Section 14. Compare/Capture/PWM (CCP)

HIGHLIGHTS

This section of the manual contains the following major topics:

14.1 Introduction

Each CCP (Capture/Compare/PWM) module contains a 16-bit register which can operate as a 16-bit capture register, as a 16-bit compare register or as a 10-bit PWM master/slave Duty Cycle register. The CCP modules are identical in operation, with the exception of the operation of the special event trigger.

Each CCP module has 3 registers. Multiple CCP modules may exist on a single device. Throughout this section we use generic names for the CCP registers. These generic names are shown in Table 14-1.

Table 14-1: Specific to Generic CCP Nomenclature

Generic Name	CCP1	CCP2	Comment
CCPxCON	CCP1CON	CCP2CON	CCP control register
CCPRxH	CCPR1H	CCPR2H	CCP High byte
CCPRxL	CCPR1L	CCPR2L	CCP Low byte
CCPx	CCP1	CCP2	CCP pin

Table 14-2 shows the resources of the CCP modules, in each of its modes. While Table 14-3 shows the interactions between the CCP modules, where CCPx is one CCP module and CCPy is another CCP module.

Table 14-2: CCP Mode - Timer Resource

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

Table 14-3: Interaction of Two CCP Modules

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same TMR1 time-base.
Capture	Compare	The compare should be configured for the special event trigger, which clears TMR1.
Compare	Compare	The compare(s) should be configured for the special event trigger, which clears TMR1.
PWM	PWM	The PWMs will have the same frequency, and update rate (TMR2 interrupt).
PWM	Capture	None
PWM	Compare	None

14.2 Control Register

Register 14-1: CCPxCON Register

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

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

bit 5:4 **DCxB1:DCxB0**: PWM Duty Cycle bit1 and bit0

Capture Mode:
Unused

Compare Mode:
Unused

PWM Mode:

These bits are the two LSbs (bit1 and bit0) of the 10-bit PWM duty cycle. The upper eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

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

`0000` = Capture/Compare/PWM off (resets CCPx module)
`0100` = Capture mode, every falling edge
`0101` = Capture mode, every rising edge
`0110` = Capture mode, every 4th rising edge
`0111` = Capture mode, every 16th rising edge
`1000` = Compare mode,
Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)
`1001` = Compare mode,
Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set)
`1010` = Compare mode,
Generate software interrupt on compare match
(CCPIF bit is set, CCP pin is unaffected)
`1011` = Compare mode,
Trigger special event (CCPIF bit is set)
`11xx` = PWM mode

Legend
R = Readable bit W = Writable bit
U = Unimplemented bit, read as ‘0’ - n = Value at POR reset

14.3 Capture Mode

In Capture mode, CCPRxH:CCPRxL captures the 16-bit value of the TMR1 register when an event occurs on pin CCPx. An event is defined as:

- Every falling edge
- Every rising edge
- Every 4th rising edge
- Every 16th rising edge

An event is selected by control bits CCPxM3:CCPxM0 (CCPxCON<3:0>). When a capture is made, the interrupt request flag bit, CCPxIF, is set. The CCPxIF bit must be cleared in software. If another capture occurs before the value in register CCPRx is read, the previous captured value will be lost.

Note: Timer1 must be running in timer mode or synchronized counter mode for the CCP module to use the capture feature. In asynchronous counter mode, the capture operation may not work.

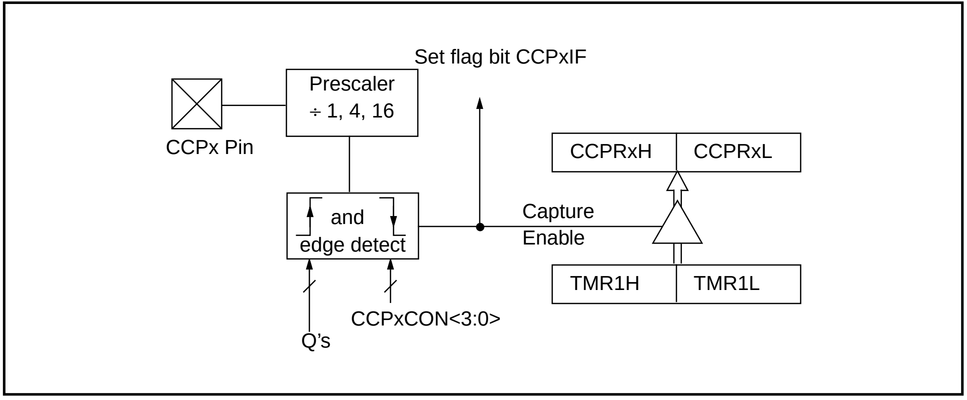
As can be seen in Figure 14-1, a capture does not reset the 16-bit TMR1 register. This is so Timer1 can also be used as the timebase for other operations. The time between two captures can easily be computed as the difference between the value of the second capture that of the first capture. When Timer1 overflows, the TMR1IF bit will be set and if enabled an interrupt will occur, allowing the time base to be extended to greater than 16-bits.

14.3.1 CCP Pin Configuration

In Capture mode, the CCPx pin should be configured as an input by setting its corresponding TRIS bit.

Note: If the CCPx pin is configured as an output, a write to the port can cause a capture condition.

Figure 14-1: Capture Mode Operation Block Diagram



The prescaler can be used to get a very fine average resolution on a constant input frequency. For example, if we have a stable input frequency and we set the prescaler to 1:16, then the total error for those 16 periods is 1 T_{CY}. This gives an effective resolution of $T_{CY}/16$, which at 20 MHz is 12.5 ns. This technique is only valid where the input frequency is “stable” over the 16 samples. Without using the prescaler (1:1), each sample would have a resolution of T_{CY}.

14.3.2 Changing Between Capture Modes

When the Capture mode is changed, a capture interrupt may be generated. The user should keep the CCPxIE bit clear to disable these interrupts and should clear the CCPxIF flag bit following any such change in operating mode.

14.3.2.1 CCP Prescaler

There are four prescaler settings, specified by bits CCPxM3:CCPxM0. Whenever the CCP module is turned off, or the CCP module is not in capture mode, the prescaler counter is cleared. This means that any reset will clear the prescaler counter.

Switching from one capture prescale setting to another may generate an interrupt. Also, the prescaler counter will not be cleared, therefore the first capture may be from a nonzero prescaler. Example 14-1 shows the recommended method for switching between capture prescale settings. This example also clears the prescaler counter and will not generate the interrupt.

Example 14-1: Changing Between Capture Prescalers

```
CLRF    CCP1CON        ; Turn CCP module off
MOVLW   NEW_CAPT_PS    ; Load the W reg with the new prescaler
                       ;   mode value and CCP ON
MOVWF   CCP1CON        ; Load CCP1CON with this value
```

To clear the Capture prescaler count, the CCP module must be configured into any non-capture CCP mode (Compare, PWM, or CCP off modes).

14.3.3 Sleep Operation

When the device is placed in sleep, Timer1 will not increment (since it is in synchronous mode), but the prescaler will continue to count events (not synchronized). When a specified capture event occurs, the CCPxIF bit will be set, but the capture register will not be updated. If the CCP interrupt is enabled, the device will wake-up from sleep. The value in the 16-bit TMR1 register is **not** transferred to the 16-bit capture register, but since the timer was not incrementing, this value should not have any meaning. Effectively, this allows the CCP pin to be used as another external interrupt.

14.3.4 Effects of a Reset

The CCP module is off, and the value in the capture prescaler is forced to 0.

14.4 Compare Mode

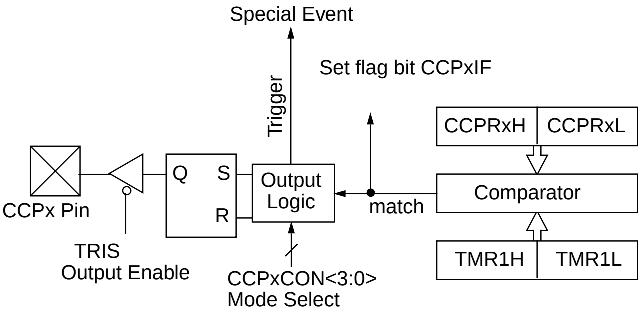
In Compare mode, the 16-bit CCPRx register value is constantly compared against the TMR1 register pair value. When a match occurs, the CCPx pin is:

- Driven High
- Driven Low
- Remains Unchanged

The action on the pin is based on the value of control bits CCPxM3:CCPxM0 (CCPxCON<3:0>). At the same time, a compare interrupt is also generated.

> **Note:** Timer1 must be running in Timer mode or Synchronized Counter mode if the CCP module is using the compare feature. In Asynchronous Counter mode, the compare operation may not work.

Figure 14-2: Compare Mode Operation Block Diagram



14.4.1 CCP Pin Operation in Compare Mode

The user must configure the CCPx pin as an output by clearing the appropriate TRIS bit.

Note:	Clearing the CCPxCON register will force the CCPx compare output latch to the default low level. This is not the Port I/O data latch.

Selecting the compare output mode, forces the state of the CCP pin to the state that is opposite of the match state. So if the Compare mode is selected to force the output pin low on match, then the output will be forced high until the match occurs (or the mode is changed).

14.4.2 Software Interrupt Mode

When generate Software Interrupt mode is chosen, the CCPx pin is not affected. Only a CCP interrupt is generated (if enabled).

14.4.3 Special Event Trigger

In this mode, an internal hardware trigger is generated which may be used to initiate an action.

The special event trigger output of CCPx resets the TMR1 register pair. This allows the CCPRx register to effectively be a 16-bit programmable period register for Timer1.

For some devices, the special trigger output of the CCP module resets the TMR1 register pair, and starts an A/D conversion (if the A/D module is enabled).

Note:	The special event trigger will not set the Timer1 interrupt flag bit, TMR1IF.

14.4.4 Sleep Operation

When the device is placed in sleep, Timer1 will not increment (since in synchronous mode), and the state of the module will not change. If the CCP pin is driving a value, it will continue to drive that value. When the device wakes-up, it will continue form this state.

14.4.5 Effects of a Reset

The CCP module is off.

14.5 PWM Mode

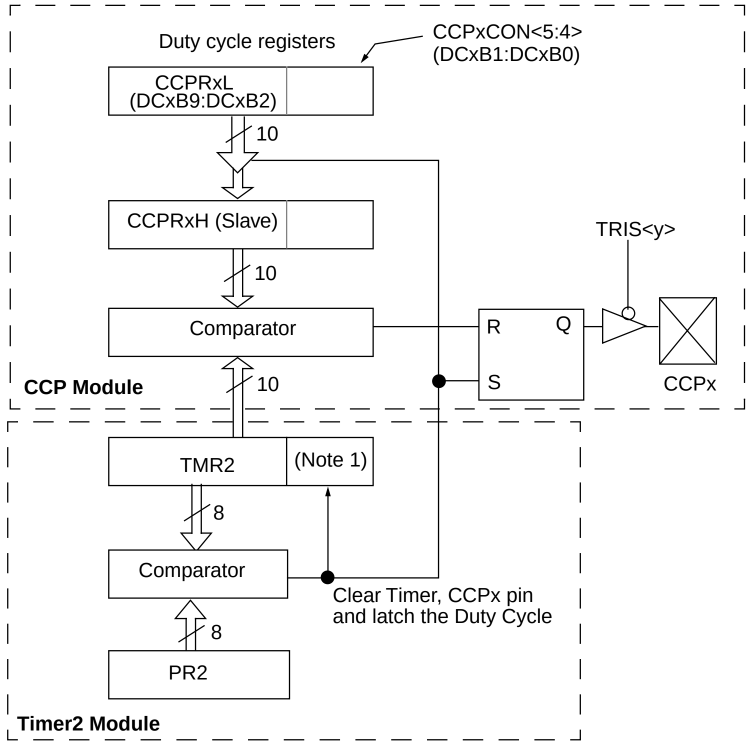
In Pulse Width Modulation (PWM) mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCPx pin is multiplexed with the PORT data latch, the corresponding TRIS bit must be cleared to make the CCPx pin an output.

> **Note:** Clearing the CCPxCON register will force the CCPx PWM output latch to the default low level. This is not the port I/O data latch.

Figure 14-3 shows a simplified block diagram of the CCP module in PWM mode.

For a step by step procedure on how to set up the CCP module for PWM operation, see Subsection **14.5.3 "Set-up for PWM Operation."**

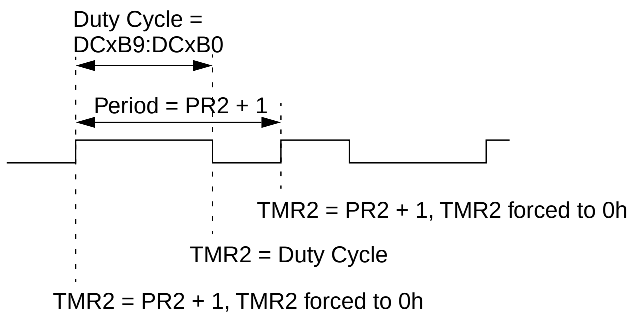
Figure 14-3: Simplified PWM Block Diagram



Note 1: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time-base.

A PWM output (Figure 14-4) has a time-base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

Figure 14-4: PWM Output


Duty Cycle =
DCxB9:DCxB0
Period = PR2 + 1
TMR2 = PR2 + 1, TMR2 forced to 0h
TMR2 = Duty Cycle
TMR2 = PR2 + 1, TMR2 forced to 0h

14.5.1 PWM Period

The PWM period is specified by writing to the PR2 register. The PWM period can be calculated using the following formula:

$$\text{PWM period} = [(PR2) + 1] \bullet 4 \bullet T_{OSC} \bullet (\text{TMR2 prescale value}), \text{ specified in units of time}$$

PWM frequency (F_{PWM}) is defined as 1 / [PWM period].

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCPx pin is set (exception: if PWM duty cycle = 0%, the CCPx pin will not be set)
- The PWM duty cycle is latched from CCPRxL into CCPRxH

Note: The Timer2 postscaler is not used in the determination of the PWM frequency. The postscaler could be used to have a servo update rate at a different frequency than the PWM output.

14.5.2 PWM Duty Cycle

The PWM duty cycle is specified by writing to the CCPRxL register and to the DCxB1:DCxB0 (CCPxCON<5:4>) bits. Up to 10-bit resolution is available: the CCPRxL contains the eight MSbs and CCPxCON<5:4> contains the two LSbs. This 10-bit value is represented by DCxB9:DCxB0. The following equation is used to calculate the PWM duty cycle:

$$\text{PWM duty cycle} = (\text{DCxB9:DCxB0 bits value}) \bullet \text{Tosc} \bullet (\text{TMR2 prescale value}), \text{ in units of time}$$

The DCxB9:DCxB0 bits can be written to at any time, but the duty cycle value is not latched into CCPRxH until after a match between PR2 and TMR2 occurs (which is the end of the current period). In PWM mode, CCPRxH is a read-only register.

The CCPRxH register and a 2-bit internal latch are used to double buffer the PWM duty cycle. This double buffering is essential for glitchless PWM operation.

When CCPRxH and 2-bit latch match the value of TMR2 concatenated with the internal 2-bit Q clock (or two bits of the TMR2 prescaler), the CCPx pin is cleared. This is the end of the duty cycle.

Maximum PWM resolution (bits) for a given PWM frequency:

$$= \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \quad \text{bits}$$

Note: If the PWM duty cycle value is longer than the PWM period, the CCPx pin will not be cleared. This allows a duty cycle of 100%.

14.5.2.2 Minimum Resolution

The minimum resolution (in time) of each bit of the PWM duty cycle depends on the prescaler of Timer2.

Table 14-4: Minimum Duty Cycle Bit Time

Prescaler Value	T2CKPS1:T2CKPS0	Minimum Resolution (Time)
1	0 0	TOSC
4	0 1	TCY
16	1 x	4 TCY

Example 14-2: PWM Period and Duty Cycle Calculation

Desired PWM frequency is 78.125 kHz,
Fosc = 20 MHz
TMR2 prescale = 1

$$1/78.125\text{ kHz} = [(\text{PR2}) + 1] \bullet 4 \bullet 1/20\text{ MHz} \bullet 1$$

$$12.8\ \mu\text{s} = [(\text{PR2}) + 1] \bullet 4 \bullet 50\text{ ns} \bullet 1$$

$$\text{PR2} = 63$$

Find the maximum resolution of the duty cycle that can be used with a 78.125 kHz frequency and 20 MHz oscillator:

$$1/78.125\text{ kHz} = 2^{\text{PWM RESOLUTION}} \bullet 1/20\text{ MHz} \bullet 1$$

$$12.8\ \mu\text{s} = 2^{\text{PWM RESOLUTION}} \bullet 50\text{ ns} \bullet 1$$

$$256 = 2^{\text{PWM RESOLUTION}}$$

$$\log(256) = (\text{PWM Resolution}) \bullet \log(2)$$

$$8.0 = \text{PWM Resolution}$$

At most, an 8-bit resolution duty cycle can be obtained from a 78.125 kHz frequency and a 20 MHz oscillator, i.e., 0 ≤ DCxB9:DCxB0 ≤ 255. Any value greater than 255 will result in a 100% duty cycle.

In order to achieve higher resolution, the PWM frequency must be decreased. In order to achieve higher PWM frequency, the resolution must be decreased.

Table 14-5 lists example PWM frequencies and resolutions for Fosc = 20 MHz. The TMR2 prescaler and PR2 values are also shown.

Table 14-5: Example PWM Frequencies and Bit Resolutions at 20 MHz

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	5.5

14.5.3 Set-up for PWM Operation

The following steps configure the CCP module for PWM operation:

1. Establish the PWM period by writing to the PR2 register.
2. Establish the PWM duty cycle by writing to the DCxB9:DCxB0 bits.
3. Make the CCPx pin an output by clearing the appropriate TRIS bit.
4. Establish the TMR2 prescale value and enable Timer2 by writing to T2CON.
5. Configure the CCP module for PWM operation.

14.5.4 Sleep Operation

When the device is placed in sleep, Timer2 will not increment, and the state of the module will not change. If the CCP pin is driving a value, it will continue to drive that value. When the device wakes-up, it will continue from this state.

14.5.5 Effects of a Reset

The CCP module is off.

14.6 Initialization

The CCP module has three modes of operation. Example 14-3 shows the initialization of capture mode, Example 14-4 shows the initialization of compare mode, and Example 14-5 shows the initialization of PWM mode.

Example 14-3: Capture Initialization

```
    CLRF   CCP1CON       ; CCP Module is off
    CLRF   TMR1H         ; Clear Timer1 High byte
    CLRF   TMR1L         ; Clear Timer1 Low byte
    CLRF   INTCON        ; Disable interrupts and clear T0IF
    BSF    STATUS, RP0   ; Bank1
    BSF    TRISC, CCP1   ; Make CCP pin input
    CLRF   PIE1          ; Disable peripheral interrupts
    BCF    STATUS, RP0   ; Bank0
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x06          ; Capture mode, every 4th rising edge
    MOVWF  CCP1CON       ;
    BSF    T1CON, TMR1ON ; Timer1 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the CCP Interrupt flag bit
;
Capture_Event
    BTFSS  PIR1, CCP1IF
    GOTO   Capture_Event
;
; Capture has occurred
;
    BCF    PIR1, CCP1IF  ; This needs to be done before next compare
```

Example 14-4: Compare Initialization

```
    CLRF   CCP1CON       ; CCP Module is off
    CLRF   TMR1H         ; Clear Timer1 High byte
    CLRF   TMR1L         ; Clear Timer1 Low byte
    CLRF   INTCON        ; Disable interrupts and clear T0IF
    BSF    STATUS, RP0   ; Bank1
    BCF    TRISC, CCP1   ; Make CCP pin output if controlling state of pin
    CLRF   PIE1          ; Disable peripheral interrupts
    BCF    STATUS, RP0   ; Bank0
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x08          ; Compare mode, set CCP1 pin on match
    MOVWF  CCP1CON       ;
    BSF    T1CON, TMR1ON ; Timer1 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the CCP Interrupt flag bit
;
Compare_Event
    BTFSS  PIR1, CCP1IF
    GOTO   Compare_Event
;
; Compare has occurred
;
    BCF    PIR1, CCP1IF   ; This needs to be done before next compare
```

Example 14-5: PWM Initialization

```
    CLRF   CCP1CON       ; CCP Module is off
    CLRF   TMR2          ; Clear Timer2
    MOVLW  0x7F          ;
    MOVWF  PR2           ;
    MOVLW  0x1F          ;
    MOVWF  CCPR1L        ; Duty Cycle is 25% of PWM Period
    CLRF   INTCON        ; Disable interrupts and clear T0IF
    BSF    STATUS, RP0   ; Bank1
    BCF    TRISC, PWM1   ; Make pin output
    CLRF   PIE1          ; Disable peripheral interrupts
    BCF    STATUS, RP0   ; Bank0
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x2C          ; PWM mode, 2 LSbs of Duty cycle = 10
    MOVWF  CCP1CON       ;
    BSF    T2CON, TMR2ON ; Timer2 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the TMR2 Interrupt flag bit
;
PWM_Period_Match
    BTFSS  PIR1, TMR2IF
    GOTO   PWM_Period_Match
;
; Update this PWM period and the following PWM Duty cycle
;
    BCF    PIR1, TMR2IF
```

14.7 Design Tips

Question 1: ***What timers can I use for the capture and compare modes?***

Answer 1:

The capture and compare modes are designed around Timer1, so no other timer can be used for these functions. This also means that if multiple CCP modules (in parts with more than one) are being used for a capture or compare function, they will share the same timer.

Question 2: ***What timers can I use with the PWM mode?***

Answer 2:

The PWM mode is designed around Timer2, so no other timer can be used for this function. (It is the only timer with a period register associated with it.) If multiple CCP modules (in parts with more than one) are doing PWM they will share the same timer, meaning they will have the same PWM period and frequency.

Question 3: ***Can I use one CCP module to do capture (or compare) AND PWM at the same time, since they use different timers as their reference?***

Answer 3:

The timers may be different, but other logic functions are shared. However you can switch from one mode to the other. For a device with two CCP modules, you can also have CCP1 set up for PWM and CCP2 set up for capture or compare (or vice versa) since they are two independent modules.

Question 4: ***How does a reset affect the CCP module?***

Answer 4:

Any reset will turn the CCP module off. See the section on resets to see reset values.

Question 5: ***I am setting up the CCP1CON module for "Compare Mode, trigger special event" (1011) which resets TMR1. When a compare match occurs, will I have both the TMR1 and the CCP1 interrupts pending (TMR1IF is set, CCP1IF is set)?***

Answer 5:

The CCP1IF flag will be set on the match condition. TMR1IF is set when Timer1 overflows, and the special trigger reset of Timer1 is not considered an overflow. However, if both the CCPR1L and CCPR1H registers are set at FFh, then an overflow occurs at the same time as the match, which will then set both CCP1IF and TMR1IF.

Question 6: ***How do I use Timer2 as a general purpose timer, with an interrupt flag on rollover?***

Answer 6:

Timer2 always resets to zero when it equals PR2 and flag bit TMR2IF always gets set at this time. By putting FFh into PR2, you will get an interrupt on overflow at FFh, as you would with Timer0, for instance. Quite often it is desirable to have an event occur at a periodic rate, perhaps an interrupt driven event. Normally an initial value would be placed into the timer so that the overflow will occur at the desired time. This value would have to be placed back into the timer every time it overflowed to make the interrupts occur at the same desired rate. The benefit of Timer2 is that a value can be written to PR2 that will cause it to reset at your desired time interval. This means you do not have the housekeeping chore of reloading the timer every time it overflows, since PR2 maintains its value.

Question 7: ***I am using a CCP module in PWM mode. The duty cycle being output is almost always 100%, even when my program writes a value like 7Fh to the duty cycle register, which should be 50%. What am I doing wrong?***

Answer 7:

1. The value in CCPRxL is higher than PR2. This happens quite often when a user desires a fast PWM output frequency and will write a small value in the PR2. In this case, if a value of 7Eh were written to PR2, then a value 7Fh in CCPRxL will result in 100% duty cycle.
2. If the TRIS bit corresponding to the CCP output pin you are using is configured as an input, the PWM output cannot drive the pin. In this case the pin would float and duty cycle may appear to be 0%, 100% or some other floating value.

Question 8: ***I want to determine a signal frequency using the CCP module in capture mode to find the period. I am currently resetting Timer1 on the first edge, then using the value in the capture register on the second edge as the time period. The problem is that my code to clear the timer does not occur until almost twelve instructions after the first capture edge (interrupt latency plus saving of registers in interrupt) so I cannot measure very fast frequencies. Is there a better way to do this?***

Answer 8:

You do not need to zero the counter to find the difference between two pulse edges. Just take the first captured value and put it into another set of registers. Then when the second capture event occurs, just subtract the first event from the second. Assuming that your pulse edges are not so far apart that the counter can wrap around past the last capture value, the answer will always be correct. This is illustrated by the following example:

1. First captured value is FFFEh. Store this value in two registers.
2. The second capture value is 0001h (the counter has incremented three times).
3. 0001h - FFFEh = 0003, which is the same as if you had cleared Timer1 to zero and let it count to 3. (Theoretically, except that there was a delay getting to the code that clears Timer1, so actual values would differ).

The interrupt overhead is now less important because the values are captured automatically. For even faster inputs do not enable interrupts and just test the flag bit in a loop. If you must also capture very long time periods, such that the timer can wrap around past the previous capture value, then consider using an auto-scaling technique that starts with a large prescale and shorten the prescale as you converge on the exact frequency.

14.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Mid-Range MCU family (that is they may be written for the Base-Line, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the CCP modules are:

Title	**Application Note #**
Using the CCP Modules	AN594
Implementing Ultrasonic Ranging	AN597
Air Flow Control Using Fuzzy Logic	AN600
Adaptive Differential Pulse Code Modulation	AN643

14.9 Revision History

Revision A

This is the initial released revision of the CCP module description.