Chapter 11

Clocks, Watchdog Timer / Timers
Read Sections 12-16 of

Data Sheet for PIC18F46K20
Updated: 4/19/12
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.

<table>
<thead>
<tr>
<th>C statement</th>
<th>Assembly Language</th>
<th>Scaling factor</th>
<th>Time to Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma config WDTPS = 1</td>
<td>_WDTPS_1_2H</td>
<td>1:1</td>
<td>4 ms</td>
</tr>
<tr>
<td>#pragma config WDTPS = 32768</td>
<td>_WDTPS_32768_2H</td>
<td>1:32768</td>
<td>131.072 sec</td>
</tr>
</tbody>
</table>
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:

```c
/** DECLARATIONS ***********************************************************/
#define PBO PORTBbits.RBO

void main (void)
{
  TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)
  INTCON2bits.RBPU = 0; // enable PORTB internal pullups
  WPUBits.WPUBO = 1; // enable pull up on RB0
  ANSELH = 0x00; // AN8-12 are digital inputs (AN12 on R
  TRISBbits.TRISBO = 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.RDO = ~PORTDbits.RDO;
    Delay1KTCYx(500);
    while (PBO == 0)
      PORTDbits.RDO = PBO;
  }
}
```
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.

```c
/* Declarations */
#define DDR0 PORTBbits.RB0
#pragma config WDTPS = 2048  // about 10 sec.

unsigned char count = 0;

void main(void)
{
    TRISB = 0b00000000;    // PORTD bits 7:0 are all outputs [0]
    INTCON2bits.RB0 = 0;   // enable PORTE internal pullup
    UPEbits.WPU0 = 1;      // enable pull up on RD0
    ABSEL = 0x00;          // AB0-12 are digital inputs (AB12 on RD0)
    TRISBbits.TRISB0 = 1;  // PORTE bit C [connected to switch] is input [1]

    // Setting the WD registers
    RCON = 0x00010000;
    UDTCON = 1;

    PORTBbits.RD1 = 1;     // This indicates that program just reset
    Delay1KTCYc(100);

    while (1)
    {
        ClrVds();
        count = count + 1;
        PORTDbits.DD1 = 0;    // Clear DD1
        PORTBbits.RD0 = ~PORTDbits.RD0;
        Delay1KTCYc(20);     // Note that if delay must be within 5 sec WD time
        while (FB0 == 0)     // PORTB bit 0 is DD0
            PORTBbits.RD0 = DD0;
        if (count == 20)
            { count = 0;       // upon each reset the count is reset
                Sleep();
            }
    }
}
```
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.

<table>
<thead>
<tr>
<th>C language</th>
<th>Assembly Language</th>
<th>Brownout Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma config BORV = 45</td>
<td>_BORV_45_2L</td>
<td>4.5 V</td>
</tr>
<tr>
<td>#pragma config BORV = 42</td>
<td>_BORV_42_2L</td>
<td>4.2 V</td>
</tr>
<tr>
<td>#pragma config BORV = 27</td>
<td>_BORV_27_2L</td>
<td>2.7 V</td>
</tr>
<tr>
<td>#pragma config BORV = 20</td>
<td>_BORV_20_2L</td>
<td>2.0 V</td>
</tr>
</tbody>
</table>
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 resistor/capacitor with Fosc/4 output on OSC2 (RC)
6. External resistor/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)

*some versions do not have an internal oscillator and
some versions may have additional modes
MCU Clock Source Diagram

Note 1: Operates only when HFINTOSC is the primary oscillator.
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 resistor/capacitor with Fosc/4 output on OSC2 (RC)
- 2 MHz operation is attained with $R = 3.9K$ and $C = 30 \ \text{pF}$, Fosc/4 is 500 KHz with these values
- Frequency = $1/[RC(4.2)]$; can vary slightly

External clock source
Connected to OSC1
Clock Examples

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

### OSCCON Register

<table>
<thead>
<tr>
<th>RW-3</th>
<th>R/W-0</th>
<th>R/W-1</th>
<th>RW-1</th>
<th>R-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLEN</td>
<td>IRCF2</td>
<td>IRCF1</td>
<td>IRCF0</td>
<td>OSTS(1)</td>
<td>IOFS</td>
<td>SC51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 7</td>
<td>bit 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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
### Programming Example

```c
#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
    }
}  // 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:

**PIC18F2XXK20/4XXK20**
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

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 FFFFFFFH 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

Instruction cycle = 4 clock cycle
Timer0

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

Instruction cycle = 4 clock cycle
# TIMER0 Registers

## REGISTERS ASSOCIATED WITH TIMER0

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMR0L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>TMR0H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>INTCON</td>
<td>GIE/GIEH</td>
<td>PEIE/GIEL</td>
<td>TMR0IE</td>
<td>INT0IE</td>
<td>RBIE</td>
<td>TMR0IF</td>
<td>INT0IF</td>
<td>RBIF</td>
</tr>
<tr>
<td>T0CON</td>
<td>TMR0ON</td>
<td>T08BIT</td>
<td>T0CS</td>
<td>T0SE</td>
<td>PSA</td>
<td>T0PS2</td>
<td>T0PS1</td>
<td>T0PS0</td>
</tr>
<tr>
<td>TRISA</td>
<td>RA7(1)</td>
<td>RA6(1)</td>
<td>RA5</td>
<td>RA4</td>
<td>RA3</td>
<td>RA2</td>
<td>RA1</td>
<td>RA0</td>
</tr>
</tbody>
</table>

Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from TOCKI 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
Interrupt
- When Timer0 overflows from FFH to 00 in the 8-bit mode and from FFFFH to 0000 in the 16-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>
<thead>
<tr>
<th>TABLE 12-1: REGISTERS ASSOCIATED WITH TIMER0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>TMR0L</td>
</tr>
<tr>
<td>TMR0H</td>
</tr>
<tr>
<td>INTCON</td>
</tr>
<tr>
<td>TOCON</td>
</tr>
<tr>
<td>TRISA</td>
</tr>
</tbody>
</table>
Example: Explain the setting

What are the setting if
TIMER0 Register is set to C7?
Control Word to Initialize Timer0

1 1 0 0 0 1 1 1 = C7H

- Timer0 On
- 8-bit Timer
- Internal Clock
- Rising Edge
- Prescaler Enabled
- Prescaler 1:256

Clock Source
1 = TOCK1
0 = Instruction Cycle

Prescaler Select Bits

1 = Enables Timer0
0 = Stops Timer0

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

1 = Falling Edge
0 = Rising Edge

1 = No Prescaler
0 = Prescaler Assigned

Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from TOCK1 max. prescale.
Example - Set TMR0 as an 8-bit timer

- Every instruction cycle the register is updated $\rightarrow 4 \times \text{(Clock\_Period)}$
- With a pre-scale=1:256 (divide the clock by 256) $\rightarrow \text{pre\_scale} \times 4 \times \text{(Clock\_Period)}$
- 8-bit register allows counting 256 values $\rightarrow (2^n) \times \text{pre\_scale} \times 4 \times \text{(Clock\_Period)}$
- 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?

\[ 256 \times 256 \times 4 \times 0.1 \times 10^{-6} = \text{Every 26.21 msec} \]
Example –

- 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 –

- 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).

  1 sec/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

We can actually design a real-time clock with this!
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
<table>
<thead>
<tr>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Reset Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTCON</td>
<td>GIE/GIEH</td>
<td>PEIE/GIEL</td>
<td>TMR0IE</td>
<td>INT0IE</td>
<td>RBIE</td>
<td>TMR0IF</td>
<td>INT0IF</td>
<td>RBIF</td>
<td>50</td>
</tr>
<tr>
<td>RCON</td>
<td>IPEN</td>
<td>SBOREN</td>
<td>—</td>
<td>RI</td>
<td>TO</td>
<td>PD</td>
<td>POR</td>
<td>BOR</td>
<td>58</td>
</tr>
<tr>
<td>PIR1</td>
<td>PSPIF(1)</td>
<td>ADIF</td>
<td>RCIF</td>
<td>TXIF</td>
<td>SSPIF</td>
<td>CCP1IF</td>
<td>TMR2IF</td>
<td>TMR1IF</td>
<td>62</td>
</tr>
<tr>
<td>PIE1</td>
<td>PSPIE(1)</td>
<td>ADIE</td>
<td>RCIE</td>
<td>TXIE</td>
<td>SSPIE</td>
<td>CCP1IE</td>
<td>TMR2IE</td>
<td>TMR1IE</td>
<td>62</td>
</tr>
<tr>
<td>IPR1</td>
<td>PSPIF(1)</td>
<td>ADIP</td>
<td>RCIP</td>
<td>TXIP</td>
<td>SSPIP</td>
<td>CCP1IP</td>
<td>TMR2IP</td>
<td>TMR1IP</td>
<td>62</td>
</tr>
<tr>
<td>PIR2</td>
<td>OSCFIF</td>
<td>C1IF</td>
<td>C2IF</td>
<td>EEIF</td>
<td>BCLIF</td>
<td>HLVDIF</td>
<td>TMR3IF</td>
<td>CCP2IF</td>
<td>62</td>
</tr>
<tr>
<td>PIE2</td>
<td>OSCFIE</td>
<td>C1IE</td>
<td>C2IE</td>
<td>EEIE</td>
<td>BCLIE</td>
<td>HLVDIE</td>
<td>TMR3IE</td>
<td>CCP2IE</td>
<td>62</td>
</tr>
<tr>
<td>IPR2</td>
<td>OSCFIF</td>
<td>C1IF</td>
<td>C2IF</td>
<td>EEIF</td>
<td>BCLIF</td>
<td>HLVDIF</td>
<td>TMR3IP</td>
<td>CCP2IP</td>
<td>62</td>
</tr>
<tr>
<td>TRISB</td>
<td>PORTB Data Direction Control Register</td>
<td>62</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRISC</td>
<td>PORTC Data Direction Control Register</td>
<td>62</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMR1L</td>
<td>Timer1 Register, Low Byte</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMR1H</td>
<td>Timer1 Register, High Byte</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1CON</td>
<td>RD16</td>
<td>T1RUN</td>
<td>T1CKPS1</td>
<td>T1CKPS0</td>
<td>T1OSCEN</td>
<td>T1SYNC</td>
<td>TMR1CS</td>
<td>TMR1ON</td>
<td>60</td>
</tr>
<tr>
<td>TMR3H</td>
<td>Timer3 Register, High Byte</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMR3L</td>
<td>Timer3 Register, Low Byte</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3CON</td>
<td>RD16</td>
<td>T3CCP2</td>
<td>T3CKPS1</td>
<td>T3CKPS0</td>
<td>T3CCP1</td>
<td>T3SYNC</td>
<td>TMR3CS</td>
<td>TMR3ON</td>
<td>61</td>
</tr>
<tr>
<td>CCP1RL</td>
<td>Capture/Compare/PWM Register 1, Low Byte</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP1RH</td>
<td>Capture/Compare/PWM Register 1, High Byte</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP1CON</td>
<td>P1M1</td>
<td>P1M0</td>
<td>DC1B1</td>
<td>DC1B0</td>
<td>CCP1M3</td>
<td>CCP1M2</td>
<td>CCP1M1</td>
<td>CCP1M0</td>
<td>61</td>
</tr>
<tr>
<td>CCP2L</td>
<td>Capture/Compare/PWM Register 2, Low Byte</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP2H</td>
<td>Capture/Compare/PWM Register 2, High Byte</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP2CON</td>
<td>—</td>
<td>—</td>
<td>DC2B1</td>
<td>DC2B0</td>
<td>CCP2M3</td>
<td>CCP2M2</td>
<td>CCP2M1</td>
<td>CCP2M0</td>
<td>61</td>
</tr>
</tbody>
</table>
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.
Timer1 (4 of 5)

- **Interrupt**
  - When the Timer1 overflows from FFFFH to 0000, it sets the TMR1IF (Timer1 Interrupt Flag) –Bit0 in the Peripheral Interrupt Register1 (PIR1).
  - This flag can be used two ways
    - A software loop can be set up to monitor the flag.
    - An interrupt can be generated.
  - Flag must be cleared to start the timer again.
Resetting Timer1 using CCP module

- When a CCP module associated with Timer1 is loaded with a 16-bit number and setup in the Compare mode, the count in Timer1 and the number in the CCP module are compared at every cycle.
- When a match is found, Timer1 is reset.
TMR1 Example

- Generate 100 usec clock; assuming internal clock is 10MHz (do example on page 349).
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.
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
    - Count = Td / [Inst. Clock Cycle(4) x Prescaler x PostScaler x clock period]
    - $8\text{msec}/[4 \times 16 \times 16 \times ((32^6)^{-1})-1]=249$

```
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             ; enable global interrupt
movwf INTCON,A         ; enable TMR2, set prescaler to 16, set
movlw 0x7E             ; postscaler to 16
movwf T2CON,A          ; enable TMR2 overflow interrupt
bsf PIE1, TMR2IE,A      ;
```

PR2=249
RCON: IPEN=1
IPR1: TMR2IP=1; TMR2IF=CLR
INTCON=C0; GLOBAL INT
T2CON=7E; TMR2 ENABLE AND SCALING SETUP
PIE1: TMR2IE=SET
# TMR2

## Registers Associated with PWM and Timer2

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTCON</td>
<td>GIE</td>
<td>GIE</td>
<td>TMR0IE</td>
<td>INTOIE</td>
<td>RIE</td>
<td>TMR0IF</td>
<td>INTOIF</td>
<td>RBIF</td>
</tr>
<tr>
<td>RCON</td>
<td>IPEN</td>
<td>SBOREN</td>
<td>—</td>
<td>R1</td>
<td>TO</td>
<td>PD</td>
<td>POR</td>
<td>BOR</td>
</tr>
<tr>
<td>PIR1</td>
<td>PSPIF</td>
<td>ADIF</td>
<td>RCIF</td>
<td>TXIF</td>
<td>SSPIF</td>
<td>CCP1IF</td>
<td>TMR2IF</td>
<td>TMR1IF</td>
</tr>
<tr>
<td>PIE1</td>
<td>PSPIE</td>
<td>ADIE</td>
<td>RCIE</td>
<td>TXIE</td>
<td>SSPIE</td>
<td>CCP1E</td>
<td>TMR2E</td>
<td>TMR1E</td>
</tr>
<tr>
<td>IPR1</td>
<td>PSPIP</td>
<td>ADIP</td>
<td>RCIP</td>
<td>TXIP</td>
<td>SSPIP</td>
<td>CCP1P</td>
<td>TMR2P</td>
<td>TMR1P</td>
</tr>
<tr>
<td>TRISB</td>
<td>PORTB</td>
<td>Data Direction Control Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRISC</td>
<td>PORTC</td>
<td>Data Direction Control Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMR2</td>
<td>Timer2 Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR2</td>
<td>Timer2 Period Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2CON</td>
<td>—</td>
<td>T2OUTPS3</td>
<td>T2OUTPS2</td>
<td>T2OUTPS1</td>
<td>T2OUTPS0</td>
<td>TMR2ON</td>
<td>T2CKPS1</td>
<td>T2CKPS0</td>
</tr>
<tr>
<td>CCFR1L</td>
<td>Capture/Compare/PWM Register 1, Low Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCFR1H</td>
<td>Capture/Compare/PWM Register 1, High Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP1CON</td>
<td>P1M1</td>
<td>P1M0</td>
<td>DC1B1</td>
<td>DC1B0</td>
<td>CCP1M3</td>
<td>CCP1M2</td>
<td>CCP1M1</td>
<td>CCP1M0</td>
</tr>
<tr>
<td>CCFR2L</td>
<td>Capture/Compare/PWM Register 2, Low Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCFR2H</td>
<td>Capture/Compare/PWM Register 2, High Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP2CON</td>
<td>—</td>
<td>—</td>
<td>DC2B1</td>
<td>DC2B0</td>
<td>CCP2M3</td>
<td>CCP2M2</td>
<td>CCP2M1</td>
<td>CCP2M0</td>
</tr>
<tr>
<td>ECCP1AS</td>
<td>ECCPASE</td>
<td>ECCPAS2</td>
<td>ECCPAS1</td>
<td>ECCPAS0</td>
<td>PSSAC1</td>
<td>PSSAC0</td>
<td>PSSBD1</td>
<td>PSSBD0</td>
</tr>
<tr>
<td>PWM1CON</td>
<td>PRSEN</td>
<td>PDC6</td>
<td>PDC5</td>
<td>PDC4</td>
<td>PDC3</td>
<td>PDC2</td>
<td>PDC</td>
<td>PDC0</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>DC1B1</th>
<th>DC1B0</th>
<th>CCP1M3-CCP1M0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CCP1 Mode Select</td>
</tr>
<tr>
<td>0000</td>
<td></td>
<td>CCP1 is off</td>
</tr>
<tr>
<td>0001</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0010</td>
<td></td>
<td>Compare mode. Toggle CCP1 output pin on match. (CCP1IF bit is set.)</td>
</tr>
<tr>
<td>0011</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0100</td>
<td></td>
<td>Capture mode, every falling edge</td>
</tr>
<tr>
<td>0101</td>
<td></td>
<td>Capture mode, every rising edge</td>
</tr>
<tr>
<td>0110</td>
<td></td>
<td>Capture mode, every 4th rising edge</td>
</tr>
<tr>
<td>0111</td>
<td></td>
<td>Capture mode, every 16th rising edge</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>Compare mode. Initialize CCP1 pin LOW, on compare match force CCP1 pin HIGH. (CCP1IF is set.)</td>
</tr>
<tr>
<td>1001</td>
<td></td>
<td>Compare mode. Initialize CCP1 pin HIGH, on compare match force CCP1 pin LOW. (CCP1IF is set.)</td>
</tr>
<tr>
<td>1010</td>
<td></td>
<td>Compare mode. Generate software interrupt on compare match. (CCP1IF bit is set, CCP1 pin is unaffected.)</td>
</tr>
<tr>
<td>1011</td>
<td></td>
<td>Compare mode. Trigger special event. (CCP1IF bit is set, and Timer1 or Timer3 is reset to zero.)</td>
</tr>
<tr>
<td>11xx</td>
<td></td>
<td>PWM mode</td>
</tr>
</tbody>
</table>
## REGISTER 15-1: **T3CON**: TIMER3 CONTROL REGISTER

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD16</td>
<td>T3CCP2</td>
<td>T3CKPS1</td>
<td>T3CKPS0</td>
<td>T3CCP1</td>
<td>T3SYNC</td>
<td>TMR3CS</td>
<td>TMR3ON</td>
<td></td>
</tr>
<tr>
<td>bit 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bit 0</td>
</tr>
</tbody>
</table>

### 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
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)
  - 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.
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.
Example

- What is the value of CCP1CON and TxCON if
  - We are using capture mode; rising edge
  - Based on TMR3 with no pre-scale, using external clock source (Use timer1 as clock for CCP1)

We need to change T3CON
CCP1CON = 0x05
T3CON = 0x42
CCP1 is INPUPT
Demo Board Pinout

Different setting for PIC18F45K20!
C- Example (CCP1 on RC2)

```
CCP1CON=0x05;  //Capture mode on every rising edge
T3CON=0x0;     //Timer1 for capture
T1CON=0x0;     //Timer1 internal clk, 1:1 prescaler
TRISB=0;       //make PORTB output port
TRISD=0;       //make PORTD output port
TRISCbits.TRISC2=1; //make CCP1 pin an input
CCPR1L=0;      //CCPR1L = 0
CCPR1H=0;      //CCPR1H = 0
while(1)
{
    TMR1H=0;    //clear Timer1
    TMR1L=0;

    PIR1bits.CCP1IF=0; //clear CCP1IF flag
    while(PIR1bits.CCP1IF==0); //wait for 1st rising edge

    T1CONbits.TMR1ON=1;  //start Timer1
    PIR1bits.CCP1IF=0;  //clear CCP1IF for next edge
    while(PIR1bits.CCP1IF==0); //wait for 2nd rising edge

    T1CONbits.TMR1ON=0; //stop Timer1
    PORTB=CCPR1L;
    PORTD=CCPR1H;      //display the clock count
}
```

Measure the period of Pulse - Counting the number of pulses between two rising edges

NOTE: Different setting for PIC18F45K20!
Measuring Period of an Incoming Pulse

NOTE: Different setting for PIC18F45K20!
Applications of Capture Mode

- Event arrival time recording
  - Swimming competition, need to compare different swimmer times
  - Number of events captured limited by the number of capture channels
- 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
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.
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 1

Toggle output RC2 (CCP1) every 10 pulses; assume a 1-Hz pulse is connected to the Timer(3) pin.

What will be the values for CCP1CON, CCPRIH/L and T3CON assuming we need to use compare mode and no pre-scaling is used

```
CCP1CON=0x20;  //Compare mode, toggle upon match
T3CON=0x42;    //Timer3 for Compare, 1:1 prescaler
TRISCbits.TRISC2=0; //CCP1 pin an output
TRISCbits.TRISC0=1; //T3CLK pin an input
CCPR1L=10;
CCPR1H=0;     //load CCPR1L
while(1)
{
    TMR3H=0;
    TMR3L=0;
    PIR1bits.CCP1IF=0;   //clear CCP1IF flag
    T3CONbits.TMR3ON=1; //turn on Timer3
    while(PIR1bits.CCP1IF==0); //wait for CCP:
        //CCP toggles CCP pin upon match
    T3CONbits.TMR3ON=0; //stop Timer3
}
```

**CCP1CON = 0x20**
**T3CON = 0x42**
CCPRIH=0
CCPRIL=10d
Example 2

Assume \( F_{\text{osc}} = 10 \) MHz. Use CCP to generate a square wave with 50% duty cycle and 40 msec period. Assume 1:1 prescaler. Use Timer(1).

- Find the values for CCP1CON, CCPRIH/L and T1CON assuming we need to use compare mode and no pre-scaling is used.

```
CCP1CON=0x02; //Compare mode, toggle upon match
T3CON=0x0;    //Timer1 for Compare, 1:1 prescaler
T1CON=0x0;    //Timer1 internal clk, 1:1 prescaler
TRISCbits.TRISC2=0; //make CCP1 pin an output
TRISCbits.TRISC0=1; //make T1CLK pin an input
CCPR1H=0xC3;  //load CCPR1H
CCPR1L=0x50;  //load CCPR1L
while(1)
{
    TMRLH=0;        //clear Timer1
    TMRL=0;
    PIR1bits.CCP1IF=0; //clear CCP1IF flag
    T1CONbits.TMR1ON=1; //turn on Timer1
    while(PIR1bits.CCP1IF==0); //wait for CCP1IF
    //CCP toggles CCP1 pin upon match
    T1CONbits.TMR1ON=0; //stop Timer1
}
```

Timer uses \( F_{\text{osc}}/4 = 0.4 \)usec

- CCP1CON = 0x02
- T1CON = 0x0
- CCPR1H/L=C350 (50,000)

See Notes
Pulse Width Modulation
Basic Idea

For example 75% of 249 = 186.75
→ 0.75 is equivalent to
→ DC1B2 + B1 = 11
CCP in the Pulse Width Modulation (PWM) Mode

- 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.
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

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

\[
\text{PWM duty cycle} = (\text{CCPRxL} : \text{CCPxCON<5:4>} ) \times T_{\text{OSC}} \times (\text{TMRy prescale factor})
\]

or \[\text{CCPR1L} = [\text{PR2}+1] \times \text{DutyCycle}\]
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.
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: \( PR2 = \frac{32MHz}{[4 \times 4 \times 10KHz]} - 1 = 199 \)
    
  \[ PR2 = \text{Fosc} / [4 \times N \times \text{Fdesired}] - 1 ; \text{N is the prescaler value} \]
  - Duty Cycle Value: \( \text{CCPR1L} = [PR2+1] \times \text{DutyCycle} \)
    
    \[ = 200 \times 40\% = 80.00 \]

<table>
<thead>
<tr>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>DCxB1</td>
<td>DCxB0</td>
<td>CCPxM3</td>
<td>CCPxM2</td>
<td>CCPxM1</td>
<td>CCPxM0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( \text{CCPRL Register} = 80d \)
- \( \text{PR2 Register} = 199d \)

DCxB0 & B1 = 00
PWM Example 3

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

Remember:
- 1msec x 4MHz = 4000 counts
- 40% → 1600 = 640h
- 60% → 2400 = 960h

Load TMR3H/L = 640 and CCP1H/L = 640

Load TMR3H/L = 960 and CCP1H/L = 960

1KHz 40% duty cycle waveform
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

```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;
    T3CON = 0x00;
    T1CON = 0x0;
    TRISD = 0x0;
    TRISCbits.TRISC2 = 1; // Set RC2 // Make sure the input
                      // is a square signal with no offset.
    CCP1L = 0;
    CCP1H = 0;
    while (1)
    {
        TMR1H = 0;
        TMR1L = 0;
        PORTbits.R0 = PORTbits.RD0;
        PIRLbits.CCP1IF = 0;
        PORTbits.RDO = PORTbits.RD0;

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

        while (PIRLbits.CCP1IF == 0); // wait for the second rising edge
        T1CONbits.TMR1ON = 0;
        PulsePeriod[0] = CCP1L; // the number of counts are here!
        PulsePeriod[1] = CCP1H;
    }
}
```
Programing ECCP (CCP1) In PIC18F46K20 - 2

It performs well for frequencies less than 5KHz when the clock is 4 usec.
Example of PWM using CCP1(RC2)

```c
// main program
void main (void)
{
    OSCON = 0x40;  // IRCF = 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;
    }
}
```

Note that we use OSCTUNE to adjust the frequency.

CCPR1L sets the value of the duty cycle.

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

\[
\text{PR2} \times \text{DC}\% = 50.00 = 100 \times 0.5 \rightarrow \text{for 50\% Duty Cycle}
\]
Controlling a DC Motor Using PWM

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