Chapter 8: Timers and CCP Modules

The PIC18 Microcontroller

Han-Way Huang

Minnesota State University, Mankato
Introduction

- **Time** is represented by the count in a timer.
- There are many applications that cannot be implemented without a timer:
  1. Event arrival time recording and comparison
  2. Periodic interrupt generation
  3. Pulse width and period measurement
  4. Frequency and duty cycle measurement
  5. Generation of waveforms with certain frequency and duty cycle
  6. Time references
  7. Event counting
  8. Others
The PIC18 Timer System

- A PIC18 microcontroller may have up to 5 timers: Timer0…Timer 4.
- Timer0, Timer1, and Timer3 are 16-bit timers whereas Timer2 and Timer4 are 8-bit.
- When a timer rolls over, an interrupt may be generated if it is enabled.
- Both Timer2 and Timer4 use instruction cycle clock as the clock source whereas the other three timers may also use external clock input as the clock source.
- A PIC18 device may have one, two, or five CCP modules.
- CCP stands for **Capture, Compare, and Pulse Width Modulation**.
- Each CCP module can be configured to perform capture, compare, or PWM function.
- In **capture** operation, the CCP module copy the contents of a timer into a capture register on an signal edge.
- In **compare** operation, the CCP module compares the contents of a CCPR register with that of Timer1 (or Timer3) in every clock cycle. When these two registers are equal, the associated pin may be pulled to high, or low, or toggled.
- In **PWM** mode, the CCP module can be configured to generate a waveform with certain frequency and duty cycle.
**Timer0**

- Can be configured as an 8-bit or 16-bit timer or counter.
- Can select the internal instruction cycle clock or the T0CKI signal as the clock signal.
- The user can choose to divide the clock signal by a prescaler before connecting it to the clock input to Timer0.
- The T0CON register controls the operation of Timer0.

---

**Figure 8.1b Timer0 block diagram in 16-bit mode (redraw with permission of Microchip)**
The PIC18 Microcontroller

TMR0ON: Timer0 on/off control bit
- 0 = stops Timer0
- 1 = Enables Timer0

T08BIT: Timer0 8-bit/16-bit control bit
- 0 = Timer0 is configured as a 16-bit timer
- 1 = Timer0 is configured as an 8-bit timer

T0CS: Timer0 clock source select
- 0 = Instruction cycle clock
- 1 = Transition on T0CKI pin

T0SE: Timer0 source edge select bit
- 0 = Increment on falling edge transition on T0CKI pin
- 1 = Increment on rising edge transition on T0CKI pin

PSA: Timer0 prescaler assignment bit
- 0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.
- 1 = Timer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.

T0PS2:T0PS0: Timer0 prescaler select bits
- 000 = 1:2 prescaler value
- 001 = 1:4 prescaler value
- 010 = 1:8 prescaler value
- 011 = 1:16 prescaler value
- 100 = 1:32 prescaler value
- 101 = 1:64 prescaler value
- 110 = 1:128 prescaler value
- 111 = 1:256 prescaler value

Figure 8.2 T0CON register (reprint with permission of Microchip)
- Timer0 can operate as a timer or as a counter.
- When the clock source is the instruction cycle clock, it operates as a timer.
- When the clock source is the T0CKI pin, it operates as a counter.
- As shown in Figure 8.1b, when PIC18 reads the TMR0L register, the upper half of Timer0 is latched into the TMR0H register. This makes sure that the PIC18 always reads a 16-bit value that its upper byte and lower byte belong to the same time.

**Example 8.2** Write a subroutine to create a time delay that is equal to 100 ms times the contents of the PRODL register assuming that the crystal oscillator is running at 32 MHz.

**Solution:** The 100 ms delay can be created as follows:
1. Place the value 15535 into the TMR0 high byte and the TMR0L register so that Timer0 will overflow in 50000 clock cycles.
2. Choose instruction cycle clock as the clock source and set the prescaler to 16 so that Timer0 will roll over in 100 ms.
3. Enable Timer0.
4. Wait until Timer0 to overflow.
```assembly
delay
    movlw 0x83        ; enable TMR0, select internal clock,
    movwf T0CON,A   ; set prescaler to 16
loopd
    movlw 0x3C       ; load 15535 into TMR0 so that it will
    movwf TMR0H,A    ; roll over in 50000 clock cycles
    movlw 0xAF       ;
    movwf TMR0L,A    ;
    bcf INTCON,TMR0IF,A ; clear the TMR0IF flag
wait
    btfss INTCON,TMR0IF,A ;
    bra wait         ; wait until 100 ms is over
    decfsz PRODL,F,A
    bra loopd
return
```

The PIC18 Microcontroller
In C language,

```c
void delay (char cx)
{
    int i;
    T0CON = 0x83;  /* enable TMR0, select instruction clock, prescaler set to 16 */
    for (i = 0; i < cx; i++) {
        TMR0 = 15535;  /* load 15535 into TMR0 so that it rolls */
                      /* over in 50000 clock cycles */
        INTCONbits.TMR0IF = 0;
        while(!(INTCONbits.TMR0IF)); /* wait until TMR0 rolls over */
    }
    return;
}
```
Timer1

- Is a 16-bit timer/counter depending upon the clock source.
- An interrupt may be requested when Timer1 rolls over from 0xFFFF to 0x0000.
- Timer1 can be reset when the CCP module is configured to compare mode to generate a special event trigger.
- Timer1 operation is controlled by the T1CON register.
- Timer1 can be configured to use the oscillator connected to the T1OSO and T1OSI pins.
- The Timer1 oscillator is primarily intended for a 32 KHz crystal.
- Timer1 can be used to create time delays and measure the frequency of an unknown signal.
Figure 8.3 Timer1 block diagram: 16-bit mode (redraw with permission of Microchip)
RD16: 16-bit read/write mode enable bit
   0 = Enables read/write of Timer1 in two 8-bit operations
   1 = Enable read/write of Timer1 in 16-bit operation

T1CKPS1:T1CKPS0: Timer1 input clock prescale select bits
   00 = 1:1 prescale value
   01 = 1:2 prescale value
   10 = 1:4 prescale value
   11 = 1:8 prescale value

T1OSCEN: Timer1 oscillator enable bit
   0 = Timer1 oscillator is shut off
   1 = Timer1 oscillator is enabled

T1SYNC: Timer1 external clock input synchronization select bit
When TMR1CS = 1
   0 = Synchronize external clock input
   1 = Do not synchronize external clock input
When TMR1CS = 0
   This bit is ignored.

TMR1CS: Timer1 clock source select bit
   0 = Instruction cycle clock (FOSC/4)
   1 = External clock from pin RC0/T1OSO/T13CKI

TMR1ON: Timer1 on bit
   0 = Stop Timer1
   1 = Enables Timer1

Figure 8.4. T1CON contents (redraw with permission of Microchip)
**Example 8.3** Use Timer0 as a timer to create a one-second delay and use Timer1 as a counter to count the rising (or falling) edges of an unknown signal (at the T1CKI pin) arrived in one second which would measure the frequency of the unknown signal. Write a program to implement this idea assuming that the PIC18 MCU is running with a 32 MHz crystal oscillator.

**Solution:**
A one-second delay can be created by placing 10 in PRODL and calling the `delay` function in Example 8.2.

Timer1 should be configured as follows:

- 16-bit mode
- prescaler value set to 1
- disable oscillator
- do not synchronize external clock input
- select external T1CKI pin signal as the clock source
- Timer1 may overflow many times in one second.
- The user must enable the Timer1 overflow interrupt and keep track of the number of times that it interrupts.

The setting of Timer1 interrupt is as follows:

- Enable priority interrupt
- Place Timer1 interrupt at high priority
- Enable only Timer1 roll-over interrupt
#include <p18F8680.inc>

```
t1ov_cnt
  set 0x00 ; Timer1 rollover interrupt count
freq
  set 0x01 ; to save the contents of Timer1 at the end
org 0x00
  goto start

; high priority interrupt service routine
org 0x08
  btfss PIR1,TMR1IF,A ; skip if Timer1 roll-over interrupt
  retfie ; return if not Timer1 interrupt
  bcf PIR1,TMR1IF,A ; clear the interrupt flag
  incf t1ov_cnt,F,A ; increment Timer1 roll-over count
  retfie

; dummy low priority interrupt service routine
org 0x18
  retfie

start
  clrf t1ov_cnt,A ; initialize Timer1 overflow cnt to 0
  clrf freq,A ; initialize frequency to 0
  clrf freq+1,A ; "
  clrf TMR1H ; initialize Timer1 to 0
  clrf TMR1L ; "
  clrf PIR1 ; clear all interrupt flags
  bsf RCON,IPEN,A ; enable priority interrupt
```
movlw 0x01 ; set TMR1 interrupt to high priority
movwf IPR1,A ;
movwf PIE1,A ; enable Timer1 roll-over interrupt
movlw 0x87 ; enable Timer1, select external clock, set
movwf T1CON,A ; prescaler to 1, disable crystal oscillator
movlw 0xC0 ; enable global and peripheral interrupt
movwf INTCON,A ;
movlw 0x0A
movwf PRODL,A ; prepare to call delay to wait for 1 second
call delay ; Timer1 overflow interrupt occur in this second
movff TMR1L,freq ; save frequency low byte
movff TMR1H,freq+1 ; save frequency high byte
bcf INTCON,GIE,A ; disable global interrupt

forever
nop
bra forever
end

The C language version of the program is in the following slides.

Copyright @ 2005 Thomson Delmar Learning

H. Huang Transparency No.8-15
#include <p18F8680.h>
unsigned int t1ov_cnt;
unsigned short long freq;
void high_ISR(void);
void low_ISR(void);
#pragma code high_vector = 0x08 // force the following statement to
void high_interrupt (void) // start at 0x08
{
    _asm
    goto high_ISR
    _endasm
}
#pragma code //return to the default code section
#pragma interrupt high_ISR //return to the default code section
void high_ISR (void)
{
    if(PIR1bits.TMR1IF){
        PIR1bits.TMR1IF = 0;
        t1ov_cnt ++;
    }
}
void delay (char cx);  /* prototype declaration */
void main (void)
{
    char t0_cnt;
    char temp;
    t1ov_cnt = 0;
    freq = 0;
    TMR1H = 0;  /* force Timer1 to count from 0 */
    TMR1L = 0;  /* */
    PIR1 = 0;  /* clear Timer1 interrupt flag */
    RCONbits.IPEN = 1;  /* enable priority interrupt */
    IPR1 = 0x01;  /* set Timer1 interrupt to high priority */
    PIE1 = 0x01;  /* enable Timer1 roll-over interrupt */
    T1CON = 0x83;  /* enable Timer1 with external clock, prescaler 1 */
    INTCON = 0xC0;  /* enable global and peripheral interrupts */
    delay (10);  /* create one-second delay and wait for interrupt */
    INTCONbits.GIE = 0;  /* disable global interrupt */
    temp = TMR1L;
    freq = t1ov_cnt * 65536 + TMR1H * 256 + temp;
}
**Timer2**
- 8-bit timer TMR2 and 8-bit period register PR2.

- TMR2 is counting up and comparing with PR2 in every clock cycle.
- When TMR2 equals PR2, the EQ signal will reset TMR2.
- A postscaler is applied to the EQ signal to generate the TMR2 interrupt.
- The TMR2 output is fed to the synchronous serial port module.
- The operation of Timer2 is controlled by T2CON register.
The PIC18 Microcontroller

Figure 8.6. T2CON control register (redraw with permission of Microchip)
Example 8.4 Assume that the PIC18F8680 is running with a 32 MHz crystal oscillator. Write an instruction sequence to generate periodic interrupts every 8 ms with high priority using Timer2.

Solution: By setting the prescaler and postscaler to 16 and loading 249 into the PR2 register, Timer2 will generate periodic interrupt every 8 ms:

```
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 INTCAN,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
```
Timer3
- Timer3 consists of two 8-bit registers TMR2H and TMR2L.
- Timer3 can choose to use either the internal (instruction cycle clock) or external signal as the clock source.
- The block diagram of Timer3 is quite similar to that of Timer1.
- Reading TMR3L will load the high byte of Timer3 into the TMR3H register.
- Timer3 operation is controlled by the T3CON register.
Figure 8.7 Timer3 block diagram: 16-bit mode (redraw with permission of Microchip)
<table>
<thead>
<tr>
<th>Value after reset</th>
<th>RD16</th>
<th>T3CCP2</th>
<th>T3CKPS1</th>
<th>T3CKPS0</th>
<th>T3CCP1</th>
<th>T3SYNC</th>
<th>TMR3CS</th>
<th>TMR3ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

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

T3SYNC: Timer3 external clock input synchronization select bit
- When TMR3CS = 1
  - 0 = Synchronizes external clock input
  - 1 = Do not synchronize external clock input
- When TMR3CS = 0
  - This bit is ignored.

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

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

Figure 8.8. T3CON contents (redraw with permission of Microchip)
**Timer4**
- Only available to the PIC18F8X2X and PIC6X2X devices.
- The block diagram of Timer4 is shown in Figure 8.9.
- 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.

---

**Figure 8.9 Timer4 block diagram (redraw with permission of Microchip)**
C Library Functions for Timers

Functions for disabling timers

void CloseTimer0 (void);
void CloseTimer1 (void);
void CloseTimer2 (void);
void CloseTimer3 (void);
void CloseTimer4 (void);

Functions for configuring timers

void OpenTimer0 (unsigned char config);
void OpenTimer1 (unsigned char config);
void OpenTimer2 (unsigned char config);
void OpenTimer3 (unsigned char config);
void OpenTimer4 (unsigned char config);

The arguments to these functions are a bit mask that is created by ANDing the values from each category.

Include the timers.h file in order to use these library functions.
### Enable Timer0 Interrupt
- **TIMER_INT_ON**: enable interrupt
- **TIMER_INT_OFF**: disable interrupt

### Timer Width
- **T0_8BIT**: 8-bit mode
- **T0_16BIT**: 16-bit mode

### Clock Source
- **T0_SOURCE_EXT**: external clock source
- **T0_SOURCE_INT**: internal clock source

### External Clock Trigger
- **T0_EDGE_FALL**: External clock on falling edge
- **T0_EDGE_RISE**: External clock on rising edge

### Prescale Value
- **T0_PS_1_n**: 1: n prescale (n = 1, 2, 4, 8, 16, 32, 64, 128, or 256)

### Example
```
OpenTimer0 (TIMER_INT_ON & T0_8BIT & T0_SOURCE_INT & T0_PS_1_32);
```
Functions for Reading Timer Values

unsigned int ReadTimer0 (void);
unsigned int ReadTimer1 (void);
unsigned char ReadTimer2 (void);
unsigned int ReadTimer3 (void);
unsigned char ReadTimer4 (void);

unsigned int cur_time;
cur_time = ReadTimer1();

Functions for writing values into timers

void WriteTimer0 (unsigned int timer);
void WriteTimer1 (unsigned int timer);
void WriteTimer2 (unsigned char timer);
void WriteTimer3 (unsigned int timer);
void WriteTimer4 (unsigned char timer);

writeTimer0 (15535);
Capture/Compare/PWM (CCP) Modules

- Each CCP module requires the use of timer resource.
- In capture or compare mode, the CCP module may use either Timer1 or Timer3 to operate.
- In 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.
- The operation of a CCP module is controlled by the CCPxCON register.
<table>
<thead>
<tr>
<th>Value after reset</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCxB1:DCxB0: PWM duty cycle bit 1 and bit 0 for CCP module x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>capture mode:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>compare mode:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM mode:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>These two bits are the lsbs (bit 1 and bit 0) of the 10-bit PWM duty cycle.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCPxM3:CCPxM0: CCP module x mode select bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000 = capture/compare/PWM disabled (resets CCPx module)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 = reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010 = compare mode, toggle output on match (CCPxIF bit is set)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100 = capture mode, every falling edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101 = capture mode, every rising edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110 = capture mode, every 4th rising edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111 = capture mode, every 16th rising edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000 = compare mode, initialize CCP pin low, on compare match force CCP pin high (CCPxIF bit is set)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001 = compare mode, initialize CCP pin high, on compare match force CCP pin low (CCPxIF bit is set)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010 = compare mode, generate software interrupt on compare match (CCP pin unaffected, CCPxIF bit is set).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011 = compare mode, trigger special event (CCPxIF bit is set)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>For CCP1 and CCP2: Timer1 or Timer3 is reset on event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>For all other modules: CCPx pin is unaffected and is configured as an I/O port.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11xx = PWM mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.10 CCPxCON register (x = 1,...,5) (redraw with permission of Microchip)
CCP Module Configuration

- Each module is associated with a control register (CCPxC) and a data register (CCPRx).
- The data register in turn consists of two 8-bit registers: CCPRxL and CCPRxH.
- The CCP modules utilize Timers 1, 2, 3, or 4, depending on the module selected.
- Timer1 and Timer3 are available to modules in capture or compare mode.
- Timer2 and Timer4 are available to modules in PWM mode.
- The assignment of a particular timer to a module is determined by the bit 6 and bit 3 of the T3CON register as shown in Figure 8.11.
Figure 8.11 CCP and Timer interconnect configurations (redraw with permission of Microchip)
**CCP in Capture Mode**
- Main use of CCP is to capture event arrival time
- An event is represented by a signal edge.
- The PIC18 event can be one of the following:
  1. every falling edge
  2. every rising edge
  3. every 4th rising edge
  4. every 16th rising edge

![Diagram of CCP in Capture Mode](image_url)

Figure 8.13 Capture mode operation block diagram (redraw with permission of Microchip)
Capture Operation

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

Microchip C Library Functions for CCP in Capture Mode

- Need to include the file `capture.h` in order to use these functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseCapture\x</td>
<td>Disable capture channel \x</td>
</tr>
<tr>
<td>OpenCapture\x</td>
<td>Configure capture channel \x</td>
</tr>
<tr>
<td>ReadCapture\x</td>
<td>Read a value from CCP channel \x</td>
</tr>
</tbody>
</table>
void OpenCapture1 (unsigned char config);
void OpenCapture2 (unsigned char config);
void OpenCapture3 (unsigned char config);
void OpenCapture4 (unsigned char config);
void OpenCapture5 (unsigned char config);

There are two values for the parameter \texttt{config}: interrupt enabling and the edge to capture.

Interrupt enabling

\begin{itemize}
  \item \texttt{CAPTURE\_INT\_ON} : interrupt enabled
  \item \texttt{CAPTURE\_INT\_OFF} : interrupt disabled
\end{itemize}

Edge to capture

\begin{itemize}
  \item \texttt{Cx\_EVERY\_FALL\_EDGE} : capture on every falling edge
  \item \texttt{Cx\_EVERY\_RISE\_EDGE} : capture on every rising edge
  \item \texttt{Cx\_EVERY\_4\_RISE\_EDGE} : capture on every 4th rising edge
  \item \texttt{Cx\_EVERY\_16\_RISE\_EDGE} : capture on every 16th rising edge
\end{itemize}
Applications of Capture Mode

- Event arrival time recording
- Period measurement
- Pulse width measurement
- Interrupt generation
- Event counting
- Time reference
- Duty cycle measurement
Example 8.5 Period measurement. Use the CCP channel 1 in capture mode to measure the period of an unknown signal assuming that the PIC18 MCU is running with a 16 MHz crystal oscillator. Use the number of clock cycles as the unit of period. The period of the unknown signal is shorter than 65536 clock cycles.

Solution:
Either two consecutive rising edges or two falling edges must be captured. The difference of these two edges becomes the period of the signal. The required timers settings are

- CCP1 (RC2): input
- Timer1: 16-bit mode, use instruction clock as clock source, 1:1 prescaler
- Timer3: select Timer1 as base timer for the CCP1 capture mode
- CCP1: capture on every rising edge
- Disable CCP1 interrupt
```assembly
#include <p18F8720.inc>
org 0x00
goto start
org 0x08
goto
org 0x18
retfie

start
bsf TRISC,CCP1,A ; configure CCP1 pin for input
movlw 0x81 ; use Timer1 as the time base
movwf T3CON,A ; of CCP1 capture
bcf PIE1,CCP1IE,A ; disable CCP1 capture interrupt
movlw 0x81 ; enable Timer1, prescaler set to 1,
movwf T1CON,
movlw 0x05 ; 16-bit, use instruction cycle clock
movwf CCP1CON,A ; set CCP1 to capture on every rising edge
movwf CCP1CON,A ;
bcf PIR1,CCP1IF,A ; clear the CCP1IF flag
edge1
btfss PIR1,CCP1IF,A ; wait for the first edge to arrive
bra edge1
movff CCPR1H,PRODH ; save the first edge
movff CCPR1L,PRODL ;
```

The PIC18 Microcontroller

Copyright @ 2005 Thomson Delmar Learning

H. Huang Transparency No.8-37
The PIC18 Microcontroller

```assembly
bcf PIR1, CCP1IF,A ; clear the CCP1IF flag
edge2 btfss PIR1, CCP1IF,A ; wait for the second edge to arrive
bra edge2 ; "
clrfrf CCP1CON ; disable CCP1 capture
movfrf PRODL, W, A
subwfrf CCPR1L, W, A ; subtract first edge from 2nd edge
movwrf PRODL, A ; and leave the period in PRODH:PRODL
movfrf PRODH, W, A ; "
subwfrf CCPR1H, W, A ; "
movwrf PRODH, A ; "
forever goto forever ;
end
```

The C language version of the program is in the next slide.
#include <p18F8720.h>
void main (void)
{
    unsigned int period;

    TRISCbits.TRISC2 = 1;   /* configure CCP1 pin for input */
    T3CON = 0x81;           /* use Timer1 as the time base for CCP1 capture */
    PIE1bits.CCP1IE = 0;    /* disable CCP1 capture interrupt */
    PIR1bits.CCP1IF = 0;    /* clear the CCP1IF flag */
    T1CON = 0x81;           /* enable 16-bit Timer1, prescaler set to 1 */
    CCP1CON = 0x05;        /* capture on every rising edge */
    while (!(PIR1bits.CCP1IF)); /* wait for 1st rising edge */
    PIR1bits.CCP1IF = 0;
    period = CCPR1;        /* save the first edge (CCPR1 is accessed as a 16-bit value) */
    while (!(PIR1bits.CCP1IF)); /* wait for the 2nd rising edge */
    CCP1CON = 0x00;        /* disable CCP1 capture */
    period = CCPR1 - period;
}
- The clock period of an unknown signal could be much longer than $2^{16}$ clock cycles.
- One will need to keep track of the number of times that the timer overflows.
- Each timer overflow adds $2^{16}$ clock cycles to the period.

Let

$$ovcnt = \text{timer overflow count}$$
$$diff = \text{the difference of two edges}$$
$$edge1 = \text{the captured time of the first edge}$$
$$edge2 = \text{the captured time of the second edge}$$

**Case 1**: $\text{edge2} \geq \text{edge1}$

$$\text{period} = ovcnt \times 2^{16} + \text{diff}$$

**Case 2**: $\text{edge1} > \text{edge2}$

$$\text{period} = (ovcnt - 1) \times 2^{16} + \text{diff}$$

- The Timer1 overflow interrupt should be enabled after the first signal edge is captured.
- Timer1 interrupt service routine simply increments `ovcnt` by 1 and returns.
Example 8.6 Write a program to measure the period of a signal connected to the CCP1 pin assuming that the instruction clock is running at 5 MHz. Make the program more general so that it can also measure the period of a signal with very low frequency.

Solution:

```c
#include <p18F8720.inc>

ov_cnt set 0x00 ; timer overflow count
per_hi set 0x01 ; high byte of edge difference
per_lo set 0x02 ; low byte of edge difference
org 0x00
start
org 0x08
hi_pri_ISR ; go to the high-priority service routine
org 0x18
retfie
```
start   clrf   ov_cnt, A ; initialize overflow count by 1
bcf    INTCON, GIE A ; disable all interrupts
bsf    RCON, IPEN A ; enable priority interrupt
bcf    PIR1, TMR1IF, A ; clear the TMR1IF flag
bsf    IPR1, TMR1IP, A ; set Timer1 interrupt to high priority
bsf    TRISC, CCP1, A ; configure CCP1 pin for input
movlw  0x81 ; use Timer1 as the time base
movwf  T3CON, A ; of CCP1 capture
bcf    PIE1, CCP1IE, A ; disable CCP1 capture interrupt
movlw  0x81 ; enable Timer1, prescaler set to 1,
movwf  T1CON, A ; 16-bit mode, use instruction cycle clock
movlw  0x05 ; set CCP1 to capture on every rising edge
movwf  CCP1CON, A ; 
bcf    PIR1, CCP1IF, A ; clear the CCP1IF flag
edge1  btfss PIR1, CCP1IF, A ; wait for the first edge to arrive
goto   edge1 ; 
movff  CCPR1H, per_hi ; save the high byte of captured edge
movff  CCPR1L, per_lo ; save the low byte of captured edge
bcf    PIR1, TMR1IF, A
movlw  0xC0
iorwf  INTCON, F, A ; enable global interrupts
bsf    PIE1, TMR1IE ; enable Timer1 overflow interrupt
edge2  btfss PIR1,CCP1IF,A ; wait for the 2nd edge to arrive
goto    edge2
movf    per_lo,W,A
subwf   CCPR1L,W,A
movwf   per_lo,A ; save the low byte of edge difference
movf    per_hi,W,A
subwfb  CCPR1H,W,A
movwf   per_hi,A ; save the high byte of edge difference
btfsc   STATUS,C,A
goto    forever
decf    ov_cnt,A ; 1st edge is larger, so decrement overflow count
negf    per_lo,F ; compute its magnitude
comf    per_hi,F ; “
movlw   0x00 ; “
addwfc  per_hi,F ; “
forever  nop
goto    forever
hi_pri_ISR btfss PIR1,TMR1IF,A ; high priority interrupt service routine
retfie  ; not Timer1 interrupt, so return
incf    ov_cnt
bcf     PIR1,TMR1IF,A ; clear Timer1 overflow interrupt flag
retfie
end
The PIC18 Microcontroller

```c
#include <p18F8720.h>
#include <timers.h>
#include <capture.h>

unsigned int ov_cnt, temp;
unsigned short long period; /* 24-bit period value */

void high_ISR(void);
void low_ISR(void);

#pragma code high_vector = 0x08 // force the following statement to void

void high_interrupt (void) // start at 0x08
{
  _asm
    goto high_ISR
  _endasm
}

#pragma interrupt high_ISR

void high_ISR (void)
{
  if (PIR1bits.TMR1IF) {
    PIRbits.TMR1IF = 0;
    ov_cnt ++;
  }
}
```

Copyright @ 2005 Thomson Delmar Learning

H. Huang Transparency No.8-44
void main (void)
{
    unsigned int temp1;
    ov_cnt = 0;
    INTCONbits.GIE = 0; /* disable global interrupts */
    RCONbits.IPEN = 1; /* enable priority interrupts */
    PIR1bits.TMR1IF = 0;
    IPR1bits.TMR1IP = 1; /* promote Timer1 rollover interrupt to high priority */
    TRISCbits.TRISC2 = 1; /* configure CCP1 pin for input */
    OpenTimer1 (TIMER_INT_ON & T1_16BIT_RW & T1_PS_1_1 &
    T1_OSC1EN_OFF & T1_SYNC_EXT_OFF &
    T1_SOURCE_INT);
    OpenTimer3 (TIMER_INT_OFF & T3_16BIT_RW & T3_PS_1_1 &
    T3_SOURCE_INT & T3_PS_1_1 & T3_SYNC_EXT_ON &
    T1_SOURCE_CCP); /* turn on Timer3 and appropriate parameters */
    OpenCapture1 (CAPTURE_INT_OFF & C1_EVERY_RISE_EDGE);
    PIE1bits.CCP1IE = 0; /* disable CCP1 capture interrupt */
    PIR1bits.CCP1IF = 0;
    while(!(PIR1bits.CCP1IF));
}
temp = ReadCapture1( ); /* save the first captured edge */
PIR1bits.CCP1IF = 0;
PIR1bits.TMR1IF = 0;
INTCON |= 0xC0; /* enable global interrupts */
PIE1bits.TMR1IE = 1; /* enable Timer1 rollover interrupt */
while(!(PIR1bits.CCP1IF)); /* enable Timer1 rollover interrupt */
CloseCapture1(); /* disable CCP1 capture */
temp1 = ReadCapture1( );
if (temp1 < temp)
    ov_cnt--;
period = ov_cnt * 65536 + temp1 - temp;
}
CCP in Compare Mode

- The 16-bit CCPRx register is compared against the TMR1 (or TMR3).
- When they match, one of the following actions may occur on the associated CCPx pin:
  1. driven high
  2. driven low
  3. toggle output
  4. remains unchanged

How to Use the Compare Mode?

1. Makes a copy of the 16-bit timer value (Timer1 or Timer3)
2. Adds to this copy a delay count
3. Stores the sum in the CCPRxH:CCPRxL register pair

Special Event Trigger

- The CCP1 and CCP2 modules can also generate this event to reset TMR1 or TMR3 depending on which timer is the base timer.
- The CCP compare mode can be used to generate waveforms and create delays.
Example 8.7 Use CCP1 to generate a periodic waveform with 40% duty cycle and 1 KHz frequency assuming that the instruction cycle clock frequency is 4 MHz.

Solution: The waveform of 1 KHz waveform is shown in Figure 8.20.

![Waveform Diagram](image)

Figure 8.20 1KHz 40% duty cycle waveform

The algorithm is shown in Figure 8.21.
Figure 8.21b Logic flow for generating a 1-KHz waveform with 40% duty cycle
```c
#include <p18F8720.inc>

hi_hi equ 0x06 ; number (1600) of clock cycles that signal
hi_lo equ 0x40 ; is high
lo_hi equ 0x09 ; number (2400) of clock cycles that signal
lo_lo equ 0x60 ; is low

org 0x00 goto start;

... start
bcf TRISC,CCP1 ; configure CCP1 pin for output
movlw 0xC9 ; enable 16-bit Timer3, prescaler 1:1
movwf T3CON ; “
bcf PIR1,CCP1IF
movlw 0x09 ; CCP1 pin set high initially and
movwf CCP1CON ; pull low on match
; CCPR1 \( \leftarrow \) TMR3 + 1600 ; start a new compare operation
movlw hi_lo ; “
addwf TMR3L,W ; “
movwf CCP1L ; “
movlw hi_hi ; “
addwfc TMR3H,W ; “
addwfc TMR3H,W ; “
```
The PIC18 Microcontroller

```
bcf    PIR1,CCP1IF
hi_time btfss PIR1,CCP1IF ; wait until CCPR1 matches TMR3
bra    hi_time ;
bcf    PIR1,CCP1IF
movlw  0x08 ; CCP1 pin set low initially and
movwf   CCP1CON ; pull high on match
; CCPR1 ← CCPR1 + 2400 ; start another compare operation
movlw   lo_lo ;
addwf   CCPR1L,F ;
movlw   lo_hi ;
addwfc  CCPR1H,F ;
lo_time btfss PIR1,CCP1IF ; wait until CCPR1 matches TMR3
bra    lo_time ;
bcf    PIR1,CCP1IF
movlw  0x09 ; CCP1 pin set high initially and
movwf   CCP1CON ; pull low on match
movlw   hi_lo ; start another new compare operation
movlw   hi_hi ;
addwf   CCPR1L,F ;
movlw   hi_hi ;
addwfc  CCPR1H,F ;
bra    hi_time
end
```
#include <p18F8720.h>
void main (void)
{
    TRISCbits.TRISC2 = 0;     /* configure CCP1 pin for output */
    T3CON = 0xC9;            /* turn on TMR3 in 16-bit mode, TMR3 & TMR4 as base timer for all CCP modules */
    CCP1CON = 0x09;          /* configure CCP1 pin set high initially and pull low on match */
    CCPR1 = TMR3 + 1600;     /* start CCP1 compare operation with 1600 cycles delay */
    PIR1bits.CCP1IF = 0;
    while (1) {
        while (!(PIR1bits.CCP1IF));
        PIR1bits.CCP1IF = 0;
        CCP1CON = 0x08;          /* set CCP1 pin low initially, pull high on match */
        CCPR1 += 2400;           /* start CCP1 compare with 2400 as delay */
        while (!(PIR1bits.CCP1IF));
        PIR1bits.CCP1IF = 0;
        CCP1CON = 0x09;          /* change CCP1 setting */
        CCPR1 += 1600;
    }
}
Example 8.8 Use interrupt-driven approach to generate the waveform specified in Example 8.7.

**Solution:** This program uses a flag to select either 1600 (=0) or 2400 (=1) as the delay count for the compare operation.

```c
#include <p18F8720.inc>
hi_hi equ 0x06 ; number (1600) of clock cycles that signal is high
hi_lo equ 0x40 ;
lo_hi equ 0x09 ; number (2400) of clock cycles that signal is low
lo_lo equ 0x60 ;
flag equ 0x00 ; select 1600 (=0) or 2400 (=1) as delay
org 0x00
start bcf TRISC,CCP1 ; configure CCP1 pin for output
org 0x00
goto start
org 0x08
goto hi_ISR
org 0x18
retfie
start
movlw 0xC9 ; choose TMR3 as the base timer for
movwf T3CON ; CCP1
movlw 0x09 ; configure CCP1 pin to set high initially
```

H. Huang Transparency No.8-54
The PIC18 Microcontroller

\[
\begin{align*}
\text{movwf} & \quad \text{CCP1CON} \quad \text{; and pull low on match} \\
\text{; start a compare operation so that CCP1 pin stay high for 400 \(\mu\)s}
\end{align*}
\]

\[
\begin{align*}
\text{movlw} & \quad \text{hi_lo} \\
\text{addwf} & \quad \text{TMR3L,W} \\
\text{movwf} & \quad \text{CCPR1L} \\
\text{movlw} & \quad \text{hi_hi} \\
\text{addwfc} & \quad \text{TMR3H,W} \\
\text{movwf} & \quad \text{CCPR1H} \\
\text{bcf} & \quad \text{PIR1,CCP1IF} \\
\text{hi_lst} & \quad \text{btfss} \quad \text{PIR1,CPP1IF} \\
\text{bra} & \quad \text{hi_lst} \\
\text{bcf} & \quad \text{PIR1,CCP1IF} \\
\text{movlw} & \quad 0x02 \quad \text{; CCP1 pin toggle on match} \\
\text{movlw} & \quad \text{CCP1CON} \quad \text{; “} \\
\text{movlw} & \quad \text{lo_lo} \quad \text{; start next compare operation} \\
\text{addwf} & \quad \text{CCPR1L,F} \quad \text{; “} \\
\text{movlw} & \quad \text{lo_hi} \quad \text{; “} \\
\text{addwfc} & \quad \text{CCPR1H,F} \quad \text{; “} \\
\text{bsf} & \quad \text{IPR1,CCPR1IP} \quad \text{; set CCP1 interrupt to high priority} \\
\text{clrf} & \quad \text{flag} \quad \text{; next delay count set to 1600} \\
\text{movlw} & \quad 0xC0 \\
\text{iorwf} & \quad \text{INTCON,F} \quad \text{; enable interrupt}
\end{align*}
\]

Copyright @ 2005 Thomson Delmar Learning
The PIC18 Microcontroller

bsf PIE1,CCP1IE ; “
forever
nop ; wait for interrupt to occur
bra

hi_ISR
btfss PIR1,CCP1IF ; is the interrupt caused by CCP1?
retfie
bcf PIR1,CCP1IF
btfsc flag,0 ; prepare to add 1600 if flag is 0
goto add_2400
movlw hi_lo ; start a new compare operation
addwf CCPR1L,F ; that will keep CCP1 pin high for 1600
movlw hi_hi ; clock cycles
addwfcc CCP1H,F ; “
btg flag,0 ; “
retfie

add_2400
movlw lo_lo ; start a new compare operation that will
addwf CCPR1L,F ; keep CCP1 pin low for 2400 clock cycles
movlw lo_hi ; “
addwfcc CCP1H,F ; “
btg flag,0 ; toggle the flag
retfie
end
Example 8.9 Assume that there is a PIC18 demo board (e.g., SSE8720) running with a 16-MHz crystal oscillator. Write a function that uses CCP1 in compare mode to create a time delay that is equal to 10 ms multiplied by the contents of PRODL.

Solution:
- Set the Timer3 prescaler to 1
- Use 40000 as the delay count of the compare operation

```assembly
dly_by10ms  movlw 0x81          ; enable Timer3 in 16-bit mode, 1:1 prescaler
          movwf T3CON,A          ; use Timer1 as base times for CCP1
          movlw 0x81              ; enable Timer1 in 16-bit mode with 1:1
          movwf T1CON,A          ; prescaler
          movlw 0x0A              ; configure CCP1 to generate software
          movwf CCP1CON,A        ; interrupt on compare match
          movf TMR1L,W,A          ; to perform a CCP1 compare with
          addlw 0x40              ; 40000 cycles of delay
          movwf CCPR1L,A          ; "
          movlw 0x9C              ; "
          addwf TMR1H,W,A         ; "
          movwf CCPR1H,A          ; "
          bcf PIR1,CCP1IF,A
```
loop  
  btfss PIR1,CCP1IF,A ; wait until 40000 cycles are over
  goto loop
  dcfnsz PRODL,F,A ; is loop count decremented to zero yet?
  return 0 ; delay is over, return
  bcf PIR1,CCP1IF,A ; clear the CCP1IF flag
  movlw 0x40 ; start the next compare operation
  addwf CCPR1L,F,A ; with 40000 cycles delay
  movlw 0x9C ;
  addwf CCPR1H,F,A ;
  goto loop

In C,

```c
void dly_by10ms (unsigned char kk)
{
  CCP1CON = 0x0A; /* configure CCP1 to generate software interrupt */
  T3CON = 0x81; /* enables Timer3 to select Timer1 as base timer */
  T1CON = 0x81; /* enables Timer1 in 16-bit mode with 1:1 as prescaler */
  PIR1bits.CCP1IF = 0;
  while (kk) {
    while (!PIR1bits.CCP1IF);
    PIR1bits.CCP1IF;
    kk--;
    CCPR1 += 40000; }
  return;
}
```
Use CCP Compare to Generate Sound

- A speaker is needed to generate the sound.
- The CCP1 compare mode can be used to generate the sound.

Figure 8.22 Circuit connection for siren generation
Example 8.10 Use the circuit in Figure 8.22 to generate a siren that oscillates between 440 Hz and 880 Hz assuming the PIC18 is running with a 4 MHz crystal oscillator.

Solution: The procedure is as follows:

Step 1. Configure the CCP channel to operate in the compare mode and toggle output on match.
Step 2. Start a compare operation and enable its interrupt with a delay equals to half of the period of the sound of the siren.
Step 3. Wait for certain amount of time (say half of a second). During the waiting period, interrupts will be requested by the CCP compare match many times. The interrupt service routine simply clears the CCP flag and starts the next compare operation and then return.
Step 4. At the end of the delay choose different delay time for the compare operation so the siren sound with different frequency can be generated.
Step 5. Wait for the same amount of time as in Step 3. Again, interrupt caused by CCP compare match will be requested many times.
Step 6. Go to Step 2.
```assembly
#include <p18F452.inc>

hi_hi equ 0x02 ; delay count to create 880 Hz sound
hi_lo equ 0x38
lo_hi equ 0x04 ; delay count to create 440 Hz sound
lo_lo equ 0x70

org 0x00
goto start
org 0x08
goto hi_ISR
org 0x18
retfie

start bcf TRISC,CCP1,A ; configure CCP1 pin for output
movlw 0x81 ; Enable Timer3 for 16-bit mode, use
movwf T3CON,A ; Timer1 as the base timer of CCP1
movwf T1CON,A ; enables Timer1 for 16-bit, prescaler set to 1:1
movlw 0x02 ; CCP1 pin toggle on match
movwf CCP1CON,A ; 
bsf RCON,IPEN ; enable priority interrupt
bsf IPR1,CCP1IP ; configure CCP1 interrupt to high priority
movlw hi_hi ; load delay count for compare operation
```
movwf PRODH ; into PRODH:PRODL register pair
movlw hi_lo ; "
movwf PRODLO ; "
movlw 0xC0
iorwf INTCON,F,A ; set GIE & PIE bits
movf PRODL,W,A ; start a new compare operation with
addwf TMR1L,W,A ; delay stored in PRODH:PRODL
movwf CCPR1L,A ; "
movf PRODH,W,A ; "
addwfcc TMR1H,W,A ; "
movwf CCPR1H,A ; "
bcf PIR1,CCP1IF,A ; clear CCP1IF flag
bsf PIE1,CCP1IE ; enable CCP1 interrupt

forever
call delay_hsec ; stay for half second in one frequency
movlw lo_hi ; switch to different frequency
movw PRODH,A ; "
movlw lo_lo ; "
movwf PRODLO ; "
call delay_hsec ; stay for half second in another frequency
movlw hi_hi ; switch to different frequency
movwf PRODH,A ; "
movlw hi_lo ; "
movwf PRODL,A ;

goto forever

hi_ISR bcf PIR1,CCP1IF,A ; clear the CCP1IF flag
movf PRODL,W,A ; start the next compare operation
addwf CCPR1L,F,A ; using the delay stored in PRODH:PRODL
movf PRODH,W,A ;
addwfcc CCPR1H,F,A ;
retfie

delay_hsec movlw 0x85
movwf TMR0H,A
movlw 0xED
movwf TMR0L,A
movlw 0x83 ; enable TMR0, select instruction clock,
movwf T0CON,A ; prescaler set to 16
bcf INTCON,TMR0IF,A

loopw btfss INTCON,TMR0IF,A
goto loopw ; wait for a half second
return end
**Example 8.11** For the circuit in Figure 8.22, write a program to generate a simple song assuming that $f_{\text{osc}} = 4\text{MHz}$.

**Solution:**
- The example song to be played is a German folk song. Two tables are used by the program:
  1. Table of numbers to be added to CCPR1 register to generate the waveform with the desired frequency.
  2. Table of numbers that select the duration of each note.

```c
#include <p18F452.h>
#define base 3125  /* counter count to create 0.1 s delay */
#define NOTES 38   /* total notes in the song to be played */
#define C4 0x777
#define F4 0x598
#define G4 0x4FC
#define A4 0x470
#define B4 0x3F4
#define C5 0x3BC
#define D5 0x353
#define F5 0x2CC
```
```c
unsigned rom int per_arr[38] = { C4,A4,G4,A4,F4,C4,C4,C5,
                                B4,C5,A4,A4,F4,D5,D5,D5,
                                C5,A4,C5,C5,B4,A4,B4,C5,
                                A4,F4,D5,F5,D5,C5,A4,C5,
                                C5,B4,A4,B4,C5,A4};

unsigned rom char wait[38] = ( 3,5,3,5,3,3,5,3,3,5,3,3,5,3,3,5,3,3,5,3,3,5,3,3,3,2,2,3,3,6,
                                3,2,2,3,3,6};

void  delay (unsigned char xc);
void  high_ISR(void);
void  low_ISR(void);
#pragma code high_vector = 0x08;
void high_interrupt(void)
{
    __asm
    goto high_ISR
    __endasm
```
#pragma code low_vector = 0x18
void low_interrupt (void)
{
    _asm
        goto  low_ISR
    _endasm
}

#pragma interrupt high_ISR
void high_ISR(void)
{
    if (PIR1bits.CCP1IF) {
        PIR1bits.CCP1IF = 0;
        CCPR1 += half_cycle;
    }
}

#pragma interrupt low_ISR
void low_ISR (void)
{
    _asm
        retfie 0
    _endasm
}
void main (void) {
    int i, j;

    TRISCbits.TRISC2 = 0; /* configure CCP1 pin for output */
    T3CON = 0x81; /* enables Timer3 in 16-bit mode, Timer1 for CCP1 time base */
    T1CON = 0x81; /* enable Timer1 in 16-bit mode */
    CCP1CON = 0x02; /* CCP1 compare mode, pin toggle on match */
    IPR1bits.CCP1IP = 1; /* set CCP1 interrupt to high priority */
    PIR1bits.CCP1IF = 0; /* clear CCP1IF flag */
    PIE1bits.CCP1IE = 1; /* enable CCP1 interrupt */
    INTCON |= 0xC0; /* enable high priority interrupt */
    for (j = 0; j < 3; j++) {
        i = 0;
        half_cyc = per_arr[0];
        CCPR1 = TMR1 + half_cyc;
        while (i < NOTES) {
            half_cyc = per_arr[i]; /* get the cycle count for half period of the note */
            delay (wait[i]); /* stay for the duration of the note */
            i++;
        }
    }
}
INTCON &= 0x3F; /* disable interrupt */
delay (5);
delay (6);
INTCON |= 0xC0; /* re-enable interrupt */
}

void delay (unsigned char xc)
{
    switch (xc)
    {
    case 1: /* create 0.1 second delay (sixteenth note) */
        T0CON = 0x84; /* enable TMR0 with prescaler set to 32 */
        TMR0 = 0xFFFF - base; /* set TMR0 to this value so it overflows in 0.1 second */
        INTCONbits.TMR0IF = 0;
        while (!INTCONbits.TMR0IF);
        break;
    } /* The following function runs on a PIC18 demo board running with a 4 MHz crystal oscillator. The parameter xc specifies the amount of delay to be created */
case 2: /* create 0.2 second delay (eighth note) */
    T0CON = 0x84; /* set prescaler to Timer0 to 32 */
    TMR0 = 0xFFFF - 2*base; /* set TMR0 to this value so it overflows in 0.2 second */

    INTCONbits.TMR0IF = 0;
    while (!INTCONbits.TMR0IF);
    break;

case 3: /* create 0.4 seconds delay (quarter note) */
    T0CON = 0x84; /* set prescaler to Timer0 to 32 */
    TMR0 = 0xFFFF - 4*base; /* set TMR0 to this value so it overflows in 0.4 second */

    INTCONbits.TMR0IF = 0;
    while (!INTCONbits.TMR0IF);
    break;

case 4: /* create 0.6 s delay (3 eighths note) */
    T0CON = 0x84; /* set prescaler to Timer0 to 32 */
    TMR0 = 0xFFFF - 6*base; /* set TMR0 to this value so it overflows in 0.6 second */

    INTCONbits.TMR0IF = 0;
    while (!INTCONbits.TMR0IF);
    break;
INTCONbits.TMR0IF = 0;
while (!INTCONbits.TMR0IF);
bake;
case 6: /* create 1.2 second delay (3 quarter note) */
    T0CON = 0x84; /* set prescaler to Timer0 to 32 */
    TMR0 = 0xFFFF - 12*base; /* set TMR0 to this value so it overflows
                           in 1.2 second */
    INTCONbits.TMR0IF = 0;
    while (!INTCONbits.TMR0IF);
bake;
case 7: /* create 1.6 second delay (full note) */
    T0CON = 0x84; /* set prescaler to Timer0 to 32 */
    TMR0 = 0xFFFF - 16*base; /* set TMR0 to this value so it overflows
                           in 1.6 second */
    INTCONbits.TMR0IF = 0;
    while (!INTCONbits.TMR0IF);
bake;
    default:
bake;
break;
}
CCP in PWM Mode

Duty cycle registers

CCPRxL

CCPRxH (slave)

Comparator

TMRy (y = 2 or 4)

Comparator

PRy (y = 2 or 4)

CCPxCON<5:4>
(x = 1, ..., 5)

R

Q

CCPx

S

TRISz<k>
z = C or B or G

Figure 8.24 Simplified PWM block diagram (redraw with permission of Microchip)
The PIC18 Microcontroller

**PWM period** = \([(PRy + 1) \times 4 \times TOSC \times (TMRy\text{ prescale factor})\]

**PWM duty cycle** = \((CCPRxL:CCPxCON<5:4>) \times TOSC \times (TMRy\text{ prescale factor})\)

Procedure for using the PWM module:

**Step 1**  
Set the PWM period by writing to the PRy (y = 2 or 4) register.

**Step 2**  
Set the PWM duty cycle by writing to the CCPRxL register and CC PxCON<5:4> bits.

**Step 3**  
Configure the CCPx pin for output

**Step 4**  
Set the TMRy prescale value and enable Timer by writing to TyCON register

**Step 5**  
Configure CCPx module for PWM operation
Example 8.12 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.

Solution:
Timer setting

1. Use Timer2 as the base timer of CCP1 through CCP5 for PWM mode
2. Enable Timer3 in 16-bit mode with 1:1 prescaler
3. Set Prescaler to Timer2 to 1:4

Period register value is

$$ PR2 = \frac{32 \times 10^6}{4 \div 4 \div 10^4} - 1 = 199 $$

Duty Cycle value

$$ CCPR1L = 200 \times 40\% = 80 $$
Instruction sequence to achieve the previous setting:

movlw 0xC7 ; set period value to 199
movwf PR2,A ;
movlw 0x50 ; set duty cycle value to 80
movwf CCPR1L,A ;
movwf CCPR1H,A ;
bcf TRISC,CCP1,A ; configure CCP1 pin for output
movlw 0x81 ; enable Timer3 in 16-bit mode and use
movwf T3CON,A ; Timer2 as time base for PWM1 thru PWM5
clf TMR2,A ; force TMR2 to count from 0
movlw 0x05 ; enable Timer2 and set its prescaler to 4
movwf T2CON,A ;
movlw 0x0C ; enable CCP1 PWM mode
movwf CCP1CON,A ;
PIC18 Pulse Width Modulation C Library Functions

void ClosePWM1 (void);
void ClosePWM2 (void);
void ClosePWM3 (void);
void ClosePWM4 (void);
void ClosePWM5 (void);
void OpenPWM1 (char period);
void OpenPWM2 (char period);
void OpenPWM3 (char period);
void OpenPWM4 (char period);
void OpenPWM5 (char period);

void SetDCPWM1 (unsigned int dutycycle);
void SetDCPWM2 (unsigned int dutycycle);
void SetDCPWM3 (unsigned int dutycycle);
void SetDCPWM4 (unsigned int dutycycle);
void SetDCPWM5 (unsigned int dutycycle);
Example 8.13 Write a set of C statements to configure CCP4 to generate a digital waveform with 5 KHz frequency and 70% duty cycle assuming that the PIC18F8720 is running with a 16 MHz crystal oscillator. Use Timer4 as the base timer.

Solution:
- Set timer prescaler to 4 and set the period value to be 200
- Duty cycle value to be written is $200 \times 70\% \times 4 = 560$
- The following C statements will configure CCP1 to generate 5 KHz, 70% duty cycle waveform:

```c
TRISGbits.TRISG3 = 0; /* configure CCP4 pin for output */
OpenTimer3(TIMER_INT_OFF & T3_16BIT_RW & T3_SOURCE_INT &
T3_PS_1_1 & T3_SOURCE_CCP & T3_OSC1EN_OFF);
OpenTimer4(TIMER_INT_OFF & T4_PS_1_1 & T4_POST_1_1);
SetDCPWM4 (560);
OpenPWM4(199);
```
PWM Application 1 – Light Dimming

Figure 8.25 Using PWM to control the brightness of a light bulb
Example 8.14 Write a program to dim the lamp in Figure 8.25 to 10% of its brightness in 5 seconds assuming that the PIC18 is running with a 32 MHz crystal oscillator.

Solution:
- Set duty cycle to 100% initially. Load 99 and 400 as the initial period and duty cycle register values.
- Dim the lamp by 10% in the first second by reducing the brightness in 10 steps.
- Dim the lamp down to 10% brightness in the next 4 seconds in 40 steps.

```c
#include <p18F452.inc>
org 0x00
org 0x00
goto start
org 0x08
retfie
org 0x18
retfie

start bcf TRISC,CCP1,A ; configure CCP1 pin for output
movlw 0x81 ; Use Timer2 as the base timer for PWM1
movwf T3CON ; and enable Timer3 in 16-bit mode
movlw 0x63 ; set 100 as the period of the digital
movwf PR2,A ; waveform
```
The PIC18 Microcontroller

movlw 0x64 ; set 100 as the duty cycle
movwf CCPR1L,A ; 
movwf CCPR1H,A ; 
movlw 0x05 ; enable Timer2 and set its prescaler to 4
movwf T2CON,A ; 
movlw 0x0C ; enable PWM1 operation and set the lowest
two bits of duty cycle to 0
movlw 0x0A ; use PRODL as the loop count
movwf CCP1CON,A ; 
movwf PRODL,A ; 

loop_1s call delay ; call "delay" to delay for 100 ms
decf CCPR1L,F,A ; decrement the duty cycle value by 1
decfcsz PRODL,F,A ; check to see if loop index expired
goto loop_1s
movlw 0x28 ; repeat the next loop 40 times
movwf PRODL,A ; 

loop_4s call delay ; call "delay" to delay for 100 ms
decf CCPR1L,F,A ; decrement duty cycle value by 2
decf CCPR1L,F,A ; 
decfcsz PRODL,F,A ; is loop index expired?
goto loop_4s

forever nop
bra forever
Lamp Dimming C Program

```c
#include <p18F452.h>
#include <pwm.h>
#include <timers.h>

void delay (void);

void main (void)
{
    int i;
    TRISCbits.TRISC2 = 0;  /* configure CCP1 pin for output */
    T3CON = 0x81;  /* use Timer2 as base timer for CCP1 */
    OpenTimer2 (TIMER_INT_OFF & T2_PS_1_4 & T2_POST_1_1);
    SetDCPWM1 (400);  /* set duty cycle to 100% */
    OpenPWM1 (99);  /* enable PWM1 with period equals 100 */
    for(i = 0; i < 10; i++) {
        delay();
        CCPR1L --;  /* decrement duty cycle value by 1 */
    }
    for(i = 0; i < 40; i++) {
        delay();
        CCPR1L -= 2;  /* decrement duty cycle value by 2 */
    }
}
```

---

The PIC18 Microcontroller

H. Huang Transparency No.8-80

Copyright @ 2005 Thomson Delmar Learning
**DC Motor Control**

- DC motor speed is regulated by controlling its average driving voltage. The higher the voltage, the faster the motor rotates.
- Changing motor direction can be achieved by reversing the driving voltage.
- Motor braking can be performed by reversing the driving voltage for certain length of time.
- Most PIC18 devices have PWM functions that can be used to drive DC motors.
- Many DC motors operate with 5 V supply.
- DC motors require large amount of current to operate. Special driver circuits are needed for this purpose.
- A simplified DC motor control circuit is shown in Figure 8.26.
DC Motor Driver L293

(a) Pin Assignment

(b) Motor connection

Figure 8.27 Motor driver L293 pin assignment and motor connection
L293 Motor Driver
- The L293 has four channels and can deliver up to 1 A of current per channel.
- The L293 has separate logic supply and takes a logic input to enable or disable each channel.
- Clamping diodes are provided to drain the kickback current generated from the inductive load during the motor reversal.

Feedback
- DC motor needs the speed information to adjust the voltage output to the motor driver circuit.
- A sensing device such as Hall-effect transistor can be used to measure the motor speed.
- The Hall-effect transistor can be mounted on the shaft (rotor) of a DC motor and magnets are mounted on the armature (stator).
- The attachment of the Hall-effect transistors is shown in Figure 8.28.
- Each time the magnet passes by the Hall-effect transistor, a pulse is generated.
- CCP capture mode can be used to measure the motor speed.
- The schematic of a motor-control system is illustrated in Figure 8.29.
Magnets

Hall-effect transistor

$T$ is the time for one revolution

Figure 8.28 The output waveform of the Hall-effect transistor
All diodes are the same and could be any one of the 1N4000 series.

Figure 8.29 Schematic of a PIC18-based motor-control system
- Pin 2 and pin 7 drives the two terminals of the DC motor.
- Depending on the voltages applied to pin 2 and pin 7, the motor can be rotating in clockwise or counterclockwise direction as shown in Figure 8.30.

Figure 8.30 The L293 motor driver
**Electrical Braking**
- Once a DC motor is running, it picks up speed.
- Turning off the voltage to the motor does not stop the motor immediately.
- The momentum of the DC motor will keep the motor running after the voltage it turned off.
- An abrupt stop may be achieved by reversing the voltage applied to the motor.
- The duration of braking needs to be precisely measured.
Example 8.15 For the circuit shown in Figure 8.29, write a function in C language to measure the motor speed (in RPM) assuming that the PIC18 MCU is running with a 20 MHz crystal oscillator.

Solution:
- Motor speed can be measured by capturing two consecutive rising or falling edges.
- Let \textbf{diff} and \textbf{f} are the difference of two captured edges and the Timer1 frequency.

\[
\text{Speed} = 60 \times \frac{f}{(2 \times \text{diff})}
\]
unsigned int motor_speed(void)
{
    unsigned int edge1, diff, rpm;
    long unsigned temp;

    T3CON = 0x81; /* enables Timer3 in 16-bit mode and use Timer1 and Timer2 for CCP1 thru CCP2 operations */
    OpenTimer1(TIMER_INT_OFF & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_4); /* set Timer1 prescaler to 1:4 */
    PIR2bits.CCP2IF = 0;
    OpenCapture2(CAPTURE_INT_OFF & C2_EVERY_RISE_EDGE);
    while (!PIR2bits.CCP2IF);
    edge1 = CCPR2; /* save the first rising edge */
    PIR2bits.CCP2IF = 0;
    while (!PIR2bits.CCP2IF);
    CloseCapture2();
    diff = CCPR2 - edge1; /* compute the difference of two rising consecutive edges */
    temp = 125000ul/(2 * diff);
    rpm = temp * 60;
    return rpm;
}
Example 8.16 Write a subroutine to perform the electrical braking.

Solution: Electrical braking is implemented by setting the duty cycle to 0% and setting the voltage on the RB4 pin to high for certain amount of time. The braking program is as follows:

```
brake       bsf PORTB, RB4, A       ; reverse the applied voltage to motor
            movlw 0x00
            call brake_time
            bcf PORTB, RB4, A       ; stop braking
```

movlw
movwf
ccpr1l
brake_time
bcf
return