF		

PIR1

D7	D6	D5	D4	D3	D2	D1	D0
PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

PIR2

D7	D6	D5	D4	D3	D2	D1	D0
OSCIF	CIF	---	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF

Programming Timer interrupt

```
#pragma code high_vector=0x0008;
Void My_Hivect_Int(void)
{
                _asm
GOTO my_isr
_endasm
}
 #pragma code

#pragma interrupt my_isr
Void my_isr (void)
{
Places RETFIE here automatically
}

#include<P18F4550.h>
#define myPB1bit PORTBbits.RB1
#define myPB7bit PORTBbits.RB7
Void T0_ISR (Void);
Void T1_ISR (Void);
#pragma interrupt chk_isr
```

```
Void  chk_isr (void);
{
If (INTCONbits.TMR0IF == 1;
T0_ISR ();
If (INTCONbits.TMR1IF == 1;
T1_ISR ();
}

#pragma code My_HiPrio_Int=0x0008;
Void My_HiPrio_Int(void)
{
  _asm
GOTO chk_isr
_endasm
}
 #pragma code

Void  main (void)
{
  TRISBbits.TRISRB1=0;
  TRISBbits.TRISRB7=0;
  TRISD=0
   TRISC=255;
```

Timer Interrupt Program

Write program to generate the delay of 100 ms using Timer Interrupt Program

```
void interrupt low_priority timerinterrupt(void)
{
 if  (TMR0IF == 1)//If timer0 interrupt flag is set.....
   {
       T0CONbits.TMR0ON = 0;    // Stop the timer
     INTCONbits.TMR0IF = 0;
     TMR0H = 0xED;
     TMR0L = 0xB0;
     LATB =~LATB;
     T0CONbits.TMR0ON = 1;     // Start the timer
   }
}

void main(void)
{
   TRISB = 0x00;
   LATB = 0xFF;
   RCONbits.IPEN = 1;
   INTCONbits.GIEH = 1;
   INTCONbits.GIEL = 1;
   INTCONbits.TMR0IE = 1;
   INTCONbits.TMR0IF = 0;
   INTCON2bits.TMR0IP = 0;
   T0CON = 0x07;// Stop the timer, Run in 16-bit mode, 1:256 prescaler
   TMR0H = 0xED;
   TMR0L = 0xB0;
   T0CONbits.TMR0ON = 1; // Start the timer
   while(1);
 }
```

RCON Register

IPEN	SBOREN	---	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMR0L	Timer0 Register Low Byte								54
TMR0H	Timer0 Register High Byte								54
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	53
INTCON2	$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	53
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	54
TRISA	—	TRISA6[1]	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	56

```
/*CALCULATIONS of Delay
 * required time = 100ms
 * TMR value=0xFFFF-[(required time)/(4*Tosc*Prescaler)]
 *          =0xFFFF-[(0.1*48000000)/(4*256)]
 *          =0xFFFF-0x124F
 * TMR      =0xEDB0
 * TMRH = 0xED
 * TMRL = 0xB0
 */
```

or

No of MC = Required time/(4*Tosc*Prescaler)

Capture, Compare, and PWM (CCP) Modules

- CCP modules : Registers specially designed to perform the following functions (in conjunction with timers as resources)
- Capture: The CCP pin can be set as an input to record the arrival time of a pulse. In this CCP module may use either Timer1 or Timer3 to operate
- Compare: The CCP pin is set as an output, and at a given count, it can be driven low, high, or toggled.
- Pulse width modulation (PWM): The CCP pin is set as an output and the duty cycle of a pulse can be varied. In PWM mode, either Timer2 or Timer4 may be used.

- Pulse Width Modulation
  - ➢ Duty cycle
    Percentage ratio of on time of a pulse to its period
  - ➢ Changing of the duty cycle is defined as PWM
    CCP pin is set as an output
    Count for period and duty cycle loaded into CCP registers
    Varying the duty cycle generates PWM

The operation of a CCP module is controlled by the CCPxCON register.

CCP Modules

- Capture, Compare, and Pulse Width Modulation (PWM) module is associated with a control register (CCPxCON) and a data register (CCPRx).
- The data register in turn consists of two 8-bit register: CCPRxL and CCPRxH.
- The CCP modules utilize Timers 1, 2, 3, or 4, depending on the module selected.
- CCPR1H (high) and CCPR1L (low)
  - 16-bit Capture register   16-bit Compare register
  - Duty-cycle PWM register
- Timer1 used as clock for Capture and Compare
- Timer2 used as clock for PWM
- The assignment of a particular timer to a module is determined by the bit 6 and bit 3 of the T3CON register
- CCP1

B7	B6	B5	B4	B3	B2	B1	B0
-----	----	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0

PWM Bit & Bit 0

Mode Select Bits – See Table 11.1

Least Significant Bits 1 and 0
of the 10-bit PWM Duty Cycle

CCPxCON register

	7	6	5	4	3	2	1	0
	--	--	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
value after reset	0	0	0	0	0	0	0	0

DCxB1:DCxB0: PWM duty cycle bit 1 and bit 0 for CCP module x

capture mode:

unused

compare mode:

unused

PWM mode:

These two bits are the lsbs (bit 1 and bit 0) of the 10-bit PWM duty cycle.

CCPxM3:CCPxM0: CCP module x mode select bits

0000 = capture/compare/PWM disabled (resets CCPx module)

0001 = reserved

0010 = compare mode, toggle output on match (CCPxIF bit is set)

0100 = capture mode, every falling edge

0101 = capture mode, every rising edge

0110 = capture mode, every 4th rising edge

0111 = capture mode, every 16th rising edge

1000 = compare mode, initialize CCP pin low, on compare match force CCP pin high (CCPxIF bit is set)

1001 = compare mode, initialize CCP pin high, on compare match force CCP pin low (CCPxIF bit is set)

1010 = compare mode, generate software interrupt on compare match (CCP pin unaffected, CCPxIF bit is set).

1011 = compare mode, trigger special event (CCPxIF bit is set)

For CCP1 and CCP2: Timer1 or Timer3 is reset on event

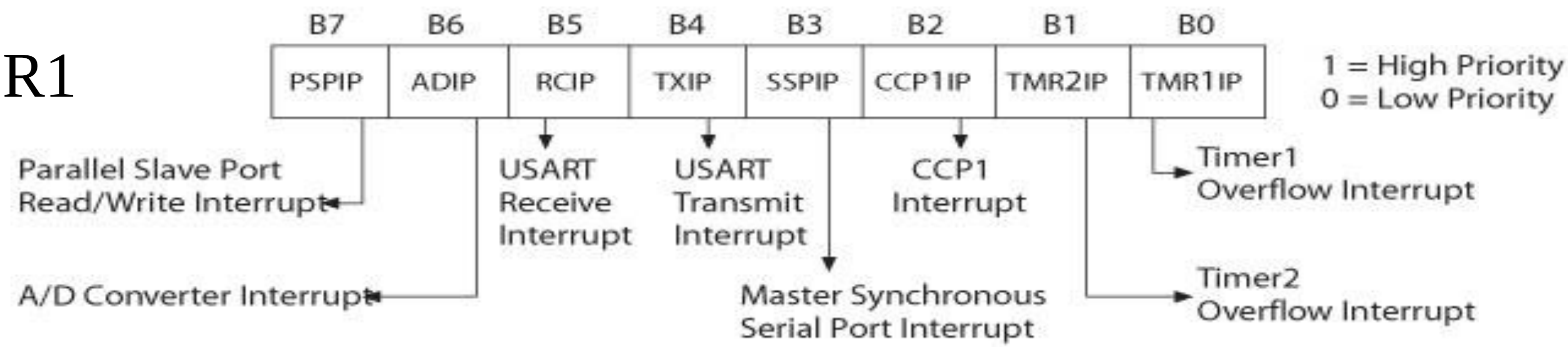
For all other modules: CCPx pin is unaffected and is configured as an I/O port.

11xx = PWM mode

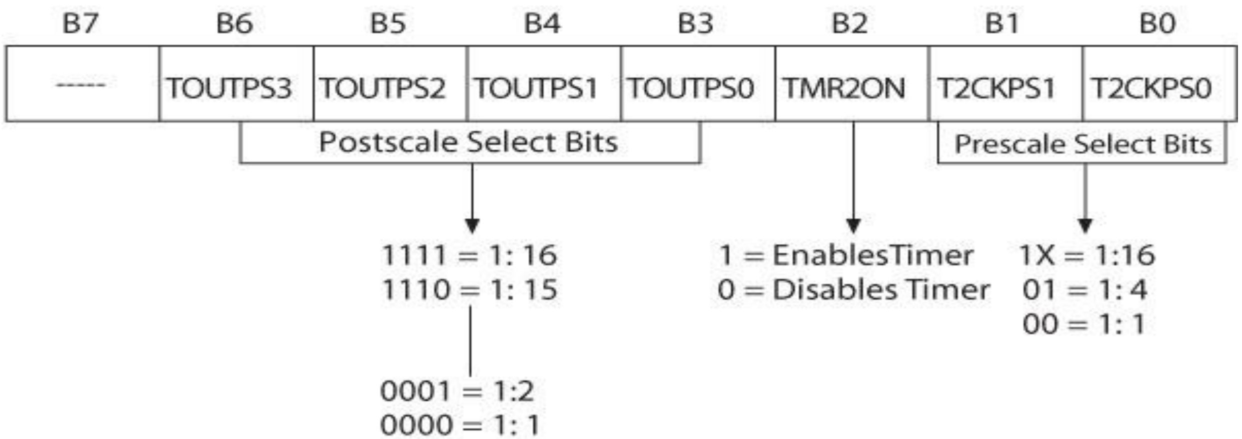
Sinhgad Institutes

Timer SFRS

IPR1



(T2CON)

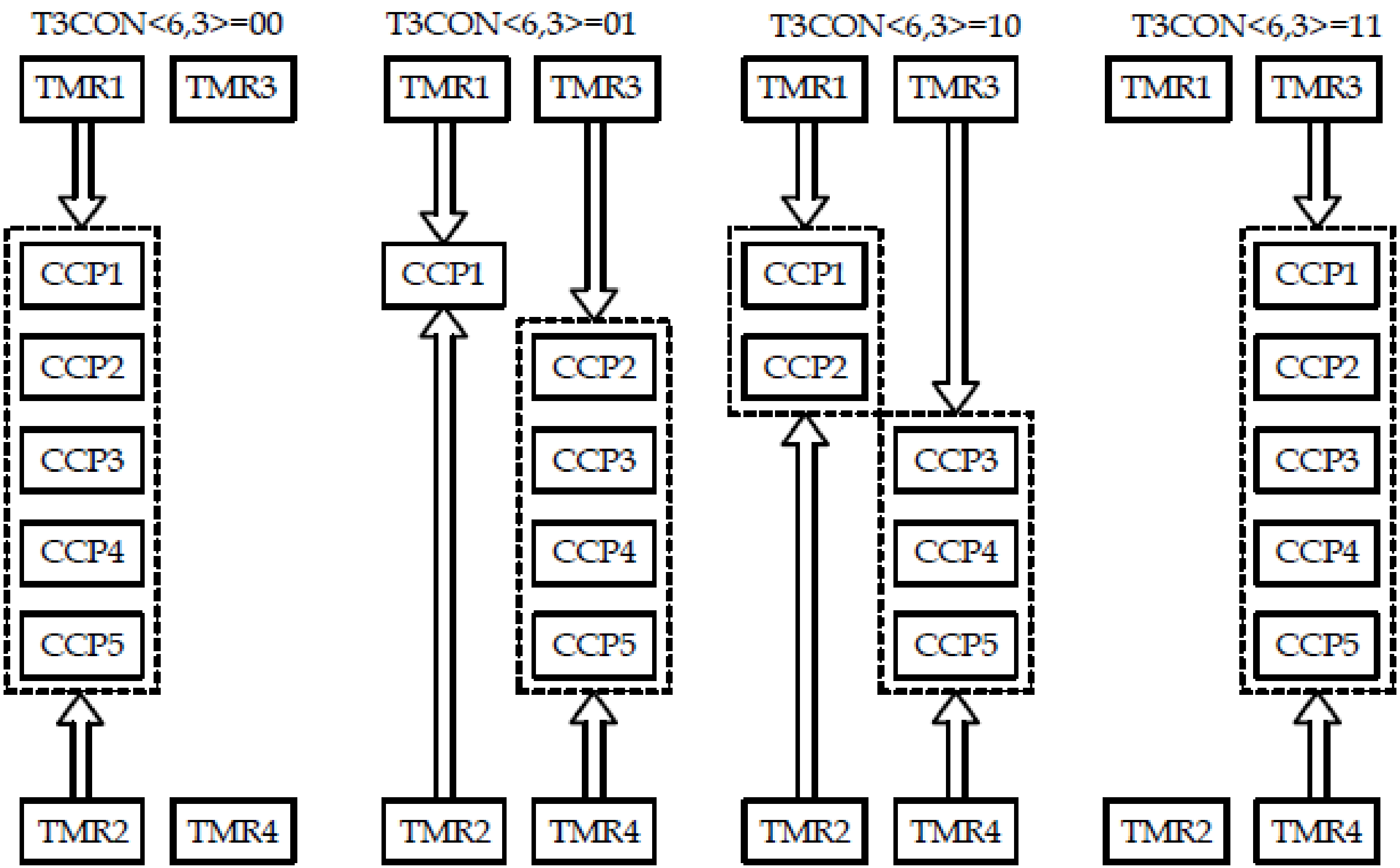


(T3CON)

	7	6	5	4	3	2	1	0
r	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
	0	0	0	0	0	0	0	0

CCP and Timer inter connected

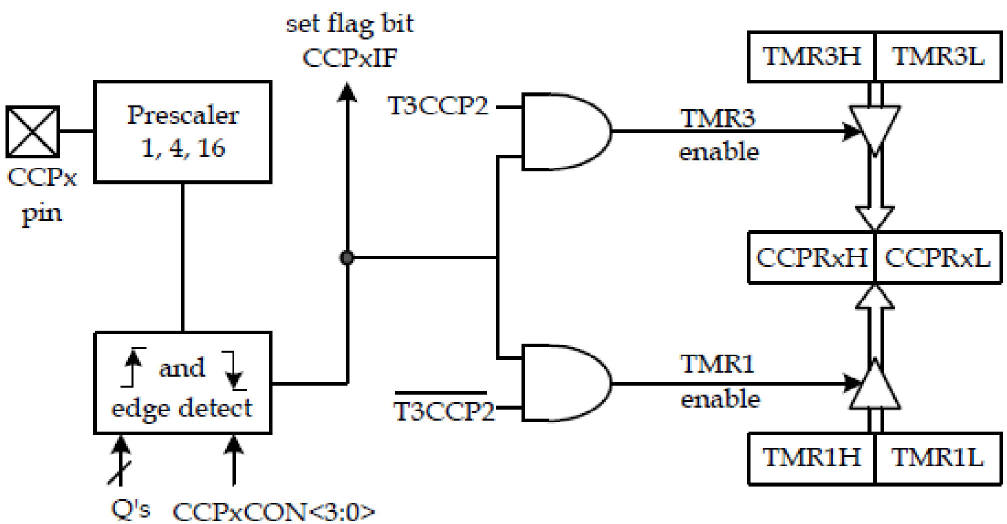
	7	6	5	4	3	2	1	0
Timer3:	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	T3SYNC	TMR3CS	TMR3ON



CCP in the Capture Mode

- CCPR1 captures the 16-bit value of Timer1 : When an event occurs on pin RC2/CCP1
- Interrupt request flag bit CCP1IF is set: Must be cleared for the next operation
- To capture an event
  - Set up pin RC2/CCP1 of PORTC as the input
  - Initialize Timer1: T1CON register
  - Initialize CCP1: CCP1CON register
- The PIC18 event can be one of the following:
  1. every falling edge
  2. every rising edge
  3. every 4th rising edge
  4. every 16th rising edge
- New Capture before Completion

  Lost Previous data



Capture operation and applications

- When a capture is made, the interrupt flag bit, CCPxIF is set. [PIR1 register]
- The CCPxIF flag must be cleared by software.
- In capture mode, the CCPx pin must be configured for input.
- The timer to be used with the capture mode must be running in timer mode or synchronous counter mode.
- To prevent false interrupt, the user must disable the CCP module when switching prescaler.

Applications of Capture Mode

- Event arrival time recording
- Period measurement
- Pulse width measurement
- Interrupt generation
- Event counting
- Time reference
- Duty cycle measurement

	B7	B6	B5	B4	B3	B2	B1	B0
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

CCP in compare mode [T1CON, PIR1]

- The 16-bit CCPRx register is compared against the TMR1 (or TMR3).
- When they match, one of the following actions associate may occur on the CCPx pin: Pin RC2/CCP1 on PORTC

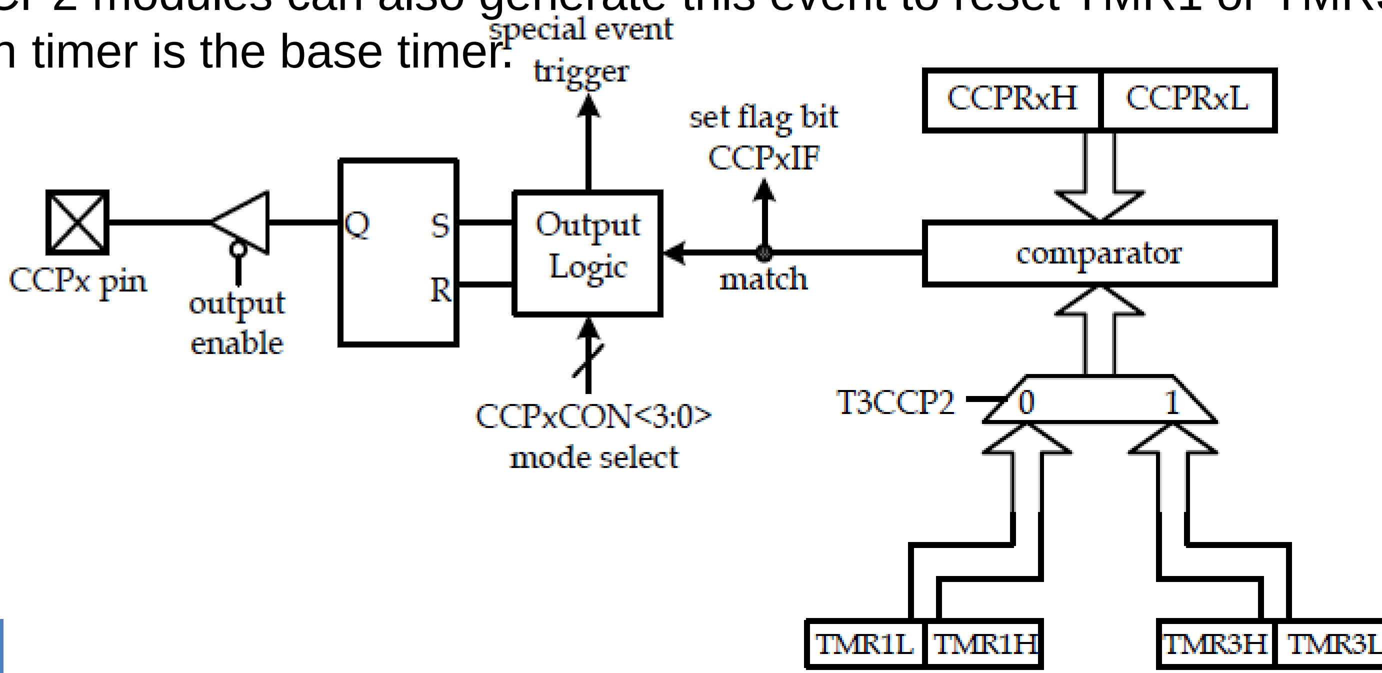
1. Driven high 2. Driven low or 3. toggle output 4. remains unchanged[Interrupt flag bit CCP1IF is set]

How to Use the Compare Mode? To set up CCP1 in Compare mode

1. Set up pin RC2/CCP1 of PORTC as output
2. Initialize Timer1 or 3 and CCP1
3. Stores the sum in the CCPRxH:CCPRxL register pair: Clear the flag CCP1IF

Special Event Trigger

- The CCP1 and CCP2 modules can also generate this event to reset TMR1 or TMR3 depending on which timer is the base timer.



Compare mode programming

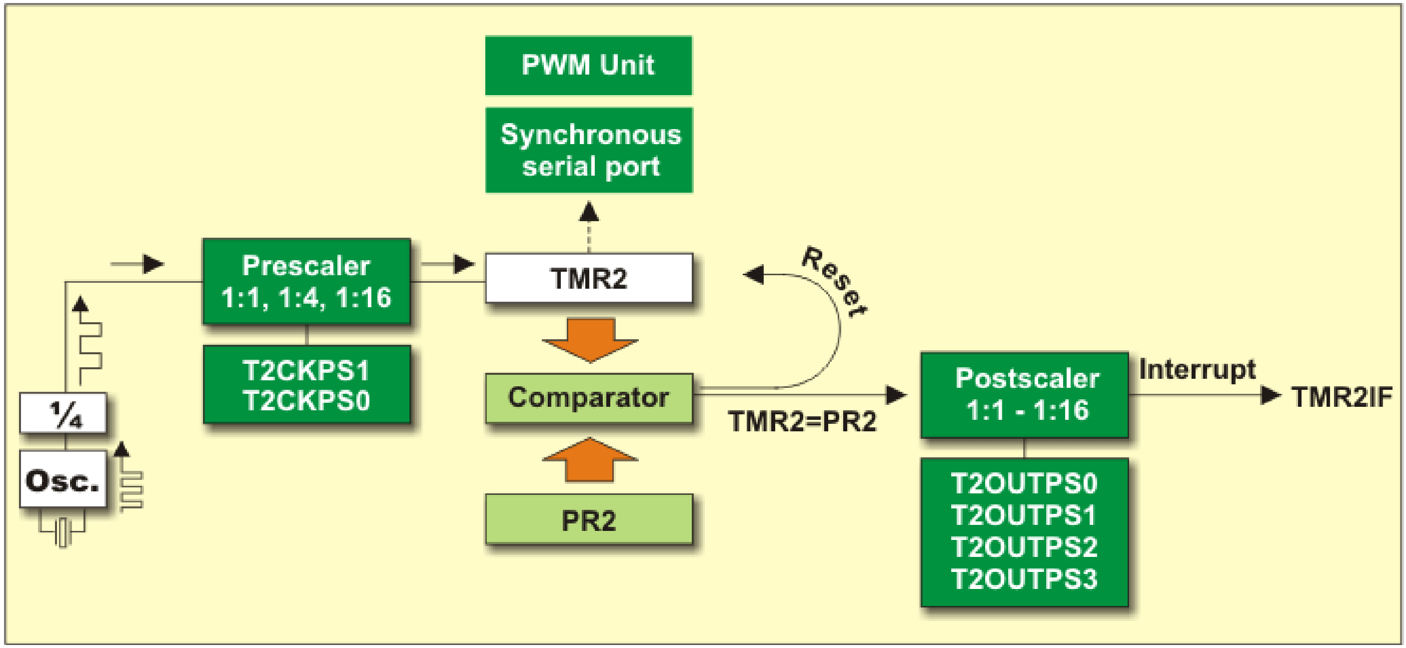
- Initialize CCP1CON
- Initialize T3CON for timer 1(or 3)
- Initialize the CCPR1H:CCPR1L registers
- Make CCp1 pin as output
- Initialize Timer1(or3) register values
- Start Timer1(or3)
- Monitor CCP1IF flag(or use as interrupt).

PWM Mode

- CCP module with Timer2
  - Output a pulse wave form for a given frequency/duty cycle
- Duty cycle
  - CCPR1 register
- Period
  - PR2 register
- When TMR2 is equal to PR2
  - TMR2 is cleared
  - Pin RC2/CCP1 of PORTC is set high
  - PWM duty-cycle byte loaded into CCPR1

Timer2- Block Diagram

- Timer2 operation : 8-bit number is loaded in PR2
- When TMR2 and PR2 match: Output pulse is generated and the timer is reset
- Output pulse goes through postscaler: Sets the flag TMR2IF



When using the TMR2 timer, :one should know:

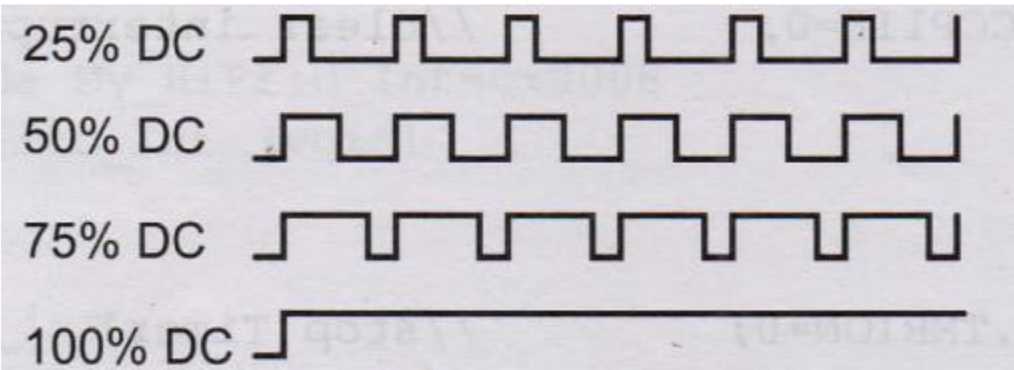
- Upon power-on, the PR2 register contains the value FFh;
- Both prescaler and postscaler are cleared by writing to the TMR2 register;
- Both prescaler and postscaler are cleared by writing to the T2CON register; and
- On any reset, both prescaler and postscaler are cleared.

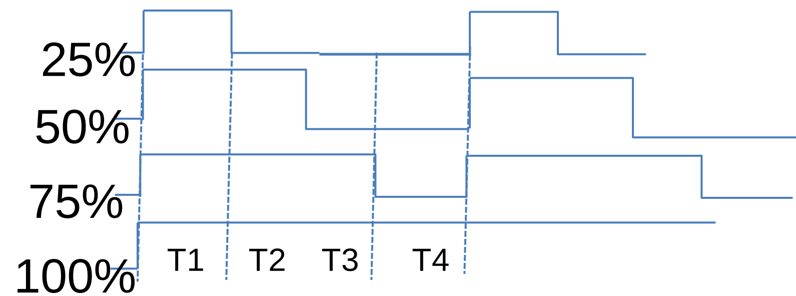
TMR2: Timer2- 8-bit

- 8-bit period register (PR2)- Fixed value
- TMR2 and PR2 are readable and writable
- TMR2 increments from 00 to the value equal to PR2
- TMR2IF flag from PIR1 reg. is raised and TMR2 reset to 00
- The clock source for TMR2 is Fosc/4 for both prescaler and Postscaler options.
- There is no external clock source ,hence cant not used as counter
- Three prescale values (Bit1-Bit0) and 16 postscale values (Bit6-Bit3)
- Flag (TMR2IF) is set when TMR2 matches PR2: Can generate an interrupt

PWM Mode

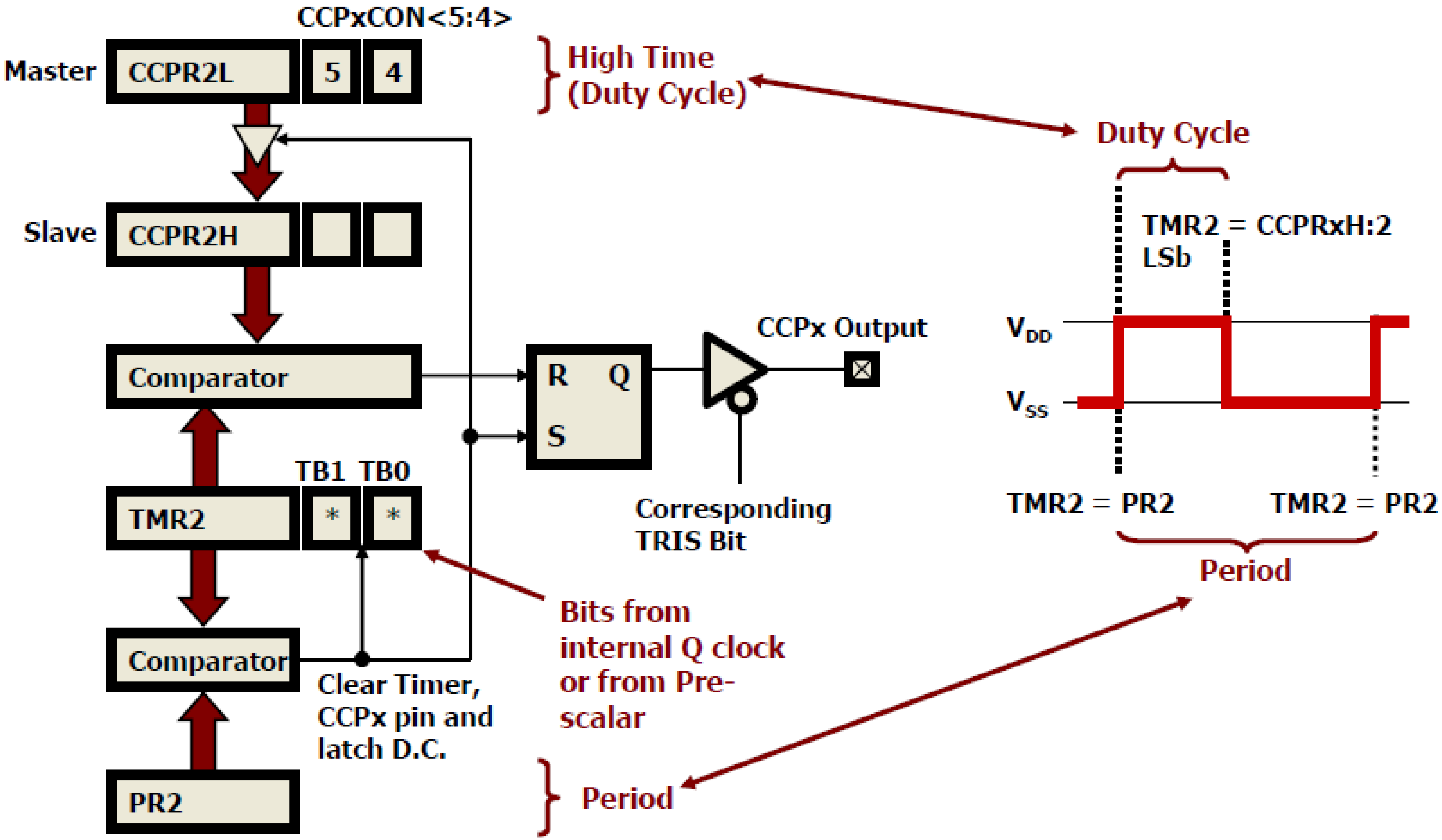
- PWM is the Feature of CCP and allows to create pulses of variable widths
- It is based on the Duty cycle of Wave with fixed duration of period





- Uses Timer2 and Period register PR2 –fixed value.
- Pin RC2/CCP1 of PORTC is set high to obtain the PWM wave
- Uses 8-bit CCPR1L register
- SFRs used CCPxCON, T2CON, PIR1 and TMR2 is cleared CCPRxL
- Used to control Speed of DC motor

PWM MODES (CCP2)



Sinhgad Institutes

PWM Configuration:

Step 1 :Calculating the Period Value

Period value : Tpwm = (PR2 + 1) *4*Tosc*N

PR2 = [Tpwm / *4*Tosc*N]- 1

Minimum *f*pwm--- PR2-255, N=16
=fosc/16382

Where Tpwm = Desired PWM Signal Period = 1 / fpwm

PR2 = TMR2’s Period Register

Tosc = System Oscillator Period = 1 / fosc

N = TMR2 Pre-scale Value (1, 4, or 16)

Maximum *f*pwm--- PR2-1, N=1

Choose Pre scaler [TMR2PRE] to ensure that PR2 is in the range of 0 to 255 for the desired PWM frequency

Step 2 :Calculating the CCPRL1 Value (Lower 8 bits)

Value to be loaded = % D * PR2

Step 3 : Calculating the Duty Cycle Value

DCPWM= (CCPRxL:CCPxCON<5:4>) *Tocs*N

where DCpwm =%DC*Twpm = Desired PWM Duty Cycle (time, not %)

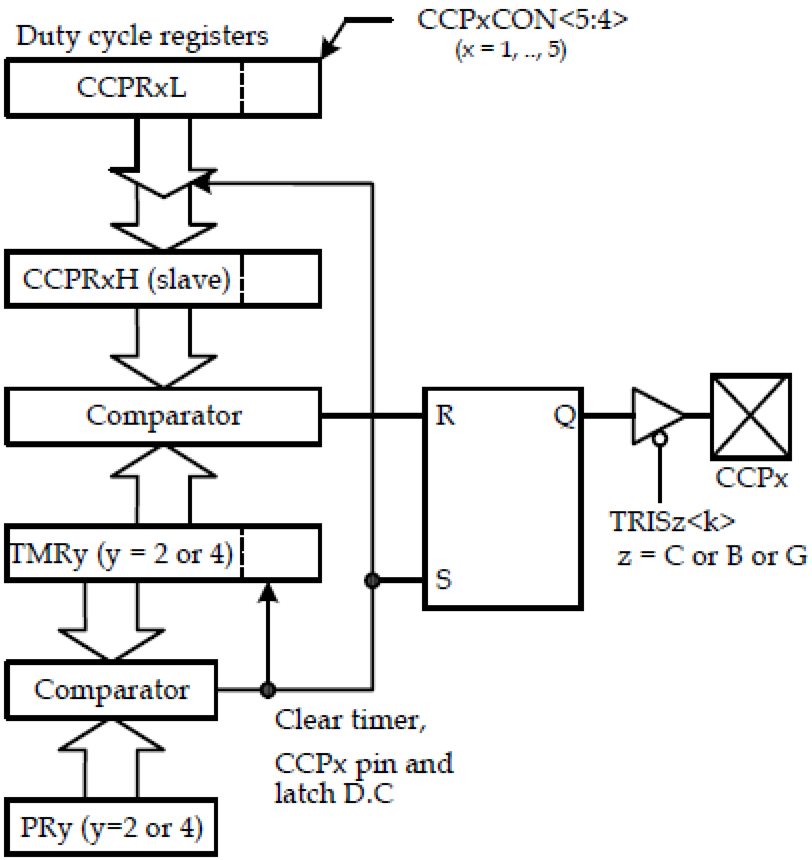
CCPR2L = Upper 8-bits of Duty Cycle Value

CCP2CON<5:4> = Low 2-bits of Duty Cycle Value

(CCPRxL:CCPxCON<5:4>) =DCpwm/(Tosc*N) ;

CCPR2L:CCP2CON<5:4> is in the range of 0 to 1023 for the desired PWM duty cycle.

CCP in PWM mode [T2CON, PIR1]



CCPxCON<5:4> for Duty cycle bits

0 0   0

0 1   0.25 :

1 0   0.50 :

2 1 1   0.75:

PWM period = [(PRy) + 1] * 4 * TOSC * N

N = Presale factor 1,4,16

PWM duty cycle =
(CCPRxL:CCPxCON<5:4>) * TOSC * N

Procedure for using the PWM module:

Step 1

Set the PWM period by writing to the PRy (y = 2 or 4) register.

Step 2

Set the PWM duty cycle by writing to the CCPRxL register and CCPxCON<5:4> bits.

Step 3

Configure the CCPx pin for output

Step 4

Set the TMRy prescale value and enable Timery by writing to TyCON register

Step 5

Configure CCPx module for PWM operation

Calculation of PR2, Pre-scaler, CCPR1L, DC1B1:B0

Calculation of PR2 & Prescalers

Find the PR2 and pre-scalers needed to get the following PWM frequencies
a) 1.22KHz, b) 4.88KHz and c) 78.125 if *f*osc=20 MHz

Ans: PR2= [*f*osc/*f*pwm*4*N]-1
a) for *f*pwm = 1.22 KHz if N=1, PR2=4097
N=4, PR2=1024
N=16, PR2=255 ---- Valid

Calculation of CCPR1L

Find value of CCPR1l and DC1B1: B0 for *f*pwm=2.5 KHz and 50% duty cycle PWM
[0 0-- 0 %, 0 1--25 %, 1 0 -- 50%, 1 1-- 75 %, 11xx- PWM]
Ans : Assume that *f*osc=10MHz, and N=16
1. PR2= [fosc/fpwm*4*N]-1= 62
2. CCPR1L= PR2*DC = 62*.5=31
3. DC1B1: DC1B0 =10

Find Minimum and Maximum value of *f*pwm

Minimum *f*pwm--- PR2-255, N=16 *f*pwm =fosc/16382
Maximum *f*pwm--- PR2-1, N=1

Configuring CCPx for PWM-programming

- Set PWM Period by writing to PR2 register
- Set PWM Duty Cycle by writing to CCPRxL and CCPxCON<5:4> bits
- Make the CCPx pin an output by clearing the appropriate TRIS bit
- Set the TMR2 prescale value, then enable TMR2 by writing to T2CON
- Clear the TMR2 register
- Configure the CCPx module for PWM mode set DC1B2 and DC1B1 for decimal portion of the duty cycle.
- Start Timer2.

Sinhgad Institutes

Configuring CCPx for PWM-programming

Write a program for 2.5 KHz and 75 % duty cycle PWM generation with N=4

B7	B6	B5	B4	B3	B2	B1	B0
------	----	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
		PWM Bit & Bit 0		Mode select bits			

B7	B6	B5	B4	B3	B2	B1	B0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

Step1: Load PR2 Value

PR2= [fosc/fpwm*4*N]-1=[10MHz/2.5KHz*4*4]-1 =249;

Step2: Set PWM Duty Cycle by writing to CCPRxL and CCPxCON<5:4> bits

CCPRxL= PR2*DC= 249*0.75= 186.75~186;

Step3: Make the CCPx pin an output by clearing the appropriate TRIS bit

TRISCbits.TRISC2=0;

Step4: Set the TMR2 pre-scaler value, then enable TMR2 by writing to T2CON

T2CON=0x01; (pre-scaler=4 00– 1:1, 01- 1:4; and 1X– 1:16) 00000001

Step5: Clear the TMR2 register

TMR2=0;

Step6: Configure the CCPx module for PWM mode set DC1B2 and DC1B1 for decimal portion of the duty cycle.

CCP1CON=0x3C; (CCPxCON<5:4> =11 for 75% DC & 11XX-- PWM)

Step7: Start Timer2.

T2CONbits. TMR2ON=1;

B7	B6	B5	B4	B3	B2	B1	B0
-----	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
	Postscale Select Bits					Prescale Select Bits	

Step8: Check for End of Period

{ PIR1bits.TMR2IF=0;

while(PIR1bits.TMR2IF= =0); }

Write a program for 1KHzand 10% duty cycle PWM generation

Solution, Assume that fosc=10MHz, and N=16

1. PR2= [fosc/fpwm*4*N]-1= 155.25   2. CCPR1L= PR2*DC = 155.25*0.1=15.52
3. DC1B1: DC1B0 =10 , CCP1CON =00101100, T2CON=00000010

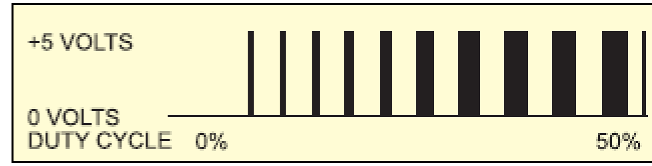
```
#include <P18F458.h>
Void main(void)
{
1.    CCP1CON=0 ;                  //clear the reg
2.    PR2=155;                     // load the PR2 value
3.    CCPR1L=15;                   // 10% DC
4.    TRISCbits.TRISC2=0;          // make PWM pin output
5.    T2CON=0x02;                  // Timer2, 16 prescalar, no post scalar
6.    CCP1CON=0x2C;                // PWM mode 00 for DC1B1:DC1BO
7.    TMR2=0;                      // Clear timer2
8.    T2CONbits.TIMER2ON=1;        // START TIMER2
9.    Ckeck for the timer flag
  While(1)
{
          PIR1bits.TMR2IF=0;   clear timer2 flag.
          While(PIR1bits.TMR2IF==0);  wait for end of period
} }
```

Sinhgad Institutes

DC Motor speed control with CCP

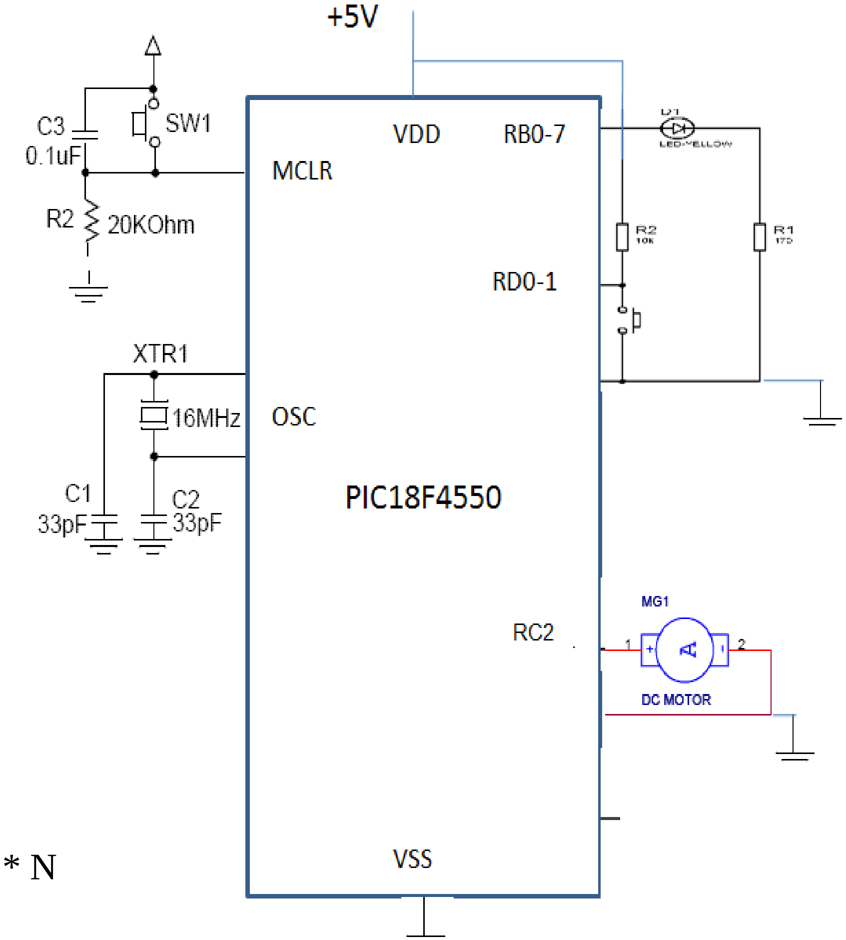
Interfacing of DC motor– PWM generation.

Sinhgad Institutes



SETUP FOR PWM OPERATION
The following steps should be taken when configuring the CCP module for PWM operation:

1. Set the PWM period by writing to the PR2 register.
2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
3. Make the CCP1 pin an output by clearing the TRISC<2> bit.
4. Set the TMR2 prescale value and enable Timer2 by writing to T2CON.
5. Configure the CCP1 module for PWM operation.



PWM period = [(PRy) + 1] * 4 * TOSC * N
N = Presale factor 2,4,16
PWM duty cycle = (CCPRxL:CCPxCON<5:4>) * TOSC * N

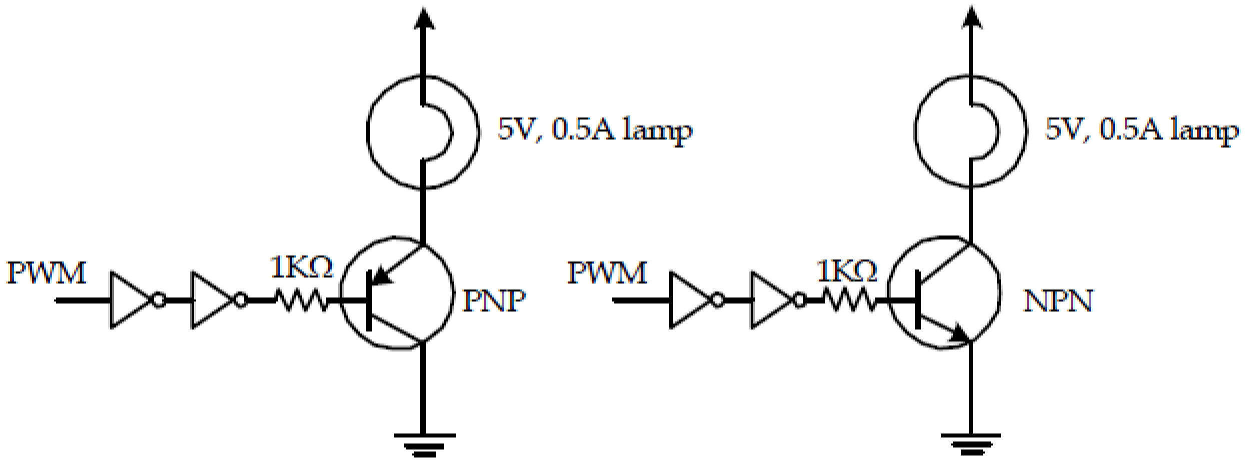
Program of DC Motor

```
/*Calculations:
  Fosc = 20MHz

  PWM Period = [(PR2) + 1] * 4 * TMR2 Prescale Value / Fosc
  PWM Period = 200us
  TMR2 Prescale = 4
  Hence, PR2 =  249 or 0xF9

  Duty Cycle = 10% of 200us
  Duty Cycle = 20us
  Duty Cycle = (CCPR1L:CCP1CON<5:4>) * TMR2 Prescale Value /
Fosc
  CCP1CON<5:4> = <1:1>
  Hence, CCPR1L = 24 or 0x18
 */
#include<p18f4550.h>

unsigned char count=0;
bit TIMER,SPEED_UP;

void timer2Init(void)
{
   T2CON   =   0b00000001;   //Prescalar = 4; Timer2 OFF
   PR2     =   0xF9;            //Period Register
}
```

```
void Interrupt_Init(void)
{
   INT1IE   =   1;     //Enable external interrupt INT1
   INTEDG1 =   0; //Interrupt on falling edge
   GIE     =   1;      // Enable global interrupt
}
void interrupt timerinterrupt(void)
{
    if (INT1IF)                  // If the external interrupt flag is 1, do .....
    {
       INT1IF   =   0;               // Reset the external interrupt flag
       if(SPEED_UP)
       {
          if(count < 8)
          {
             count++;
             CCPR1L = 0x18 + (count * 25);     //Increment duty cycle
          }
          else SPEED_UP = 0;
       }
       else
```

PWM application

as brightness control in lamp Light



Sinhgad Institutes

CCP and ECCP Modules

❖ Standard CCP Module:

- Input Capture

  Captures the timer value (16bit) when an event occurs on a CCP pin

- Output Compare

  Generate a signal on the CCP pin at a specified time

- PWM

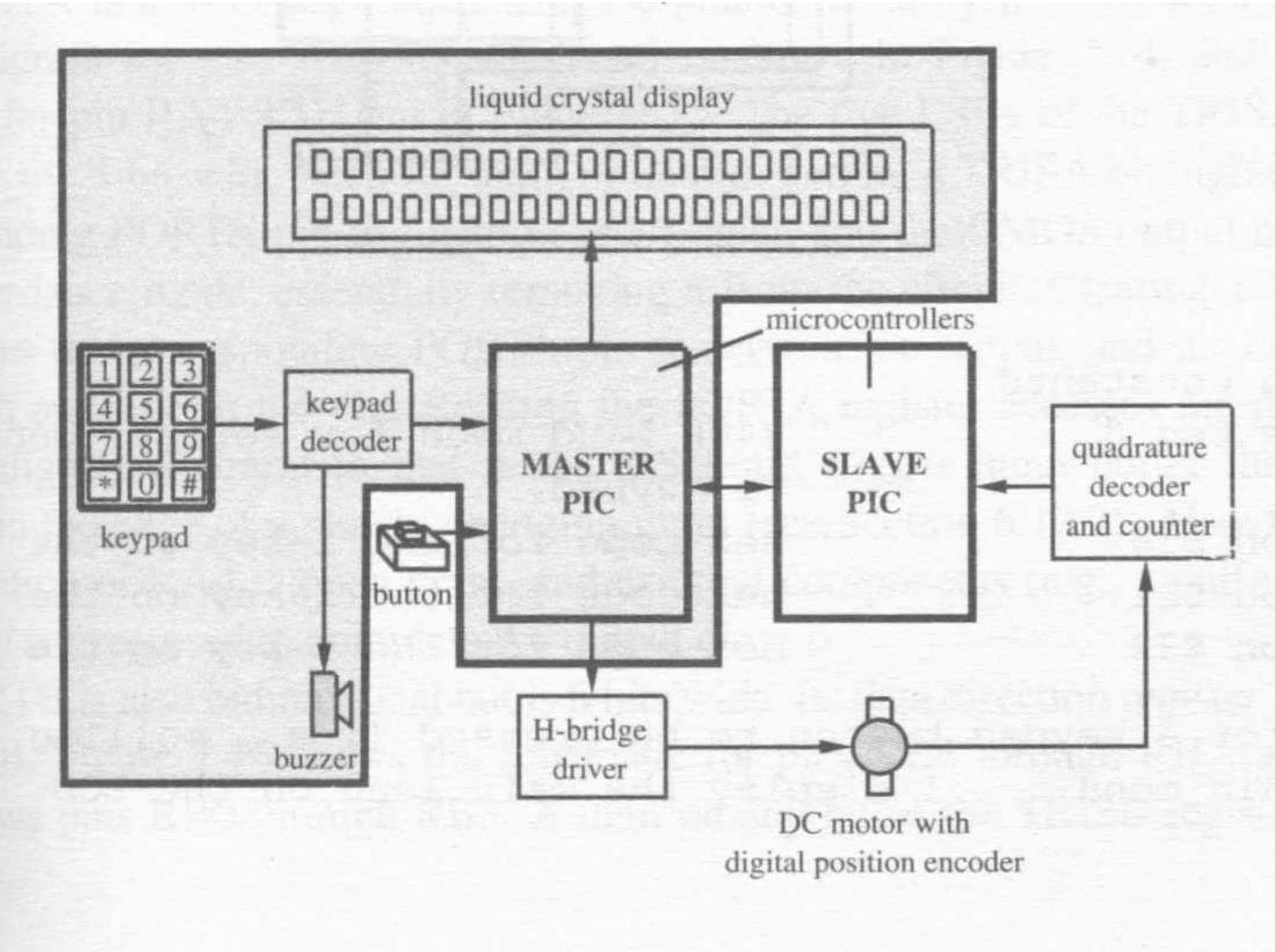
  2 Pulse Width Modulated Outputs (10 bit accuracy)

❖ Enhanced CCP Module:

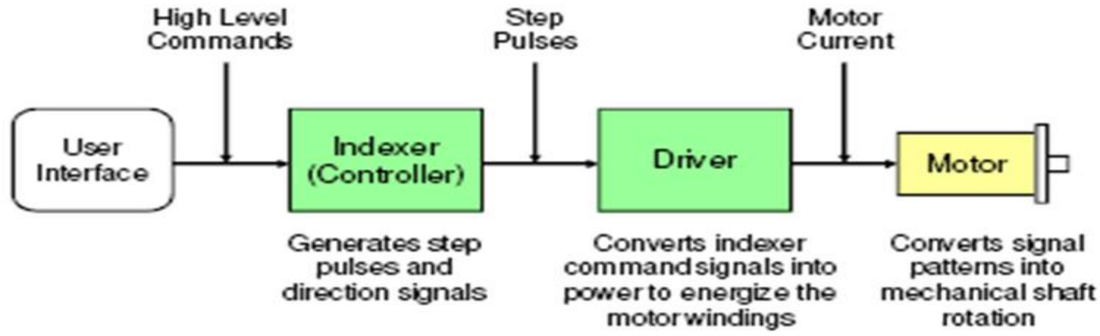
- Same as standard but with Enhanced 10-bit PWM
  - ✓ Complementary outputs to drive half or full bridge
  - ✓ Dead band control

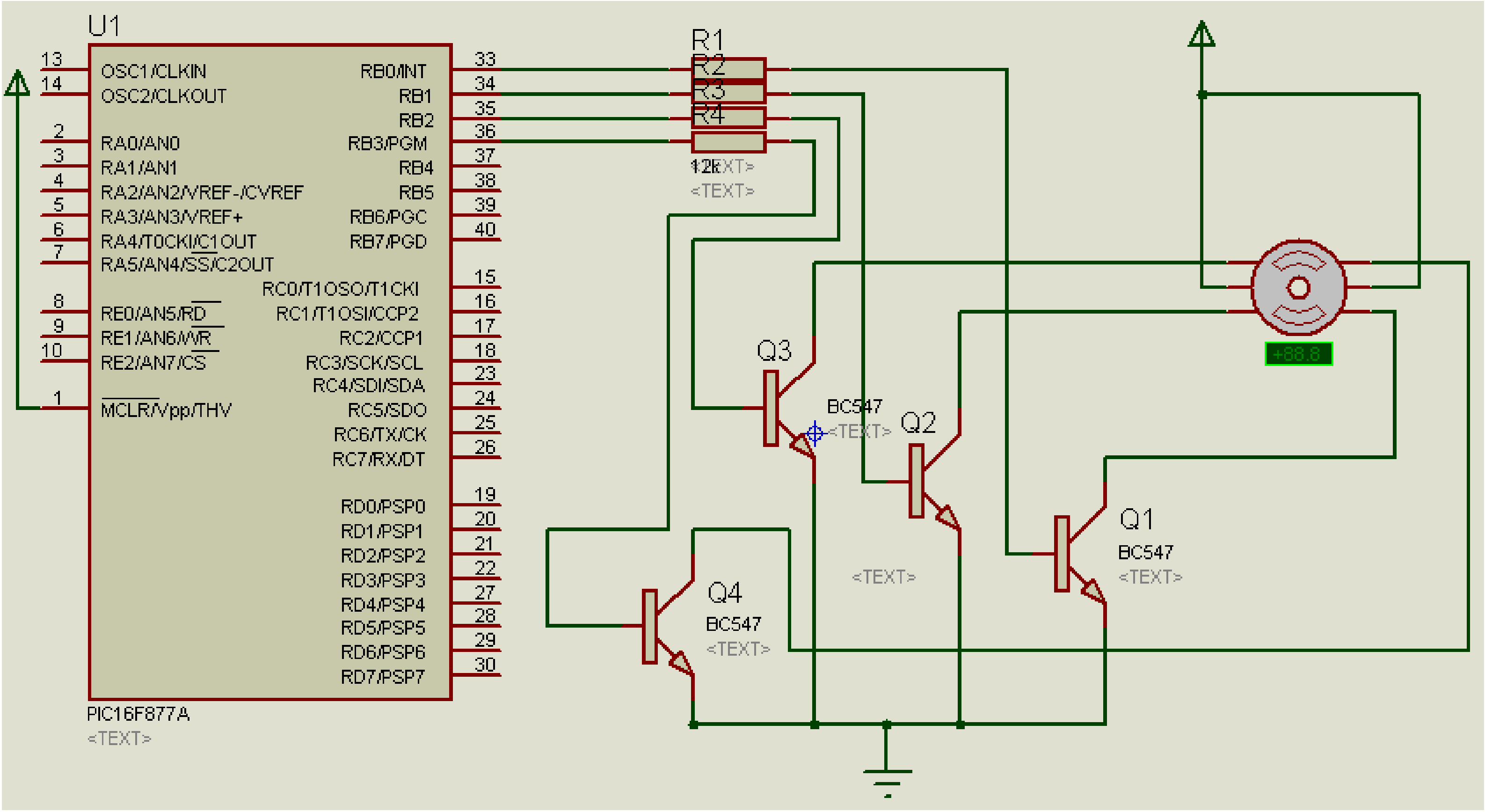
❖ Only available on CCP1

Speed control of DC motor



Stepper motor Interface

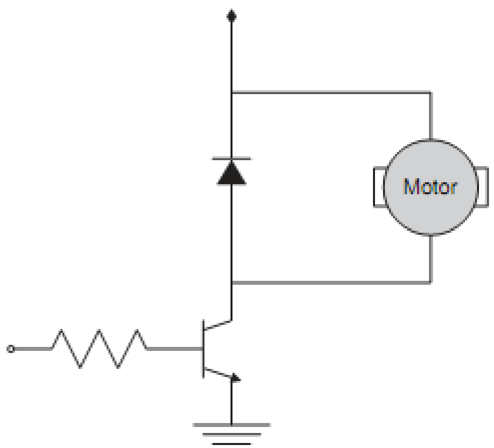




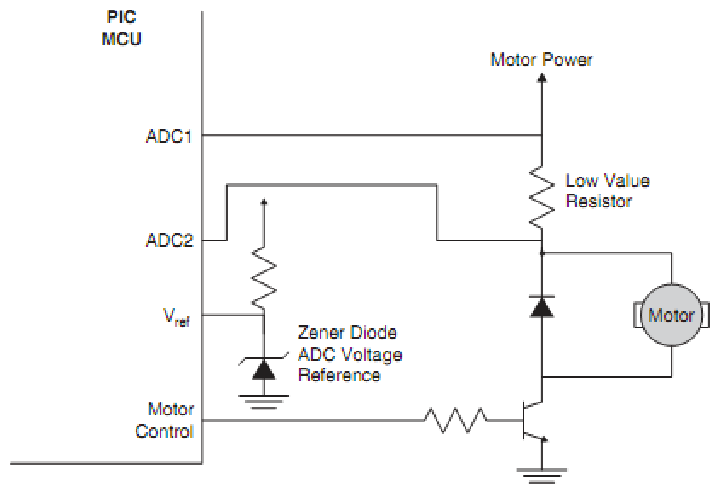
Sinhgad Institutes

Speed control of DC motor

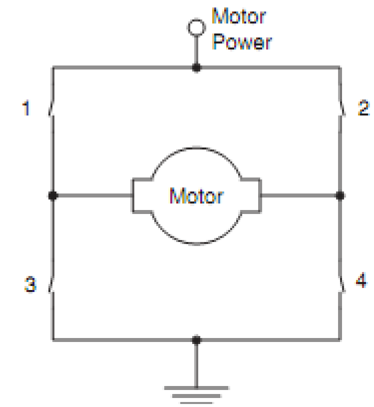
NPN transistor speed control



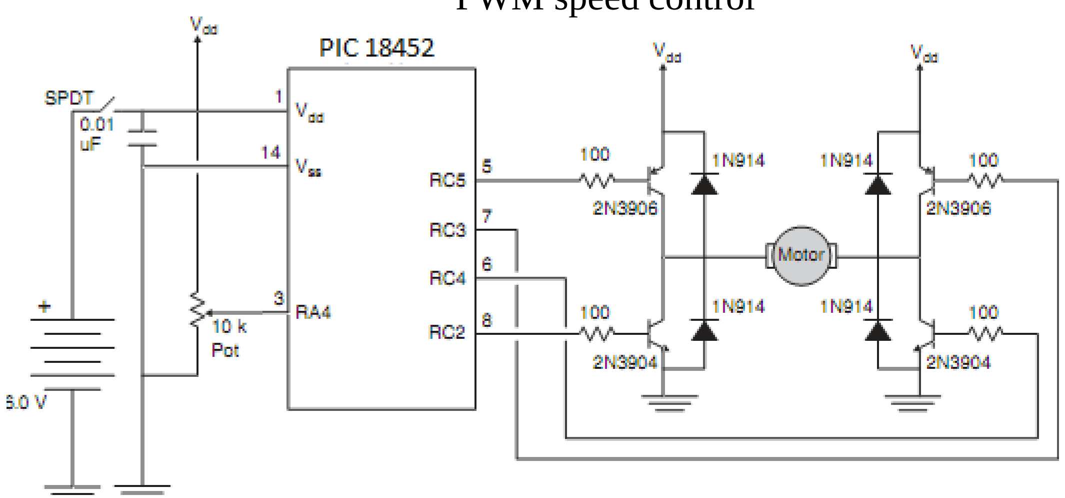
Low value register to control the speed



H- bridge speed control



PWM speed control

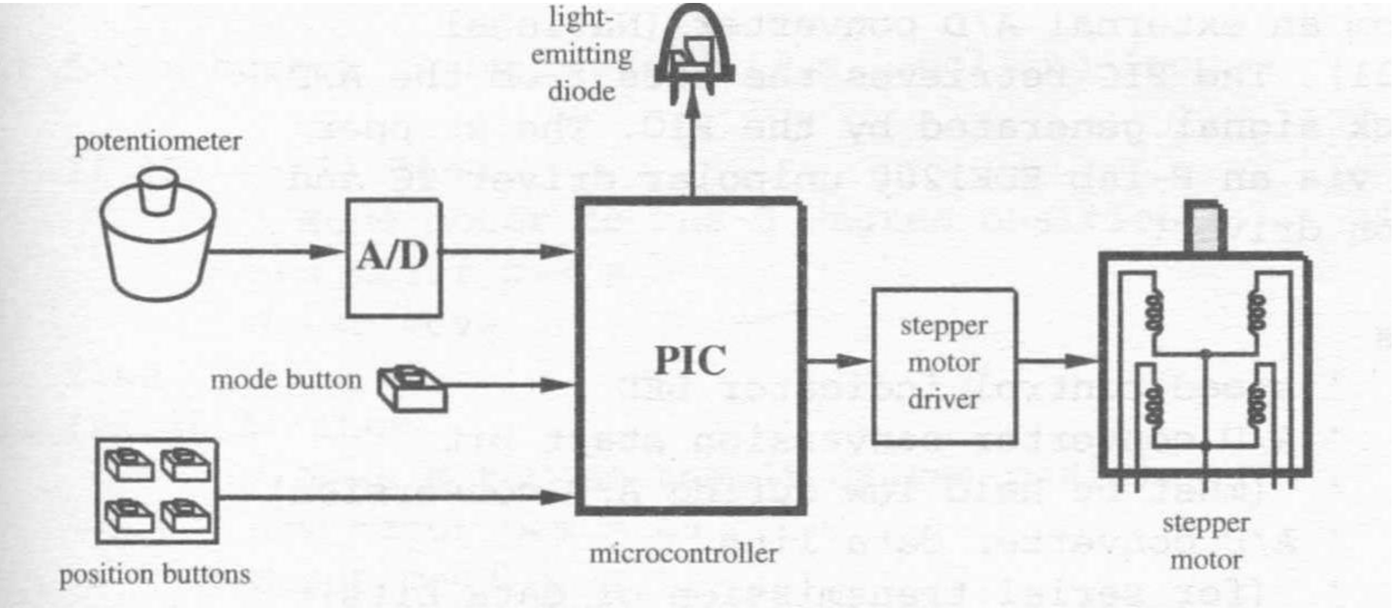


In this application a fairly complex control application is used which allows forward and reverse, as well as speed control, of a dc motor using the full H-bridge circuit.

The direction and speed of the motor are specified by the potentiometer;
- Center position: No speed
- Upward movement: Forward
- Down movement: Reverse

Uses the PWM mode to control the Speed
Uses the Timer2 with increased frequency either internal or external

Stepper motor position and speed control



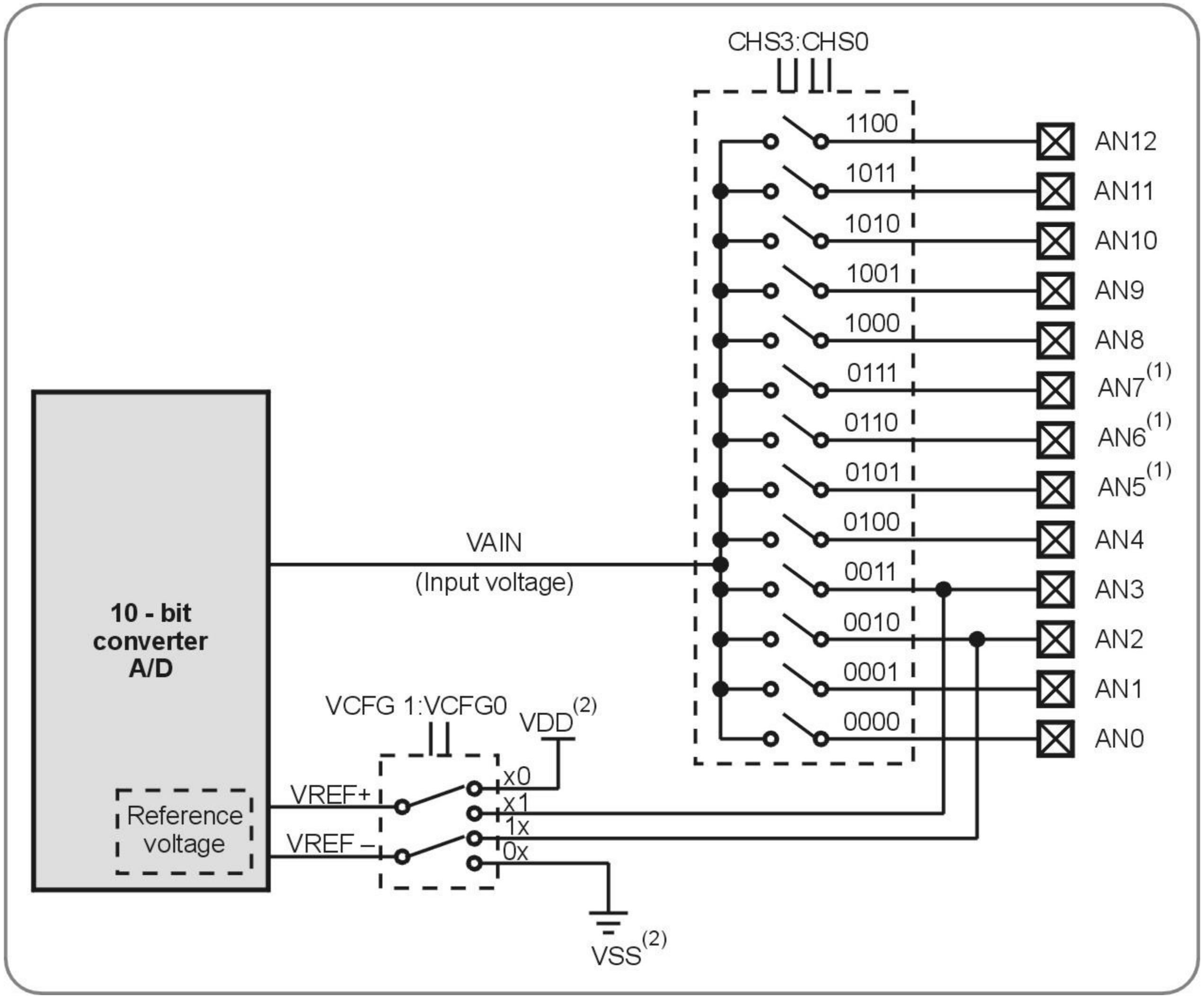
MOTOR TYPE	DRIVE HARDWARE	CONTROL SIGNALS	OPERATING CHARACTERISTICS
Dc motors	Transistor switches and H-bridges	Pulse width modulated (PWM)	Timed rotation
Bipolar stepper motors	H-bridges	Step sequence	Precise movement
Unipolar stepper motors	Transistor switches	Step sequence	Precise movement
Radio-control servos	Internal to servo	Pulse train	Positioning

Sensor interfacing using ADC

ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE – PIC 18F4550

- It has 10 bit ADC (Resolution – 10 bit)
- The ADC module has with 13 Channels or the PIC18F4550 devices. [RA0-3,5 RB0-4 RE0-3]=AN0-AN12=13 CH
- The converted binary output data is held in two registers ADRESL and ADRESH
- Vdd can be used as source for Vref or connecting to external device source
- The conversion Time is decided by Fosc--- can not be shorter than 1.6 ms (40 MHz)
- It allows the differentiation of Vref+ and Vref-
- All the features are programmed by ADCON0 , ADCON1 and ADCON2 register
- The A/D allows conversion of an analog input signal to a corresponding 10-bit digital number. The A/D module has four registers.
  - A/D Result High Register (ADRESH)
  - A/D Result Low Register (ADRESL)
  - A/D Control Register 0 (ADCON0)
  - A/D Control Register 1 (ADCON1)

ADC Block Diagram



ADC Block Diagram

- The analog reference voltage is software selectable to either the device's positive and negative supply voltage (VDD and VSS), or the voltage level on the RA3/AN3/ VREF+ pin and RA2/AN2/VREF- pin.

- 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 conversion clock must be derived from the A/D's internal RC oscillator.

- The output of the sample and hold is the input into the converter, which generates the result via successive approximation.

- A device RESET forces all registers to their RESET state. This forces the A/D module to be turned off and any conversion is aborted.

- Each port pin associated with the A/D converter can be configured as an analog input (RA3 can also be a voltage reference) or as a digital I/O.

- The ADRESH and ADRESL registers contain the result of the A/D conversion. When the A/D conversion is complete, the result is loaded into the ADRESH/ ADRESL registers, the GO/DONE bit (ADCON0<2>) is cleared, and A/D interrupt flag bit, ADIF is set.

- Channels are selected by use of CHS3:CHS0 of DADCON0 register

Sinhgad Institutes

ADC Registers

SFR	Description	Access	Reset Value	Address
ADCON0	A/D Control Register 0	Read/Write	0x00	0xFC2
ADCON1	A/D Control Register 1	Read/Write	0x00	0xFC1
ADCON2	A/D Control Register 2	Read/Write	0x00	0xFC0
ADRESH	A/D Result High Register	Read	unknown	0xFC4
ADRESL	A/D Result Low Register	Read	unknown	0xFC3

ADCON0

7	6	5	4	3	2	1	0
—	—	CHS3	CHS2	CHS1	CHS0	Go/$\overline{\text{Done}}$	ADON

ADCON1

U-0	U-0	R/W-0	R/W-0	R/W-0(1)	R/W(1)	R/W(1)	R/W(1)
—	—	VCFG0	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

ADCON2

7	6	5	4	3	2	1	0
ADFM	—	ACQ2	ACQ1	ACQ0	ADCS2	ADCS1	ADCS0

Sinhgad Institutes

ADCON0 Register

7	6	5	4	3	2	1	0
—	—	CHS3	CHS2	CHS1	CHS0	Go/$\overline{\text{Done}}$	ADON

Bit No.	Control Bit	Description	
Bit 7 - 6	Unimplemented	Read as ‘0’	
Bit 5 - 2	**CHS3:CHS0**	**Analog Channel Select bits**	
		0000 = Channel 0 (AN0)	**0001 = Channel 1 (AN1)**
		0010 = Channel 2 (AN2)	0011 = Channel 3 (AN3)
		0100 = Channel 4 (AN4)	0101 = Channel 5 (AN5)
		0110 = Channel 6 (AN6)	0111 = Channel 7 (AN7)
		1000 = Channel 8 (AN8)	1001 = Channel 9 (AN9)
		1010 = Channel 10 (AN10)	1011 = Channel 11 (AN11)
		1100 = Channel 12 (AN12)	1101 = Unimplemented
		1110 = Unimplemented	1111 = Unimplemented
Bit 1	**GO/DONE**	**A/D Conversion Status bit** 1 = A/D conversion in progress; 0 = A/D Idle	
Bit 0	**ADON**	**A/D On bit** **1 = A/D converter module is enabled** 0 = A/D converter module is disabled	

- ADCON0 reg is used to set the conversion time and select the channels
- For power saving ADC feature is turned off when Power up. And turned on with ADON bit when required.
- GO/DOWN bit is used for start and monitor the End of conversion

ADCON1 Register

ADCON1 is used to set the reference voltage
PCFG select port configuration
RA0-3, RA5 & RE0-2

U-0	U-0	R/W-0	R/W-0	R/W-0(1)	R/W(1)	R/W(1)	R/W(1)
—	—	VCFG0	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

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

bit 5 **VCFG0:** Voltage Reference Configuration bit (VREF- source)
1 = VREF- (AN2)
0 = VSS

bit 4 **VCFG0:** Voltage Reference Configuration bit (VREF+ source)
1 = VREF+ (AN3)
0 = VDD

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Calculation of A/D conversion time:
It is 12 times the Tad: conversion time / bit = Fosc/2, /4,/8,/16,/32,/64

ADCON2 Register

After conversion data in the ADRESL and ADRESH is right or left justified by ADFM bit ADON bit when required.

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit

1 = Right justified
0 = Left justified

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

bit 5-3 **ACQT2:ACQT0:** A/D Acquisition Time Select bits

111 = 20 TAD
110 = 16 TAD
101 = 12 TAD
100 = 8 TAD
011 = 6 TAD
010 = 4 TAD
001 = 2 TAD
000 = 0 TAD(1)

bit 2-0 **ADCS2:ADCS0:** A/D Conversion Clock Select bits

111 = FRC (clock derived from A/D RC oscillator)(1)
110 = FOSC/64
101 = FOSC/16
100 = FOSC/4
011 = FRC (clock derived from A/D RC oscillator)(1)
010 = FOSC/32
001 = FOSC/8
000 = FOSC/2

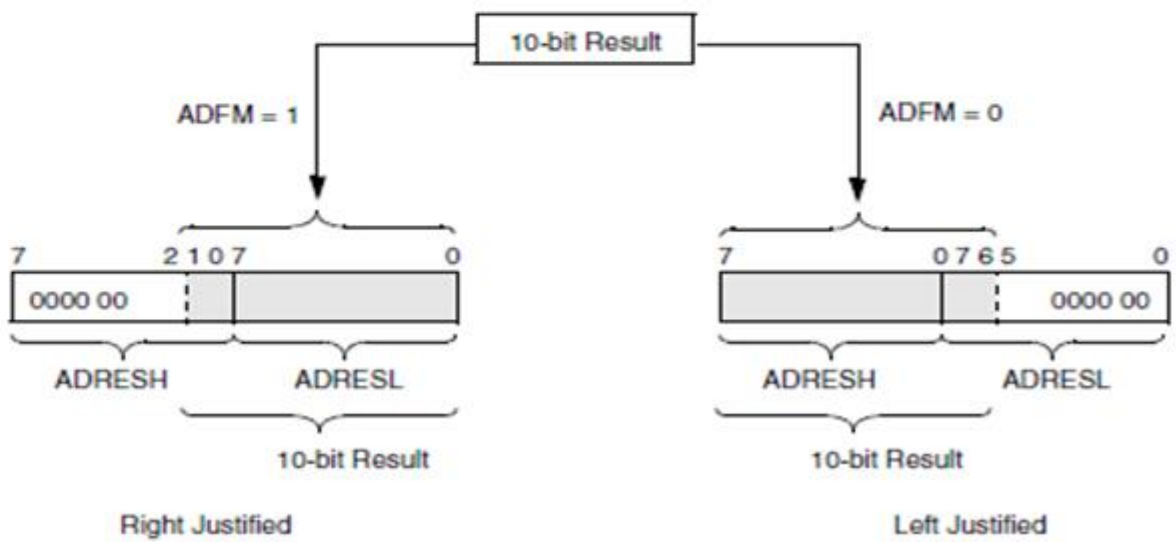
ADC Result register

A/D RESULT REGISTERS

- The ADRESH:ADRESL register pair is the location where the 10-bit A/D result is loaded at the completion of the A/D conversion. This register pair is 16-bits wide.

- **The A/D module gives the flexibility to left or right justify the 10-bit result in the 16-bit result register. The A/D Format Select bit (ADFM) controls this justification.**

- **The operation of the A/D result justification. The extra bits are loaded with ’0’s. When an A/D result will not overwrite these locations (A/D disable), these registers may be used as two general purpose 8-bit registers.**



Programming Steps

1. Configure the A/D module:
   Turn on A/D module (ADCON0) -- [BSF ADCON0, ADON]
   Configure analog pins, voltage reference and digital I/O (ADCON1)
   Select A/D input channel (ADCON0)
   Select A/D conversion clock (ADCON0)

2. Configure A/D interrupt (if desired):
   Clear ADIF bit,
   Set ADIE bit,
   Set GIE bit,
   Set PEIE bit
3. Wait the required acquisition time.
4. Start conversion:
   Set GO/DONE bit (ADCON0)
   Wait for A/D conversion to complete, by either:
   Polling for the GO/DONE bit to be cleared (interrupts disabled) OR
   Waiting for the A/D interrupt
6. Read A/D Result registers (ADRESH/ADRESL); clear bit ADIF if required.
7. For 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 2 TAD is required before the next acquisition starts.

1. Turn On ADC module
2. Initialize the port pins as input
3. Select voltage references and input channels ADCON0 & ADCON1
4. Select the conversion Speed Tad
5. Wait for the required Acquisition time
6. Activate the start of conversion bit GO/DOWN
7. Wait for the conversion to complete by polling the end of GO/DOWN bit
8. When GO/DOWN bit goes low read the ADRESL and ADRESH reg for digital output
9. repeat the sequence from step 5

Draw an interfacing diagram of ADC with PIC and Write an ALP to get data from channel 0 and display result on Port B and D every 10msec

ADCON0

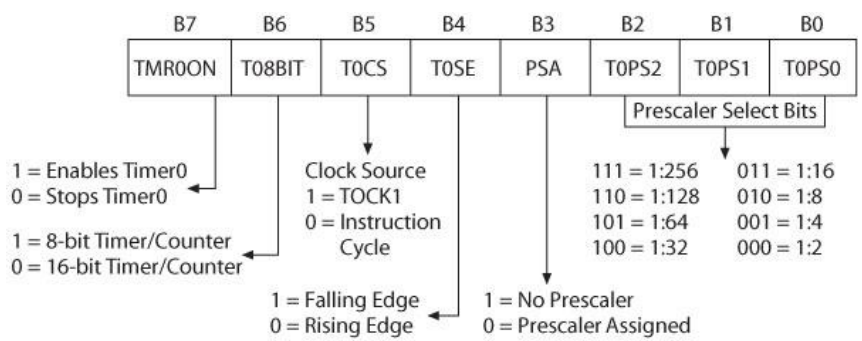
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

ADCON1

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

```
#include <P18FXXXX.h>
void T0Delay(void);
void main(void)
{
   TRISB=0;                // configure Port B as output
   TRISD=0;
   TRISAbits.TRISA0=1;     //
   ADCCON0=0X81;           //FOSC/64,CHO,ADC on
   ADCCON1=0XCE;           //FOSC/64,AN0, right Justified
   While(1)
    {
      ADCON0bits.GO=1
      while (ADCON0bits.DOWN==1);
      PORTB=ADRESL;
      PORTD=ADRESH;
      T0Delay();
    }
}
void T0Delay ( )
{
  T0CON=0x08;                        // Timer0, 16 bit, no prescaler
  TMR0H=0x9E;                        // load Higher byte in TMR0H
 TMR0L= 0x58;                        // Load Lower byte to TMR0L
 T0CONbits.TMR0ON=1;                 // start the timer for upcount
 While(INTCONbits.TMR0IF==0);        //  Check for overflow
 T0CONbits.TMR0ON=0;                 //Turnoff timer
  INTCONbits.TMR0IF==0;              // clear the Timre0 flag
 }
```

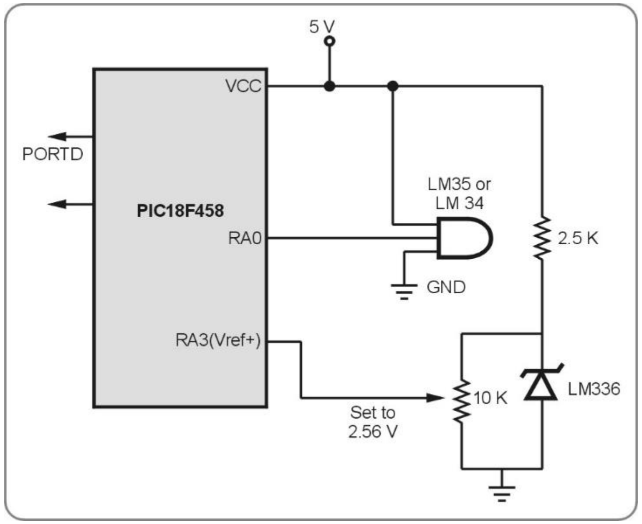
- Assume that Crystal frequency = 10 MHz
- Internal time delay T = $4/(10*10^6) = 0.4$ µs
- N= 10ms/0.4 µs = 25000
- Count= 65536-25000= $(40536)_{10}$
- Hex Value to be loaded $=(9E\ 58)_{16}$
- Load TMR0H=9E h and TMR0L=58h
- T0CON=00001000 – int Clock- No Prescaler

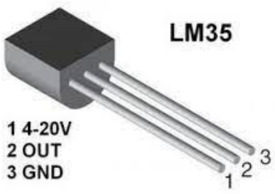


Interfacing of Temperature Sensors

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guarantee able (at +25°C)
- Rated for full -55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical

Low impedance output, 0.1 Ohm for 1 mA load






```
#include <P18FXXXX.h>
void T0Delay(void);
void main(void)
{
unsigned char Lo-bytes, Hi-bytes, bin_temp
TRISB=0;
TRISD=0;
TRISAbits.TRISA0=1;
TRISAbits.TRISA3=1;
ADCCON0=0X81;
ADCCON1=0XC5;
   While(1)
    {
        T0Delay();
        ADCON0bits.GO=1
       while (ADC)N0bits.DOWN==1);
        Lo_byte=ADRESL;
        Hi-byte=ADRESH;
       Lo_byte>>=2
       Lo_byte &=0x3F;
       Hi_byte<<6
       Hi_byte &=0xC0;
      bin_temp=Lo-bytes/Hi-bytes

    }
 }
```

LM35 Temperature Sensor Interfacing with PIC18F

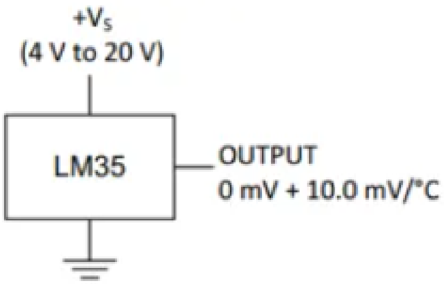
LM35 Transfer Function

The accuracy specifications of the LM35 are given with respect to a simple linear transfer function:

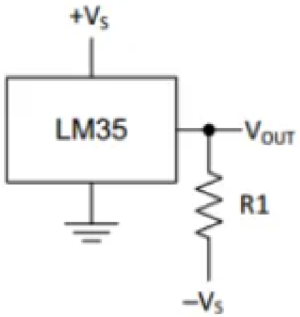
$$V_{OUT} = (10\ mv\ /\ °C) \times T$$

where V_{OUT} is the LM35 output voltage & **T** is the temperature in °C

Basic Centigrade Temperature Sensor (2°C to 150°C)



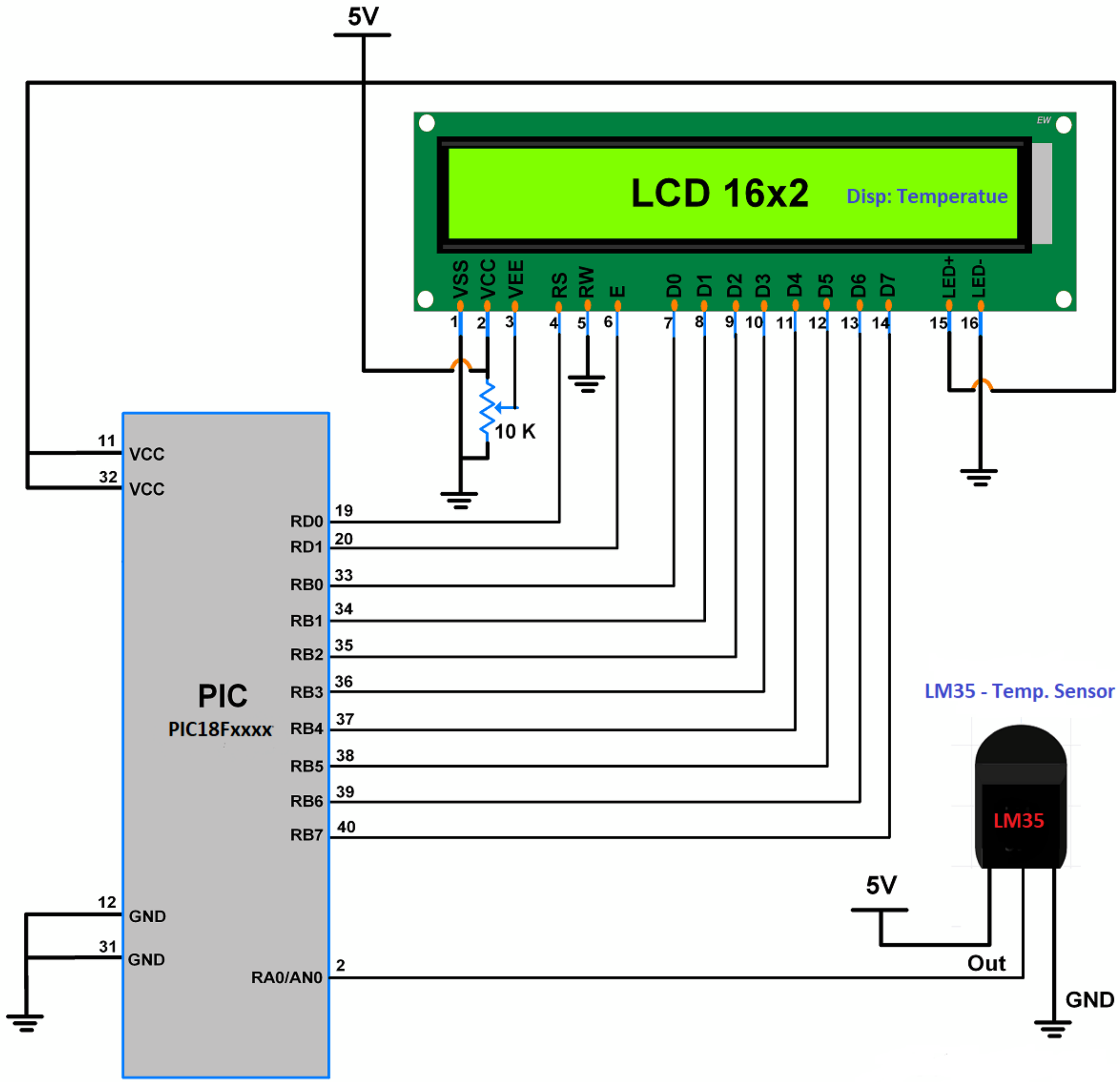
Full-Range Centigrade Temperature Sensor



Choose $R_1 = -V_S / 50\ \mu A$
V_{OUT} = 1500 mV at 150°C
V_{OUT} = 250 mV at 25°C
V_{OUT} = –550 mV at –55°C

LM35 Temperature Sensor Interfacing with PIC18F

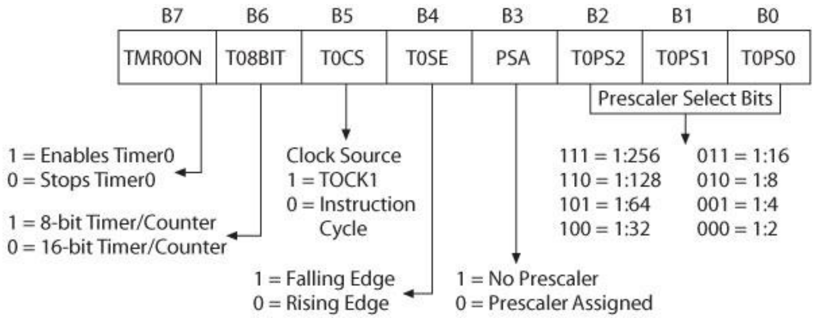
Interfacing Diagram



Interfacing of Temperature Sensors

```
#include <P18FXXXX.h>
void T0Delay(void);
void main(void)
{

unsigned char Lo-bytes, Hi-bytes, bin_temp
TRISB=0;
    TRISD=0;
    TRISAbits.TRISA0=1;
    TRISAbits.TRISA2=1;
     ADCCON0=0X81;
    ADCCON1=0XC5;
   While(1)
    {
        T0Delay();
        ADCON0bits.GO=1
       while (ADC)N0bits.DOWN==1);
        Lo_byte=ADRESL;
        Hi-byte=ADRESH;
       Lo_byte>>=2
       Lo_byte &=0x3F;
       Hi_byte<<6
       Hi_byte &=0xC0;
      bin_temp=Lo-bytes/Hi-bytes

    }
}
```

- Assume that Crystal frequency = 10 MHz
- Internal time delay $T = 4/(10*10^6) = 0.4\ \mu s$
- $N = 10ms/0.4\ \mu s = 25000$
- Count= $65536-25000= (40536)_{10}$
- Hex Value to be loaded =$(9E\ 58)_{16}$
- Load TMR0H=9E h and TMR0L=58h
- T0CON=00001000 – int Clock- No Prescaler




```
void T0Delay ( )
 {
     T0CON=0x08;              // Timer0, 16 bit, no prescaler
     TMR0H=0x9E;              // load Higher byte in TMR0H
     TMR0L= 0x58;             // Load Lower byte to TMR0L
      T0CONbits.TMR0ON=1;// start the timer for upcount
      While(INTCONbits.TMR0IF==0);      //  Check for overflow
      T0CONbits.TMR0ON=0;               //Turnoff timer
       INTCONbits.TMR0IF==0;            // clear the Timre0 flag
}
```

Sinhgad Institutes

Thanks & Regards