

# Timers

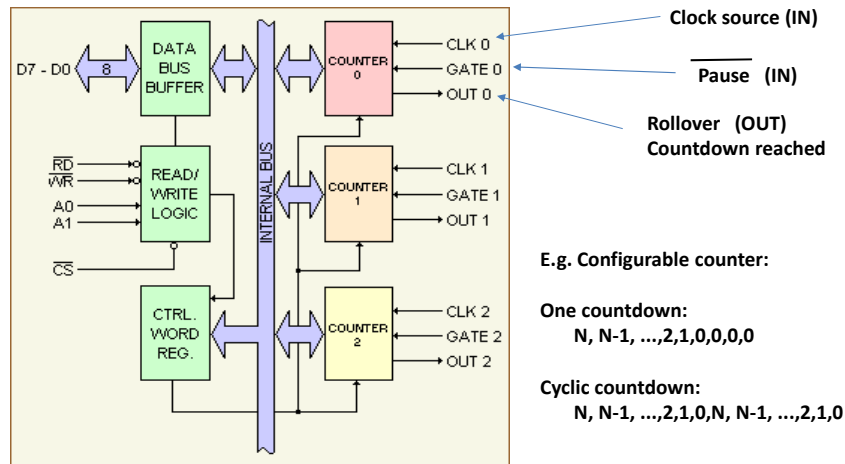
## CCP Module

Dpt. Enginyeria de Sistemes, Automàtica i Informàtica Industrial

### 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

## Generic Timers



## The PIC18 Timer System

- A PIC18F45K22 microcontroller has to 7 timers: Timer 0...Timer 6.
- Timer 0, Timer 1, Timer 3 and Timer 5 are 16-bit timers whereas Timer 2, Timer 4 and Timer 6 are 8-bit.
- When a timer rolls over, an interrupt may be generated if it is enabled.
- Timer 2, Timer 4 and Timer 6 use instruction cycle clock as the clock source whereas the other timers may also use external clock input as the clock source.
- Timer 0 is designed to act as a time base (core interrupt) while the other timers are in the peripheral group (alternate time base and CCP operation).

## The PIC18 Timer0

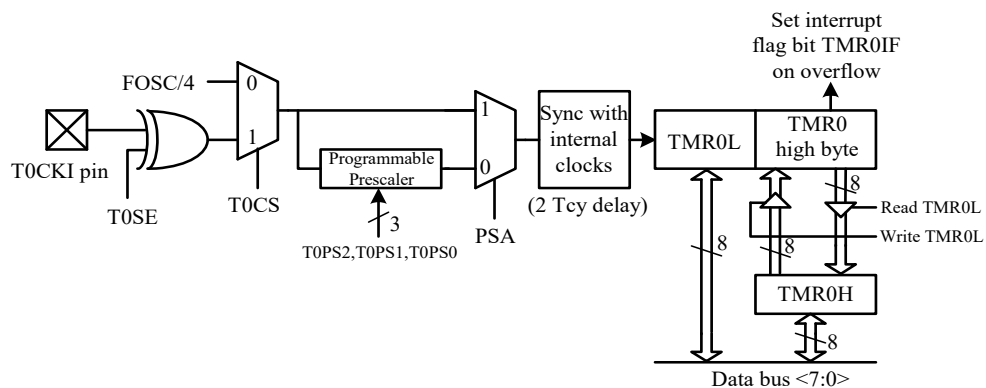


Figure 8.1b Timer0 block diagram in 16-bit mode (redraw with permission of Microchip)

## T0CON Register

	7	6	5	4	3	2	1	0
value after reset	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
	1	1	1	1	1	1	1	1

**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

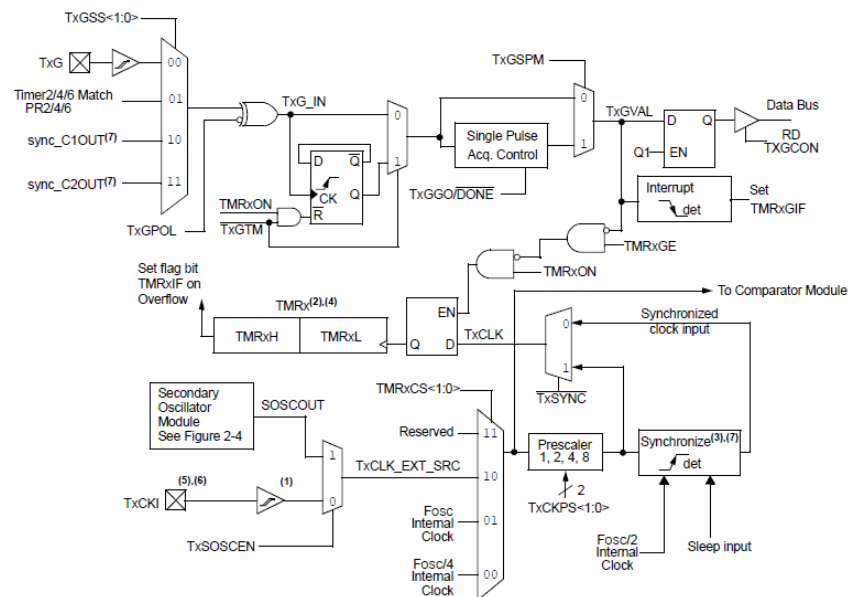
111 = 1:256 prescaler value

Figure 8.2 T0CON register (reprint with permission of Microchip)

## The PIC18 Timer1/3/5

- Is a 16-bit timer/counter depending upon the clock source.
- An interrupt may be requested when TimerX rolls over from 0xFFFF to 0x0000.
- TimerX operation is controlled by the TxCON and TxGCON register.
- These timers have a number of frequency input sources and a complex gate enable circuitry (if TMRxGE=1)
- These timers can be used to create time delays and measure the frequency of an unknown signal (using the CCP modules).

## The PIC18 Timer1/3/5





## TxGCON Register

**REGISTER 12-2: TXGCON: TIMER1/3/5 GATE CONTROL REGISTER**

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W/HC-0/u	R-x/x	R/W-0/u	R/W-0/u
TMRxGE	TxGPOL	TxGTM	TxGSPM	TxGGO/DONE	TxGVAL	TxGSS<1:0>	
bit 7							bit 0

**Legend:**

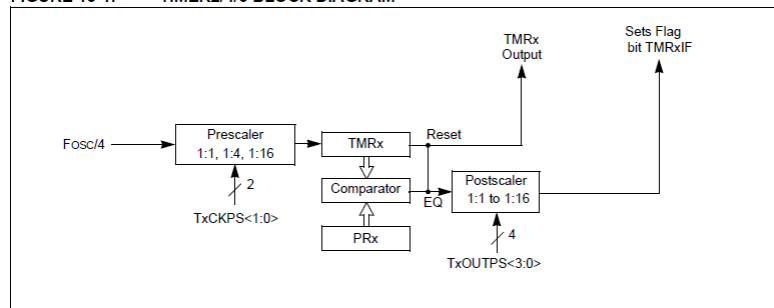
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/h = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	HC = Bit is cleared by hardware

bit 7	<b>TMRxGE:</b> Timer1/3/5 Gate Enable bit If TMRxON = 0: This bit is ignored If TMRxON = 1: 1 = Timer1/3/5 counting is controlled by the Timer1/3/5 gate function 0 = Timer1/3/5 counts regardless of Timer1/3/5 gate function	← TMRxGE=0 !!!
bit 6	<b>TxGPOL:</b> Timer1/3/5 Gate Polarity bit 1 = Timer1/3/5 gate is active-high (Timer1/3/5 counts when gate is high) 0 = Timer1/3/5 gate is active-low (Timer1/3/5 counts when gate is low)	
bit 5	<b>TxGTM:</b> Timer1/3/5 Gate Toggle Mode bit 1 = Timer1/3/5 Gate Toggle mode is enabled 0 = Timer1/3/5 Gate Toggle mode is disabled and toggle flip-flop is cleared Timer1/3/5 gate flip-flop toggles on every rising edge.	
bit 4	<b>TxGSPM:</b> Timer1/3/5 Gate Single-Pulse Mode bit 1 = Timer1/3/5 gate Single-Pulse mode is enabled and is controlling Timer1/3/5 gate 0 = Timer1/3/5 gate Single-Pulse mode is disabled	
bit 3	<b>TxGGO/DONE:</b> Timer1/3/5 Gate Single-Pulse Acquisition Status bit 1 = Timer1/3/5 gate single-pulse acquisition is ready, waiting for an edge 0 = Timer1/3/5 gate single-pulse acquisition has completed or has not been started This bit is automatically cleared when TxGSPM is cleared.	
bit 2	<b>TxGVAL:</b> Timer1/3/5 Gate Current State bit Indicates the current state of the Timer1/3/5 gate that could be provided to TMRxH:TMRxL. Unaffected by Timer1/3/5 Gate Enable (TMRxGE).	
bit 1-0	<b>TxGSS&lt;1:0&gt;:</b> Timer1/3/5 Gate Source Select bits 00 = Timer1/3/5 Gate pin 01 = Timer2/4/6 Match PR2/4/6 output (See Table 12-5 for proper timer match selection) 10 = Comparator 1 optionally synchronized output (sync_C1OUT) 11 = Comparator 2 optionally synchronized output (sync_C2OUT)	

## The PIC18 Timer2/4/6

- There are 3 8 bit timer/counters with clock source Finstr (Fosc/4) and prescaler and postscaler block logic.
- Controlled by TxCON and related to PRx registers. Can generate interrupts via TMRxIF.
- They can be a source for PWM signals (CCP modules).

**FIGURE 13-1: TIMER2/4/6 BLOCK DIAGRAM**



## T2CON Register

REGISTER 13-1: TxCON: TIMER2/TIMER4/TIMER6 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TxOUTPS<3:0>			TMRxON	TxCKPS<1:0>	
bit 7						bit 0

**Legend:**

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

bit 7	<b>Unimplemented:</b> Read as '0'
bit 6-3	<b>TxOUTPS&lt;3:0&gt;:</b> TimerX Output Postscaler Select bits
	0000 = 1:1 Postscaler
	0001 = 1:2 Postscaler
	0010 = 1:3 Postscaler
	0011 = 1:4 Postscaler
	0100 = 1:5 Postscaler
	0101 = 1:6 Postscaler
	0110 = 1:7 Postscaler
	0111 = 1:8 Postscaler
	1000 = 1:9 Postscaler
	1001 = 1:10 Postscaler
	1010 = 1:11 Postscaler
	1011 = 1:12 Postscaler
	1100 = 1:13 Postscaler
	1101 = 1:14 Postscaler
	1110 = 1:15 Postscaler
	1111 = 1:16 Postscaler
bit 2	<b>TMRxON:</b> TimerX On bit
	1 = TimerX is on
	0 = TimerX is off
bit 1-0	<b>TxCKPS&lt;1:0&gt;:</b> Timer2-type Clock Prescale Select bits
	00 = Prescaler is 1
	01 = Prescaler is 4
	1x = Prescaler is 16

## The PIC18 CCP units

- A PIC18 device may have one, two, or **five (K22)** 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 16 bit timer into a capture register on a signal edge.
- In **compare** operation, the CCP module compares the contents of a CCPR register with that of a Timer (1, 3 or 5) 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. Timers 2/4/6 are related with the PWM hardware.

## Capture/Compare/PWM (CCP) Modules

- PIC18F45K22 has 5 CCP Modules.
- Each CCP module requires the use of timer resource.
- In capture or compare mode, the CCP module may use either Timer1, Timer3 or Timer5 to operate.
- In PWM mode, either Timer2, Timer4 or Timer6 may be used.
- The operation of a CCP module is controlled by the CCPxCON register (to select the mode), the CCPTMRS0 and CCPTMRS1 registers (to select the operating timer) and the 16-bits data register CCPRx (CCPRxH and CCPRxL).
- A device pin (CCPx pin) can be associated to CCP module operation (with appropriate TRIS configuration).

## CCPxCON Register

### 14.5 Register Definitions: ECCP Control

REGISTER 14-1: CCPxCON: STANDARD CCPx CONTROL REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB<1:0>		CCPxM<3:0>		
bit 7		bit 6				

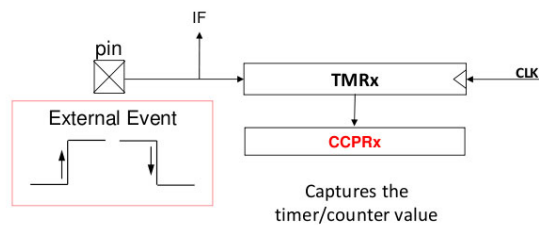


# CCPTMRS0/1 (Timer select) Register

REGISTER 14-3: CCPTMRS0: PWM TIMER SELECTION CONTROL REGISTER 0					
R/W-0	R/W-0	U-0	R/W-0	U-0	R/W-0
C3TSEL<1:0>	—		C2TSEL<1:0>	—	C1TSEL<1:0>
bit 7					bit 0
<p>bit 7-6 <b>C3TSEL&lt;1:0&gt;</b>: CCP3 Timer Selection bits</p> <p>00 = CCP3 – Capture/Compare modes use Timer1, PWM modes use Timer2</p> <p>01 = CCP3 – Capture/Compare modes use Timer3, PWM modes use Timer4</p> <p>10 = CCP3 – Capture/Compare modes use Timer5, PWM modes use Timer6</p> <p>11 = Reserved</p> <p>bit 5 <b>Unused</b></p> <p>bit 4-3 <b>C2TSEL&lt;1:0&gt;</b>: CCP2 Timer Selection bits</p> <p>00 = CCP2 – Capture/Compare modes use Timer1, PWM modes use Timer2</p> <p>01 = CCP2 – Capture/Compare modes use Timer3, PWM modes use Timer4</p> <p>10 = CCP2 – Capture/Compare modes use Timer5, PWM modes use Timer6</p> <p>11 = Reserved</p> <p>bit 2 <b>Unused</b></p> <p>bit 1-0 <b>C1TSEL&lt;1:0&gt;</b>: CCP1 Timer Selection bits</p> <p>00 = CCP1 – Capture/Compare modes use Timer1, PWM modes use Timer2</p> <p>01 = CCP1 – Capture/Compare modes use Timer3, PWM modes use Timer4</p> <p>10 = CCP1 – Capture/Compare modes use Timer5, PWM modes use Timer6</p> <p>11 = Reserved</p>					
REGISTER 14-4: CCPTMRS1: PWM TIMER SELECTION CONTROL REGISTER 1					
U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	—	—	—	C5TSEL<1:0>	C4TSEL<1:0>
bit 7					bit 0
<p><b>Legend:</b></p> <p>R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'</p> <p>u = Bit is unchanged      x = Bit is unknown      -n/n = Value at POR and BOR/Value at all other Resets</p> <p>'1' = Bit is set      '0' = Bit is cleared</p> <p>bit 7-4 <b>Unimplemented:</b> Read as '0'</p> <p>bit 3-2 <b>C5TSEL&lt;1:0&gt;</b>: CCP5 Timer Selection bits</p> <p>00 = CCP5 – Capture/Compare modes use Timer1, PWM modes use Timer2</p> <p>01 = CCP5 – Capture/Compare modes use Timer3, PWM modes use Timer4</p> <p>10 = CCP5 – Capture/Compare modes use Timer5, PWM modes use Timer6</p> <p>11 = Reserved</p> <p>bit 1-0 <b>C4TSEL&lt;1:0&gt;</b>: CCP4 Timer Selection bits</p> <p>00 = CCP4 – Capture/Compare modes use Timer1, PWM modes use Timer2</p> <p>01 = CCP4 – Capture/Compare modes use Timer3, PWM modes use Timer4</p> <p>10 = CCP4 – Capture/Compare modes use Timer5, PWM modes use Timer6</p> <p>11 = Reserved</p>					

## Capture Function

### Capture Function



**Gives a reference on the instant in which the event takes place**  
( Application: measure the period of a given signal )

## Capture Mode

- Main use of CCP is to capture **event** arrival time
- An event is represented by a signal edge.

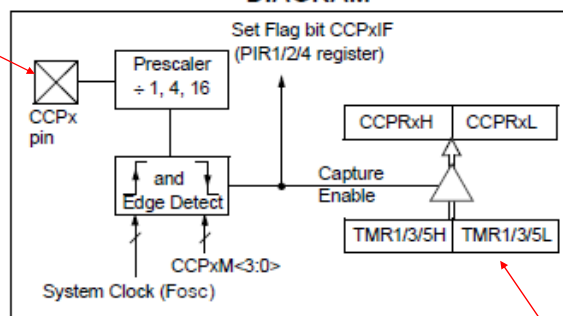
The PIC18 event can be one of the following:

1. every falling edge
2. every rising edge
3. every 4th rising edge
4. every 16th rising edge

## Capture Unit

FIGURE 14-1: CAPTURE MODE OPERATION BLOCK DIAGRAM

CCPx Pin must be configured as an Input

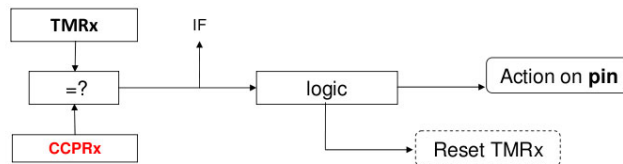


The five capture units operate independently

TimerX must be active and configured (prescaler, CLK source) properly.

## Compare Function

### Compare Function



Generates equally spaced time intervals  
(Application: Delays, Pulse Trains)

## Compare Mode

- 16-bit CCPRx register is compared against one of the Timers(1/3/5).
- 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

### Software Interrupt Mode

When Generate Software Interrupt mode is chosen (1010), the CCPx module does not assert control of the CCPx pin.

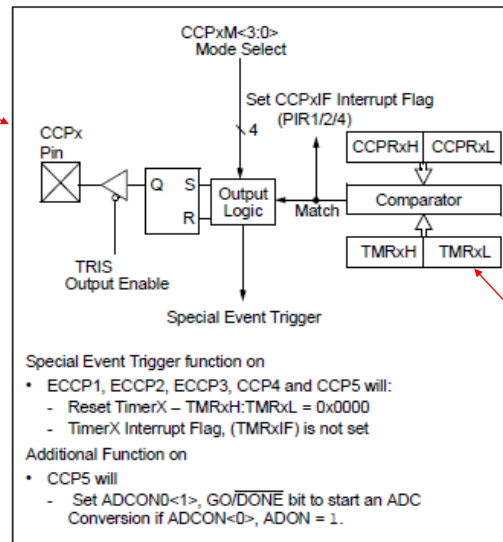
### Special Event Trigger

The CCPx modules can also generate this event (mode 1011) to reset TMR1/3/5 depending on which timer is the base timer. When this mode is selected CCP5 will start an ADC conversion, if the ADC is enabled.

## Compare Unit

FIGURE 14-2: COMPARE MODE OPERATION BLOCK DIAGRAM

CCPx Pin must be configured as an Output



TimerX must be active and configured (prescaler, CLK source) properly.

## Example: use CCP Compare to Generate Sound

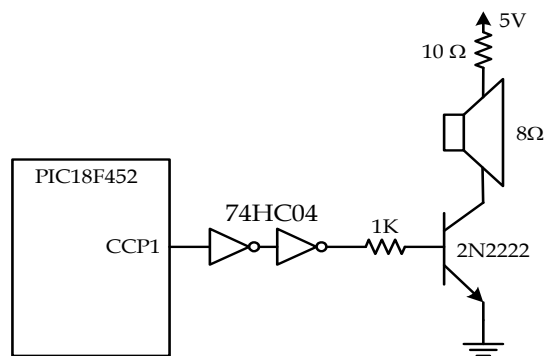


Figure 8.22 Circuit connection for siren generation

## Pitch table

Note	Frequency (Hz)							
	Octave 0	Octave 1	Octave 2	Octave 3	<b>Octave 4</b>	Octave 5	Octave 6	Octave 7
C	16.351	32.703	65.406	130.813	<b>261.626</b>	523.251	1046.502	2093.005
C#,Db	17.324	34.646	69.296	138.591	<b>277.183</b>	554.365	1100.731	2217.461
D	18.354	36.708	73.416	146.832	<b>293.665</b>	587.330	1174.659	2349.318
D#,Eb	19.445	38.891	77.782	155.563	<b>311.127</b>	622.254	1244.508	2489.016
E	20.061	41.203	82.407	164.814	<b>329.626</b>	659.255	1318.510	2367.021
F	21.827	43.654	87.307	174.614	<b>349.228</b>	698.456	1396.913	2637.021
F#,Gb	23.124	46.249	92.449	184.997	<b>369.994</b>	739.989	1474.978	2959.955
G	24.499	48.999	97.999	195.998	<b>391.995</b>	783.991	1567.982	3135.964
G#,Ab	25.956	51.913	103.826	207.652	<b>415.305</b>	830.609	1661.219	3322.438
<b>A</b>	<b>27.500</b>	<b>55.000</b>	<b>110.000</b>	<b>220.000</b>	<b>440.000</b>	<b>880.000</b>	<b>1760.000</b>	<b>3520.000</b>
A#,Bb	29.135	58.270	116.541	233.082	<b>466.164</b>	932.326	1664.655	3729.310
B	30.868	61.735	123.471	246.942	<b>493.883</b>	987.767	1975.533	3951.066

## Example code:

For the circuit in Figure 8.22, write a program to generate a simple song assuming that  $f_{OSC} = 4\text{MHz}$ .

### Solution:

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. CCP1 module is used to generate the signal -compare mode toggle CCP1 pin on match-. TIMER0 is used for note duration.

### Example XC8

```
#include <xc.h>
#define base 3125 // counter count to create 0.1 s delay
#define NOTES 38 // total notes in the song to be played

const unsigned 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};

#define C4 0x777 // semioperiod
#define F4 0x598 // for each
#define G4 0x4FC // note freq
#define A4 0x470
#define B4 0x3F4
#define C5 0x3BC
#define D5 0x353
#define F5 0x2CC

const unsigned char wait[38] = { 3,5,3,3,5,3,3,5,
                                3,3,5,3,3,5,3,3,
                                5,3,3,3,2,2,3,3,
                                6,3,5,3,3,5,3,3,
                                3,2,2,3,3,6};

unsigned int half_cycle;
```

```

void delay (unsigned char xc);

void interrupt high_ISR(void)
{
    if (PIE1bits.CCP1IE && PIR1bits.CCP1IF) {
        PIR1bits.CCP1IF = 0;
        CCPR1 += half_cycle;
    }
}

void interrupt low_priority low_ISR (void)
{
}

```

```

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++) { /* Play song several times */
        i = 0;
        half_cycle = per_arr[0];
        CCPR1 = TMR1 + half_cycle;
        while (i < NOTES) {
            half_cycle = 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 (11); /* Pause before replay song */
        INTCON |= 0xC0; /* re-enable interrupt */
    }
}

```

```

/* 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 */
/* ----- */
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;
        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;
        case 5: /* 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);
            break;
        case 6: /* 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);
            break;

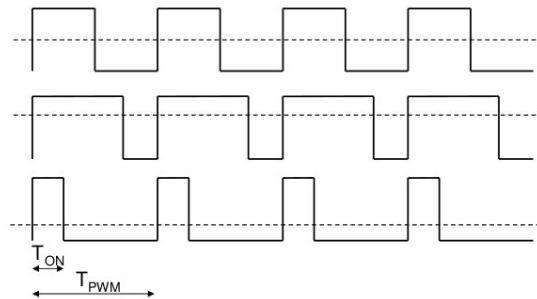
        default:
            break;
    }
}

```

# PWM function

## Pulse Width Modulation (PWM) Function

Generates (automatically) a PWM signal



$$\text{Period} = T_{PWM}$$

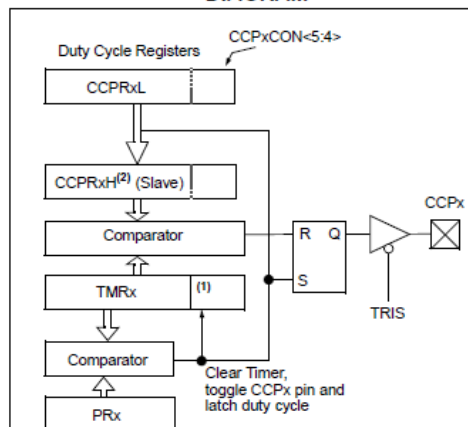
$$\text{Frequency, } F_{PWM} = 1 / T_{PWM}$$

$$\text{Duty Cycle} = (T_{ON} / T_{PWM}) \times 100$$



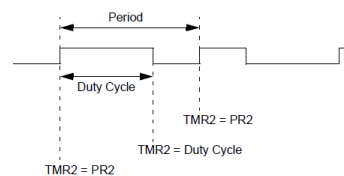
# PWM Mode

FIGURE 14-4: SIMPLIFIED PWM BLOCK DIAGRAM



**Note 1:** The 8-bit timer TMRx register is concatenated with the 2-bit internal system clock ( $F_{osc}$ ), or 2 bits of the prescaler, to create the 10-bit time base.

**Note 2:** In PWM mode, CCPRxH is a read-only register.





## Period & Duty cycle

### Pwm formulas:

EQUATION 14-1: PWM PERIOD

$$PWM\ Period = ((PRx + 1) \cdot 4 \cdot TOSC \cdot (TMRx\ Prescale\ Value))$$

Note 1:  $TOSC = 1/FOSC$

EQUATION 14-2: PULSE WIDTH

$$Pulse\ Width = (CCPRxL \cdot CCPxCON[5:4] \cdot TOSC \cdot (TMRx\ Prescale\ Value))$$

EQUATION 14-4: PWM RESOLUTION

$$Resolution = \frac{\log[4(PRx + 1)]}{\log(2)}\ bits$$

EQUATION 14-3: DUTY CYCLE RATIO

$$Duty\ Cycle\ Ratio = \frac{(CCPRxL \cdot CCPxCON[5:4])}{4(PRx + 1)}$$

TABLE 14-7: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (FOSC = 32 MHz)

PWM Frequency	1.95 kHz	7.81 kHz	31.25 kHz	125 kHz	250 kHz	333.3 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PRx Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 14-8: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (FOSC = 20 MHz)

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PRx Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 14-9: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (FOSC = 8 MHz)

PWM Frequency	1.22 kHz	4.90 kHz	19.61 kHz	76.92 kHz	153.85 kHz	200.0 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PRx Value	0x65	0x65	0x65	0x19	0x0C	0x09
Maximum Resolution (bits)	8	8	8	6	5	5

## Programming recipe.

1. Disable the CCPx pin output driver by setting the associated TRIS bit.
2. Select the 8-bit TimerX resource, (Timer2, Timer4 or Timer6) to be used for PWM generation by setting the CxTSEL<1:0> bits in the CCPTMRSx register.(1)
3. Load the PRx register for the selected TimerX with the PWM period value.
4. Configure the CCP module for the PWM mode by loading the CCPxCON register with appropriate values.
5. Load the CCPRxL register and the DCxB<1:0> bits of the CCPxCON register, with the PWM duty cycle value.
6. Configure and start the 8-bit TimerX resource:
  - Clear the TMRxIF interrupt flag bit of the PIR2 or PIR4 register.
  - Configure the TxCKPS bits of the TxCON register with the Timer prescale value.
  - Enable the Timer by setting the TMRxON bit of the TxCON register.
7. Enable PWM output pin:
  - Wait until the Timer overflows and the TMRxIF bit of the PIR2 or PIR4 register is set.
  - Enable the CCPx pin output driver by clearing the associated TRIS bit.

## Light dimmer

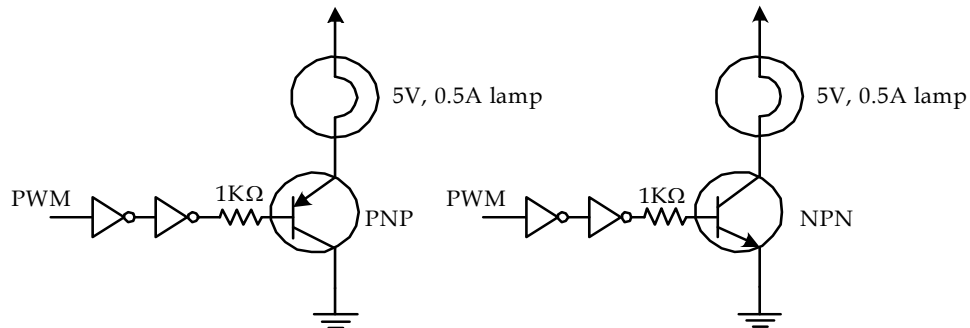


Figure 8.25 Using PWM to control the brightness of a light bulb

## 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.

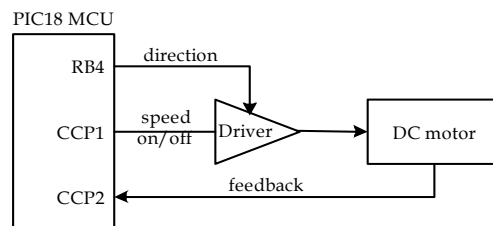


Figure 8.26A simplified circuit for DC motor control

## DC Motor Driver

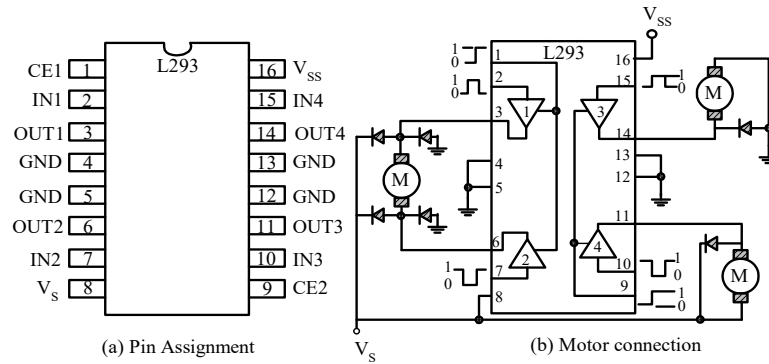


Figure 8.27 Motor driver L293 pin assignment and motor connection

## Speed sensor

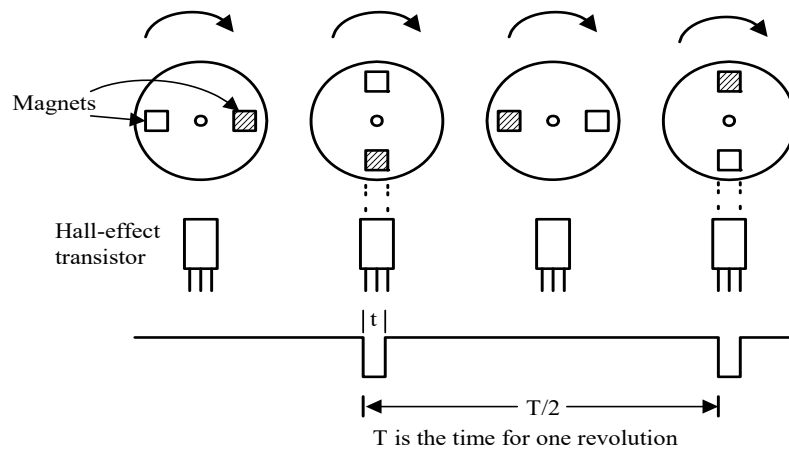
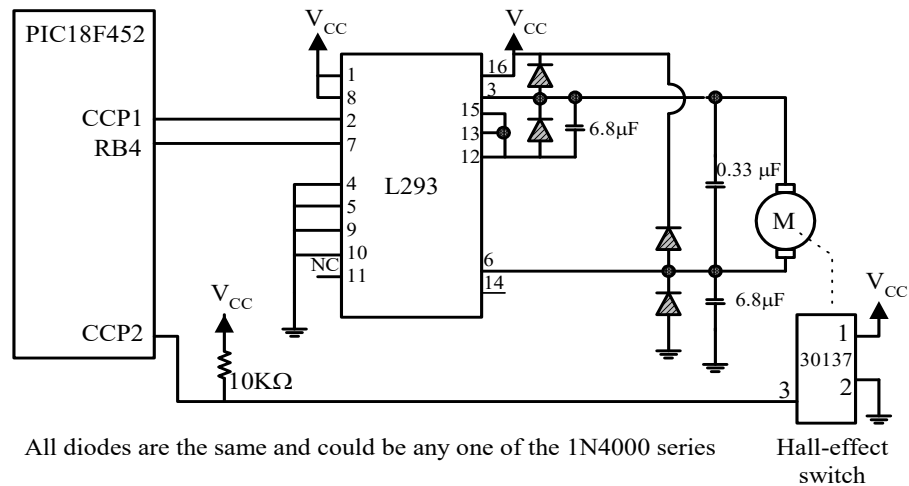


Figure 8.28 The output waveform of the Hall-effect transistor

## Motor control system



## Clockwise or counterclockwise Rotation

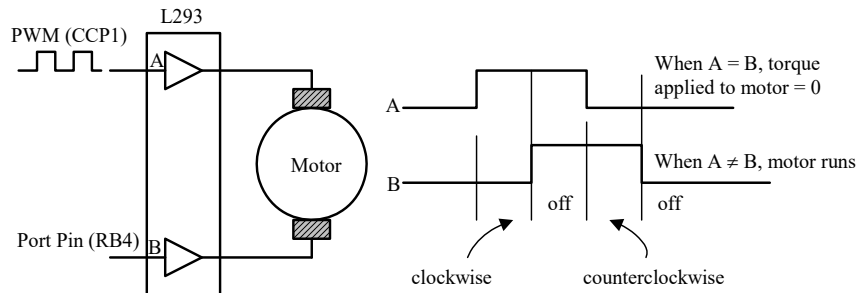
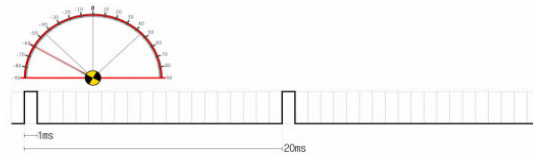
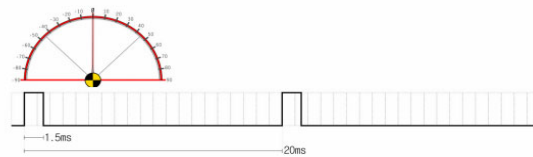


Figure 8.30 The L293 motor driver

# Servomotors



(a)



(b)

