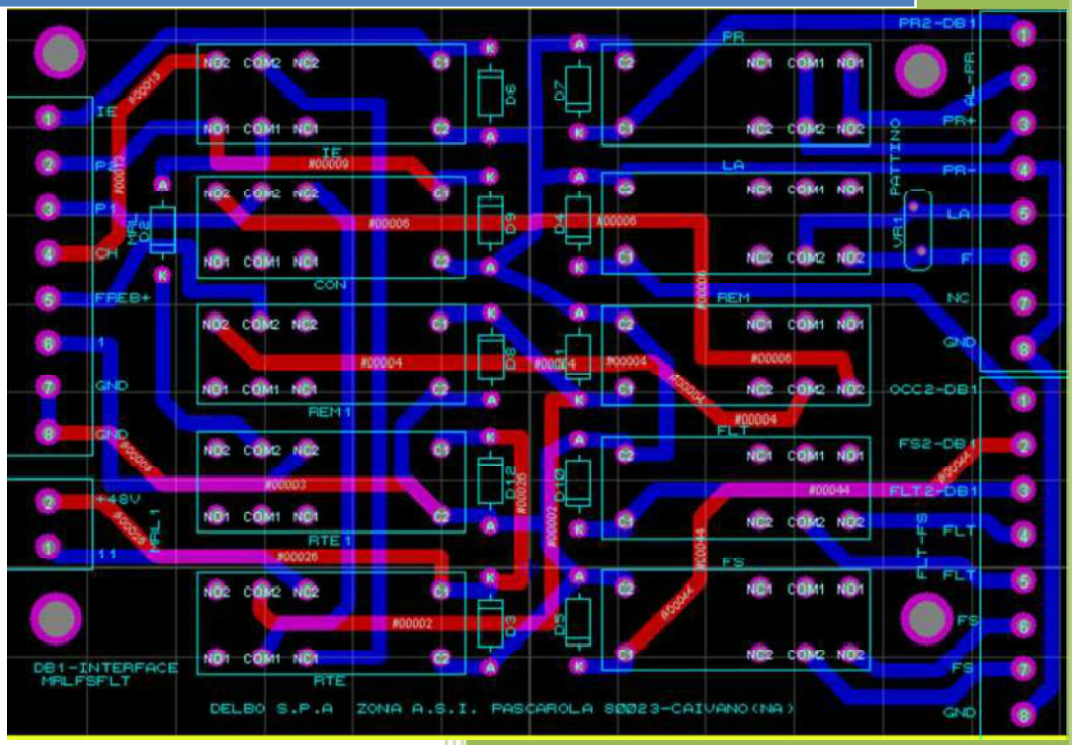
2012

MICROCHIP PIC AND CCS-C INTERRUPTS



MSEE Ibrahim GULESIN

Administrator

16/07/2012

Taking Timing Further

Brief information about interrupts, timer/counter operation, PWM pulse width modulation and CCS Pic-C applications on these topics.

- PIC16 F84A interrupts and TIMERO module
- PIC16 F877A interrupts and TIMERO, TIMER1, TIMER2 modules
- PIC16 F877A Capture/Compare/Pulse-width-modulation CCP module

Interrupts

Interrupts and timing are two most important topics in studying microcontrollers. Correct and efficient use of related hardware PIC MCU sources and software capabilities will ease many advanced engineering studies. CCS Pic-C compiler provides various high-level language features that may be utilized in related studies. Bearing in mind that, lecture notes, datasheet, compiler help and examples should be consulted for their proper use, the following information will serve as a guide to these studies.

16F84A Interrupt Structure

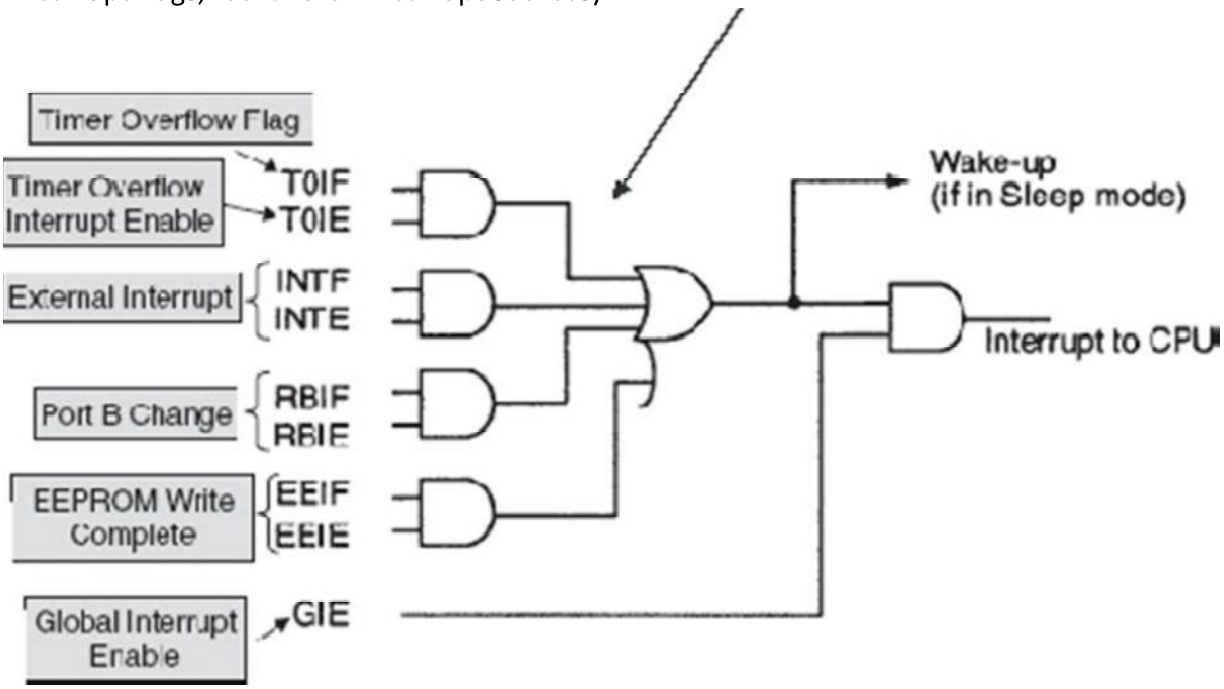
16 F84A has four maskable interrupt sources

- External interrupt source: This is the only external interrupt input. It is edge triggered. Associated pin: RBO/INT
- Timer overflow interrupt: Caused by the Timer O module. It occurs when the timer's 8-bit counter overflows
- Port B interrupt on change: This interrupt occurs when any of the higher 4 bits of Port B (RB7:RB4) changes.
- EEPROM write complete: Occurs when a write instruction to EEPROM memory is completed.

16F84A Interrupt Logic Structure

Interrupt logic structure of 16F84A is shown below. The 5 FR that controls it is the INTCON register and OPTION register. INTCON register contains enable bits of all interrupt sources and OPTION register contain interrupt edge select bit of external interrupt source

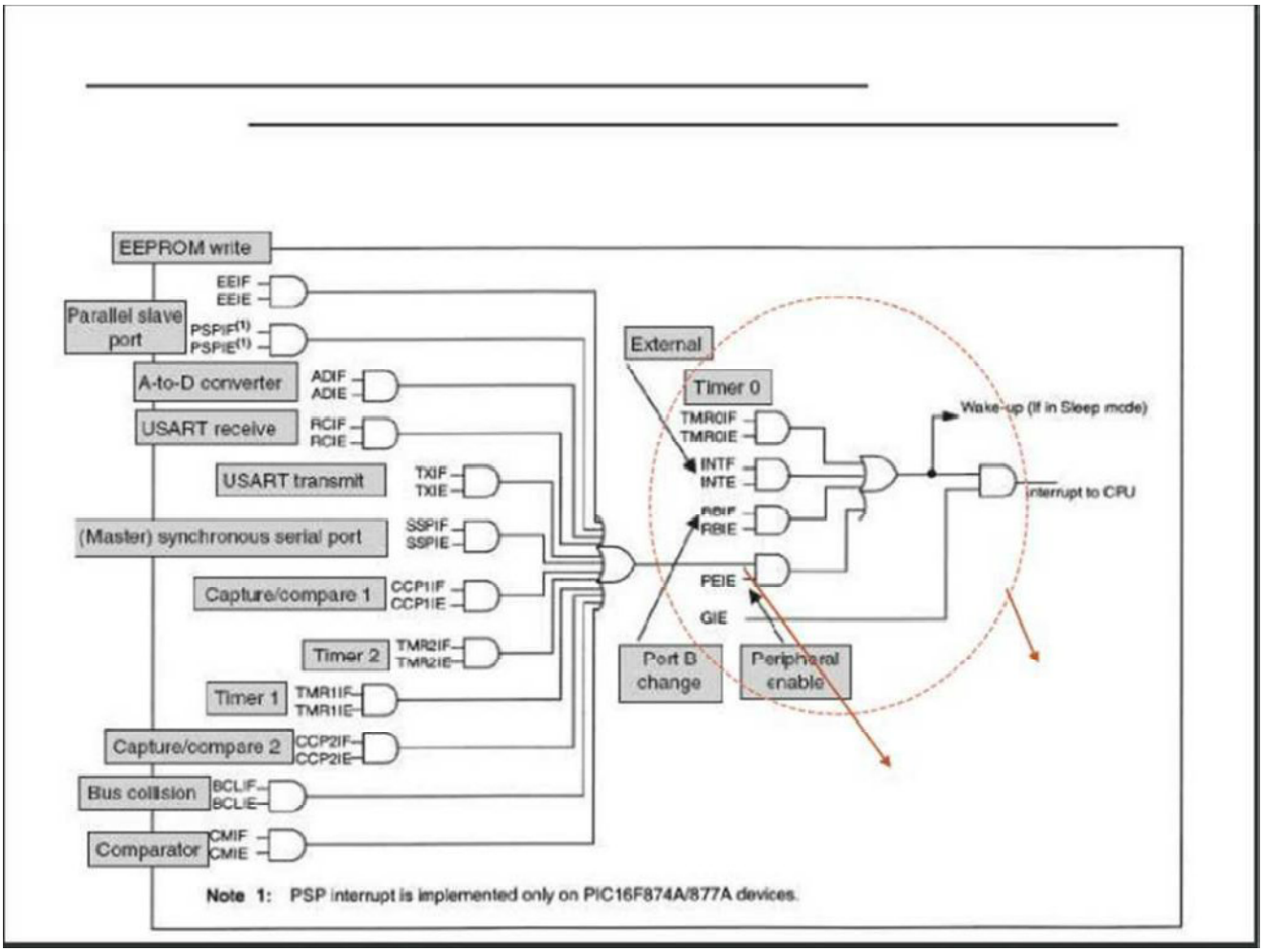
Each source has an enable line (labeled by E) and a flag line (labeled by F, actually these are the interrupt flags, rather than interrupt sources).



Note: All four 16 F84A interrupts can be enabled or disabled (maskable). Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit.

16F877A Interrupt Logic Structure

PIC16 F87XA family has up to 15 sources of interrupt, the interrupt structure (interrupt sources, individual interrupt flags and enable bits) of this family is illustrated below.



Similar to 16Ff84A interrupt structure, Previously it was EEPROM write complete in 16Ff84A. Now, with Peripheral Enable bit acts like a secondary Global Enable bit

16F877A INTCON Register and Interrupt related SFRs

Interrupt Registers

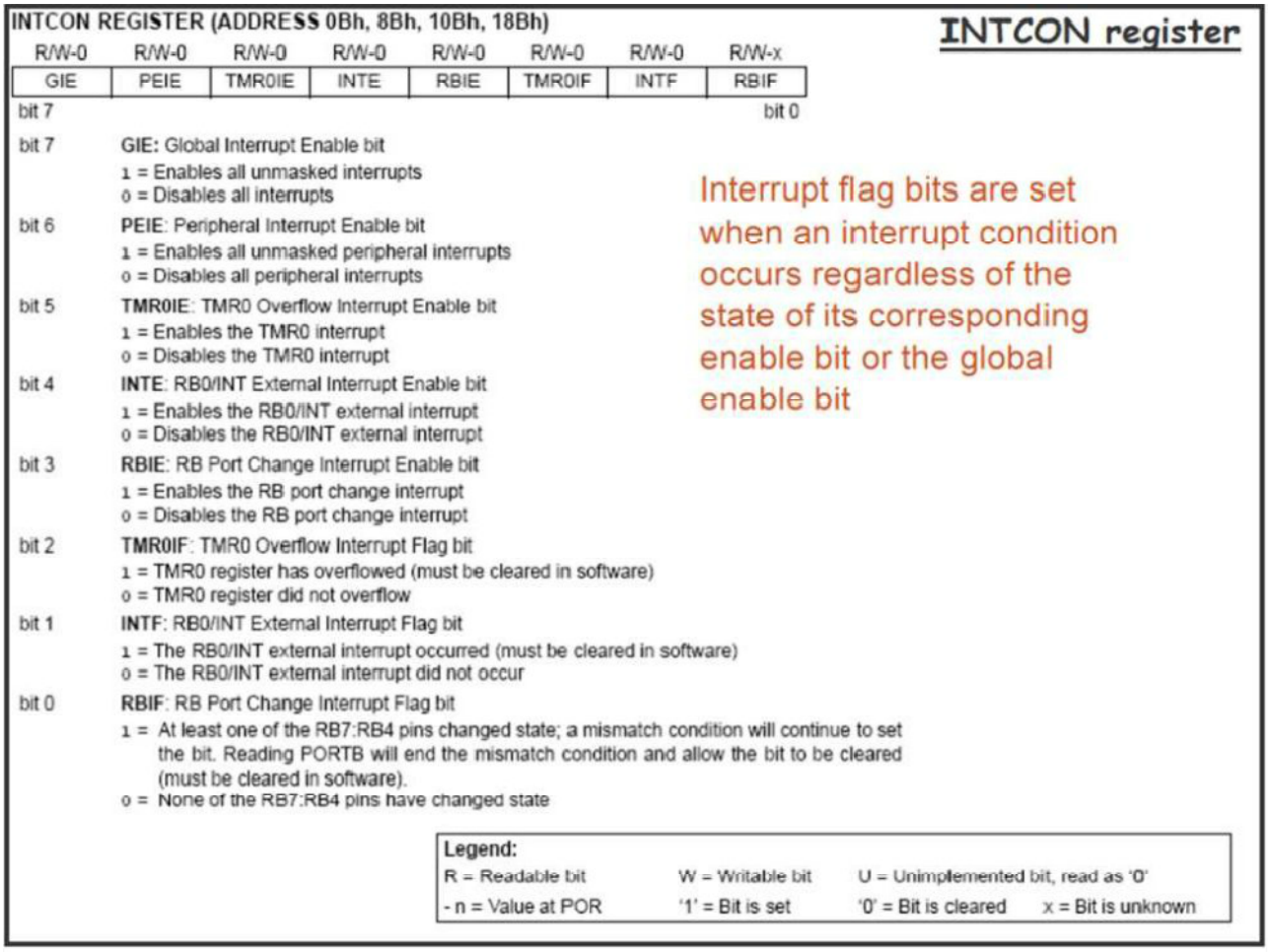
With 15 interrupt sources PIC16 F87XA family uses interrupt control register INTCON, and four special function registers S FRs (PIE1, PIE2, PIR1, PIR2)

INTCON register

- is a readable and writable register
- has individual and global interrupt enable bits.
- records individual interrupt requests in flag bits.
- contains enable and flag bits for the TMRO register overflow, RB port change, external RBO/INT pin interrupts ; and Global and Peripheral Interrupt Enable bits.

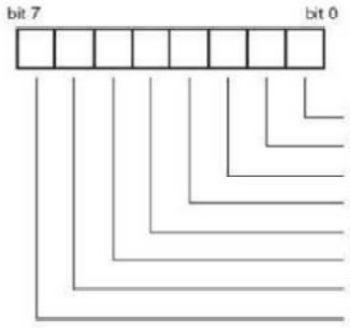
Interrupt related SFRs (PIE1, PIE2, PIR1, PIR2)

The peripheral interrupt flags are contained in the Special Function Registers, PIR1 and PIR2. The corresponding interrupt enable bits are contained in Special Function Registers, PIE1 and PIE2, and the peripheral interrupt enable bit is contained in Special Function Register, INTCON.



Peripheral Interrupt Enable and Peripheral Interrupt Request Registers

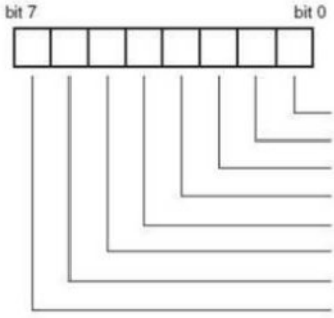
16F87XA PIE1/PIR1 Registers



	PIE1 (enable bits)	PIR1 (flag bits)
Timer 1 overflow	TMR1IE	TMR1IF*
Timer 2 overflow	TMR2IE	TMR2IF*
Capture compare 1	CCP1IE	CCP1IF*
Synchronous serial port	SSPIE	SSPIF*
USART transmit	TXIE	TXIF
USART receive	RCIE	RCIF
Analog-to-digital converter	ADIE	ADIF
Parallel slave port Read/Write**	PSPIE	PSPIF*

* Must be cleared in software
** 16F874/7 only. Reserved in 16F873/6

16F87XA PIE2/PIR2 Registers



	PIE2 (enable bits)	PIR2 (flag bits)
CCP2	CCP2IE	CCP2IF
Unimplemented	read as 0	read as 0
Unimplemented	read as 0	read as 0
Bus collision	BCLIE	BCLIF
EEPROM write	EEIE	EEIF*
Unimplemented	read as 0	read as 0
Comparator	CMIE	CMIF*
Unimplemented	read as 0	read as 0

* Must be cleared in software

CCS Pic-C: Intterupts

To enable or disable interrupts in CCS PIC-C following functions are used ;
enable_interrupts (*level)*
disable_ interrupts (*level)* where *level* is a constant defined in .h file.

These constants are defined the header file of a microcontroller such as "16f84a.h", "16f877a.h".
These constants that are used to enable or disable interrupt functions are:

GLOBAL : refer to global level interrupt

INT_RTCC : refer to specific level **TMRO** overflow interrupt (using **RTCC** as the name)

INT_RB : refer to specific level PORTB change interrupt on any RB4, RB5, RB6, RB7 pins

INT_EXT : refer to specific level external interrupt on **RBO-Int pin**

INT_EEPROM : refer to specific level **eeprom** write complete interrupt

INT_TIMERO : refer to specific level **TMRO** overflow interrupt (using **TIMERO** as the name)

INT_AD : refer to specific level **Analog to Digital Conversion** Complete interrupt

INT_TBE : refer to specific level **RS232 transmit** buffer empty interrupt

INT_RDA : refer to specific level **RS232 receive** data available interrupt

INT_TIMER1 : refer to specific level **TMR1** overflow interrupt

INT_TIMER2 : refer to specific level **TMR1** overflow interrupt

INT_CCP1 : refer to specific level **Capture or Compare** interrupt of CCP1 module

INT_CCP2 : refer to specific level **Capture or Compare** interrupt of CCP2 module

INT_SSP : refer to specific level **SPI or I2C** activity interrupt

INT_PSP : refer to specific level **Parallel Slave Port** data in interrupt

INT_BUSCOL : refer to specific level **Bus collision** interrupt

INT_COMP : refer to specific level **Comparator detect** interrupt

CCS Pic-C: Interrupts

When interrupts are enabled at global level, this will not enable any of the specific interrupts but will allow any of the specific interrupts previously enabled to become active.
When interrupts are disabled at global level, specific level interrupts are still active but are not taken into consideration by the CPU.
To be able to take care of what needs to be done when interrupts occur, programmer must write interrupt handler functions. These are usually called interrupt service routines (ISRs) and should have been defined for required interrupts.

In CCS PIC-C an #int_xxx directive followed by the xxx_isr() function is used to write ISR for each individual interrupt. Here, xxx can be the constants given previously (i.e. RTCC, RB, EXT, EEPROM, TIMERO, etc.)

The main structure of enabling interrupts and associated ISRs for individual interrupts are:

```
#int_RB
void RB_isr (void)
 {
....
....
}
#int_EXT
void EXT_isr (void)
{
....
....
}
#int_EEPROM
void EEPROM_isr (void)
{
....
....
}
.
#int_TIMERO    // Note: RTCC and Timer() are the same
void TIMERO_isr (void)
{
....
....
}


void main (void) (
enable_interrupts (INT_RB);
enable_interrupts (INT_EXT);
enable_interrupts (INT_EEPROM);
enable_interrupts (INT_TIMER0);
enable_interrupts (GLOBAL);
 ....
  ....
```

CCS Pic-C: Intterupts

External interrupt (INT_EXT) is edge sensitive. The ext_int_edge (edge) function is used to respond to an external interrupt on rising edge of the interrupting signal if the constant edge is H_TO_L or falling edge of the interrupting signal if edge is L_TO_H. The constants H_TO_L and L_TO_H are defined in .h file. Shortly, following function calls are used to select edge for the external interrupt.

ext_int_edge (H_TO_L)

ext_int_edge (L_TO_H)

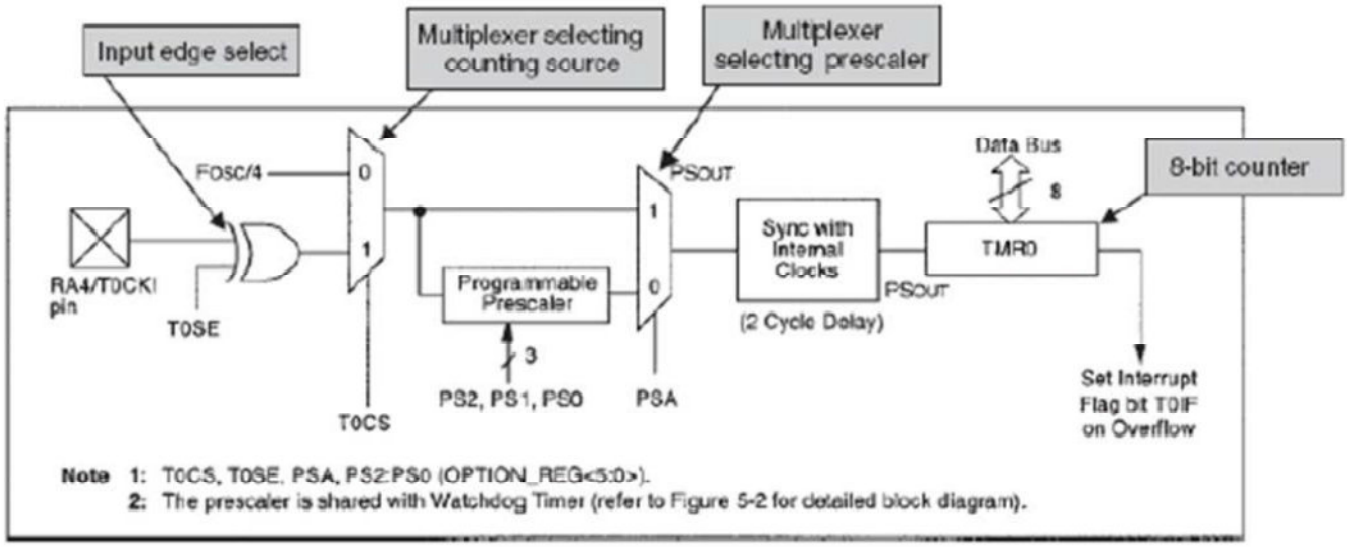
In contrast to assembly language there is no need for context saving (i.e. as storing the state of WREG, STATUS or user defined registers). C compiler takes care of the state of the MCU during ISR. Also note that interrupt service routines have void return type and they do not have any input parameters!

CCS Pic-C: TIMERO Module

PIC16F87XA TimerO module is the same timer/counter module used in PIC16F84A

TIMERO module con be setup for timing ond counting opplicotions using built-in function setup_timer_O (*mode*) function, where mode may be one or two of the constants defined in the devices .h file. These constants or RTCC_INTERNAL, RTCC_EXT_L_TO_H or RTCC_EXT_H_TO_L , referring to internal or external clock source selection. Hence, the counter can be clocked by an external pulse train or from MCU oscillator. In the case of external clock source, TMRO register being on 8-bit register in the TimerO module can be incremented during low-to-high (rising edge) or high-to-low (falling edge) source signal transitions on RA4/TOCKI pin. Prescaler that is used to divide the clock signal frequency input to the TMRO counter is obtoined by

RTCC_DIV_2,
RTCC_DIV_4,
RTCC_DIV_8,
RTCC_DIV_16,
RTCC_DIV_32,
RTCC_DIV_64,
RTCCDIV128,
RTCCDIV256.



CCS Pic-C: TIMERO Module

Timer( ) Module Operation:

Counter Mode:

In this counter mode operation pulses applied RA4 / TOCKI pin are counted in TMRO. These pulses can be input manually from a push-button, from a signal source such as a sensor. TMRO can also be loaded with an initial value.

Timer Mode:

In this mode, timing measurements and hardware-generated delays can be obtained by calculating the time between start of the counting from TMRO's preloaded initial value until timer overflow flag is set that causes interrupt, which is given by

$$(4 / F_{xt}) * A * (256 - B)$$

Where;
F_{xt} is oscillator frequency,
A is prescaler value,
B is initial value loaded to TMRO.

Example: If F_{xt} = 4MHz, A=32 and B=156 then total-time delay until TMRO overflow interrupt occurs is 3.2 msec.

CCS Pic-C: TIMERO Module

In the counter mode operation pulses applied RA4/T0CKI pin are counted in TMRO.

In timer mode, timer is incremented at time intervals of $(4/F_{xt}) * A$, where Fxt is oscillator frequency and A is prescaler value (A=RTCC_DIV_XX).

As an example, if Fxt = 4MHz, and prescaler is RTCC_DIV_8, then timer is incremented at each (4/4000000)*8=8microseconds.
Total time it takes for the timer to overflow is calculated using $(4/F_{xt})* A *(256-B)$,
Where;
F_{xt} is oscillator frequency,
A is prescaler value (A=RTCC_DIV_XX),
B is initial value loaded to TMRO.

Example: If F_{xt} = 4MHz, A=32 and B=156 then total-time delay until TMRO overflow interrupt occurs is 3.2 msec.
Functions to set a value to TMR0 and get its value are, set_timer0() and get_timer0(), respectively.

CCS Pic-C: TIMERO Module

void setup_timerO (mode)

This built-in function is used to configure TIMER0 module operation. Required parameters is given by the mode, which is a group of constant values that can be OR'ed using I operator. The constants are defined in the devices .h file (i.e. 16 F877A.h)

Constants that may be used in mode:
RTCC_I NTERNAL //refer to internal clock source selection
Or
RTCC_ExT_L_TO_H
or
RTCC_ExT_H_TO_L //referto external clock source selection
//with edge selection indicated //to increment the counter

and prescaler options can be:
RTCC_DIV_1,
RTCC_DIV_2,
RTCC_DIV_4,
RTCC_DIV_8,
RTCC_DIV_16,
RTCC_DIV_32,
RTCC_DIV_64,
RTCC_DIV_128,
RTCC_DIV_256

Note: RTCC (=Real Time Counter Clock) is another name for TIMERO.

void set_timer0 (*value*) same as set_rtcc (*value*)

This built-in function sets the 8-bit value of TMR0 by *value*. The counter counts up starting from this value. Timer0 counts upwards and when it reaches its maximum value of it will rollover to 0 and continue counting from 0

(for example if value=250, then counting will be as 250, 251, 252, 253, 254, 255, 0, 1, 2...).

Hence, after roll over, if it is required to start counting from an other value different from zero, the built-in function must be again used with required *value*.

int _get_timer0() same as get_rtcc ()
Returns the current 8-bit count value of the counter TMR0. OBSOLOTE built-in function:

setup_counters (rtcc_state, ps_state) // for old version compiler compatibility. DON'T USE!

Example instructions:

```
setup_timer_0 (RTCC_EXT_L_TO_H I RTCC_DIV_1);
                        //setup timer0 with external clock source, increment on rising
                        //edge of clock signal, prescaler=1 //such that there is no prescaling

setu p_timer_0(RTCC_INTERNAL I  RTCC_DIV_1);  //setup timer0 with internal clock source, no prescaling
setu p_timer_0(RTCC_INTERNAL I  RTCC_DIV_8);  //setup timer0 with internal clock source, prescaler=8
set_timer0(0); //set TMR0 to 0 i.e initialize it ; Note that Timer0 never stops, hence always
continues to  count !!!
set_timer0(100);       //set TMR0 to 100
data=get_timer0();   //get TMR0 value, where data must be an 8-bit variable
```

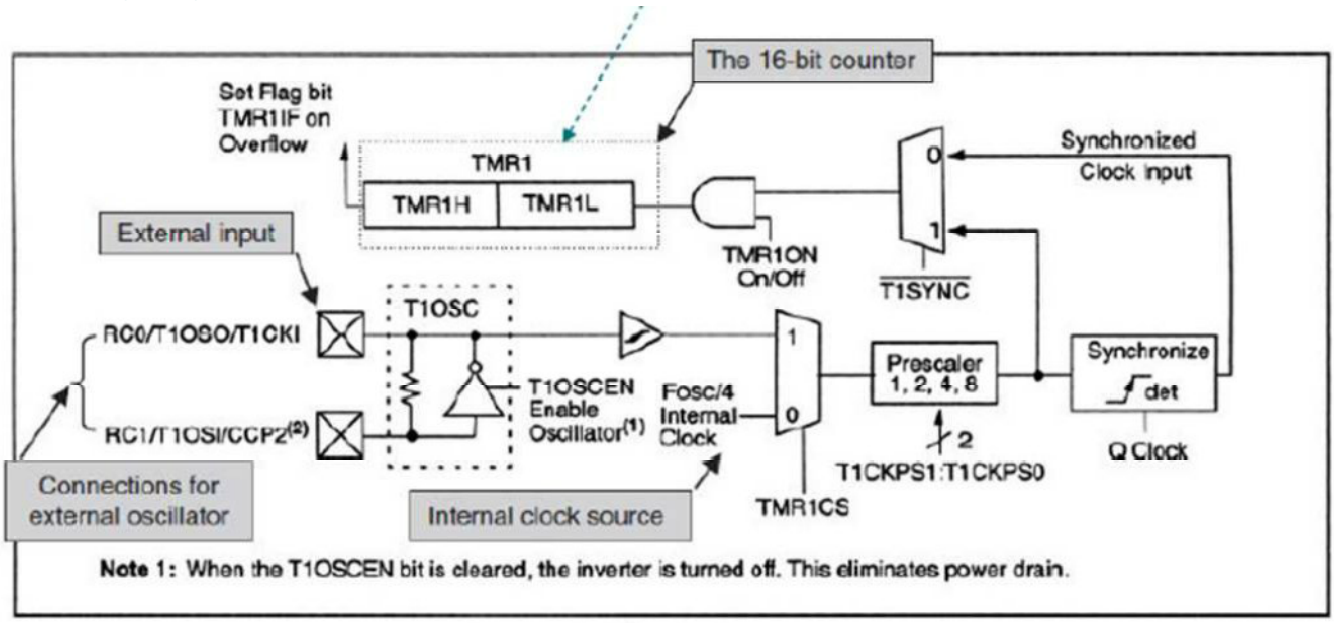
AKA is an abbreviation to "Also Known As". AKA is used frequently in the CCS PIC-C manual II

```
//Connect an LED to PORTB RB7 pin
//Connect RB0 pin to 0 voltage level than to +5 voltage level to generate interrupt
#include <16F877A.h>
#FU5E5 NOWDT,XT,PUT, NOPROTECT,BROWNOUT, NOLVP, NOCPD, NOWRT #use
delay(clock=4000000)
#use fast_io(B)
#int_EXT void EXT_isr()
{
delay_ms(20);         //software debounce
output_high(PIN_B7);        //RB7 pin at high voltage level
delay_ms(100);       //delay 0.1 seconds
output_low(PIN_B7);         //RB7 pin at low voltage level
}
void main()
{
set_tris_b(0b01111111);   //set PORTB bit 7 as output, other bits as input
output_bit(PIN_B7,0);       //RB7 pin at low voltage level
ext_int_edge(L_TO_H);      //select rising edge of external signal
enable_interrupts(INT_EXT);        //enable external interrupt on RB0 pin
enable_interrupts(GLOBAL);         //enable global interrupts
while (TRUE);//infinite loop
} //end of main
```

```
//Generate a signal ation PORTB pin RB0 using timer0 module
#include <16F877A.h>
#FU5E5 NOWDT,XT,PUT, NOPROTECT,BROWNOUT, NOLVP, NOCPD, NOWRT
#use delay(clock:4000000)
#use fast_io(B)
#int_TIMER0 void TIMER0_isr()
{
output_bit(PIN_B0,0); //RB0 at low voltage level
delay_ms(100);          //delay 0.1 seconds
}
void main()
{
set_tris_b(0x00);       //portb pins set as output
output_b(0x00);         //clear portb data
set_timer0(0);//set TMR0 initial value to zero
setup_timer_0(RTCC_INTERNALIRTCC_DIV_256);       //set timer0 mode
enable_interrupts(INT_TIMER0);       //enable timer overflow interrupt
enable_interrupts(GLOBAL);   //enable global interrupt
while(TRUE)
{
output_bit(PIN_B0,1); //RB0 at high voltage level
}
} //end of main
```

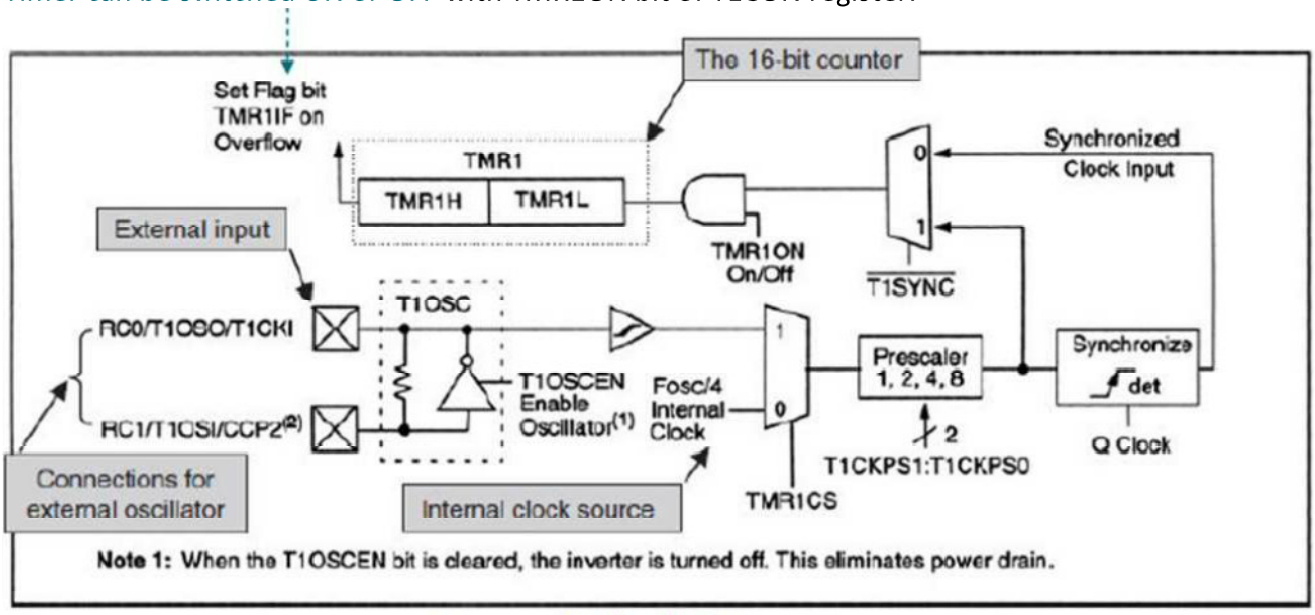
PIC16F87XA TIMER1 Module

Timer 1 module is an 16-bit timer/counter module. It is made up of two 8-bit registers, TMR1H and TMR1L, which are SFRs located at 0 F_H and 0E_H in data memory. With these two registers together, indicated as TMR1 in the block diagram, they can count from 0000 to $FFFF_H$ ($=65535_D=(2^{16}-1)$)



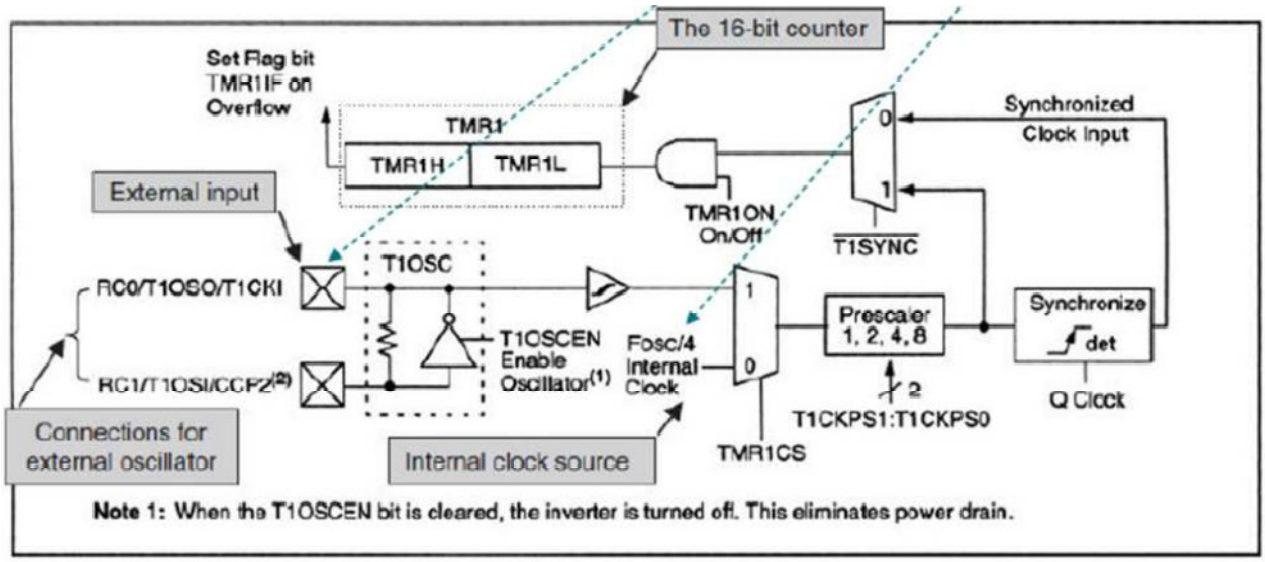
Timer 1 Block Diagram

When the count rolls over from $FFFF_H$ back to 0 the associated interrupt flag TMR1 IF is set.
Timer 1 operation is controlled by the T1CON register.
Timer can be switched ON or OFF with TMR1ON bit of T1CON register.



Timer 1 Block Diagram

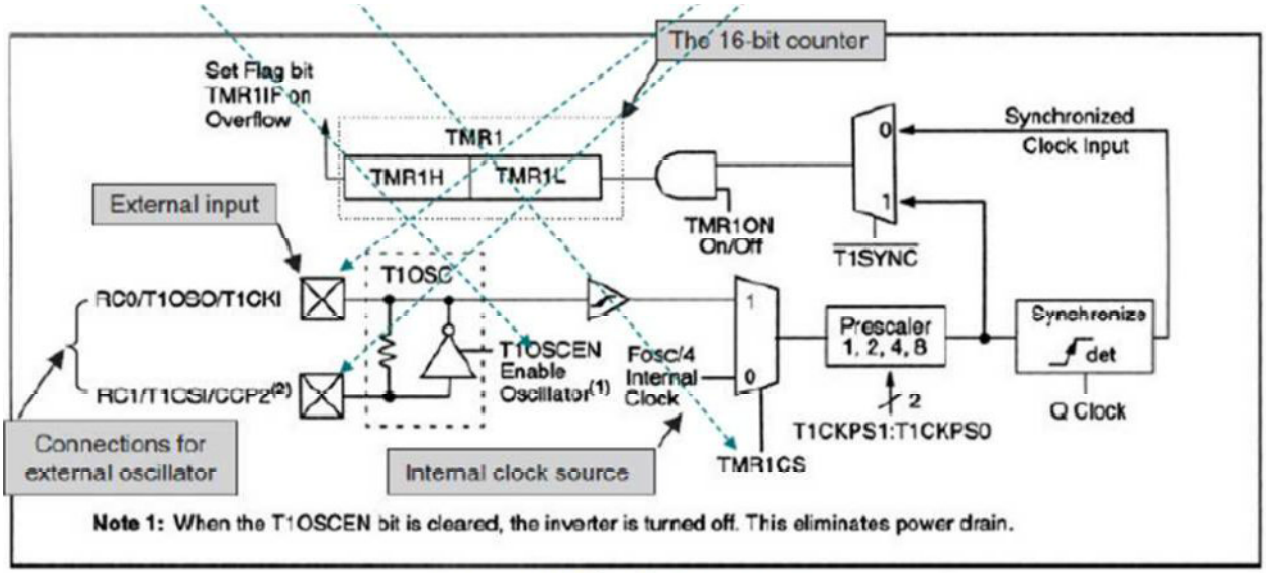
- Timer 1 has three distinct clock sources.
- For counting mode of operation, the external input T1CKI, which is shared with bit0 of PortC, must be used. And in this counting mode, counter always increments on rising edges of source.
- For timing mode of operation, internal clock oscillator (FO5C/4) source can be used.



Timer 1 Block Diagram

- TMR1C5 bit of T1CON control register is the external or internal clock source select bit of T1CON control register.

- Connecting a low frequency (i.e up to 200kHz) external oscillator between RCO/T1O5O and RC1/T1O5I pins provides the third option for clock source of TIMER 1 module. This external oscillator can be enabled by the T1O5CEN bit of T1CON.

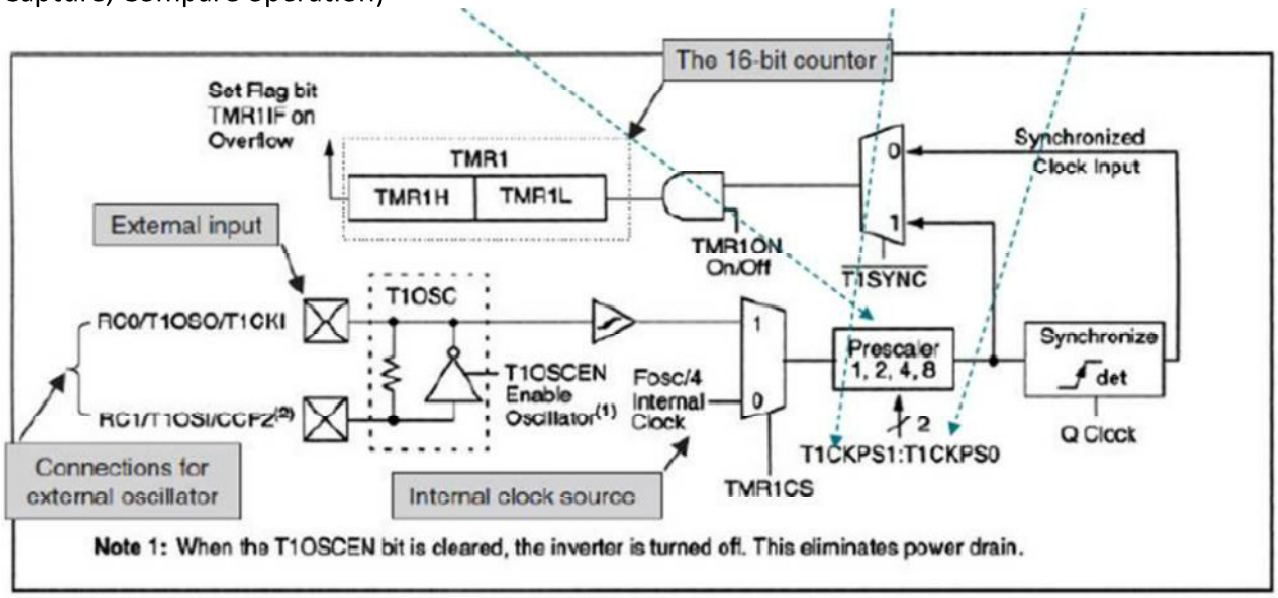


Timer 1 Block Diagram

• Prescaler can be used for all of three clock sources.

• Prescaler values (1, 2, 4, 8) are set by the TICKPS1 and TICKPS0 bits of T1CON control register.

• Synchronisation of external clock with internal clock is obtained by bit T1SYNC (required in Capture, Compare operation)



Timer 1 Block Diagram

T1CON control register

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

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

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits

11 = 1:8 prescale value
10 = 1:4 prescale value
01 = 1:2 prescale value
00 = 1:1 prescale value

Prescale bits

bit 3 **T1OSCEN:** Timer1 Oscillator Enable Control bit

1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **T1SYNC:** Timer1 External Clock Input Synchronization Control bit

When TMR1CS = 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/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

Clock source /Oscillator selection/enable bits

bit 0 **TMR1ON:** Timer1 On bit

1 = Enables Timer1
0 = Stops Timer1

Timer1 module can be turned ON or OFF

Note: In the 16-Series register of this name, bit 7 is called **RD16.** If set to 1 it enables the '16-bit Read/Write' mode

Figure 9.2 The Timer 1 control register, **T1CON** (address 10_H)

Void setup_timer_1 (mode)

This built-in function is used to configure T/MER 1 module operation. Required parameters is given by the mode, which is a group of constant values that can be OR'ed using I operator. The constants are defined in the devices .h file (i.e. 16F877A.h). T/MER 1 is a 16 bit timer. The 16-bit timer value may be read and written to using set_timerl() and get_timerl().

Constants that m ay be used in mode:

Tl_DISABLED

//to turn off Timer 1 (related to T1CON control register bit 0 TMR1ON)

Tl_INTERNAL

///nternal clock source

Tl_EXTERNAL, Tl_EXTERNAL_SYNC

//External clock source at RC0/T1CK/ pin, and synchronization to internal clock

Tl_CLK_OUT

//External low frequency oscillator as a clock source with the external oscillator

//between RC0/T1OSO and RC1/T1OS/ pins And prescaler options can be:

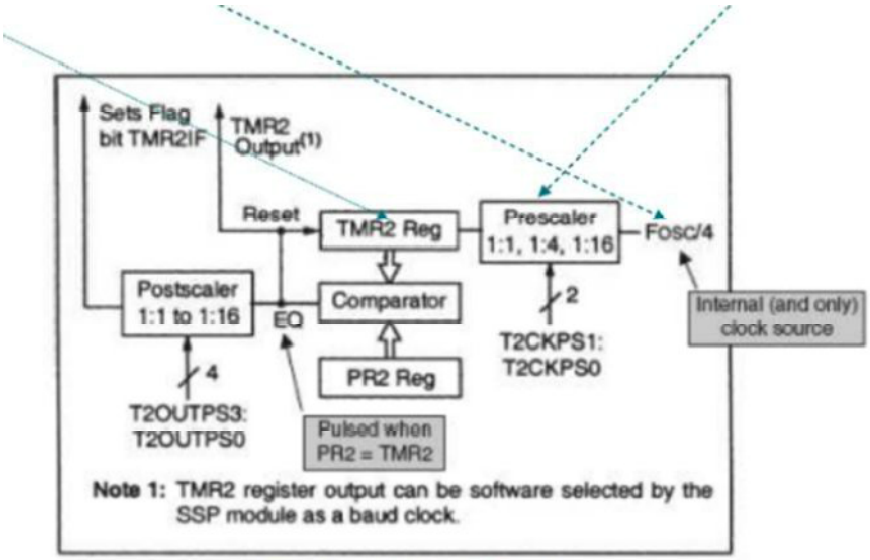
Tl_DIV_BY_l, Tl_DIV_BY_2, Tl_DIV_BY_t, Tl_DIV_BY_8

instructions: Example

```
setup_timer_1 ( T1_D/SABLED );      //Turn OFF Timer1 module
setup_timer_1 ( T1_/NTERNAL I T1_D/V_BY_4 );
setup_timer_1 ( T1_/NTERNAL I T1_D/V_BY_ 8);
//With an internal clock at 20mhz and with the T1_D/V_BY_8 mode, the timer will increment every
//1.6us. /t will overflow every 104.8576ms.
```

PIC16F87XA Timer 2, Comparator and PR2 Register

16F87XATimer 2 module is an 8-bit device. Timer 2 is driven only from the internal oscillator. The 8-bit readable/ writable timer register TMR2 is located at 11_H in data memory. Modest prescaling (1,4,16) is possible.



Timer 2 Block Diagram

T2CON control register

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

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

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

0000 = 1:1 postscale
0001 = 1:2 postscale
0010 = 1:3 postscale
•
•
•
1111 = 1:16 postscale

Postscale bits

bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

Timer2 module can be turned ON or OFF

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

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Prescale bits

Figure 9.5 The Timer 2 control register, **T2CON**

Timer 2 module has period register PR2, at memory location 92_H. The programmer can preset PR2 to a value, which is when the timer is on, continuously compared with the TMR2 register value. If TMR2 reaches PR2 value then TMR2 is cleared to zero on the next increment cycle. Hence, there are PR2+1 cycles between each reset of TMR2 register. This reset is the TMR2 output illustrated in the block diagram, and can be used as a baud rate generator by the synchronous serial port (55P). The resets forms an input to the postscaler, which can be used as an interrupt. The postscaler is controlled by the Timer 2 postscaler bits (TOUTP53, TOUTP52, TOUTP51) of the T2CON control register.

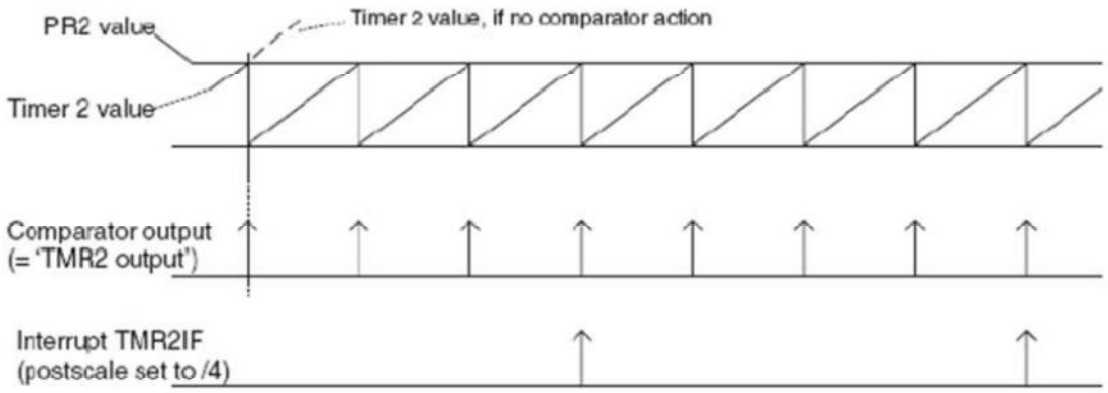


Figure 9.6 PR2 and comparator action

Void setup_timer_2 (mode, period, postscale)
This built-in function is used to configure T/MER 2 module operation. The mode specifies whether the timer is disabled or the prescaler values 1, 4, 16. The timer value may be read and written to using get_timer2() and set_timer2(). Timer 2 is an 8-bit counter/timer.
mode may be one of: T2_D/SABLED, T2_D/V_BY_1, T2_D/V_BY_4, T2_D/V_BY_16 //To dis able Timer 2, and prescaler values period is a int 0-255 that determines when the clock value //period is PR2
postscale is a number 1-16 that determines how many timer resets before an interrupt: (1 means one reset, 2 means 2, and so on).

Example instructions:
setup_timer_2 (T2_D/V_BY_4, 0xc0, 2);
// At 20mhz, the timer will increment every 800ns: Fosc= 20MHz 3 Fosc/4 =5MHz 3
// 5MHz/4=1.25MHz31/(1.25MHz=800ns)
// will overflow every 153.6us: $0xC0_H = 192_D$ 3 192x800ns 3 153.6 microseconds
// and will interrupt every 307.2us: 2x153.6microseconds=307.2microseconds.

Capture / Compare / PWM (CCP) Modules

- A register that can record the time of an event is called a 'Capture[3] register.
- A register that can generate an event by comparing for equality of a preset value in a register i.e. PR2 to a running timer is called a 'Compare[3] register.
- PIC16 Series MCUs combine these Capture and Compare functions and some more functions in their CCP modules.
- CCP modules interact with both Timer 1 and Timer 2.
- 16 F87XA has two CCP modules.
- Each module has two 8-bit registers, CCPR1L, CCPR1H and CCPR2L, CCPR2H.
- Two 8-bit registers in each module forming a 16-bit register, can be used for capture, compare or to form the duty cycle of a PWM signal.
- CCP modules are controlled by CCP1CON and CCP2CON registers.

Capture / Compare / PWM (CCP) Modules

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

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

bit 5-4 **CCPxX:CCPxY:** PWM Least Significant bits

Capture mode:
Unused.

Compare mode:
Unused.

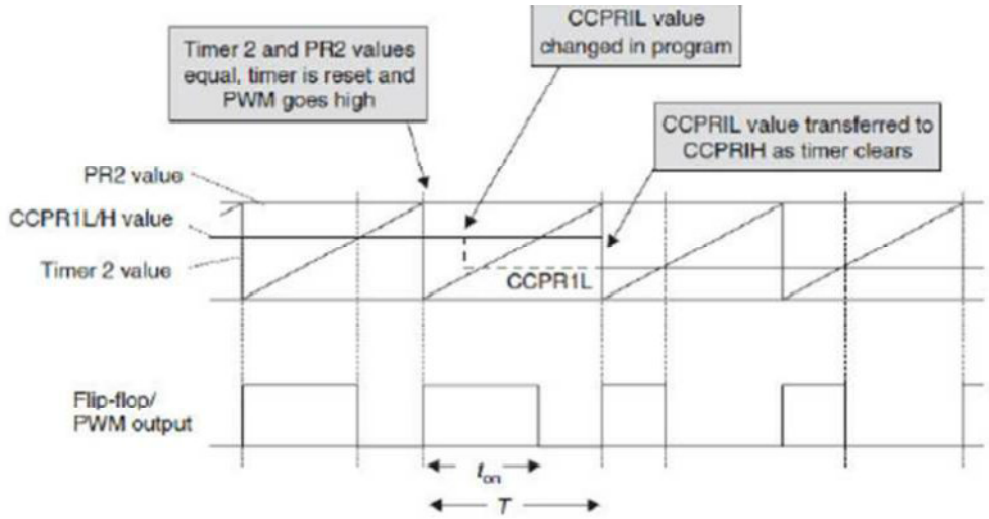
PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0:** CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

Figure 9.7 The CCP1CON/CCP2CON registers (addresses 17_H and $1D_H$ respectively)

Capture / Compare / PWM (CCP) Modules PWM Signal



$$T = (\mathbf{PR2} + 1) \times (\text{Timer 2 input clock period})$$

$$= (\mathbf{PR2} + 1) \times \{\text{Tosc} \times 4 \times (\text{Timer 2 prescale value})\} \quad (9.2)$$

$$t_{on} = (\text{pulse width register}) \times \{\text{Tosc} \times (\text{Timer 2 prescale value})\} \quad (9.3)$$

PR2 register is loaded with 249_D. clock oscillator frequency is 4 MHz

PWM period is: $T = (\mathbf{PR2} + 1) \times \{\text{Tosc} \times 4 \times (\text{Timer 2 prescale value})\}$

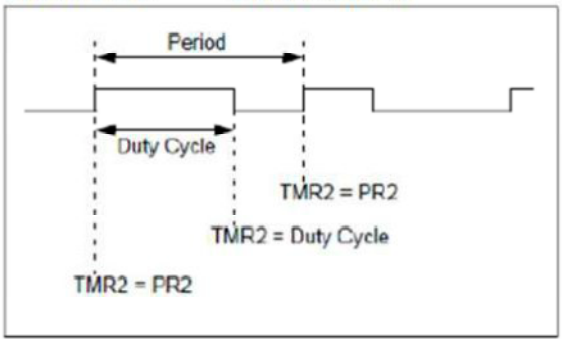
$= 250 \times (250\,\text{ns} \times 4 \times 1)$

$= 250\,\mu\text{s}$

PWM frequency $= 4.00\,\text{kHz}$

Capture / Compare / PWM (CCP) Modules PWM Signal

FIGURE 8-4: PWM OUTPUT



If the PWM duty cycle value is longer than the PWM period, the CCP1 pin will not be cleared.

PWM Period = [(PR2) + 1] • 4 • Tosc • (TMR2 Prescale Value)

PWM frequency= 1/[PWM period]

PWM Duty Cycle = (CCPR1L:CCP1CON<5:4>) • Tosc • (TMR2 Prescale Value)

The maximum PWM resolution (bits) for a given PWM frequency is given by the following formula.

EQUATION 8-1:

$$\text{Resolution} = \frac{\log\left(\frac{FOSC}{FPWM}\right)}{\log(2)} \text{ bits}$$

TABLE 8-3: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS AT 20 MHz

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFFh	0xFFh	0xFFh	0x3Fh	0x1Fh	0x17h
Maximum Resolution (bits)	10	10	10	8	7	5.5

CCS PIC-C built-in functions for PWM generation are:

void setup_ccpl(CCP_PWM) // CCP_PWM indicates that CCP module will be
// used for PWM generation,signal output to CCP1 pin

void setup_ccp2(CCP_PWM) // CCP_PWM indicates that CCP module will be
// used for PWM generation, signal output to CCP2 pin

void set_pwml_duty (value) // value is the 8- or 10-bit value indicating duty time (t),
// i.e. value of pulse-width register on

void set_pwm2_duty (value) // value is the 8- or 10-bit value indicating duty time (t),
// i.e. value of pulse-width on

PWM Period T = (PR2+1) * (Tosc *4)* (Timer2 PrescaleValue) ...(Textbook Eqn. 9.2)

t_{on}=(value of pulse width register)* (Tosc* (Timer2 PrescaleValue)) ...(Textbook Eqn. 9.3)