Chapter 11

Clocks, Watchdog Timer / Timers

Read Sections 12-16 of

Data Sheet for PIC18F46K20

Updated: 4/19/19

Reset Conditions

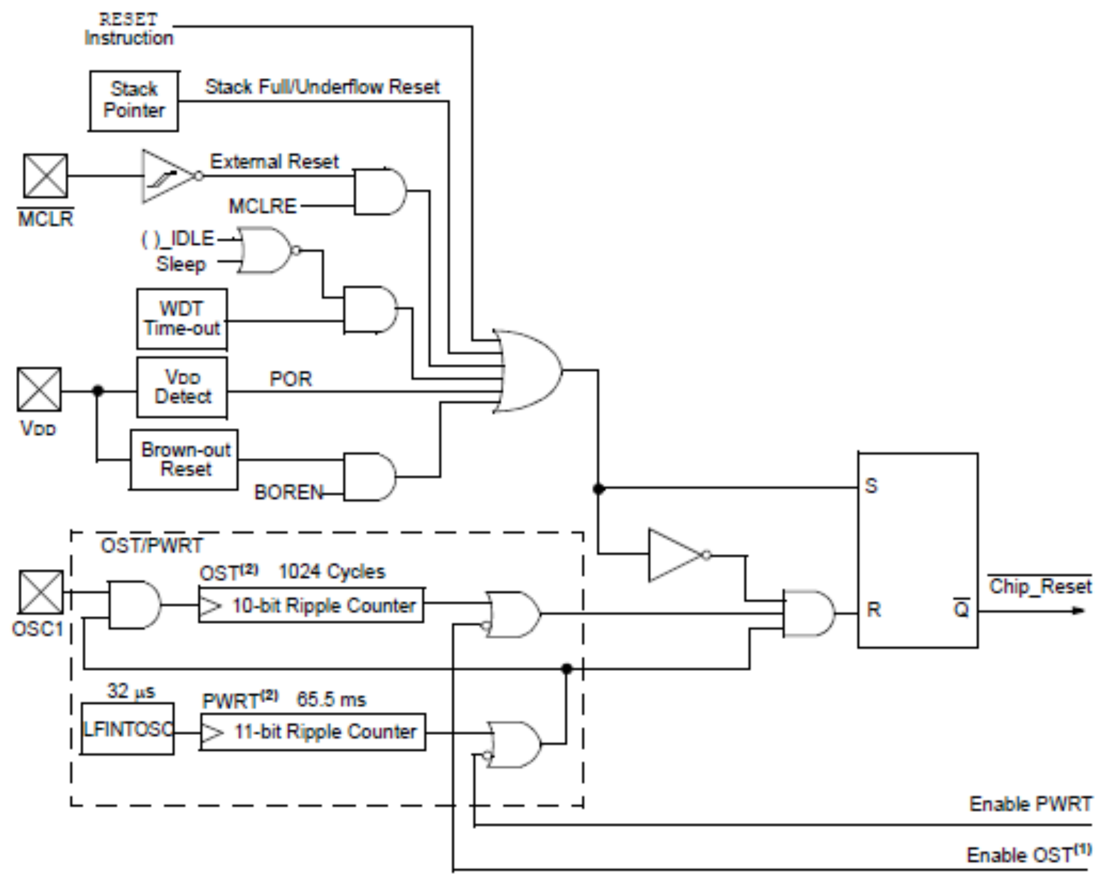
Master Clear

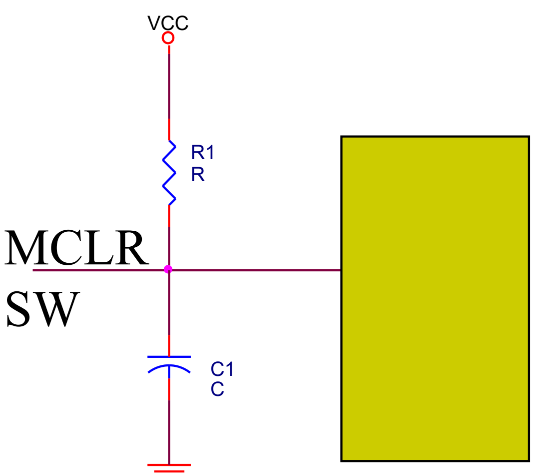
Initializes the MCU

Starts with memory 0x00

RC time constant 10-20 msec (R=10K/C=1uF)

#pragma config MCLRE = ON





Watchdog Timer

- The watchdog timer is a device that resets the microcontroller if it is allowed to expire.
- The watchdog timer is programmable to expire between 4 ms and 131 seconds.
- The watchdog timer is restarted with a ClrWdt() function in C-Language to reset it so it does not expire and cause a reset.

C statement	Assembly Language	Scaling factor	Time to Reset
#pragma config WDTPS = 1	_WDTPS_1_2H	1:1	4 ms
#pragma config WDTPS = 32768	_WDTPS_32768_2H	1:32768	131.072 sec

WD Example

- Example of how WD operates:
  - Make sure you RELEASE the program on the DEMO board
  - As you reset (GND) RB0 the WD will expire and thus the program keeps resetting → RD1 blinks.
- The time it takes for the WD to be enabled depends on the value of CONFIG2H register (WDTPS) (1024 x 4msec = 5sec) → When RB0 9s set for about 5 seconds later the WD will be enabled, resetting the program:

```
#define     PB0     PORTBbits.RB0
#pragma config WDTPS = 1024
```

```
/** D E C L A R A T I O N S *******************************************
#define     PB0     PORTBbits.RB0

void main (void)
{
    TRISD = 0b00000000;          // PORTD bits 7:0 are all outputs (0)
    INTCON2bits.RBPU = 0;        // enable PORTB internal pullups
    WPUBbits.WPUB0 = 1;          // enable pull up on RB0
    ANSELH = 0x00;               // AN8-12 are digital inputs (AN12 on R
    TRISBbits.TRISB0 = 1;        // PORTB bit 0 (connected to switch) is

    //setting the WD registers
    RCON = 0b0001000;
    WDTCON = 1;

    PORTDbits.RD1 = 1;  // This indicates that program just reset
    Delay1KTCYx(500);

    while(1)
    {
        ClrWdt();
        PORTDbits.RD1 = 0;   // Clear RD1
        PORTDbits.RD0 = ~PORTDbits.RD0;
        Delay1KTCYx(500);
        while (PB0 == 0)
            PORTDbits.RD0 = PB0;
    }

}
```

Automatic Wakeup!

/In this program the LED blinks for a few seconds and then the program goes to sleep for about 10 seconds. Then, it wakes up, following watchdog trigger.

```
/** D E C L A R A T I O N S *******************************************/
#define     PB0     PORTBbits.RB0
#pragma config WDTPS = 2048      // about 10 sec.

unsigned char count = 0;

void main (void)
{
    TRISD = 0b00000000;          // PORTD bits 7:0 are all outputs (0)
    INTCON2bits.RBPU = 0;        // enable PORTB internal pullups
    WPUBbits.WPUB0 = 1;          // enable pull up on RB0
    ANSELH = 0x00;               // AN8-12 are digital inputs (AN12 on RB0)
    TRISBbits.TRISB0 = 1;        // PORTB bit 0 (connected to switch) is input (1)

    //setting the WD registers
    RCON = 0b0001000;
    WDTCON = 1;

    PORTDbits.RD1 = 1;  // This indicates that program just reset
    Delay1KTCYx(100);

    while(1)
    {
        ClrWdt();
        count = count + 1;
        PORTDbits.RD1 = 0;   // Clear RD1
        PORTDbits.RD0 = ~PORTDbits.RD0;
        Delay1KTCYx(20);  // Note that if delay must be within 5 sec WD time
        while (PB0 == 0)
            PORTDbits.RD0 = PB0;
        if (count == 20)
        {
            count = 0;      // upon each resent the count is reset
            Sleep();
        }
    }
}
```

- Measure the current when the board is in sleep mode!
- Where does the program start when it wakes up?

Brownout Reset

- The brownout reset is programmed and used to reset the microcontroller if the power supply voltage drops below a pre-programmed value.
- The brownout reset triggers the microcontroller and waits at the reset state until the power supply voltage returns to a level higher then the programmed brownout voltage.

C language	**Assembly Language**	**Brownout Voltage**
#pragma config BORV = 45	_BORV_45_2L	4.5 V
#pragma config BORV = 42	_BORV_42_2L	4.2 V
#pragma config BORV = 27	_BORV_27_2L	2.7 V
#pragma config BORV = 20	_BORV_20_2L	2.0 V

Clocks

- The PIC18 family allows many different clocking modes for operation. Some include internal timing and some external.
- External timing sources are very accurate and are crystal- or resonator-based. A less accurate, but less expensive timing source is an RC circuit. An oscillator module or external timing signal can also be used for the microcontroller.

Clock Sources

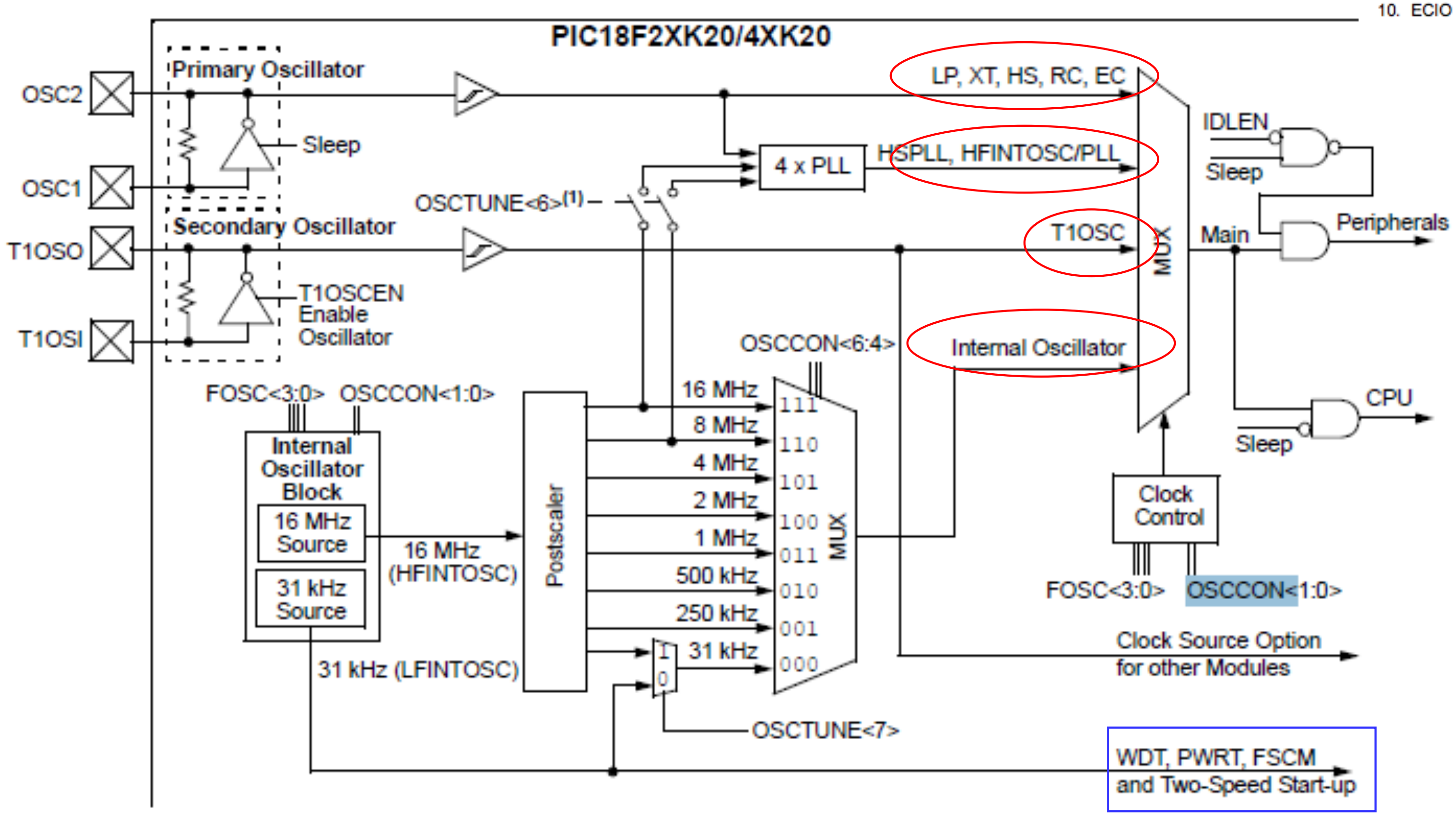
- 1. Low power crystal (LP)
- 2. Crystal or ceramic resonator (XT)
- 3. High-speed crystal or ceramic resonator (HS)
- 4. High-speed crystal or ceramic resonator with PLL (HSPLL)
- 5. External resister/capacitor with Fosc/4 output on OSC2 (RC)
- 6. External resister/capacitor with I/O on OSC2 (RCIO)
- 7. *Internal oscillator with Fosc/4 on RA6 and I/O on RA7 (INTIO1)
- 8. *Internal oscillator with I/O on RA6 and RA7 (INTIO2)
- 9. External clock with Fosc/4 (EC)
- 10. External clock with I/O on RA6 (ECIO)

Examples of External XTL or ceramic Resonator (OSC1/OSC2)

- *some versions do not have an internal oscillator and
- some versions may have additional modes

MCU Clock Source Diagram

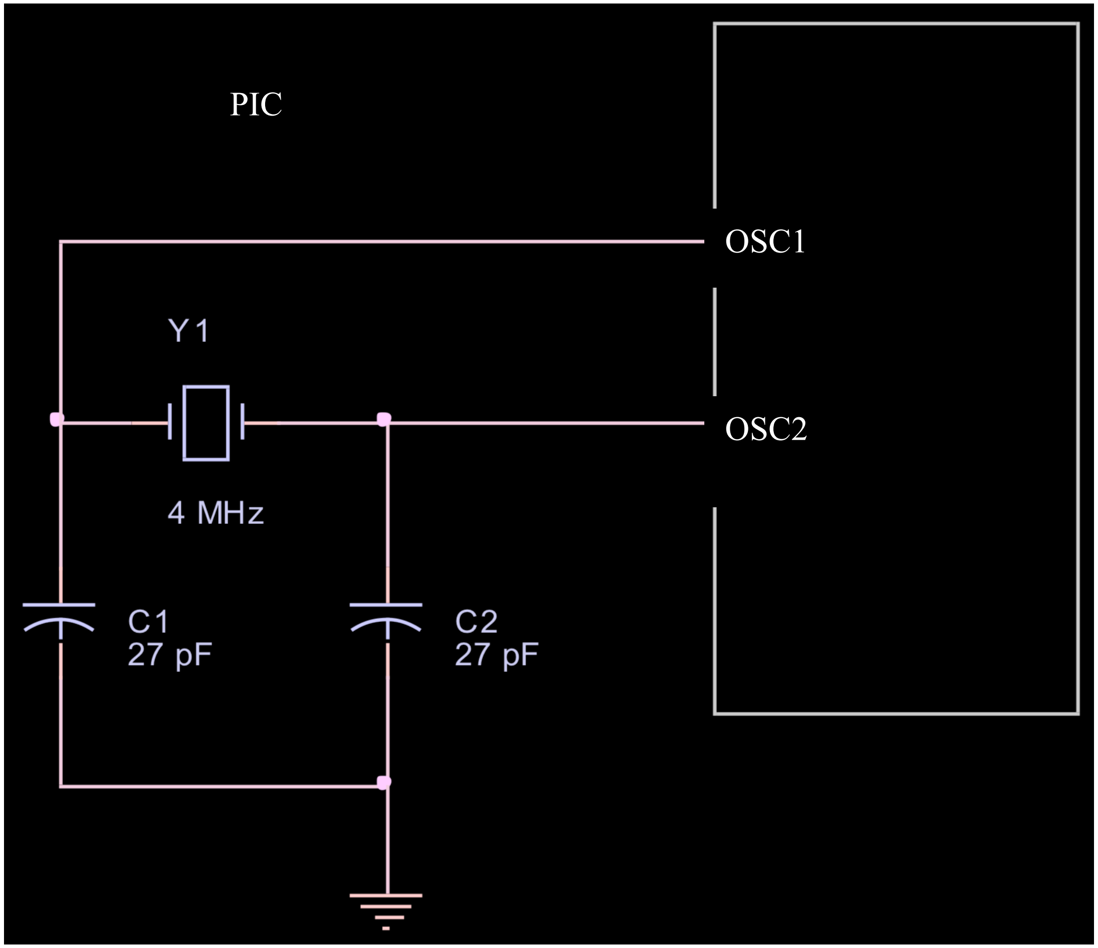
1. LP — Low-Power Crystal
2. XT — Crystal/Resonator
3. HS — High-Speed Crystal/Resonator
4. HSPLL — High-Speed Crystal/Resonator with PLL enabled
5. RC — External Resistor/Capacitor with FOSC/4 output on RA6
6. RCIO — External Resistor/Capacitor with I/O on RA6
7. INTOSC — Internal Oscillator with FOSC/4 output on RA6 and I/O on RA7
8. INTOSCIO — Internal Oscillator with I/O on RA6 and RA7
9. EC — External Clock with FOSC/4 output
10. ECIO — External Clock with I/O on RA6



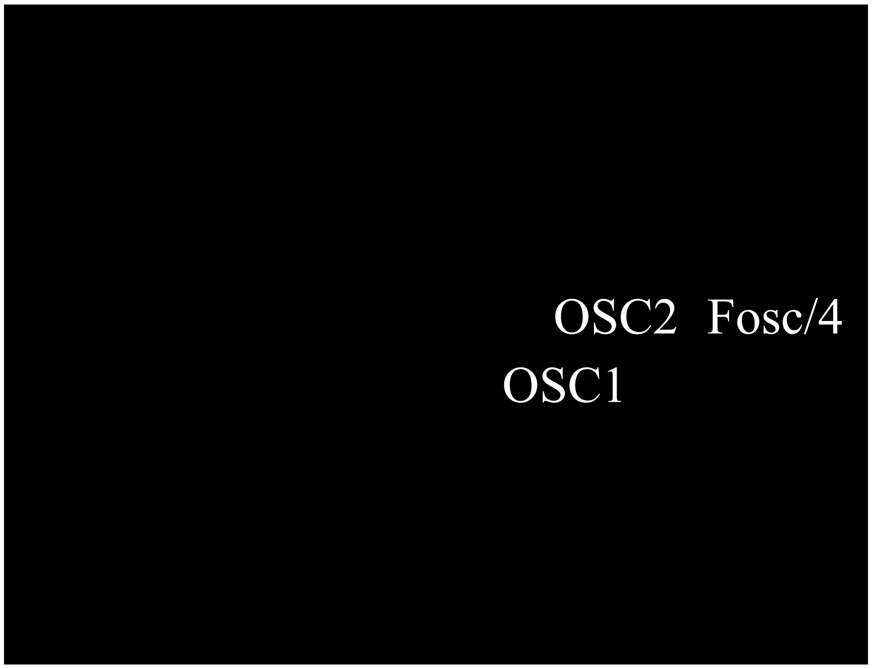
Note 1: Operates only when HFINTOSC is the primary oscillator.

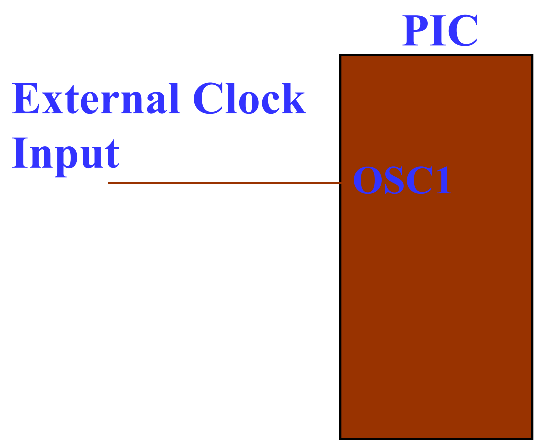
XTL / Ceramic Clock Source Connection

PLL internal function
Allows multiplying the
External clock by 4;
This is used to reduce the EMI
(Electromagnetic Interference)
on the board



RC Oscillator
Clock Source Connection





External clock source
Connected to OSC1

- External resister/capacitor with Fosc/4 output on OSC2 (RC)

2 MHz operation is attained with R = 3.9K
and C = 30 pF, Fosc/4 is 500 KHz with these values
Frequency = 1/[RC(4.2)] ; can vary slightly

Clock Examples

- #pragma config OSC = HS // high speed crystal oscillator
- #pragma config OSC = RC // RC oscillator
- #pragma config OSC = INTIO1 // internal oscillator

R/W-0	R/W-0	R/W-1	R/W-1	R-q	R-0	R/W-0	R/W-0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS(1)	IOFS	SCS1	SCS0
bit 7							bit 0

OSCCON Register

1. LP Low-Power Crystal
2. XT Crystal/Resonator
3. HS High-Speed Crystal/Resonator
4. HSPLL High-Speed Crystal/Resonator with PLL enabled
5. RC External Resistor/Capacitor with Fosc/4 output on RA6
6. RCIO External Resistor/Capacitor with I/O on RA6
7. INTOSC Internal Oscillator with Fosc/4 output on RA6 and I/O on RA7
8. INTOSCIO Internal Oscillator with I/O on RA6 and RA7
9. EC External Clock with Fosc/4 output
10. ECIO External Clock with I/O on RA6

bit 7 **IDLEN**: Idle Enable bit
1 = Device enters Idle mode on SLEEP instruction
0 = Device enters Sleep mode on SLEEP instruction

bit 6-4 **IRCF<2:0>**: Internal Oscillator Frequency Select bits
111 = 16 MHz (HFINTOSC drives clock directly)
110 = 8 MHz
101 = 4 MHz
100 = 2 MHz
011 = 1 MHz(3)
010 = 500 kHz
001 = 250 kHz
000 = 31 kHz (from either HFINTOSC/512 or LFINTOSC directly)(2)

bit 3 **OSTS**: Oscillator Start-up Time-out Status bit(1)
1 = Device is running from the clock defined by FOSC<2:0> of the CONFIG1 register
0 = Device is running from the internal oscillator (HFINTOSC or LFINTOSC)

bit 2 **IOFS**: HFINTOSC Frequency Stable bit
1 = HFINTOSC frequency is stable
0 = HFINTOSC frequency is not stable

bit 1-0 **SCS<1:0>**: System Clock Select bits
1x = Internal oscillator block
01 = Secondary (Timer1) oscillator
00 = Primary clock (determined by CONFIG1H[FOSC<3:0>]).

Programming Example

```
#pragma config MCLRE = ON          // enable master clear input
#pragma config OSC = HS            // select crystal oscillator
#pragma config WDT = ON            // set watchdog
#pragma config WDTPS = 256         // watchdog time is 1 second
#pragma config BORV = 42           // set brownout reset voltage
#pragma BOR = ON                   // brownout is on

void main(void)
// initialize system here
        while ( 1 )                // main program loop
        {
                ClrWdt();          // reset watchdog

                // system software goes here

        }
```

Basic Concepts in Counters and Timers

- In digital systems
  - Counting is a fundamental concept.
  - Clock is an essential element.
  - Count is in synchronization with the clock.
  - Count is converted in time by multiplying the count and the clock period.

Hardware Counters and Timers

- Counter is a register that can be loaded with a binary number (count) which can be decremented or incremented per clock cycle.
- Calculating time:
  - Find the difference between the beginning count and the last count
  - Multiply the count difference by the clock period
- The register can also be used as a counter by replacing the clock with a signal from an event.
- When a signal from an event arrives, the count in the register is incremented (or decremented); thus, the total number of events can be counted.

Types of Counters

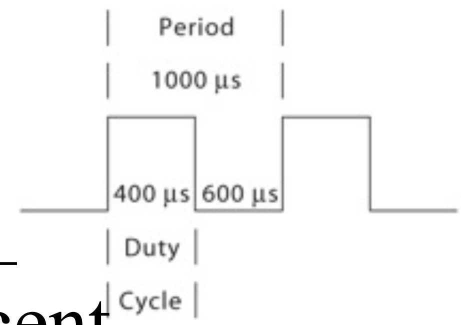
- Up-counter
  - Counter is incremented at every clock cycle
  - When count reaches the maximum count, a flag is set
  - Counter can be reset to zero or to the initial value
- Down-counter
  - Counter is decremented at every clock cycle
  - When count reaches zero, a flag is set
  - Counter can be reset 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

What are applications on timers?

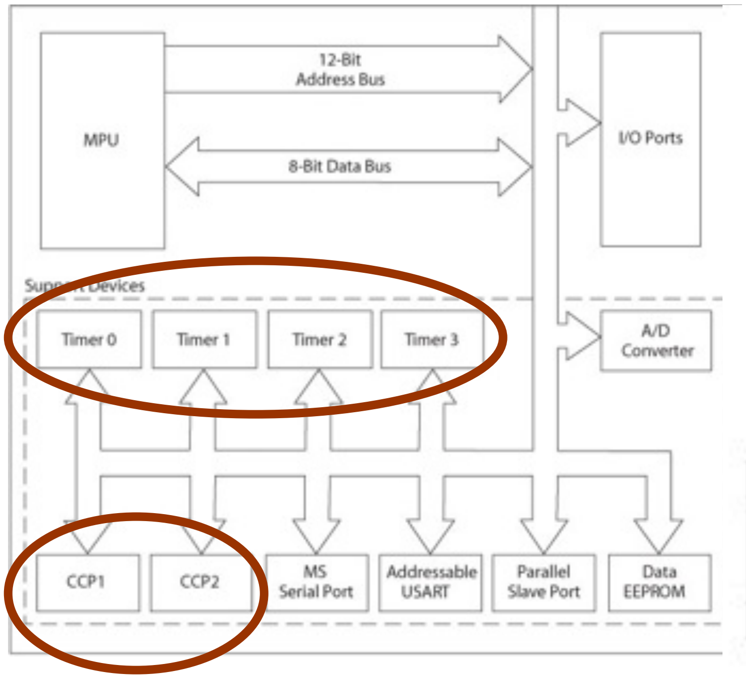
Timer Applications

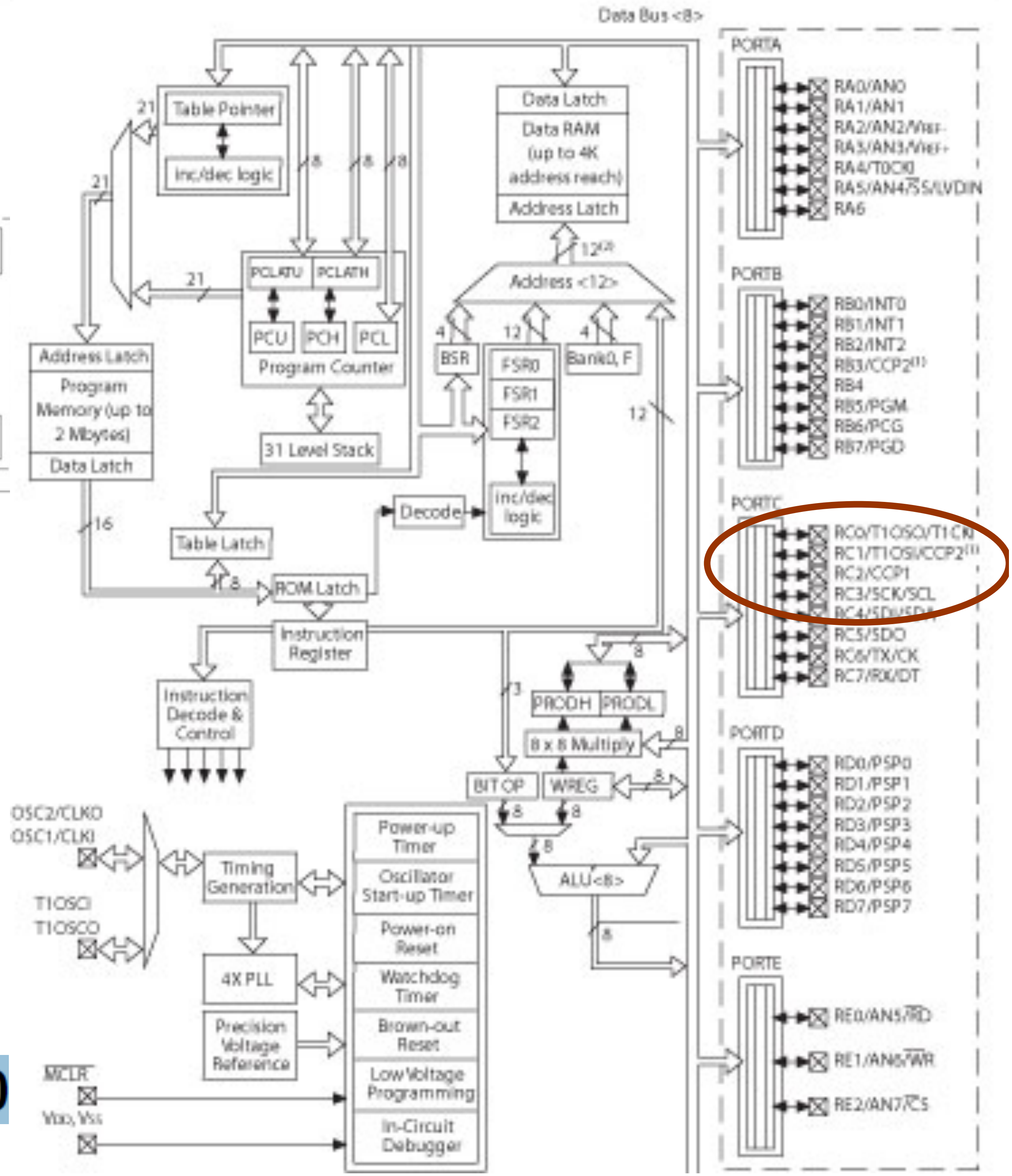
- Time delay
- Pulse wave generation
- Pulse width or frequency measurement
- Timer as an event counter

Capture, Compare, and PWM (CCP) Modules



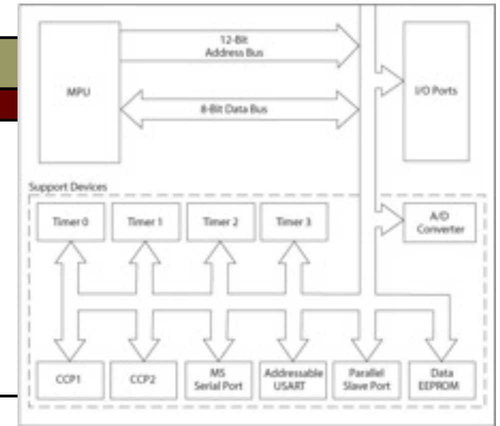
- CCP modules are commonly found in recent microcontrollers
  - 16-bit (or two 8-bit) registers specially designed to perform the following functions in conjunction with timers
    - Capture: The CCP pin can be set as an input to record the arrival time of a pulse.
    - Compare: The CCP pin is set as 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.
      - The count for the period and the duty cycle are loaded into CCP registers.
      - In this mode, the duty cycle of the output pulse can be varied.





Same for:

PIC18F2XK20/4XK20

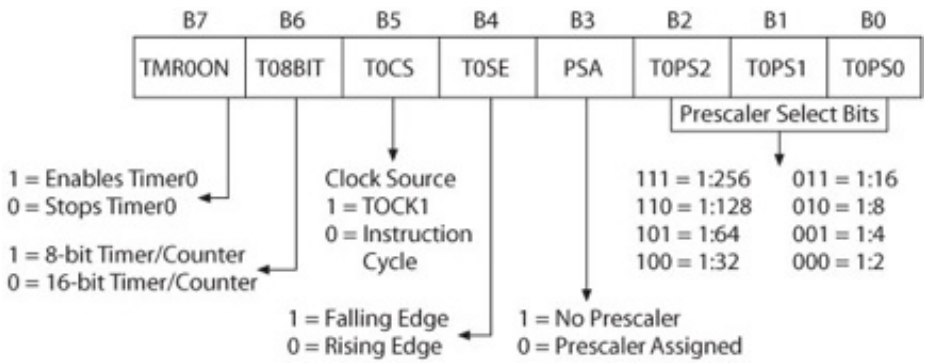


PIC18 Timers

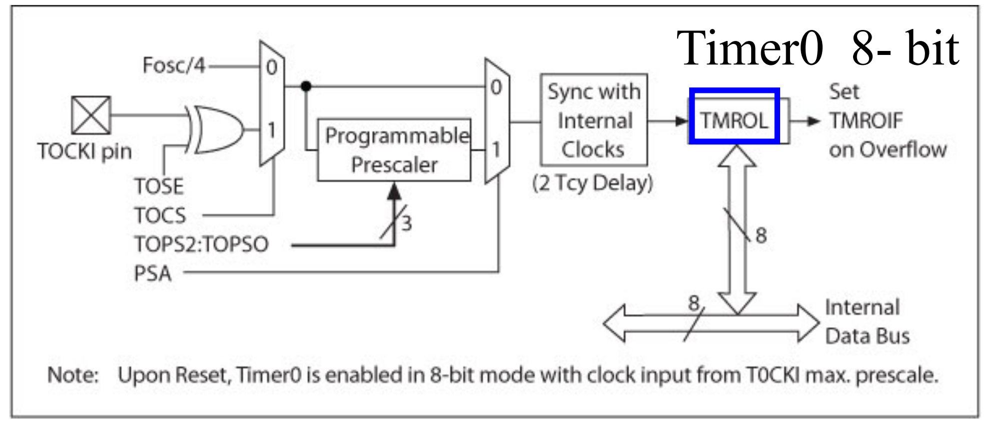
- The PIC18 microcontroller have multiple timers, and all of them are up-counters.
- Timers are divided into two groups: 8-bit and 16-bit
- Labeled as Timer0 to Timer3 or Timer4 (if available)
  - Timer0 can be set up as an 8-bit or 16-bit timer.
  - Timer1 and Timer3 are 16-bit timers.
  - Timer2 and Timer4 (if available) are 8-bit timers.
- Each timer associated with its Special Function Register (SFR): T0CON-T3CON or T4CON

Timer0

Note: TMR Flags are set when the counter reg. has reached it max.

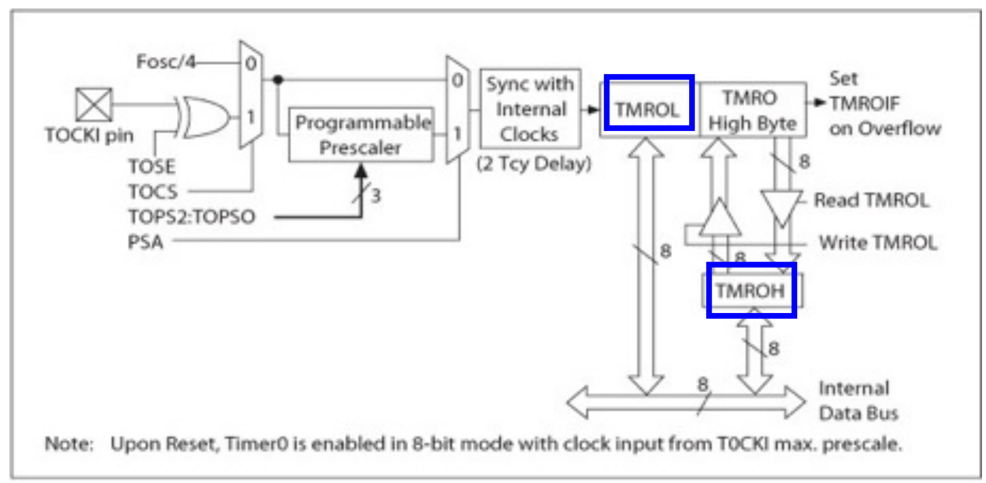


Timer0 Control Register (T0CON)



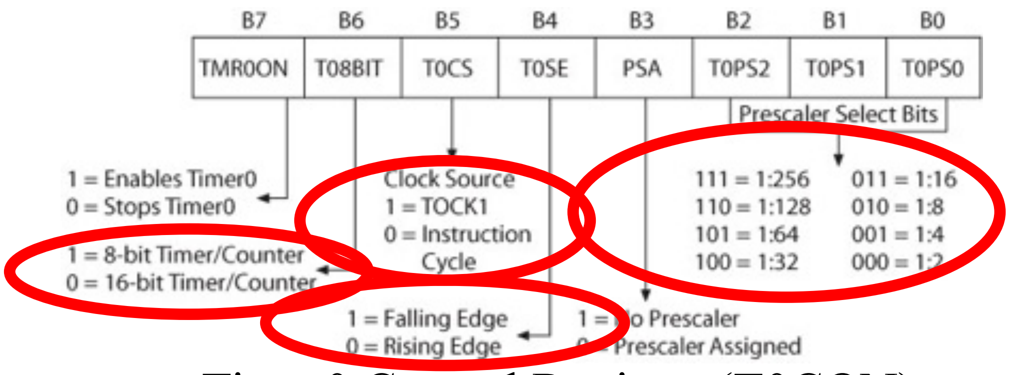
1. Can be set up as an 8-bit or 16-bit timer
2. Has eight options of pre-scale values **(Divides)**
3. Can run on internal clock source (instruction cycle) or external clock connected to pin RA4/T0CK1
4. Generates an interrupt or sets a flag when it overflows from FFH to 00 in the 8-bit mode and from FFFFH to 0000 in the 16-bit mode
5. Can be set up on either rising edge or falling edge when an external clock is used

Timer0 16-bit

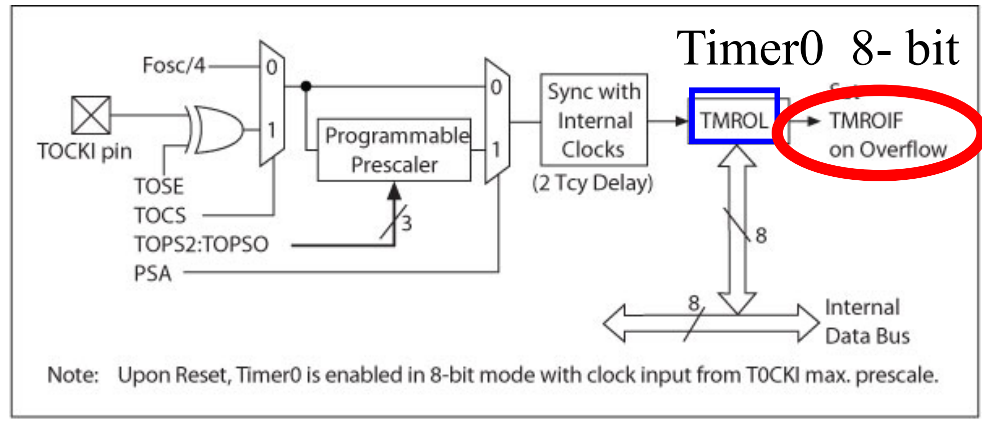


Instruction cycle = 4 clock cycle

Timer0

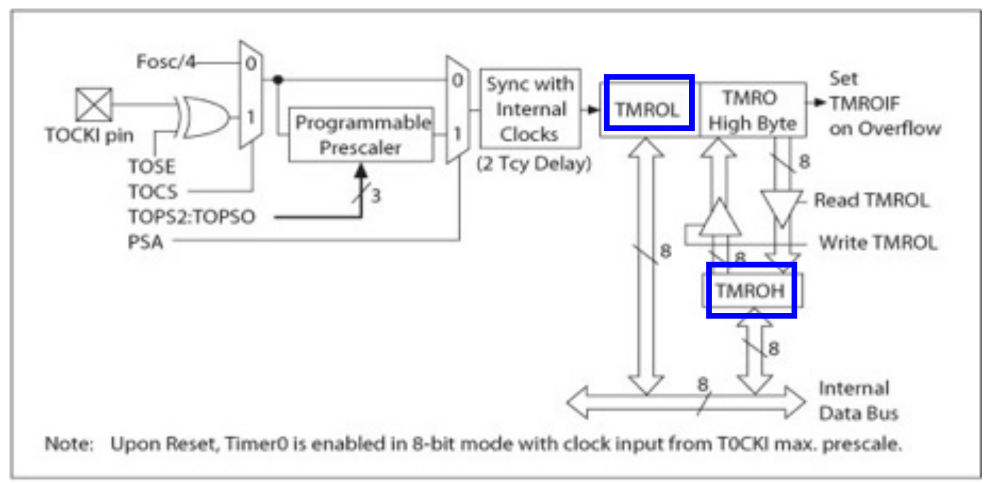


Timer0 Control Register (T0CON)



1. Can be set up as an 8-bit or 16-bit timer
2. Has eight options of pre-scale values **(Divides)**
3. Can run on internal clock source (instruction cycle) or external clock connected to pin RA4/T0CK1
4. Generates an interrupt or sets a flag when it overflows from FFH to 00 in the 8-bit mode and from FFFFH to 0000 in the 16-bit mode
5. Can be set up on either rising edge or falling edge when an external clock is used

Timer0 16-bit

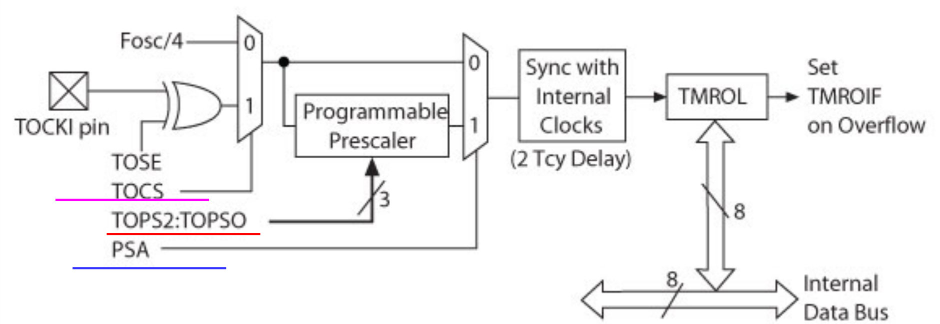


Instruction cycle = 4 clock cycle

TIMER0 Registers

REGISTERS ASSOCIATED WITH TIMER0

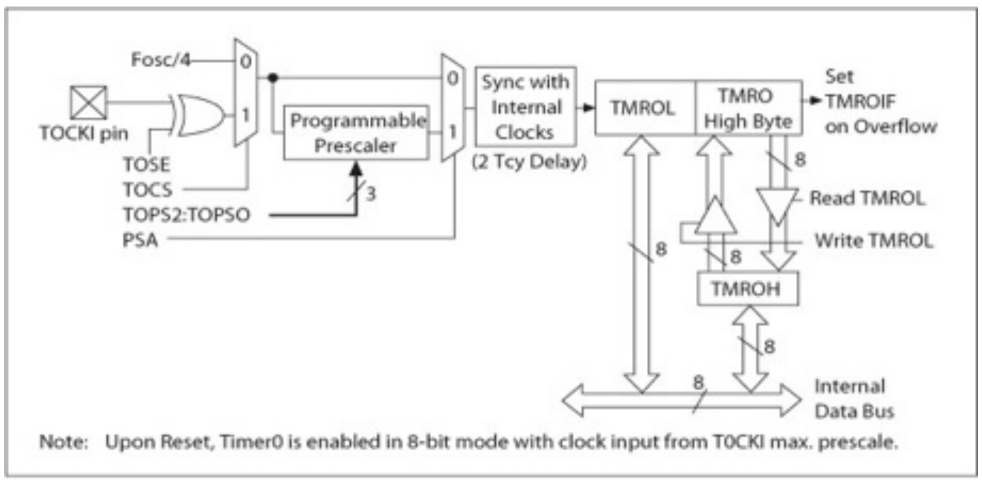
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TMR0L	Timer0 Register, Low Byte							
TMR0H	Timer0 Register, High Byte							
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
TRISA	RA7(1)	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0



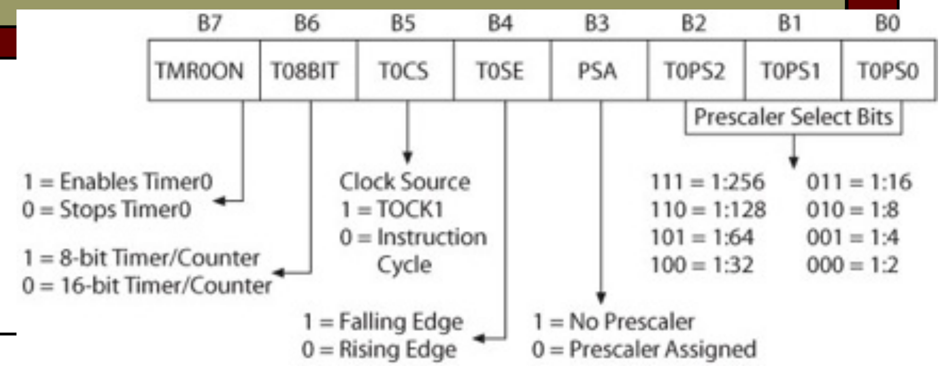
Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

Timer0

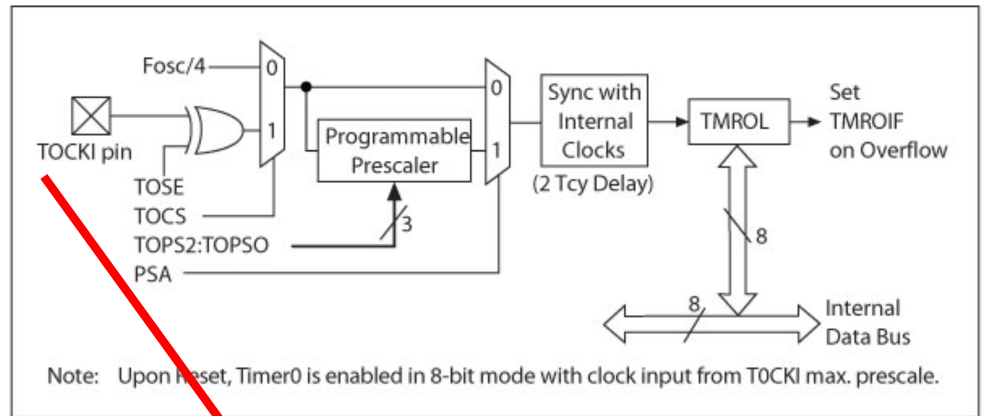
- TMROH buffer between internal data bus and TMR0 high byte
  - Read from the TMR0L register, the upper half of Timer0 is latched into the TMR0H register
  - Ensures that the PIC18 always reads a 16-bit value that its upper byte and lower byte belong to the same time (since only read 8-bits at a time)

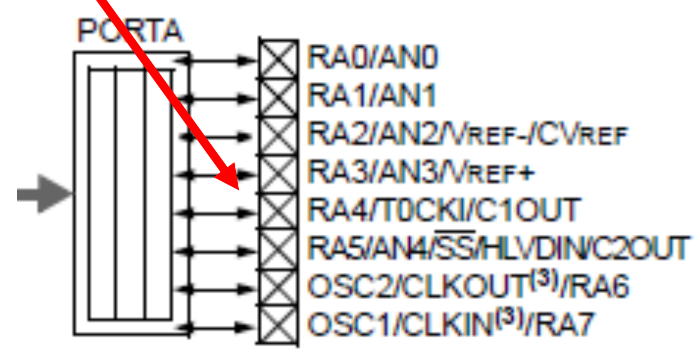


Timer0 Control Register (1 of 2)



- Timer0 as timer
  - Bit5 must be cleared to use the internal clock.
  - At each instruction cycle (four clock cycles), the timer register is incremented.
- Timer0 as a counter
  - Bit5 must be set 1 to use an external clock.
  - In this mode, input signal at PORTA-pin RA4/T0CK used as a clock.
  - When Bit4 = 1, register is incremented on the falling edge, and when Bit4 = 0, the register is incremented on the rising edge.
- Prescaler
  - Divides clock frequency by a specified ratio.
  - To use prescaler, Bit3 = 0, and three bits Bit2-Bit0 specify scaler ratio from 1:2 to 1:256





Timer0 Control Register (2 of 2)

- Interrupt
  - When Timer0 overflows from FFH to 00 in the 8-bit mode and from FFFFH to 0000 in the16-bit mode, it sets TMR0IF (Timer0 Interrupt Flag) –Bit2 in the INTCON register.
    - Flag can be used two ways: 1) a software loop can be set up to monitor the flag, or 2) an interrupt can be generated.
    - Flag must be cleared to start the timer again.
- 16-bit mode
  - When Timer0 is set in the 16-bit mode, it uses two 8-bit registers TMR0L and TMR0H.

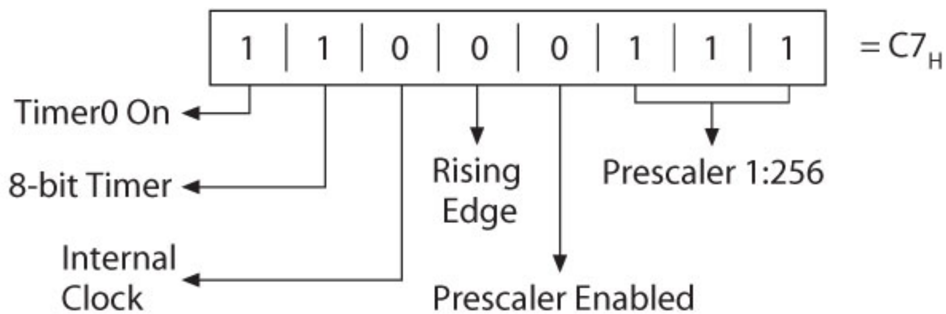
TABLE 12-1: REGISTERS ASSOCIATED WITH TIMER0

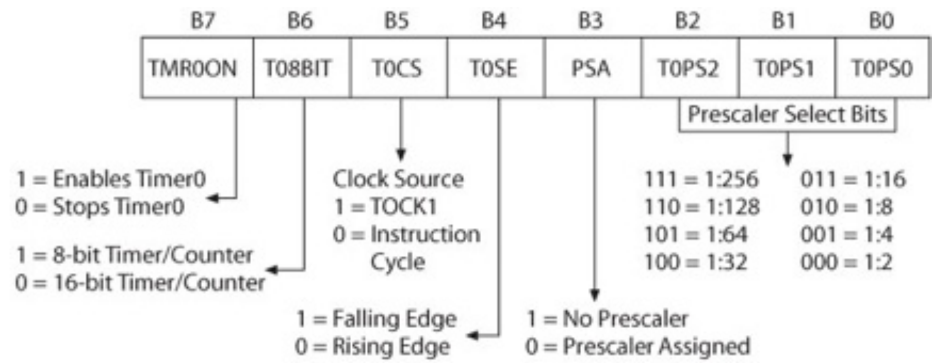
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TMR0L	Timer0 Register, Low Byte							
TMR0H	Timer0 Register, High Byte							
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
TRISA	RA7(1)	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0

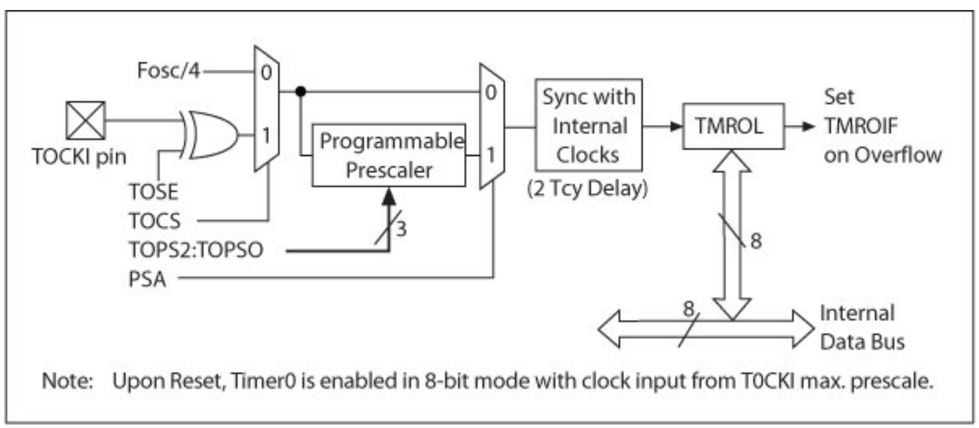
Example: Explain the setting

What are the setting if
TIMER0 Register is set to C7?

Control Word to Initialize Timer0



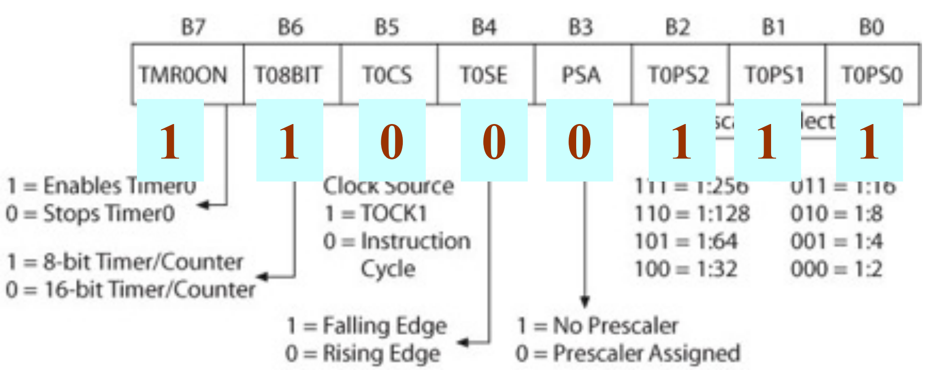




Example - Set TMR0 as an 8-bit timer

- Every instruction cycle the register is updated → 4x(Clock_Period)
- With a pre-scale=1:256 (divide the clock by 256)→ pre_scalex4x(Clock_Period)
- 8-bit register allows counting 256 values →
  - **(2^n)x pre_scalex4x(Clock_Period)**
- Assuming using a 10MHz internal clock, rising edge clock, how often the flag is set if timer 0 is set as 8-bit counter? What should TMR0 (T0CON) setup be?

256x256x4x0.1E-6=Every 26.21 msec



Example For TMR0 (1)

- Using a 16-bit TMR0 generate a high priority interrupt every 1 sec. Assume rising edge, 1:128 pre-scale, and a 10MHz crystal oscillator (internal clock).

Example For TMR0 (2)

We can actually design a real-time clock with this!

- Using a 16-bit TMR0 generate a high priority interrupt every 1 sec. Assume rising edge, 1:128 pre-scale, and a 10MHz crystal oscillator (internal clock).
- 1sec/0.4usec=2,500,000← number of counts that must be generated
  - 16 bit → Assume pre-scale 1:128
  - 2,500,000/128=19531.25 (up counter)← number of counts
  - 2^16-1=65535; (65535)-19531=46,004 → B3B4 →load B3B4 into TMR0L/H and count up to FFFF → then a flag is set!
- Code:
  - High priority →
  - RCON →
  - INTCON→
  - INTCON2→
  - INTCON3→
  - PIR1→
  - TCON→

High priority → ORG 0x08
RCON → IPEN = 1
INTCON→ Set GIEH/L ; PEIE ; TMR0IE ; clear FLAG
INTCON2→ set TMR0IP (priority)
INTCON3→ All zero
PIR1→ clear all flags
TCON→ TMR0ON=1 ; T0PS=110
Load B3B4 into TMR0L/H and count up to FFFF → generate interrupt

Note: TMR Flags are set when the counter reg. has reached it max.

Example For TMR0 (3)

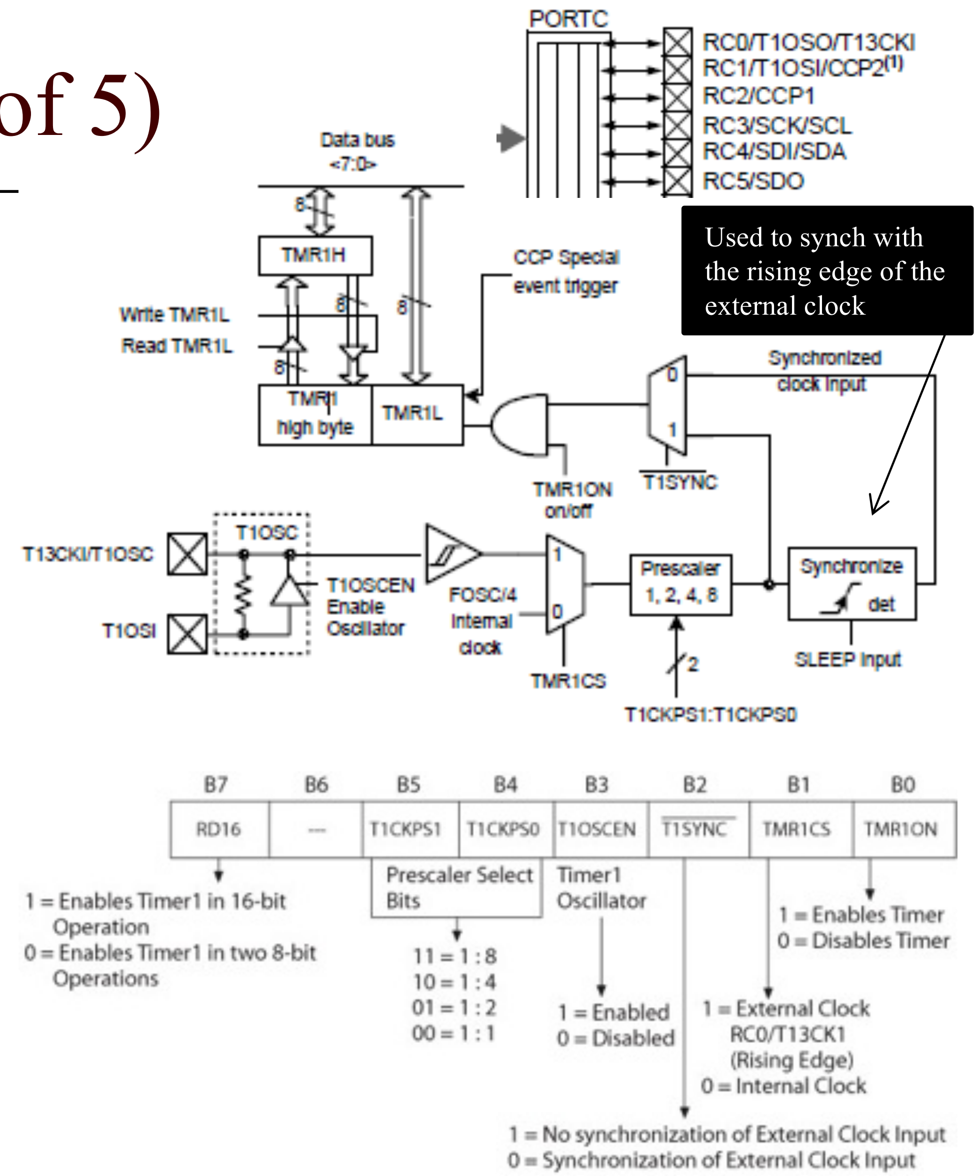
```
        Title "PIC18F452 EX11-2 One Second Delay With Interrupt"
        List p=18F452, f =inhx32
        #include <p18F452.inc>                  ;This is a header file

                ORG     00
                GOTO    MAIN

                ORG     0x08
                GOTO    TMR0_ISR

MAIN:           CLRF    INTCON3                 ;Disable all INT flags
                CLRF    PIR1                    ;Clear all internal peripheral flags
                BSF     RCON, IPEN              ;Enable priority - RCON <7>
                BSF     INTCON2,TMR0IP          ;Set Timer0 as high-priority
                MOVLW   B'11100000'             ;Set Timer0:global interrupt, high
                IORWF   INTCON,1                ;piority, overflow, interrupt flag
                MOVLW   B'10000110'             ;Enable Timer0: 16-bit, internal clock,
                MOVWF   T0CON                   ; prescaler- 1:128

DELAY_1s:
                MOVLW   0xB3                    ;High count of B3B4H
                MOVWF   TMR0H                   ;Load high count in Timer0
                MOVLW   0xB4                    ;Low count of B3B4H
                MOVWF   TMR0L                   ;Load low count in Timer0
                BCF     INTCON, TMR0IF          ;Clear TIMR0 overflow flag ñ Start counter
HERE:           GOTO    HERE                    ;Wait here for an interrupt

                ORG     0x100
TMR0_ISR:
                MOVLW   0xB3                    ;High count of B3B4H
                MOVWF   TMR0H                   ;Load high count in Timer0
                MOVLW   0xB4                    ;Low count of B3B4H
                MOVWF   TMR0L                   ;Load low count in Timer0
                BCF     INTCON, TMR0IF          ;Clear TIMR0 overflow flag ñ Star
                RETFIE  FAST                    ;Return
                END
```

When flag is set the MPU transfer the program to high priority interrupt vector location 0x08

When the Interrupt service routine is executed, the TMR0 is reloaded, interrupts are cleared, back to MAIN

Timer1 – 16-bit (1 of 5)

- A 16-bit counter/timer with two 8-bit registers (TMR1H and TMR1L); both registers are readable and writable
- Four options of prescale value (Bit5-Bit4)
- Clock source (Bit1) can be internal (instruction cycle) or external (pin RC0/T13CK1) on rising edge
- Sets flag or generates an interrupt when it overflows from FFFFH to 0000

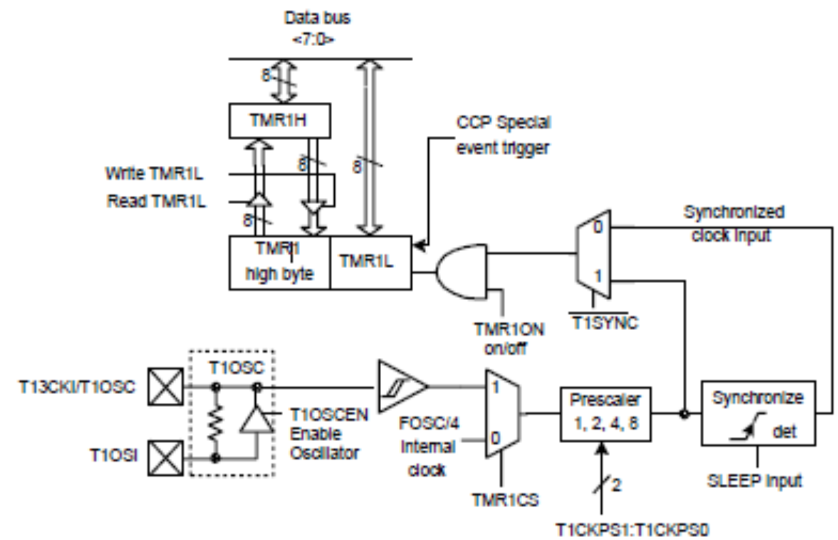


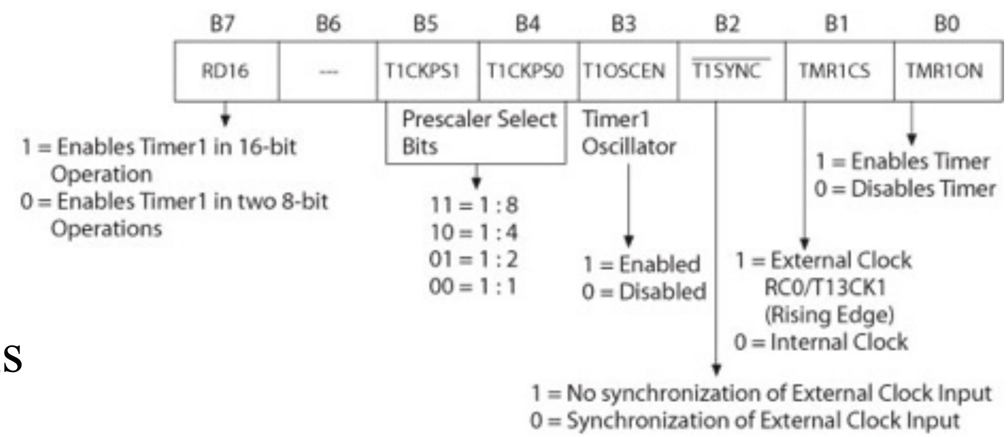
REGISTERS ASSOCIATED WITH CAPTURE, COMPARE, TIMER1 AND TIMER3

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	59
RCON	IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	58
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	62
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	62
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	62
PIR2	OSCFIF	C1IF	C2IF	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF	62
PIE2	OSCFIE	C1IE	C2IE	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE	62
IPR2	OSCFIP	C1IP	C2IP	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP	62
TRISB	PORTB Data Direction Control Register								62
TRISC	PORTC Data Direction Control Register								62
TMR1L	Timer1 Register, Low Byte								60
TMR1H	Timer1 Register, High Byte								60
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	60
TMR3H	Timer3 Register, High Byte								61
TMR3L	Timer3 Register, Low Byte								61
T3CON	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{T3SYNC}$	TMR3CS	TMR3ON	61
CCPR1L	Capture/Compare/PWM Register 1, Low Byte								61
CCPR1H	Capture/Compare/PWM Register 1, High Byte								61
CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	61
CCPR2L	Capture/Compare/PWM Register 2, Low Byte								61
CCPR2H	Capture/Compare/PWM Register 2, High Byte								61
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	61

Timer1 (3 of 5)

- Timer1 Operation
  - Can operate in three modes:
    - timer,
    - synchronous counter,
    - asynchronous counter
  - Bit0 enables or disables the timer
  - When Bit1 = 0, it operates as a timer and increments count at every instruction cycle.
    - When Bit1 = 1, it operates as a counter and increments count at every rising edge of the external clock.
  - When Bit3 = 1, Timer1 oscillator is enabled which is used for low frequency operations.



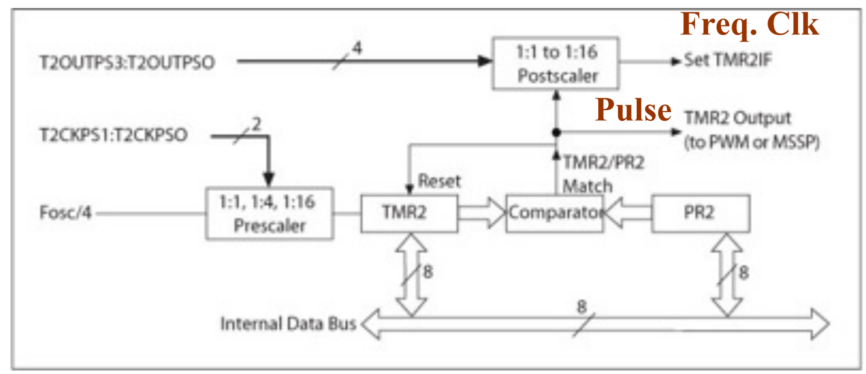


TMR1 Example

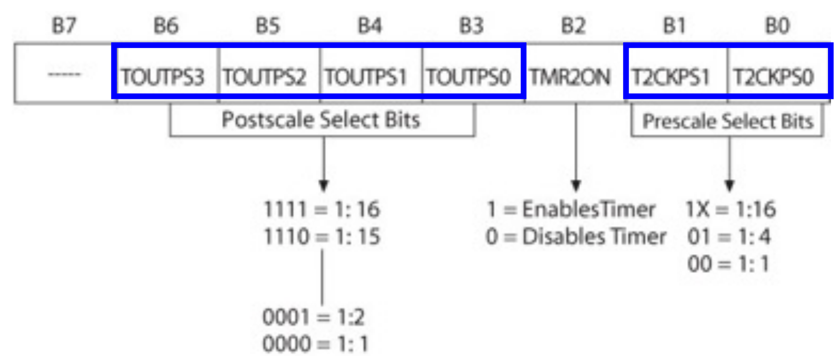
- Generate 100 usec clock; assuming internal clock is 10MHz (See the handout).

Timer2

- Two 8-bit registers (TMR2 and PR2)
- An 8-bit number is loaded in PR2 and the timer is turned on, which is incremented every instruction cycle.
- When the count in the timer register and the PR register match, an output pulse is generated and the timer register is set to zero.
- The output pulse goes through a postscaler that divides the frequency by the scale factor and sets the flag TMR2IF-
  - Bit1 in the Peripheral Interrupt Register1 (PIR1) that can be used to generate an interrupt.



Master Synchronous Serial Port (*MSSP*)



TMR2

REGISTERS ASSOCIATED WITH PWM AND TIMER2

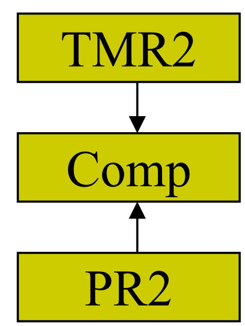
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
RCON	IPEN	SBOREN	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
TRISB	PORTB Data Direction Control Register							
TRISC	PORTC Data Direction Control Register							
TMR2	Timer2 Register							
PR2	Timer2 Period Register							
T2CON	—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
CCPR1L	Capture/Compare/PWM Register 1, Low Byte							
CCPR1H	Capture/Compare/PWM Register 1, High Byte							
CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
CCPR2L	Capture/Compare/PWM Register 2, Low Byte							
CCPR2H	Capture/Compare/PWM Register 2, High Byte							
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
ECCP1AS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1	PSSBD0
PWM1CON	PRSEN	PDC6	PDC5	PDC4	PDC3	PDC2	PDC	PDC0

Example for Timer2

- Generate a periodic high-priority interrupt every 8-msec using Timer2. Assume a 32-MHz crystal oscillator.
  - Assume post/pre scaled values are 16
  - Loaded value in PR2 will be
    - PR2 = [Td / [Inst. Clock Cycle(4) x Prescaler x PostScaler x clock period)]] - 1
    - PR2 = [8msec/[4x16x16x(1/32MHZ)]] -1 =249

Remember, we start with 0 count → -1 is needed

PR2=249
RCON: IPEN=1
IPR1: TMR21P=1; TMR2IF=CLR
INTCON=C0; GLOBAL INT
T2CON=7E; TMR2 ENABLE AND SCALING SETUP
PIE1: TMR2IE=SET



Example for Timer2 - continue

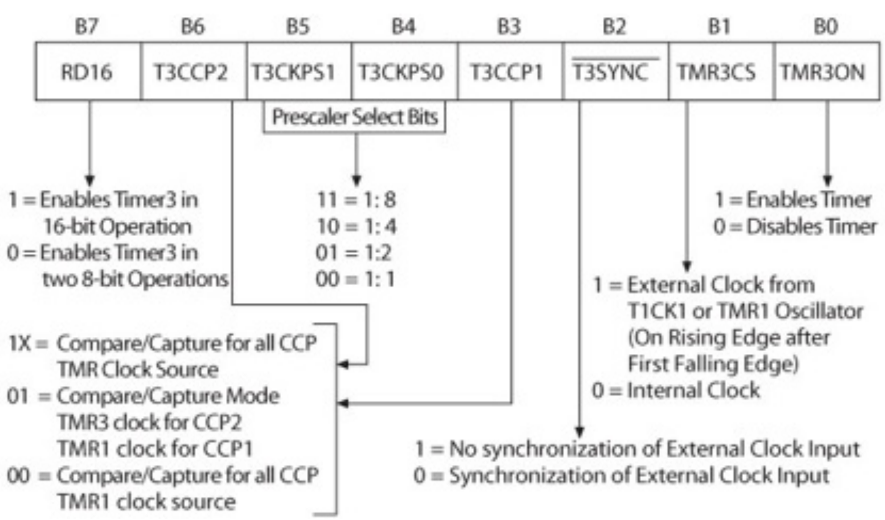
- Actual code:

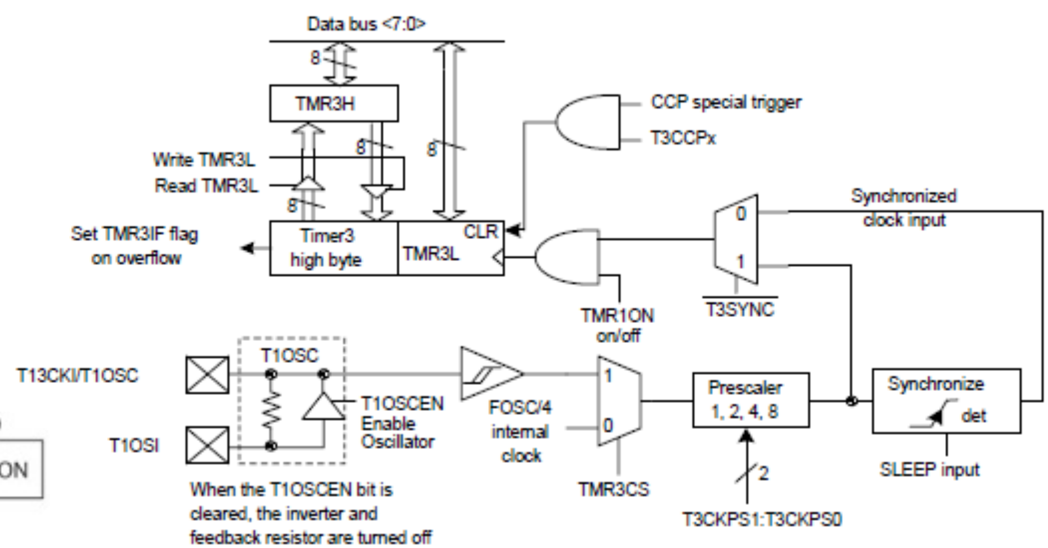
```
movlw    D'249'          ; load 249 into PR2 so that TMR2 counts up
movwf    PR2,A           ; to 249 and reset
bsf      RCON,IPEN,A     ; enable priority interrupt
bsf      IPR1,TMR2IP,A   ; place TMR2 interrupt at high priority
bcf      PIR1,TMR2IF,A   ;
movlw    0xC0
movwf    INTCON,A        ; enable global interrupt
movlw    0x7E            ; enable TMR2, set prescaler to 16, set
movwf    T2CON,A         ; postscaler to 16
bsf      PIE1,TMR2IE,A   ; enable TMR2 overflow interrupt
```

PR2=249
RCON: IPEN=1
IPR1: TMR21P=1; TMR2IF=CLR
INTCON=C0; GLOBAL INT
T2CON=7E; TMR2 ENABLE AND SCALING SETUP
PIE1: TMR2IE=SET

Timer3

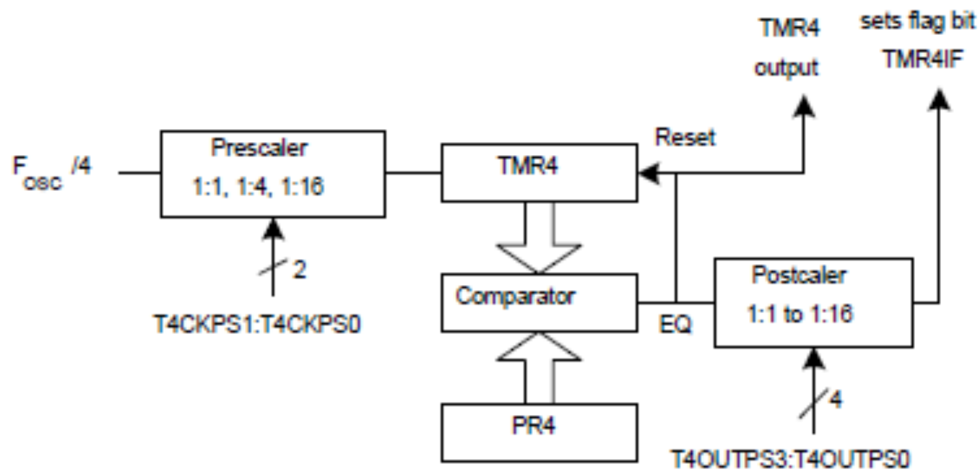
- Similar to Timer1





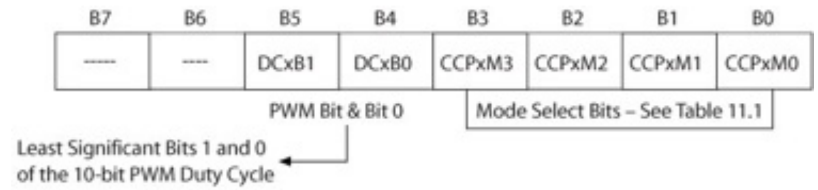
Timer4

- Only available to the PIC18F8X2X and PIC6X2X devices
- The value of TMR4 is compared to PR4 in each clock cycle
- When the value of TMR4 equals that of PR4, TMR4 is reset to 0
- The contents of T4CON are identical to those of T2CON
- ….similar to Timer2 (Two 8-bit registers)

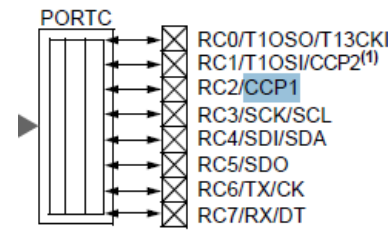


CCP & ECCP

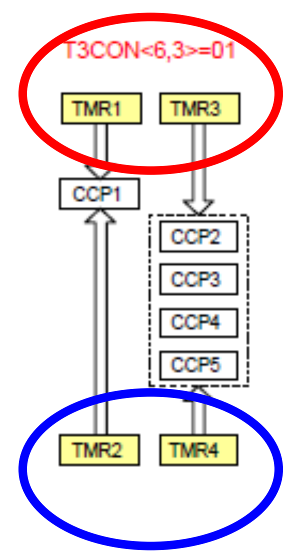
CCP (Capture, Compare, and PWM) Modules

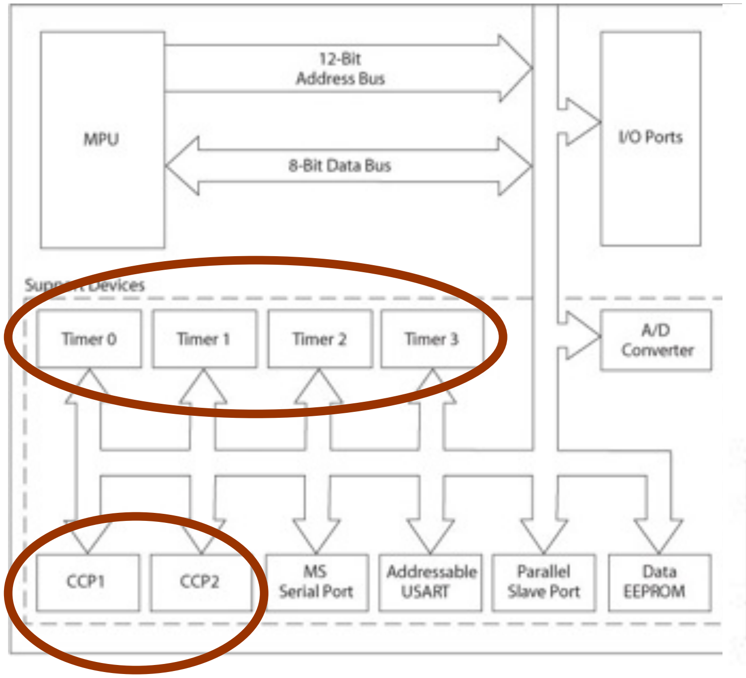


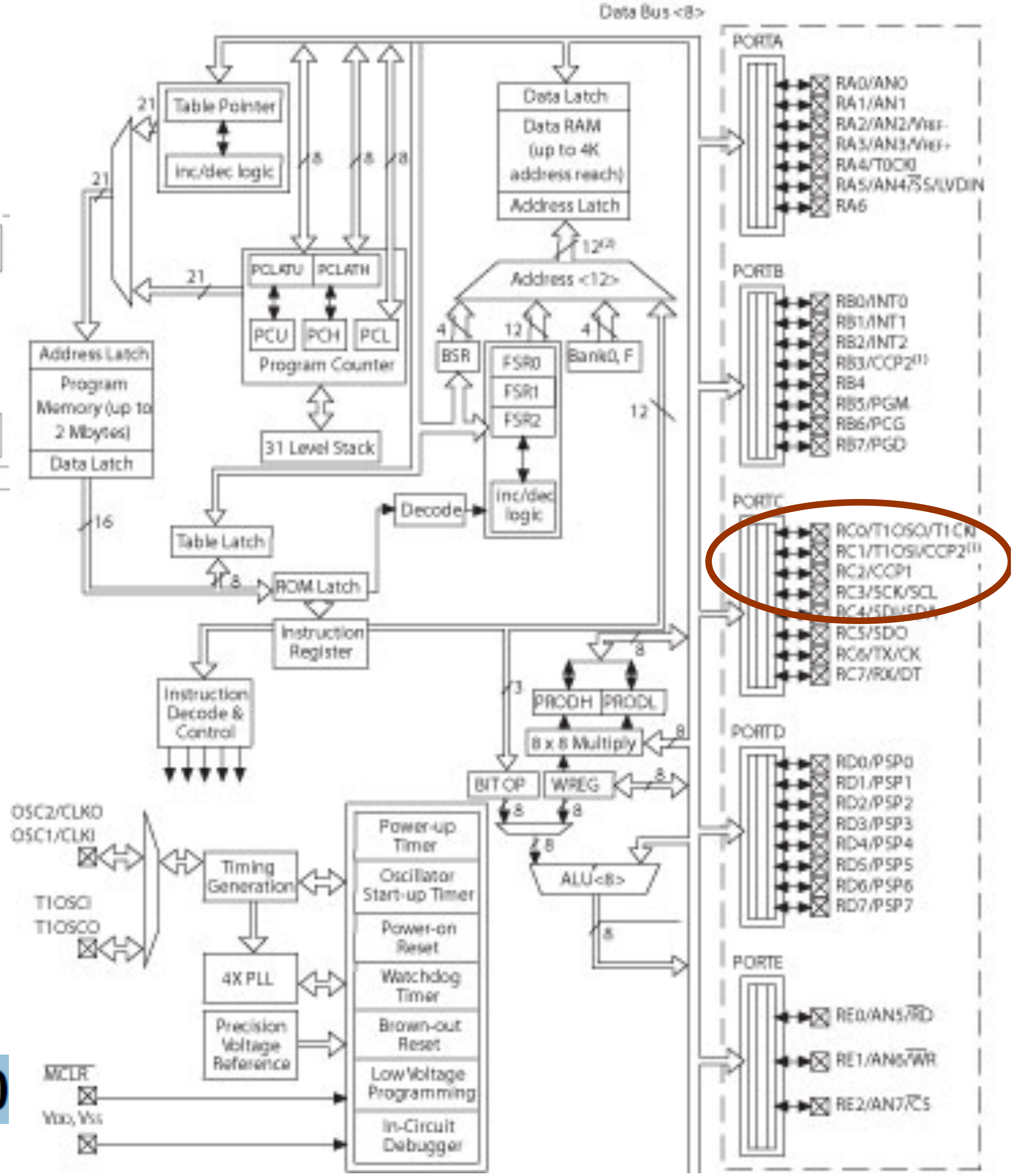
- PIC18 Device may have 1, 2, or 5 CCP modules
  - Each CCP module requires the use of a timer resource
  - Capture or compare mode, the CCP module may use either Timer1 or Timer3 to operate.
  - PWM mode, either Timer2 or Timer4 may be used



- The operations of all CCP modules are identical, with the exception of the special event trigger mode present on CCP1 and CCP2
- Each module is associated with
  - A control register (CCPxCON)
  - A data register (CCPRx) which consists of two 8-bit register: CCPRxL and CCPRxH
- The assignment of a particular timer to a module is determined by the bit 6 and bit 3 of the T3CON







Same for:

PIC18F2XK20/4XK20

Control Register (CCP1CON)

REGISTER 16-1: CCP1CON: ENHANCED CAPTURE/COMPARE/PWM CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
bit 7							bit 0

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

bit 7-6 **P1M<1:0>:** Enhanced PWM Output Configuration bits

<u>If CCP1M<3:2> = 00, 01, 10:</u>

xx = P1A assigned as Capture/Compare input/output; P1B, P1C, P1D assigned as port pins

<u>If CCP1M<3:2> = 11:</u>

00 = Single output: P1A, P1B, P1C and P1D controlled by steering (See **Section 16.4.7 "Pulse Steering Mode"**).

01 = Full-bridge output forward: P1D modulated; P1A active; P1B, P1C inactive

10 = Half-bridge output: P1A, P1B modulated with dead-band control; P1C, P1D assigned as port pins

11 = Full-bridge output reverse: P1B modulated; P1C active; P1A, P1D inactive

bit 5-4 **DC1B<1:0>**: PWM Duty Cycle bit 1 and bit 0

<u>Capture mode:</u>

Unused.

<u>Compare mode:</u>

Unused.

<u>PWM mode:</u>

These bits are the two LSbs of the 10-bit PWM duty cycle. The eight MSbs of the duty cycle are found in CCPR1L.

bit 3-0 **CCP1M<3:0>**: Enhanced CCP Mode Select bits

0000 = Capture/Compare/PWM off (resets ECCP module)

0001 = Reserved

0010 = Compare mode, toggle output on match

0011 = Reserved

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 CCP1 pin low, set output on compare match (set CCP1IF)

1001 = Compare mode, initialize CCP1 pin high, clear output on compare match (set CCP1IF)

1010 = Compare mode, generate software interrupt only, CCP1 pin reverts to I/O state

1011 = Compare mode, trigger special event (ECCP resets TMR1 or TMR3, sets CC1IF bit)

1100 = PWM mode; P1A, P1C active-high; P1B, P1D active-high

1101 = PWM mode; P1A, P1C active-high; P1B, P1D active-low

1110 = PWM mode; P1A, P1C active-low; P1B, P1D active-high

1111 = PWM mode; P1A, P1C active-low; P1B, P1D active-low

REGISTER 15-1: T3CON: TIMER3 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

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

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

bit 6,3 **T3CCP<2:1>:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the capture/compare clock source for CCP1 and CP2
01 = Timer3 is the capture/compare clock source for CCP2 and
Timer1 is the capture/compare clock source for CCP1
00 = Timer1 is the capture/compare clock source for CCP1 and CP2

bit 5-4 **T3CKPS<1:0>:** Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

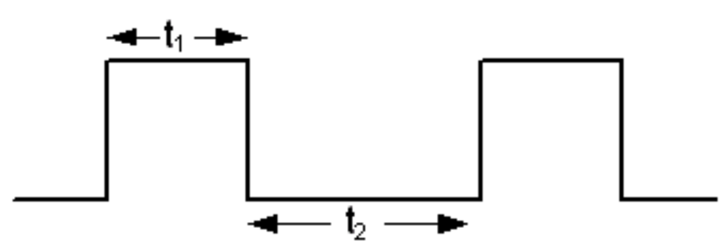
bit 2 **$\overline{\text{T3SYNC}}$:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the device clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses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

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T13CKI (on the rising edge after the first falling edge)
0 = Internal clock (Fosc/4)

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

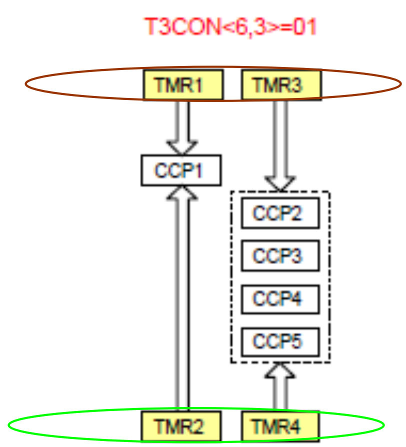
Applications of CCP

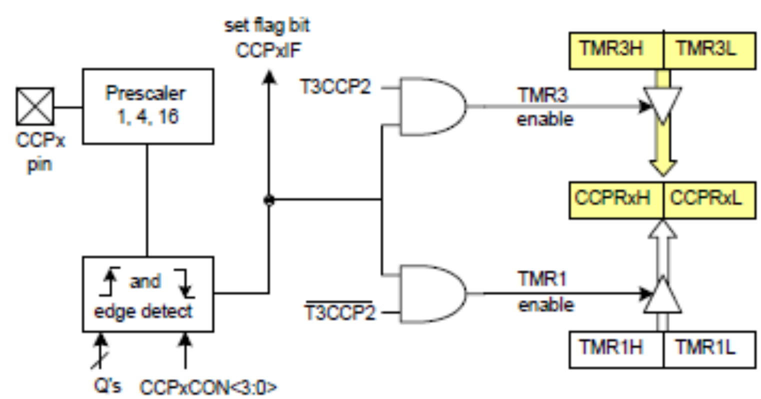
- Event arrival time recording
  - Swimming competition, need to compare different swimmer times
- Period measurement
  - Capture function configured to capture the timer values corresponding to two consecutive rising or falling edges
- Pulse width measurement
  - Capture function configured to capture two adjacent rising and falling edges
- Interrupt generation
  - All capture inputs can serve as edge-sensitive interrupt sources
- Event counting
  - Event represented by signal edge
  - CCP channel used in conjunction with a timer or another CCP channel to counter number of events that occur during a timer interval
- Time reference
  - CCP capture module used with another CCP channel in compare mode
  - Detect event, add desired response time, compare mode determine when to activate response
- Duty cycle measurement
  - Percentage of time signal is high within a period



Basic operation

- Each CCP module is comprised of two 8-bit registers: CCPR1H (high) and CCPR1L (low) → Total of 16-bits
  - Called capture and compare register
- Can operate as 16-bit Capture register, 16-bit Compare register, or duty-cycle PWM register
- Timer1 and Timer3 are used as clock resources for Capture and Compare registers
- Timer2 and Timer4 (if available) are used as clock sources as PWM modules

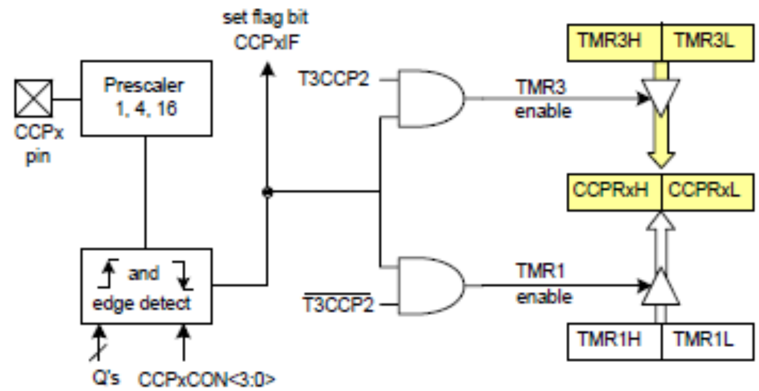




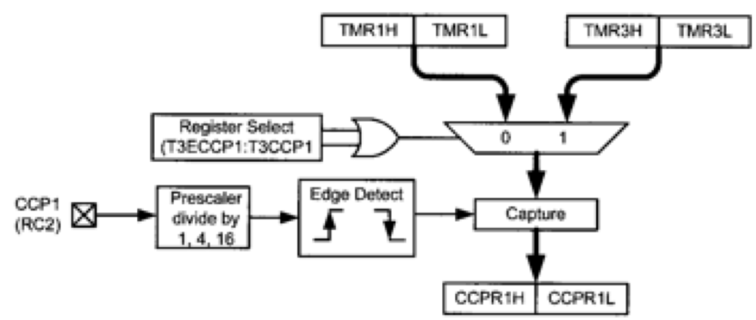
Capture Mode

CCP in the Capture Mode (1 of 2)

- When do events arrive?
  - Physical time represented by the count value in a counter
  - An event is represented by a signal edge
  - Main use of CCP is to capture **event** arrival time by latching in the count value when the signal arrives
- The PIC18 event can be one of the following
  - Every falling edge
  - Every rising edge
  - Every 4th rising edge
  - Every 16th rising edge
- CCPR1 register captures the 16-bit value of Timer1 (or Timer3) when an event occurs on pin RC2/CCP1.
- When a capture occurs, the interrupt request flag bit CCP1IF (Bit2 in PIR1) is set and must be cleared for the next operation.



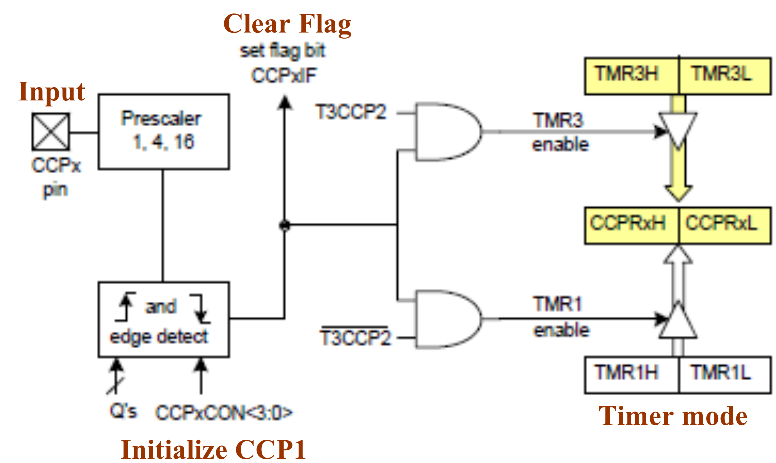
Different View….



Using TMR1 or TMR3

CCP in the Capture Mode (2 of 2)

- To capture an event:
  - Set up pin RC2/CCP1 of PORTC as the input.
  - Initialize Timer1 in the timer mode or synchronized counter mode by writing to T1CON/ T3CON register.
    - Asynch mode does not work
  - Initialize CCP1 by writing to the CCP1CON register.
  - Clear the CCP1IF flag to continue the next operation when a capture occurs.
    - Clear CCP1IE and CCP1IF to avoid a false interrupt when capture mode is changed.

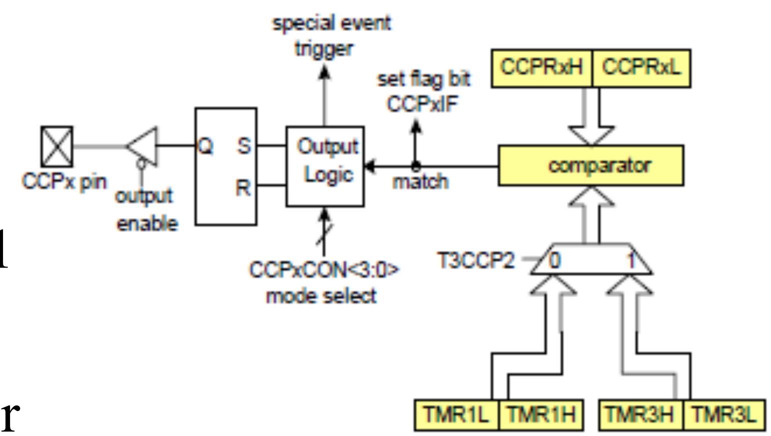


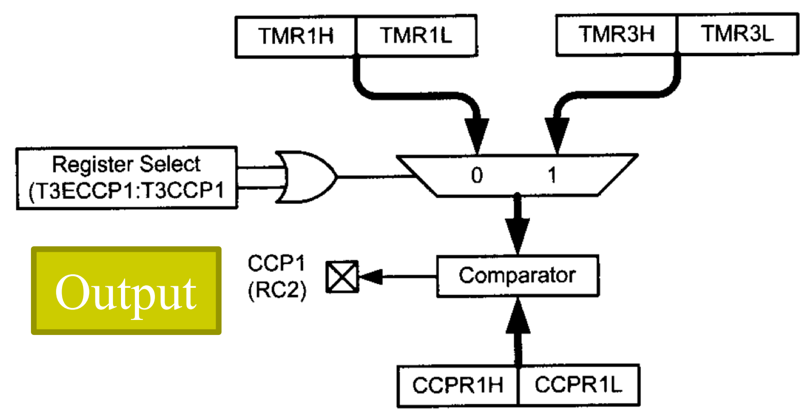
Key Registers to Set:
CCPxCON
TMR3 or TMR1
CCPx is Input
CCPRxL/H

Compare Mode

CCP in the Compare Mode (1 of 2)

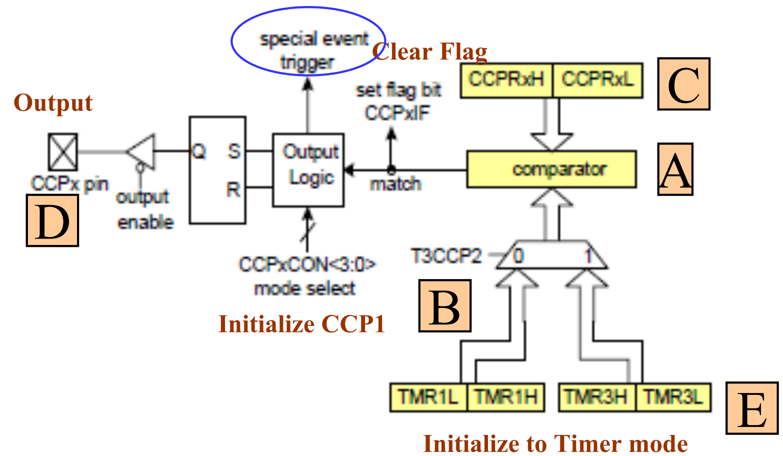
- CCP compare applications
  - Generation of a single pulse, a train of pulses, periodic waveform with certain duty cycle, specified time delay
- 16-bit value loaded by the user in CCPR1 (or CCPRx) is constantly compared with the TMR1 (or TMR3) register when the timers are running in either timer mode or synchronized counter mode.
- When a **match** occurs, the pin RC2/CCP1 on PORTC is driven high, low, or toggled based on mode select bits in the CCP1CO (Bit3-Bit0 in CCP1 control register), and the interrupt flag bit CCP1IF is set.





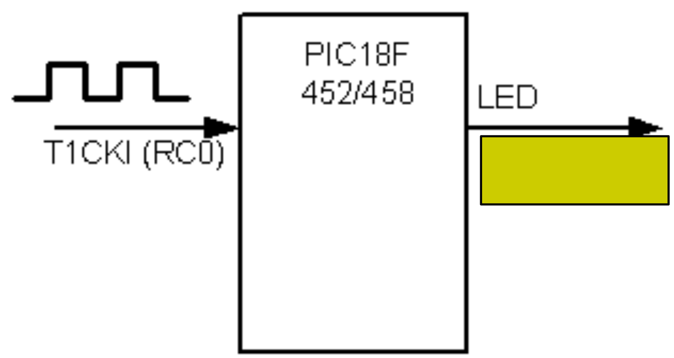
CCP in the Compare Mode (2 of 2)

- To set up CCP1 in the Compare mode:
  - Set up pin RC2/CCP1 of PORTC as **output**. D
  - Initialize Timer1 in the timer mode or the synchronized counter mode by writing to the **T1CON/ T3CON** register. E
  - Initialize CCP1 by writing to the **CCP1CON** register. B
  - Clear the flag CCP1IF, which is set when a compare occurs, and must be cleared to continue to the next operation.
  - For a special event trigger, an internal hardware trigger is generated that can be used to initiate an action.
  - The special event trigger output resets Timer1.



Order of Setup:

A---E



Example

Measure the period of the input clock

1- Assume the clock is coming from RC2 → CCP1
3- Use TMR1

```c
// main program
void main (void)
{

  //** Clock Selection ****
     //OSCCON = 0x40;           // IRCFx = 100 // 2 MHz clock --> 2usec
     //OSCTUNEbits.PLLEN = 0;  // x4 PLL disabled

  OSCCON = 0x70;              // IRCFx = 111 (8 MHz) or --> 0.125 usec
  OSCTUNEbits.PLLEN = 1;      // x4 PLL enabled = 32MHz

  // *** Initializing the CCP1 (ECCP)
  CCP1CON = 0x05;
  T3CON = 0x00;
  T1CON = 0x0;
  TRISD = 0x0;
  TRISCbits.TRISC2 = 1; // Set RC2 // Make sure the input
              // is a square signal with no offset.
  CCPR1L = 0;
  CCPR1H = 0;
  while(1)
  {
    TMR1H = 0;
    TMR1L = 0;
    PIR1bits.CCP1IF=0;
    PORTDbits.RD0=~PORTDbits.RD0;

    while(PIR1bits.CCP1IF==0); // Wait for the first rising edge
    T1CONbits.TMR1ON=1;
    PIR1bits.CCP1IF=0;

    while(PIR1bits.CCP1IF==0); // wait for the second rising edge
    T1CONbits.TMR1ON=0;
    PulsePeriod[0]=CCPR1L; // the number of counts are here!
    PulsePeriod[1]=CCPR1H;

  }
```

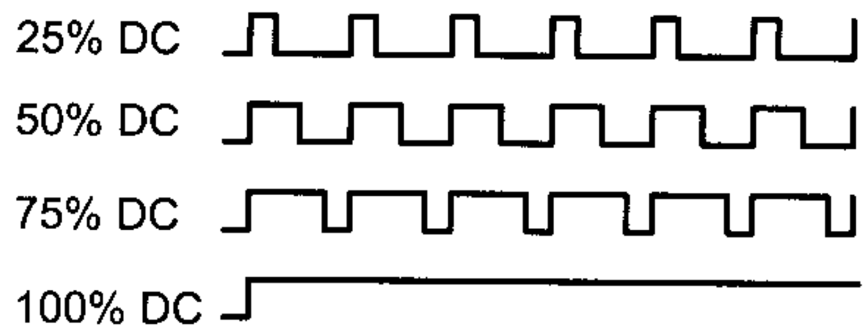
Pulse Width Modulation

Basic Idea

Reg: CCP1CON

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			

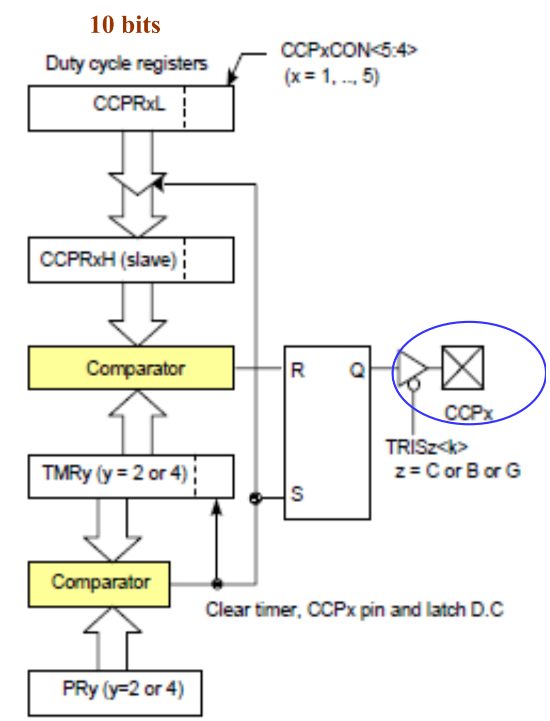
DC1B2	DC1B1	Decimal points
0	0	0
0	1	0.25
1	0	0.5
1	1	0.75



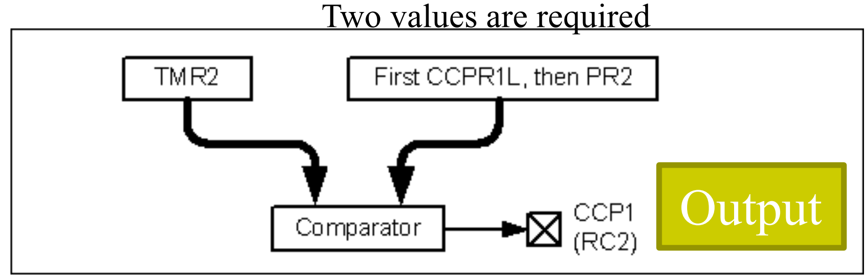
For example 75% of COUNT (e.g. Pry=249) = 186.75
→ 0.75 is equivalent to **DC1B1 & DC1B0 = 11**

CCP in the Pulse Width Modulation (PWM) Mode (0 of 3)

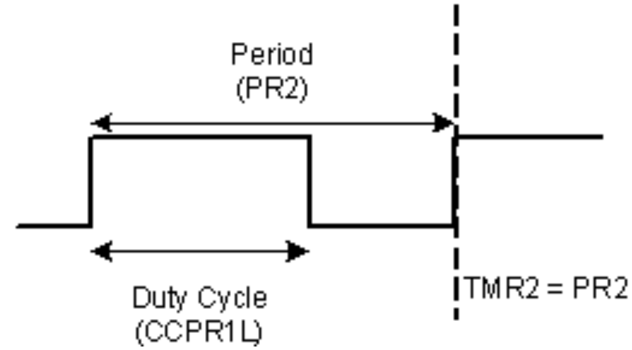
- CCPx pin can output a 10-bit resolution periodic digital waveform with programmable duty cycle
- Duty cycle to be generated is a 10-bit value
  - Upper 8-bits stored in CCPRxH register
  - Lower 2-bits stored in bit 5 and bit 4 of
- CCPxCON register Duty cycle value compared with TMRy cascaded with 2-bit clock in every instruction cycle
  - When values are equal, CCPx pin pulled low
- TMRy register compared to PRy register in every clock cycle, when equal following events occur on next increment cycle
  - CCPx pin pulled high
  - TMRy register cleared
  - PWM duty cycle is latched from CCPRxl into CCPRxH



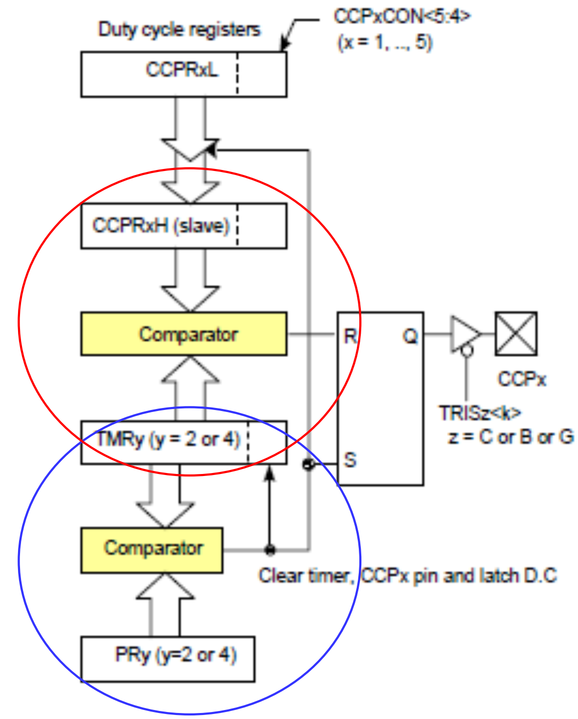
Two values are required



CCP in the Pulse Width Modulation (PWM) Mode (1 of 3)



- A CCP module in conjunction with Timer2 can be set up to output a pulse wave form for a given frequency and a duty cycle.
- The CCP module uses a 10-bit number to specify the duty cycle.
- The 8-bit number loaded into the PR2 register specify the PWM period.
- PWM period and duty cycle can be calculated using the following



PWM period = $[(PRy) + 1] \times 4 \times T_{OSC} \times$ (TMRy prescale factor)

Time unit = instruction cycle

PWM duty cycle = $(CCPRxL{:}CCPxCON\langle 5{:}4\rangle) \times T_{OSC} \times$ (TMRy prescale factor)

or **CCPR1L = [PR2+1]*DutyCycle** *Time unit = crystal oscillator cycle*

(in sec)

CCP in the PWM Mode (2 of 3)

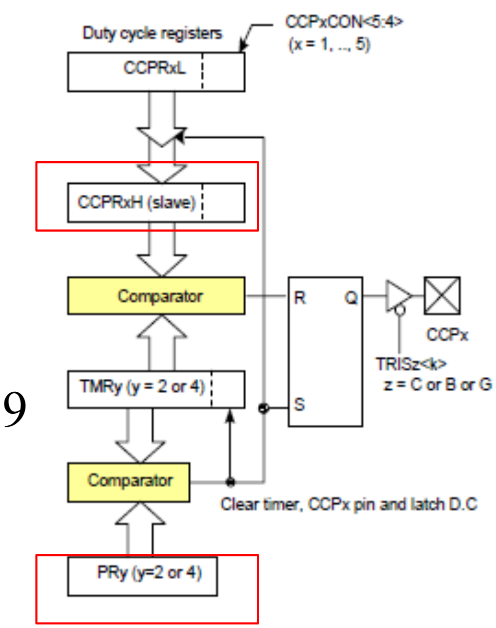
- When TMR2 is equal to PR2, the following three events occur in the next increment cycle:
  - TMR2 is cleared.
  - Pin RC2/CCP1 of PORTC is set high.
  - The PWM duty-cycle byte is latched from CCPR1L into CCPR1H.
    - When CCPR1H and TMR2 match again for the specified duty cycle, the CCP1 pin is cleared.

CCP in the PWM Mode (3 of 3)

- To Initialize CCP1 in the PWM mode:
  - Set up pin RC2/CCP1 of PORTC as output.
  - Set up PWM period by writing to the PR2 register.
  - Set up PWM duty cycle by writing to CCPR1L register and Bit5-Bit4 of CCP1CON register.
  - Set up TMR2 prescale value and Timer2 in timer mode by writing to T2CON register.
  - Enable CCP1 module in the PWM mode.
  - Set up CCP1 by writing to the CCP1CON register.

Example of Register Setting for PWM

- Configure CCP1 in PWM mode to generate a digital waveform with 40% duty cycle and 10 KHz frequency assuming that the PIC18 MCU is running with a 32 MHz crystal oscillator. Assuming prescale=4 for timer 2.



- Timer setting
  - Use Timer2 as the base timer of CCP1 for PWM mode
  - Set Prescaler to Timer2 to 1:4
  - Period register value: PR2 = 32MHz /[4x4x10KHz]-1 = 199
    - **PR2 = Fosc /[4xNxFdesired]-1 ; N is the prescaler value**
  - Duty Cycle Value: CCPR1L = [PR2+1]*DutyCycle

= 200x40% = 80.00

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			

DCxB0 & B1 = 00

CCPRL Register= 80d

PR2 Register = 199d

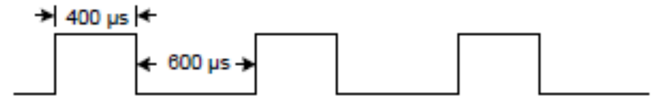
B7	B6	B5	B4	B3	B2	B1	B0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	T3SYNC	TMR3CS	TMR3ON
		Prescaler Select Bits					

T3CON

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			

CCP1CON

PWM Example 3

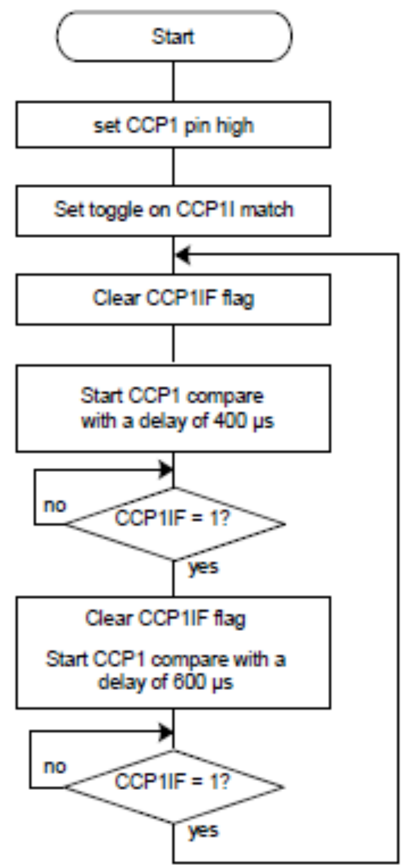


1KHz 40% duty cycle waveform

- Use CCP1 to generate a periodic waveform with 40% duty cycle and 1 KHz frequency assuming that the instruction cycle clock - use timer3 as the base timer, set prescale = 1, assume high priority. Assume 4MHz crystal oscillator.

Setup:

High priority int. vector setup
CCP1 is set to output
T3CON = C9; turn on TMR3 in 16-bit mode, TMR3 as base timer for all CCP modules
CCP1CON=09 ; configure CCP1 pin set high initially and pull low on match



Load TMR3H/L = 640 and CCPR1H/L = 640

Remember:

1msec x 4MHz = 4000 counts
40% → (PRy=4000); 4001*0.40 → 1600 = 640h
60% → 2400 = 960h

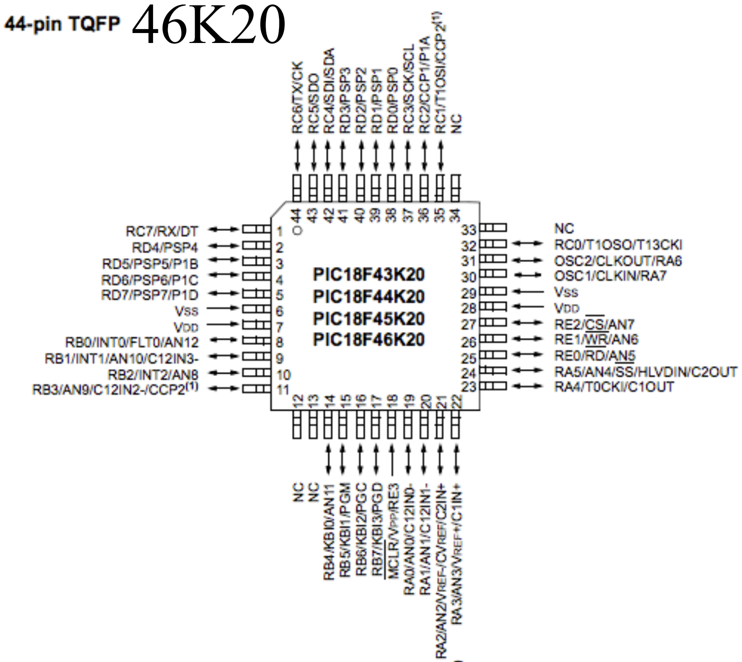
Load TMR3H/L = 960 and CCPR1H/L = 960

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).

Programing ECCP (CCP1) In PIC18F46K20 - 1

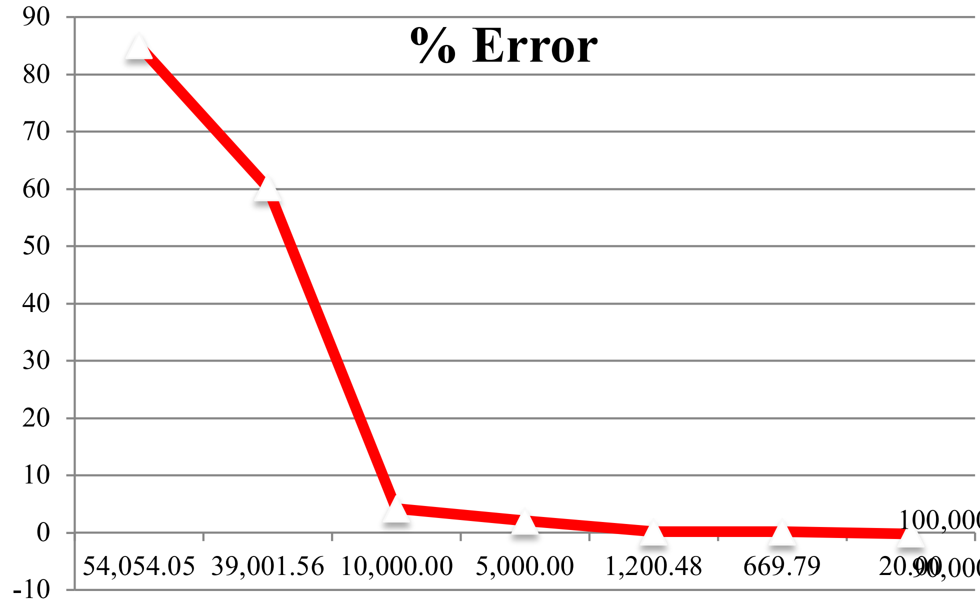
- Write a program that measures the period of the incoming signal; at RC2 (CCP1) – This is the ECCP in 46K20

44-pin TQFP

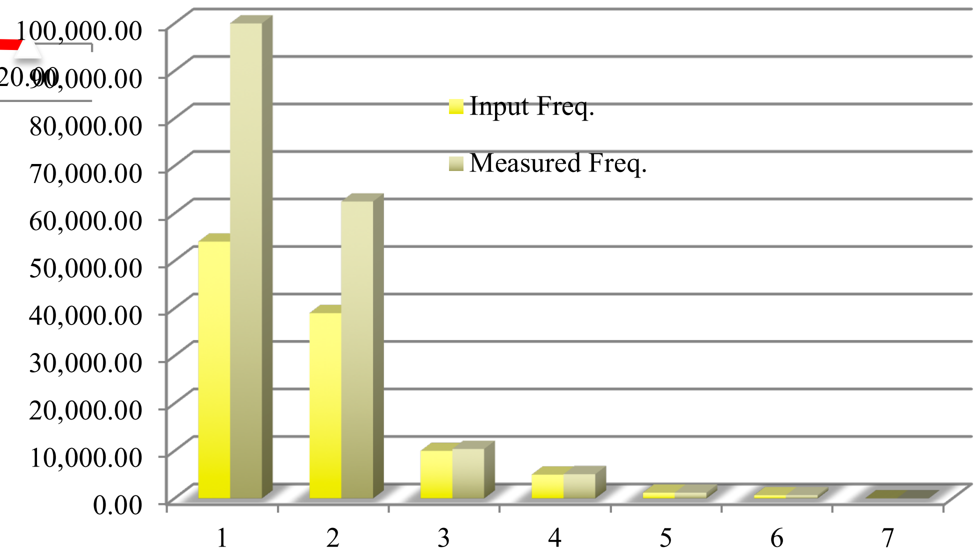



```c
void main (void)
{

 //** Clock Selection ****
//OSCCON = 0x40;          // IRCFx = 100 // 2 MHz clock --> 2usec
//OSCTUNEbits.PLLEN = 0;  // x4 PLL disabled

OSCCON = 0x70;            // IRCFx = 111 (8 MHz) or --> 0.125 usec
OSCTUNEbits.PLLEN = 1;    // x4 PLL enabled = 32MHz

 // *** Initializing the CCP1 (ECCP)
 CCP1CON = 0x05;s
 T3CON = 0x00;
 T1CON = 0x0;
 TRISD = 0x0;
 TRISCbits.TRISC2 = 1; // Set RC2 // Make sure the input
                       // is a square signal with no offset.
 CCPR1L = 0;
 CCPR1H = 0;
 while(1)
 {
     TMR1H = 0;
     TMR1L = 0;
     PIR1bits.CCP1IF=0;
     PORTDbits.RD0=~PORTDbits.RD0;

     while(PIR1bits.CCP1IF==0); // Wait for the first rising edge
     T1CONbits.TMR1ON=1;
     PIR1bits.CCP1IF=0;

     while(PIR1bits.CCP1IF==0); // wait for the second rising edge
     T1CONbits.TMR1ON=0;
     PulsePeriod[0]=CCPR1L; // the number of counts are here!
     PulsePeriod[1]=CCPR1H;

 }
```

Programing ECCP (CCP1) In PIC18F46K20 - 2



Performance of the
Period Measurements
With 2MHz clock



It performs well for frequencies less than 5KHz when the clock is 4 usec.

Example of PWM using CCP1(RC2)

Note that we use OSCTUNE to adjust The frequency

CCPR1L sets the value of the duty cycle

```
// main program
void main (void)
{
  OSCCON = 0x40;          // IRCFx = 100 // 2 MHz clock --> 2usec
  OSCTUNEbits.PLLEN = 0;  // x4 PLL disabled

  TRISC = 0xFB;
  TRISD = 0x00;
  CCP1CON = 0x3C;
  PR2=100;  // Note: refer to section 11.4.1 or datasheet.
  T2CON=0x01;
  OSCTUNE = 0b00010011;  // this is to adjust the period of the pulses
  while(1)
    {
      // For CCPR1L=25; the period is 8822 usec; DC= 223 usec
      CCPR1L = 50;    //Can be 25 or 50% duty cycle
      TMR2=0x0;
      PIR1bits.TMR2IF=0;
      T2CONbits.TMR2ON=1;
      //PORTDbits.RD0 = ~PORTDbits.RD0;
      while(PIR1bits.TMR2IF==0);
      PORTDbits.RD0 = ~PORTDbits.RD0;
    }
}
```

101*4*4*0.5usec = 808 usec = Period

CCPR1L . <DC1B2:DC1B1>=PR2*DC%

50.00 = 100 * 0.5 → for 50% Duty Cycle

Controlling a DC Motor Using PWM

This input can change the speed or used for ON/OFF