**Timers:**

A timer is a digital sequential circuit that can count at a precise and programmable frequency. A **timer** on a µC is simply a counter. The input clock to a timer can be internal or external. The input clock (internal) frequency to a timer can be **prescaled** so that it is some fraction of the system clock frequency.**Auto Reload Timer.** Compared to a standard timer, this timer automatically reloads its counting value when the count is over, therefore sparing a waste of CPU resource.



A timer on a microcontroller is simply a counter that is clocked at a particular frequency, and generates an action when a particular count is reached. The amount of time to trigger the event depends on the clock frequency of the counter, and the size of the counter.

Delay can be software based (register based) or Hardware based (timer based) Software delays are easy to implement, but hardware timers are much better at creating accurate time delays.

**Timers can be used for:**

* Switch Debouncing (delay generation)
* Waveform Generation
* Pulse Width Measurement
* Generation of Pulse Width Modulation (PWM)
* Period and Frequency Measurement
* Speed Measurement by encoder (Motor applications)
* Delay Generation
* Measuring time between events
* Count external events
* Power-up timer

**Power-up timer :**

The power-up timer on the PIC18 implements a fixed delay (typical value is 72 ms) after power-up. Its intended use is to provide time for the power supply to stabilize before fetching of the first instruction.

**Timer selection in microcontrollers:**

* Clock pulse source (internal or external)
* Clock pulse edge for external clock pulse (with use of external clock pulse , timer works as counter)
* For internal clock pulse ( selection of cpu clock division ratio factor)
* Interrupt generation or not

**Timers in PIC18: Timer0:**

This timer is an 8/16 bit counter which works with internal or external clock pulse.Timer0 includes a programmable prescaler that divides the timer input clock, whose source is FOSC/4. The prescalar has eight values ranging from 256:1 down to 2:1 that is configured via configuration bits. The interrupt flag TMR0IF is set when the counter rolls over, which is 0xFF to 0x00 in 8-bit mode or 0xFFFF to 0x0000 in 16-bit mode.

****

**Clock pulse division circuit:**

**Pulse width measurement with using of Timer 0 (Timer0 is 16 bit timer)**

A fundamental timer application is time measurement between two external

events. In the digital world, an external event is either a rising or falling edge on an input pin. The time between two edges of the same type (falling-to-falling edge or rising-to-rising edge) is the period of a square wave, while the time between a rising-to-falling edge and falling-to-rising edge is high pulse width or low pulse width,respectively.

**Example**: Measuring of human –activated pushbutton switch. (The time between two edege (falling and rising edge)



In this section, the RB0/INT0 input and the Timer0 subsystem is used for pulse

width measurement using the steps shown in Table

**Steps**

1. Configure INT0 to generate an interrupt on a falling edge. Configure Timer0 to

be clocked by the internal clock and clear Timer0.

2. If the interrupt service routine is triggered by an INT0 falling edge, start Timer0

and reconfigure INT0 to be rising edge triggered.

3. If the interrupt service routine is triggered by a rising edge, turn off Timer1 and

copy the Timer1 value, which represents the elapsed time from the falling edge

to the rising edge.

**Calculation:**

Equation shows how to convert the Timer0 value to elapsed time, where TOSC is the internal clock period( 1(FOSC) and TMROPRE is the prescaler value:

Pulse width = TMR0\*TOSC\*4\*TMR0PRE

As an example, assume FOSC = 40 MHz (TOSC = 25ns), 16-bit mode, and

TMR0PRE = 1. The minimum pulse width is 1\* 25 ns \* 4 \* 1 = 100 ns,

The time from the falling edge to the timer being turned on by the ISR is not counted as part of the pulse width, while any timer tics that elapse from the rising edge to the timer being read by the ISR is erroneously added to the pulse width.

**Timer1 and Timer3:**

Timer1 and 3 are 16-bit read/write counters. These timers support prescaler mode and interrupt generation due to counter roll over. These timer are used with CCP module.

**Timer2:**

Figure shows the Timer2 subsystem, which consists of an 8-bit timer, a prescaler (1, 4, 16), an 8-bit period register (PR2), and a postscaler (1 through 16).

The Prescalar is a mechanism for generating clock for timer by CPU clock. Every CPU has a clock source and the frequency of this source decides the rate at which instructions are executed by the processor. The Prescalar is used to divide this clock frequency and produce a clock for TIMER. **The prescaler divides the timer input clock, whose source is FOSC/4 (the instruction cycle clock). The 8-bit period register is used as the comparison value against the timer value; when they are equal, this asserts the Timer2 equal signal (TMR2\_EQ) and resets the timer to zero. The postscaler sets the Timer2 interrupt flag (TMR2IF) for every 1 of n TMR2\_EQ events**, where n is 1 through 16. A postscaler value of 1:1 means that the TMR2IF bit is set for each TMR2\_EQ event, while a postscaler value of 1:16 means the TMR2IF bit is set once for every 16 TMR2\_EQ events. a prescaler (1, 4, 16), an 8-bit period register (PR2), and a postscaler (1 through 16). The prescaler divides the timer input clock, whose source is FOSC/4 (the instruction cycle clock).





**The (CCP) module:** The (CCP) module is software programmable which is used for applying time-based operations. This module requires time resources, and operates in one of three modes:

**1. A Capture input**

**2. A Compare output**

**3. A Pulse Width Modulation (PWM) output**

**Input Capture.**

Input Capture can measure external frequencies or time intervals by copying the value from a free running timer into the input capture register when an external event occurs.

Input capture is a method of dealing with [input](http://en.wikipedia.org/wiki/Input) signals in an [embedded system](http://en.wikipedia.org/wiki/Embedded_system). In capture mode, the CCP module captures the contents of Timer 1 and Timer3 when an event occurs on the CCP input pin’s

**Application of Input Capture:**

* Generate interrupts
* Measure period, frequency, pulse width
* Count pulses
* Decode remote control (TV remote control)
* Count events: Every falling edge, Every rising edge ,Every 4 th rising edge or Every 16 th rising edge

**Example application:** The time of flight of a sonar echo can be measured using polling, interrupts, or input capture

# Input Capture Registers

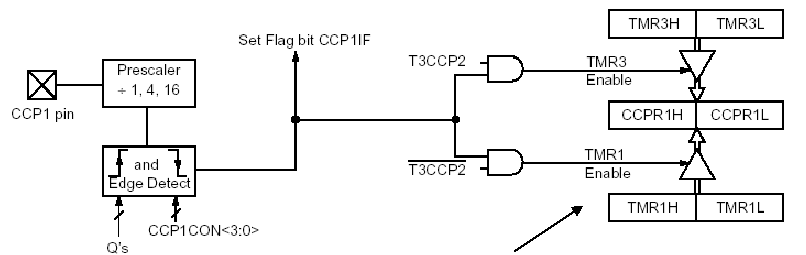
Input capture registers are the set of 16- bit registers that are used to apture the contents of the free running counter, when a control signal (generated internally or externally) is received

Rising or falling edge

detect, with interrupt flag

set.

TMR1 or TMR3 16-bit value transferred to 16-bit capture register on edge detect.



**PULSE WIDTH MEASUREMENT USING CAPTURE MODE**

The pulse width measurement scheme of Section 13.2 has an accuracy shortcoming in that time between a falling edge occurrence and the Timer0 being turned on is not counted as part of the pulse width. Another accuracy problem is that any timer tics that elapse between a rising edge occurrence and Timer0 being read is incorrectly counted as part of the pulse width. The capture subsystem solves these problems by causing an automatic transfer of the timer register contents to a capture register on occurrence of an event



Pulse width measurement using Timer1 and capture mode

One significant difference between this scheme and the previous scheme is that

the timer is always in operation; it is not started and then stopped as was done by using timer 0. This means that the elapsed timer tics between the two captured timer values must be computed, and that timer overflow can occur between the two capture events because of the free-running nature of the timer. Figure shows the case where timer overflow does not occur; the capture A value represents the falling edge timer value and the capture B value the rising edge timer value. The pulse width in timer tics is TimerDelta = B - A because timer overflow has not occurred. Figure shows the case when the timer overflows and the pulse width in timer tics is computed as TimerDelta = (#oflows-1) \* 216 + (0A) + B. The #oflows variable counts the number of times the timer overflows, while the 0–A delta value is the number of timer tics to the first overflow.

****

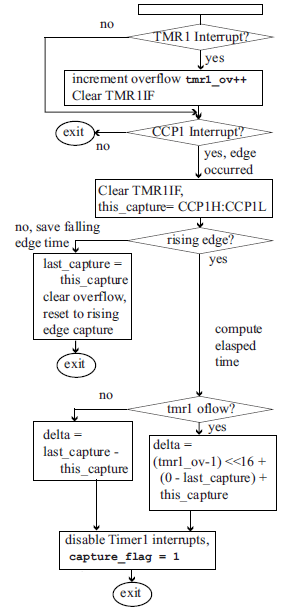
Computing the elapsed timer tics between two events.

If no overflow has occurred and the elapsed timer tics is computed as:

**delta = this\_capture – last capture**

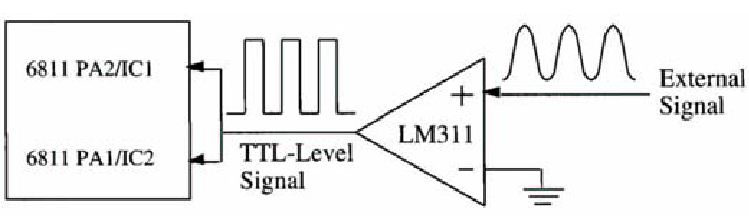
If timer overflow has occurred and the pulse width is computed as per Figure :

**delta = ((tmr1\_ov-1) << 16) + (0 – last\_capture) + this\_capture**



ISR for pulse width measurement using Timer1 and capture mode.

**Pulse width measurement using two input capture channels improves resolution of measurement. (CCP1 and CCP2)**

****

**Output Compare:**

The compare mode of Timer1 or Timer3 is compared against the CCPR (16 bit register). If a match is occurs, the output pin is driven high, low, toggles, or an interrupt signal is requested. Output Compare can time an external event by sending a value stored inside the output compare register. Used for outputting waveforms to control actuators or is used to generate time delays for I/O functions . This mode can be used for:

* Generate exact timing signals or events
* Compare system clock to trigger clock When registers match, changes state of output pin
* creating pulse waveforms such as square waves, variable duty-cycle waves, and single pulses Output

Application:

* Generate periodic interrupts
* Generate square waves
* Generate pulse width encoded signals

Output compare is the ability to trigger an [output](http://en.wikipedia.org/wiki/Output) based on a timestamp in memory, without interrupting the execution of code by a [processor](http://en.wikipedia.org/wiki/Central_processing_unit) or [microcontroller](http://en.wikipedia.org/wiki/Microcontroller).

# Output Compare Registers

Output compare registers are the set of 16- bit registers that can be initialized with an specific count. • They are continuously compared to the contents of the free running counter. • Whenever there is a match, a flag is set or they generate an output signal.

**period register:**

A timer can be programmed to roll over at any point using the **period** register.

An 8- bit timer would typically roll over to 0x00 once it reached 0xFF. However, if the period register is set to 0x7F, then timer will roll over to 0x00 once it increments past 0x7F. for example PR is the period register for TMR

***Example of application: ( Generate Delayed Pulse)***

Modern automobile engines use a microcontroller to control spark timing. A key function that it performs is to generate a spark at a prescribed time either before or after the piston has achieved a certain position. Typically, a sensor generates a pulse when the piston has reached top dead center. The microcontroller must generate a signal to output a spark at a prespecified delay time after this signal. Imagine that the optical sensor on your motor kit corresponds to the sensor on the engine crankshaft that detects when the piston has reached top dead center.

**CCP(PWM mode):**

CCP modules have a PWM mode in conjunction with Timer 2, as shown in Fig. Here Timer 2 runs with a period determined by the main crystal, prescaler and **Period Register 2**. When the **Timer 2 comparator causes the count to reset, it also sets the PWM latch. This gives the PWM repeat period.** A second CCP equality comparator matches the duty cycle number which is set up by the program in CCPR1L .Taken together this gives a duty-cycle .This datum is loaded into the CCPR1H and Timer 2 rolls over.

* The PWM period is set by the Timer 2/PR2 time-out.
* The duty cycle is set by the 10-bit datum in CCPR1L:CCP1CON[5:4].
* The duty cycle datum can be glitchlessly changed by the software at any time by updating the slave registers and will take effect in the PWM period following this update.
* The RC2/CCP1 pin direction should be set to output. CCP1 is output for generated PWM signal and Configure the CCP1 module for PWM operation

PWM capability is provided through the use of Timer2 and the CCPR1 registers as shown in Figure. The PR2 register sets the period of the generated square wave, while the CCPR1H register provides the duty cycle. A match of the PR2 register and TMR2 value sets the CCP1 pin high, clears the TMR2 register, and transfers the CCPR1L value to CCPR1H to fix the duty cycle. A match of TMR2 with CCPR2H resets the CCP1 pin, thus terminating the high portion of the square wave. Both PR2 and the duty cycle are extended to 10 bits of precision.

PWM Period = (PR2+ 1) \* 4 \* TOSC \* TMR2\_PRE

PWM Duty Cycle = (CCPR1L:CCP1CON[5:4])\*TOSC\*TMR2\_PRE

The duty cycle as a percentage of the resulting square wave is given by Equation





**CCP in PWM mode (with Timer 2)**

**Sample Question:** Assume a FOSC of 10 MHz. Give the PR2 and prescale values for the PWM mode that generates a square wave with a 75% duty cycle and a period of 6 kHz on the CCP1 output pin. Use the prescale value that gives the largest PR2 value. Only give the upper 8-bit value for the duty cycle register (CCPR1).

*Answer:* From Equations :

Timer2 PWM period = (PR2+1) \* (4/FOSC)\* PRE (recall that the postscaler is NOT used for PWM period). Then:

(1/6 kHz) = (PR2+1) \*(4/10 MHz) \* PRE

PR2 = [(10 MHz/4) / (6 KHZ \* PRE) ] – 1

For PRE = 1, PR2 = 416 ( > 255, so too large). For PRE = 4, PR2 = 103.

For PRE = 16, PR2 = 25, so use PRE = 4, PR2 = 103.

For a 75% duty cycle, CCPR1 = 0.75 \*(PR2+1) = 0.75\*104 = 78

**Frequency measurement**

Another use of the timer/counter is to measure frequencies. **For this purpose, a number of cycles must be counted over a known time.** The timer/counter cannot be used to generate this time reference, so an external time reference must be used. For simplicity it can be assumed that an external time reference exists with a base of 1 sec, this can be fed into the external interrupt line so that interrupts will be generated every second. At the first interrupt, the counter will be cleared and at the next interrupt the value in the counter will be the frequency. If the counter is used without the prescaler, the range of frequencies would be from 0 to 255 Hz. If the prescaler, with a scale of 1:256, is used, the range of frequencies is now 0 to 65280 Hz, but since the prescaler cannot be read there can be an error in that measurement which can be as much as 255 cycles. This error can be significant if the same 1:256 scale is used for low values of frequency. Therefore it is better to use an additional file register to count the number of times the counter has overflowed and keep the prescaler at low values e.g. 1:2 where the error is low or even not to use the prescaler at all. The ISR responds to two sources, counter overflow and time reference rising clock edge. When the counter overflows the file register, say n cycles is incremented by one. The actions for the time reference interrupt were given above.

**Watchdog timer:**

Watchdog is a mechanism which microcontroller uses to defend itself against programs getting stuck. As with any other electrical circuit, so with a microcontroller too can occur failure, or some work impairment. Unfortunately,microcontroller also has program where problems can occur as well. When this happens, microcontroller will stop working and will remain in that state until someone resets it. Because of this, watchdog mechanism has been introduced. After a certain period of time, watchdog resets the microcontroller (microcontroller in fact resets itself). Watchdog works on a simple principle: if timer overflow occurs, microcontroller is reset, and it starts executing a program all over again. In this way, reset will occur in case of both correct and incorrect functioning. Next step is preventing reset in case of correct functioning, which is done by writing zero in WDT register A watchdog timer provides a means of graceful recovery from a system problem. This could be a program that goes into an endless loop, or a hardware problem that prevents the rogram from operating correctly. If the program fails to reset the watchdog at some predetermined interval, a hardware reset will be initiated. The bug may still exist, but at least the system has a way to recover. This is especially useful for unattended systems.

One use of the watchdog timer is to place a maximum wait time for some external event to occur.Another use of the WDT timer is to wake the PIC18 from sleep mode, a lowpower standby mode entered by executing the PIC18 instruction SLEEP (has no arguments). Sleep mode stops the internal clock, thus freezing all register contents and dramatically lowering power consumption. One way to exit sleep mode is for the WDT to expire; even though the internal clock is stopped, the WDT continues running because it has an independent, internally generated clock source. Sleep mode exit caused by WDT expiration is called WDT wake-up.

**Glossary:**

## Postscaler

Would like to generate an interrupt when a timer rolls over ;

A Postscaler is used as a second counter – only generate an interrupt after timer rolls over N times.