Tutorial Introduction

PURPOSE
- To explain how to configure and use the Timer Interface Module in common applications

OBJECTIVES:
- Identify the steps to set up and implement an output compare function.
- Identify the steps to set up and implement an input capture function.
- Identify the steps to set up and implement both unbuffered and buffered pulse width modulation functions.
- Identify the differences between unbuffered and buffered PWMs.

CONTENTS:
- 20 pages
- 6 questions

LEARNING TIME:
- 45 minutes

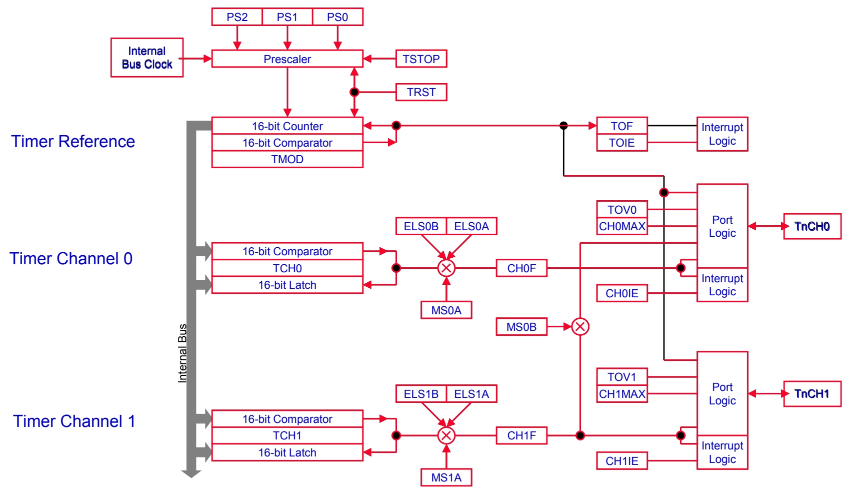
PREREQUISITE:
- 68HC08 CPU Training Module

Welcome to this tutorial on the 68HC08 Timer Interface Module. This tutorial covers the configuration and use of the (TIM) in detail. Please note that on subsequent pages, you will find reference buttons in the upper right that access additional content.

Upon completion of the tutorial, you'll be able to set up and perform an output compare function, an input capture function, and several pulse width modulation (PWM) functions demonstrating the differences between unbuffered and buffered PWMs.

The recommended prerequisite for this tutorial is the 68HC08 CPU training module. Click the Forward arrow when you're ready to begin the tutorial.

Timer Interface Module Block Diagram



Let’s begin this tutorial with a conceptual overview of the Timer Interface Module.

The 68HC08 TIM has user selectable timer channels instead of the traditional dedicated input capture and output compare functions. Each channel can be user programmed to perform either an output compare function, an input capture function, or a pulse width modulation function. These functions provide the flexibility to configure the timer exactly as needed for specific applications. You can also reconfigure the channels in an application to perform different subtasks.

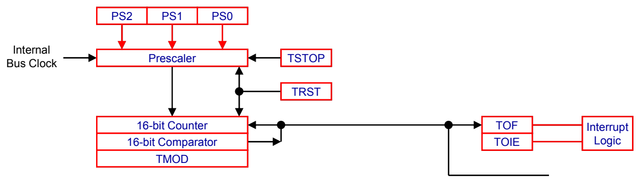
Each 68HC08 derivative MCU is implemented with a TIM of either 2, 4, 6, or 8 programmable timer channels. This block diagram is of a two channel TIM.

Each timer channel shares a single time reference that is derived from a 16-bit counter, a prescaler, and a modulo comparator. The prescaler is programmable to provide the common 16-bit reference counter with a clock that is derived from the bus frequency and one of seven divide-by ratios.

The modulo comparator allows the 16-bit timer to effectively count up only to the value loaded in the modulo comparator register, TMOD, before resetting and restarting another count-up sequence. Later we will see how this can be useful in generating PWMs.

Interrupts can be obtained from the counter overflow and either input or output timer channel activity. Each interrupt source has its own interrupt vector to allow servicing an interrupt without time consuming polling for the source.

Timer Interface Module
Time Reference



Time Reference, Part 1

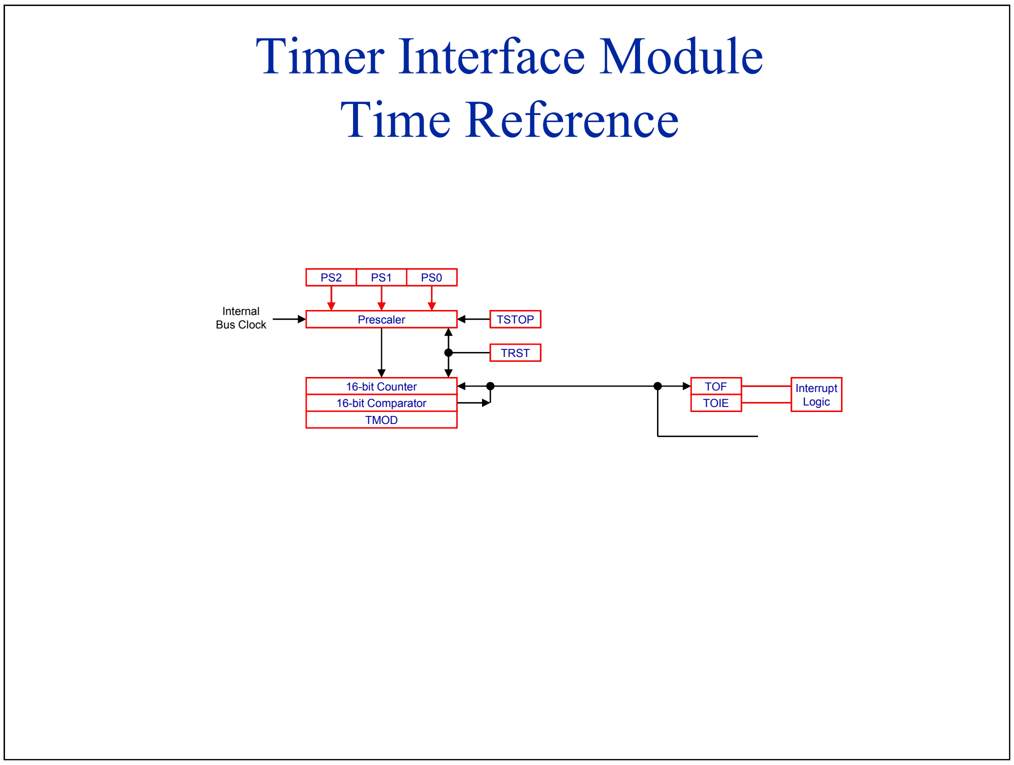
Let’s take a closer look at the features of the time reference. The time reference is generated by a 16-bit counter, a prescaler, and a modulo comparator.

For normal operations, the 16-bit counter runs continuously as a free-running counter. You can stop or reset the counter under program control. This is particularly useful when you want to synchronize the TIM reference to another timer or clock.

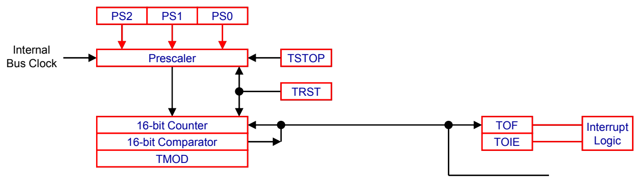
The prescaler provides the 16-bit counter with a clock value derived from the 68HC08 internal bus clock and a divide-by ratio. Using PS0, PS1, and PS2, you can set the divide-by ratio to one of seven values: divide-by-1, divide-by-2, or any other binary division up to divide-by-64. You can also select an external clock.

In its highest resolution mode (divide-by-1), the TIM supports a resolution of 125ns when the part is operating at the standard internal bus frequency of 8MHz (1 divided by 8 MHz). To clock the 16-bit reference at a slower rate, the prescaler can further divide down the internal bus frequency, as slow as divide-by-64.

The 16-bit time reference includes a 16-bit modulo register, TMOD, that is compared with the 16-bit counter. When the two match, the timer overflow flag (TOF) is set and the 16-bit counter is reset and begins another count-up sequence. By default, the modulo register is set to hexFFFF allowing the reference to count through all 16-bits. As we’ll see later in this tutorial, we can write a different value to the modulo register to generate the period of a pulse width modulated signal.

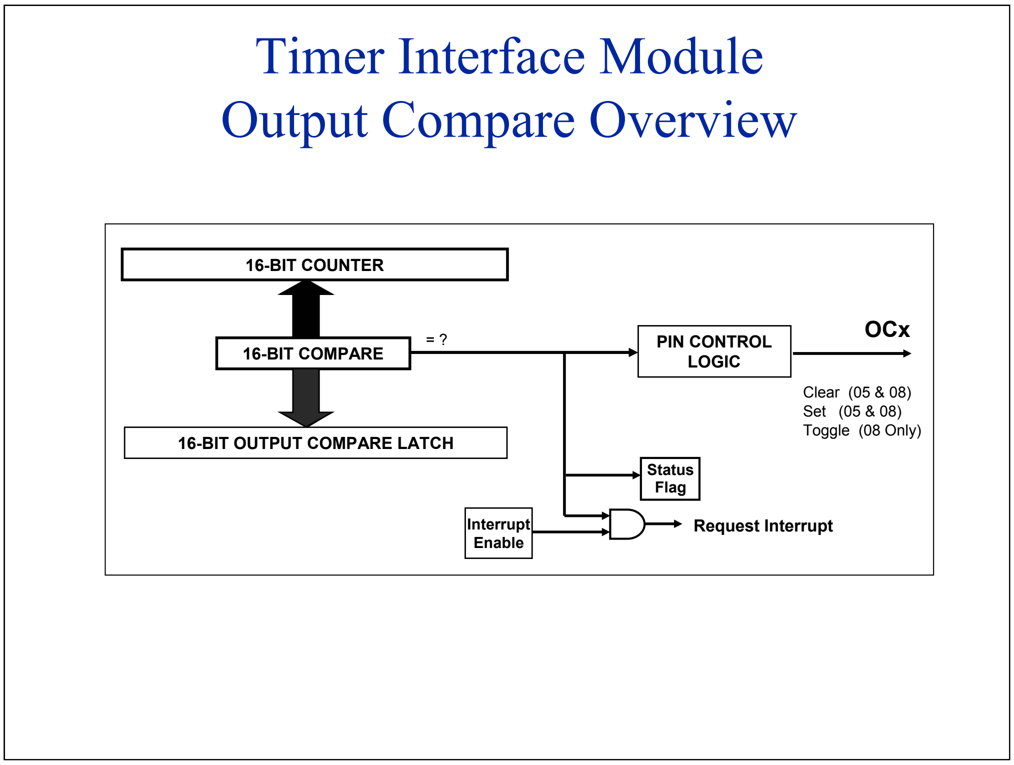


Timer Interface Module
Time Reference

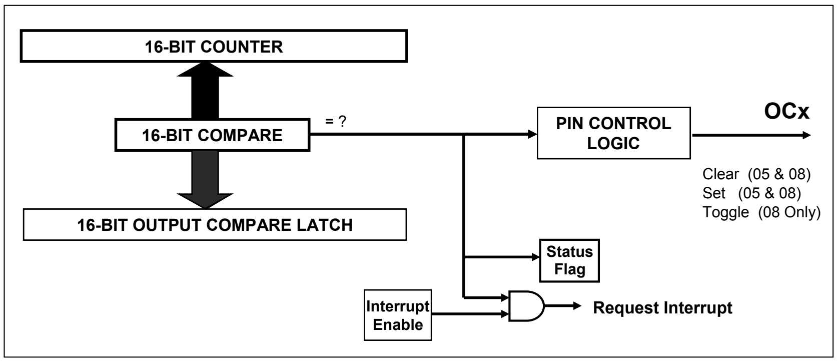


Time Reference, Part 2

You can time events longer than 65,536 cycles (16-bits) using the timer overflow feature. When the timer overflow interrupt flag (TOIE) is set, an interrupt event will be generated each time the TOF flag is set. To time an event of long duration, you can write code to count the number of overflows that occur. The prescaler provides the capability for optimizing resolution vs. timer overflow processing overhead.



Timer Interface Module
Output Compare Overview



Let's take a look at each of the TIM functions beginning with output compare.

The output compare function uses the 16-bit time reference, a comparator, a 16-bit output compare latch, an output pin, pin control logic, and interrupt logic. To demonstrate the output compare function, we'll look at a simple delay example.

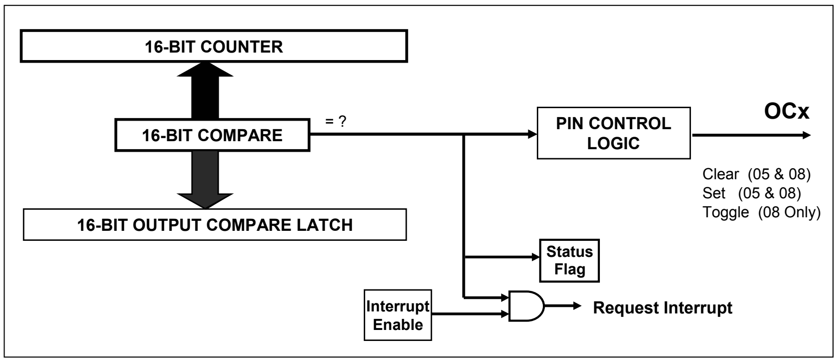
We'll start the example by enabling interrupts.

Next, select an offset value based on the desired delay and add this to the value of the 16-bit free running counter.

Write the sum of the counter value and the offset to the output compare latch.

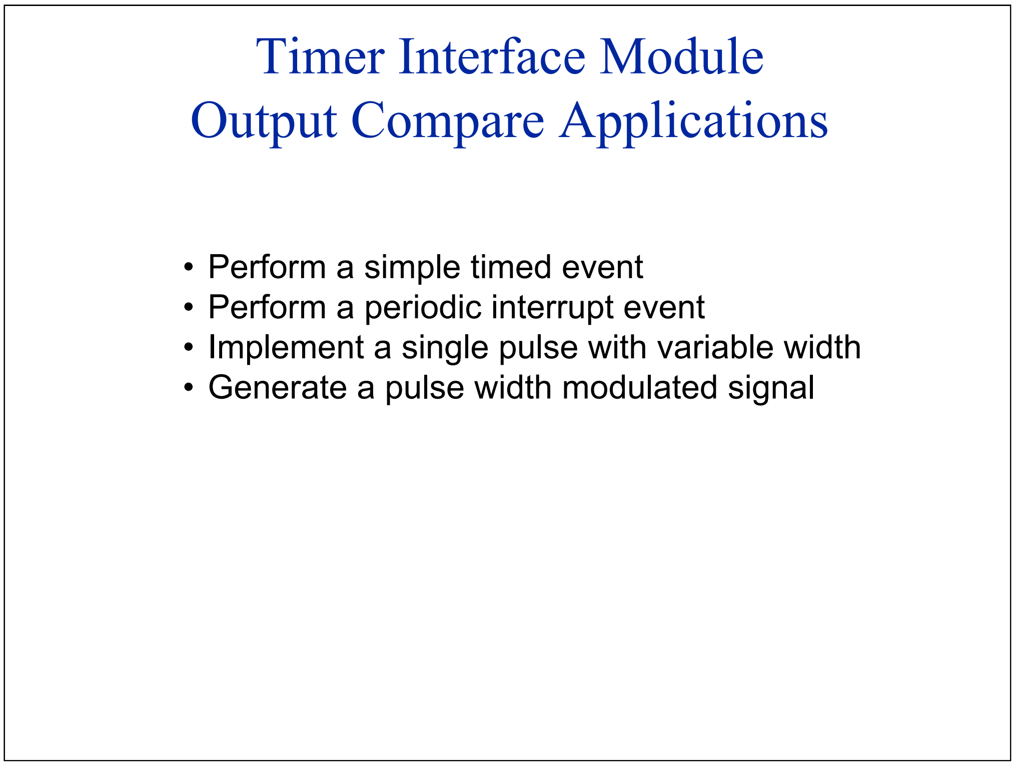
When the free running counter matches the output compare latch a status flag is set indicating the timed delay has occurred.

Timer Interface Module
Output Compare Overview



Optionally, you can set, clear, or toggle an output pin when the counter matches the value in the compare latch.

An optional interrupt can be generated. This approach allows the user to very accurately time and create external events using the output compare i/o pins.



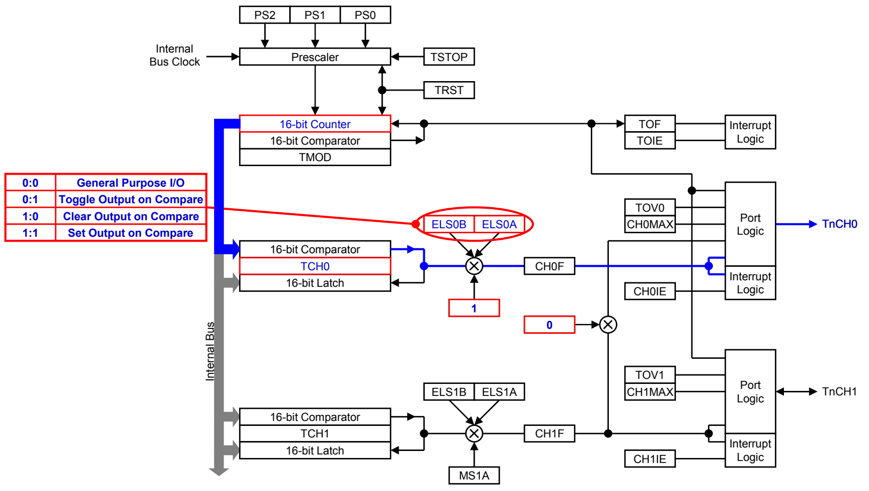
Timer Interface Module
Output Compare Applications

- Perform a simple timed event
- Perform a periodic interrupt event
- Implement a single pulse with variable width
- Generate a pulse width modulated signal

You can use the output compare function for a variety of common embedded control timing functions. In the previous example, we used the output compare function to perform a simple delay. We can expand the example to perform a periodic interrupt by enabling output compare interrupt. When the service routine receives an output compare interrupt, it calculates a new output compare value before resuming normal processing.

Using the output pin set, clear, and toggle options, you can implement a single pulse output with variable width. Using the same techniques, you can generate a variable output frequency or a pulse width modulated signal. We'll review these techniques in more detail later in the tutorial.

Timer Interface Module
Output Compare Function



Let’s look at a more detailed example of an output compare function using Channel 0.

We start by clearing MS0B to zero to isolate the two timer channels from each other and setting MS0A to one to select the output function.

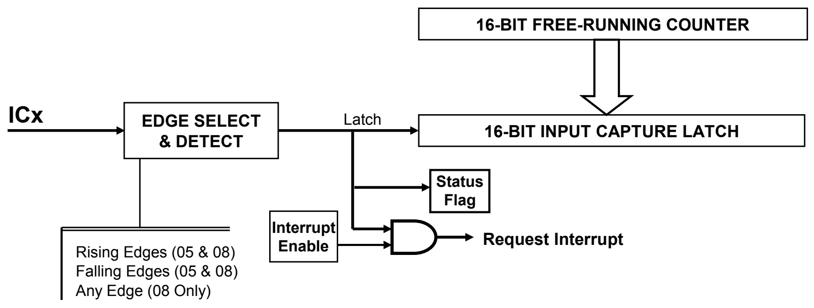
Then you use ELS0A and ELS0B to program the pin operation. Note the toggle option. This can be used to simplify the generation of pulses.

Output Compare is used to generate timed signals. The value stored in TCH0 is constantly being compared to the 16-bit counter.

When the value in the 16-bit counter equals that of TCH0, the programmed output signal transition (ELS0B:ELS0A) is forced on the signal pin TxCH0. This can also generate a timer channel interrupt, if enabled. Typically, an interrupt service routine would be used to schedule the next timer output signal transition. You simply read the value in TCH0, add a time offset, and save the result in TCH0. Once again, as soon as the counter value catches up and equals TCH0, the programmed activity occurs.

It is assumed that the counter is free-running, that is, it counts up to hex FFFF and back to zero. If not, you have to take this into account when calculating the new value to write to TCH0.

Timer Interface Module
Input Capture Overview



Next, let's look at another important TIM function, input capture.

The input capture function uses the free-running 16-bit counter, a 16-bit capture latch, an input pin with edge select and detect logic, and interrupt logic. This function allows external timing events to be referenced to the 16-bit free running counter.

You can configure the input pin to look for a rising edge, a falling edge, or either edge. In this example we will select rising edges.

We'll also enable interrupts.

Once the selected edge is detected, the input capture latch is loaded with the value of the free-running counter, recording a time-stamp of when the event occurred.

A status flag is set and an interrupt can optionally be generated.

Timer Interface Module
Input Capture Applications

- Perform time reference to an external event
- Measure an input period
- Measure the width of an input pulse
- Provide additional external interrupts

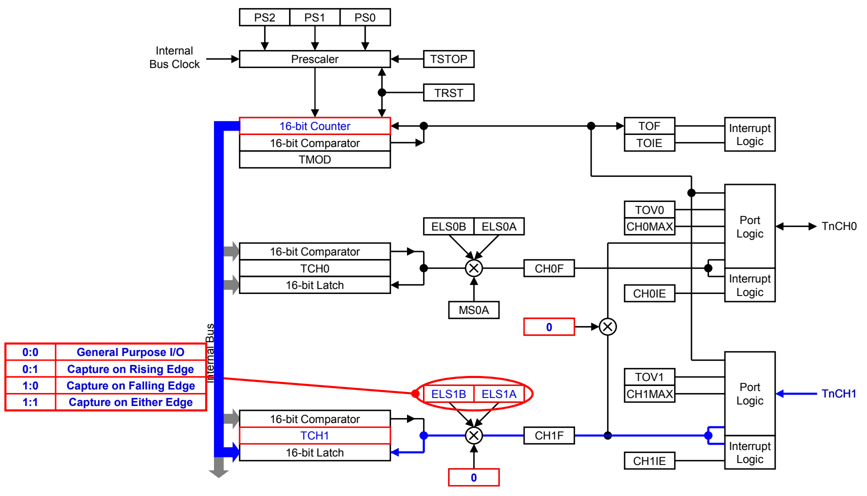
You can use the input capture function for a variety of common embedded control timing functions. In the previous example, we used the input capture function to perform absolute time reference to an external event. We can expand this to measure an input period by enabling the input capture interrupt. When the service routine receives an input capture interrupt, it temporarily saves a copy of the input capture latch and then returns to normal processing until the selected edge is detected again. The service routine subtracts the two input capture values to determine the period. For long periods, the service routine adjusts for the number of timer overflow interrupts that occurred between the first and second edges.

Using input capture, you can measure the width of an input pulse. This function is similar to measuring a period except that the second capture is configured to detect the falling edge instead of the initial rising edge. For very short pulse widths, you can use the two timer channels to look at the same signal with one channel detecting the rising edge and the other channel detecting the falling edge. This enables measurements down to 125 nsec.

If the capture capability is not needed, you can use the input capture pins as extra interrupt lines.

Next we'll look at a more detailed example of the input capture function.

Timer Interface Module
Input Capture Function



0:0	General Purpose I/O
0:1	Capture on Rising Edge
1:0	Capture on Falling Edge
1:1	Capture on Either Edge

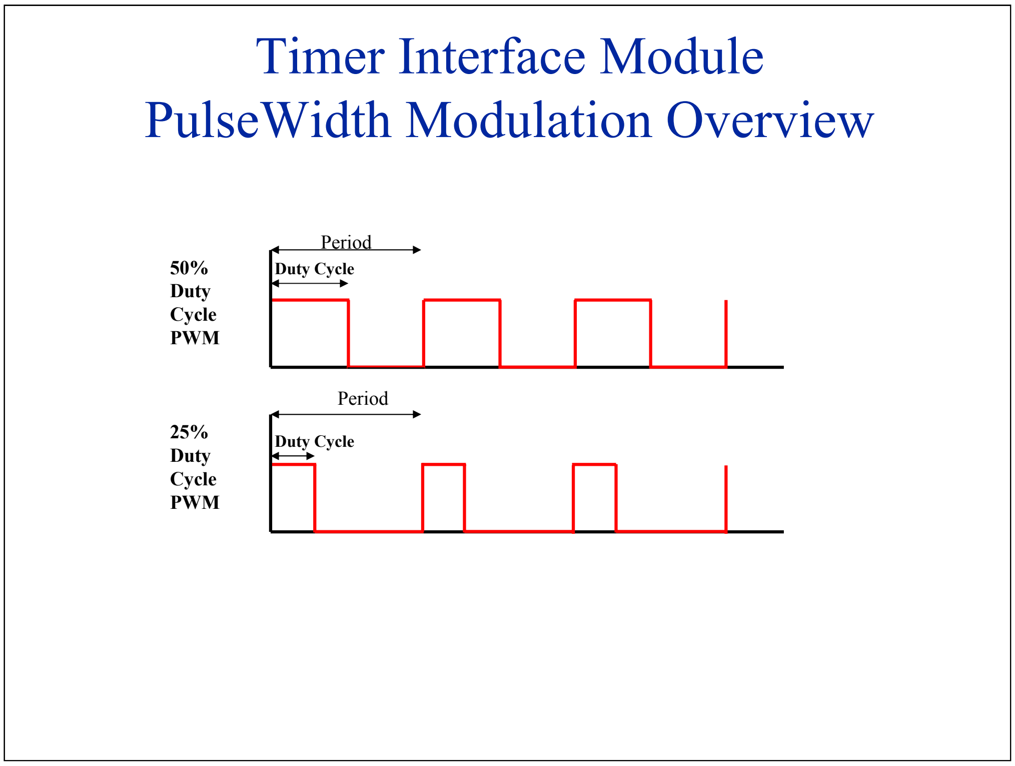
Let’s use Channel 1 to perform an input capture.

Once again, we will clear MS0B to zero to isolate the Channels 0 and 1 from each other and we’ll also clear MS1A to zero to select the input function.

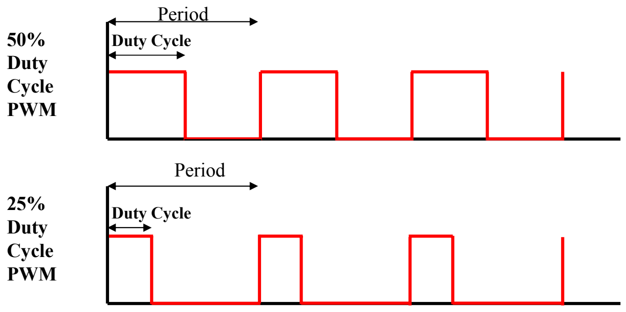
ELS1A & ELS1B are used to program the pin operation. Note the option to capture on either edge. It simplifies pulse width measurement, which reduces software overhead.

Input Capture is used to measure timed signals. The timer channel register TCH1 is used to latch the value in the 16-bit counter.

When the programmed transition (ELS1B:ELS1A) occurs at the pin (TxCH1), the value in the 16-bit counter is latched into the timer channel register TCH1. An interrupt can optionally be generated at the transition. Typically, an interrupt service routine is used to measure and calculate the time between signal transitions, thereby determining either pulse or period widths. Knowing the value of two transitions, you can calculate the pulse or period width.



Timer Interface Module PulseWidth Modulation Overview



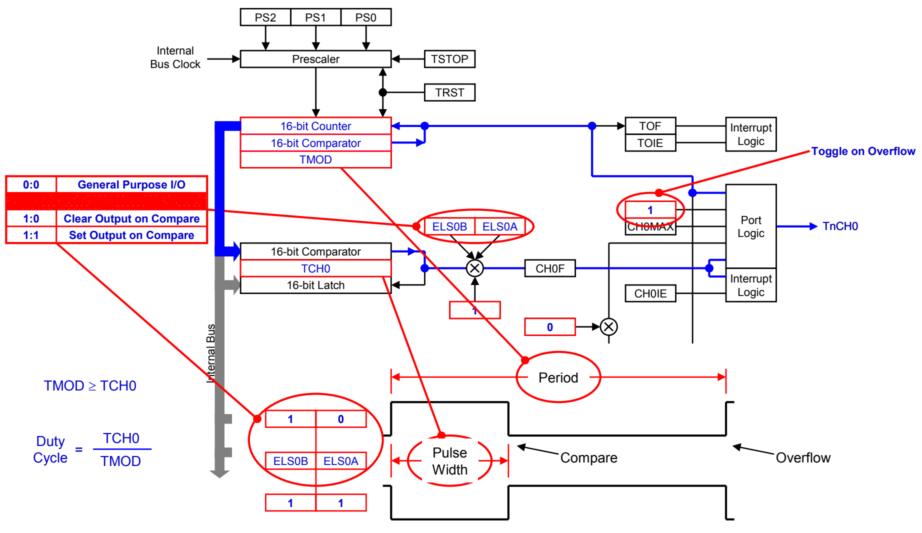
Pulse width modulation is used to generate a waveform with a fixed period and variable duty cycle. The duty cycle is the relative time spent in the high and low portions of the signal period. Pulse width modulators can have different speeds and resolution. The speed is defined by the period and the resolution is defined by the number of discrete steps the duty cycle can be set to. An 8-bit PWM allows the duty cycle to be specified in 256 steps, or 2 raised to the 8th power. For example, a non-zero duty as little as 0.39%, or 1 divided by 256, could be specified with an 8-bit PWM.

A common use for pulse width modulators is digital-to-analog conversions using some external filtering. The generated analog voltage is proportional to the duty cycle. Theoretically, a 50% duty cycle would generate approximately half of the maximum analog voltage and a 25% duty cycle would generate a quarter of the maximum analog voltage.

Pulse width modulators are also commonly used to control a motor and to control battery charging current.

Next, we'll take a closer look at PWM functions, including unbuffered pulse width modulation and buffered pulse width modulation.

Timer Interface Module
Unbuffered Pulse Width Modulation



Unbuffered PWM Block Diagram - Part 1

First, let’s cover the procedures in setting up and implementing unbuffered pulse width modulation and then follow with buffered PWM. Along the way, we’ll also identify differences between the two types.

Unbuffered PWM operates like the output compare function. Let's demonstrate it using Timer Channel 0.

Just as with the Output Compare function, clearing MS0B to zero isolates the two channels from each other, and setting MS0A to one selects the output function. Similarly, ELS0A and ELS0B are used to program the pin operation.

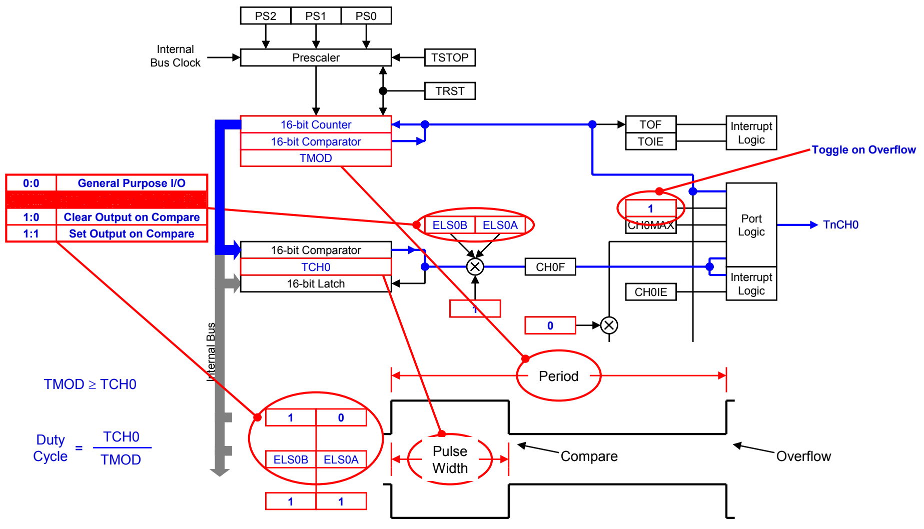
Recall that a PWM signal consists of a Period and a Pulse Width. For maximum flexibility, both of these parameters can be programmable.

To generate the signal as shown, we'll configure ELS0B:ELS0A as 1:0, or "Clear Output on Compare".

We then use the Timer Channel Register (TCH0) to determine the pulse width.

The Counter Modulo Register determines the period by forcing a counter overflow and resetting the Counter Register when they’re equal (ie., TCNT (16-bit counter) = TMOD).

Timer Interface Module
Unbuffered Pulse Width Modulation



Unbuffered PWM Block Diagram - Part 2

For proper operation, TMOD should be greater than or equal to TCH0. The duty cycle is simply TCH0 divided by TMOD.

As mentioned earlier, operation is identical to the Output Compare function. The value stored in TCH0 is constantly being compared to the 16-bit timer.

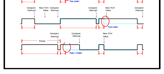
When the value in the 16-bit Counter equals that of TCH0, the programmed output signal transition (ie., "Clear Output on Compare") is forced on the signal pin TxCH0.

An additional bit that must be programmed is the TOV0 bit, "Toggle on Overflow". Setting TOV0 to one will toggle the output every time the 16-bit Counter overflows.

When the value in the 16-bit Counter equals that of TMOD, an overflow is forced, resetting the Counter to zero, and since TOV=1, the output pin is toggled.

An alternative waveform option is possible by programming ELS0B:ELS0A as 1:1, or "Set Output on Compare".

Please note that the "Toggle Output on Compare" option should not be selected, as this can result in not knowing which of the two possible output waveforms will be generated. Additional care must be taken when using Unbuffered PWM, as the next examples will illustrate.



Unbuffered PWM Pulses- Part 1

To understand the limitation of using Unbuffered PWM, let's look at what happens when we change the duty cycle.

As discussed earlier, to generate a PWM waveform that looks like the one on this page, we need to set up the timer channel to clear the output upon compare, and enable the toggle output on counter overflow function. One should always use a timer interrupt service routine to manipulate the pulse width in order to avoid spurious output signals caused by unsynchronized writes to the timer channel registers. Our choices are the Output Compare interrupt and the Counter Overflow interrupt.

For the purposes of this first example, let's use the Counter Overflow interrupt.

Now, let's see what happens when we use the counter Overflow interrupt to change the PWM operation to a very small duty cycle. Please note that the duty cycle waveforms shown in the diagram are not drawn to scale.

While the interrupt service request occurs at the moment the output signal transitions, it will take time for the interrupt service routine to set up the timer channel for the next output compare operation. This delay is also dependent upon whether or not other interrupts are currently being serviced. So, in a very busy application, this delay could become virtually unpredictable. Naturally, one should always consider the worst possible case.

In the situation shown here, the overflow interrupt service routine takes too much time to calculate and set up for the next timer channel transition (ie., an output compare).

Since the value in the counter has already surpassed the new value in the timer channel register, no output compare operation is executed and the state of the output pin doesn't change.

This proceeds until the next counter overflow interrupt, at which time the output level is toggled; in this case, from high to low.

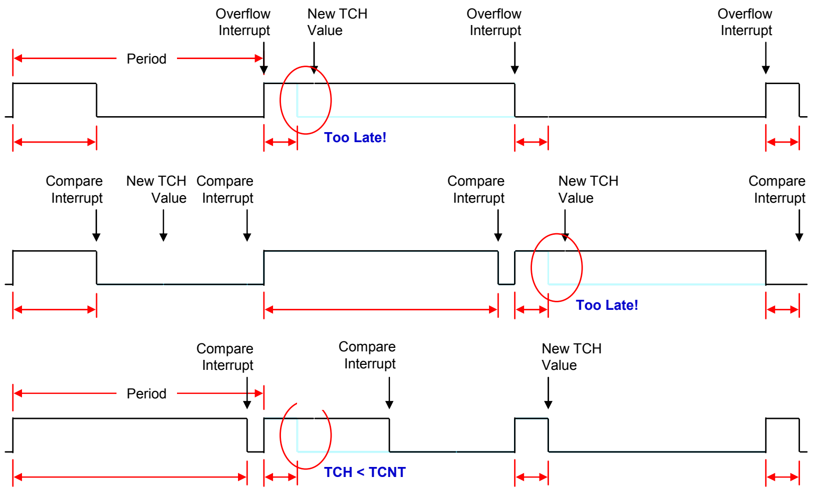
Note that the output compare operation is still configured with the new duty cycle value.

However, since the programmed operation is to clear the pin upon compare, the pin state doesn't change.

In fact, only after the next counter overflow operation do we finally get the duty cycle that we are looking for.

In summary, when using the Counter Overflow interrupt service routine to synchronize modifications to the PWM duty cycle, changing to a (application specific) very small duty cycle will cause (potentially) undesirable operation. The signal will experience 100% and then 0% duty cycles before settling on the programmed value. This will be true every time a change to a very small duty cycle is requested. So, maybe using the Counter Overflow interrupt is not our best choice.

Timer Interface Module
Unbuffered Pulse Width Modulation



Unbuffered PWM Pulses - Part 2

Instead, let's try using the output compare interrupt to synchronize PWM duty cycle changes. As you can see, this interrupt will occur before the counter overflow.

For the purposes of this example, let's change the PWM operation to a very large duty cycle.

Just as before, the interrupt service request occurs at the moment the output signal transitions, and it will take time for the interrupt service routine to set up the timer channel for the next output compare operation. But this delay should be well in advance of the next PWM period.

In fact, it's very likely that this calculation and set up will cause another Output Compare interrupt to occur within the same PWM period. This has no adverse affect on the output pin activity and this second interrupt service request can be safely ignored.
As expected, the Counter Overflow will toggle the output.

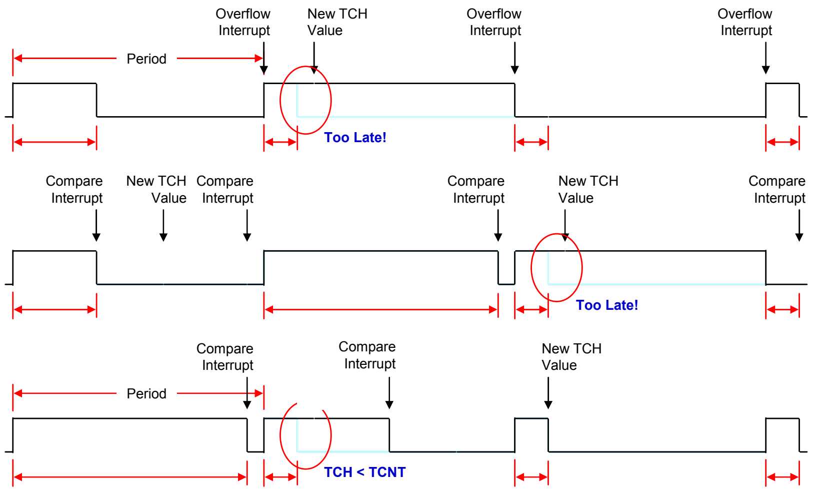
And the output compare will perform the correct transition.

But, what happens when we now reprogram the PWM for a very small duty cycle?

We discover that we have the same problem as before. Due to potential interrupt service routine response delays, the calculation for the new (very small) duty cycle could take an unacceptably long amount of time.

And just as before, we see the PWM operation transition through a 100% and then 0% duty cycle before settling on the programmed value. This will be true every time a change from a very large duty cycle to a very small duty cycle is requested. Close analysis of this situation may show this to be acceptable if the transition between a long pulse and a short pulse results in no perceivable glitch, assuming there is a 100% duty cycle pulse followed by a 0% duty cycle pulse inserted between them. However, the key is calculating the longest output compare interrupt service routine response time, which determines the conditions when this occurs. Depending on the application, a longer response time could make this kind of operation undesirable. Comparing the two interrupt sources, using the output compare interrupt only presents a potential problem when transitioning from very large to very small duty cycles. This is superior to using the counter overflow interrupt which will exhibit the above operation every time a transition is made to a small duty cycle, regardless of previous value.

Timer Interface Module
Unbuffered Pulse Width Modulation



Unbuffered PWM Pulses - Part 3

Let's investigate a possible solution to this scenario. Once again, we'll start with a very large duty cycle.

And we'll try to transition to a very small duty cycle.

Using the Output Compare interrupt, as we've discussed, the process of calculating the new duty cycle value may take too long.

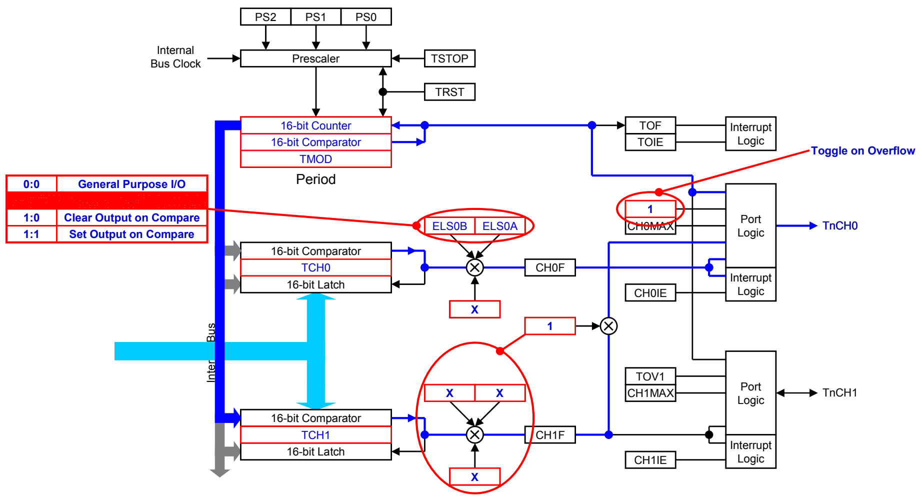
This situation can be detected by comparing the new value to either the current Counter value (probably just the lower 8-bits) or some other predetermined worst case value.

Instead of using the new, small duty cycle, an intermediate value will be written for one cycle which forces a 50% duty cycle waveform.

The next cycle (i.e., the Output Compare interrupt service routine) now programs the timer channel with the intended small duty cycle. The time required to perform this task is well in advance of the next PWM period.

And operation from here on is as desired. Now, the waveform transitions from 99% to 50% to 1%, which is likely acceptable in most applications.

Timer Interface Module
Buffered Pulse Width Modulation



For situations where unbuffered PWM will not work, you can try the Buffered mode which can remove any possibility of spurious output signals. It requires the use of two timer channel registers to control the signal driven on the TxCH0 pin. The TxCH1 pin cannot be used by the timer; however, it is completely available as a general purpose I/O port pin.

The two timer channels are functionally linked by setting MS0B to one.

Setting MS0B to one will force the TxCH0 pin to be an output, so the configuration of MS0A doesn't matter (ie., "don't care").

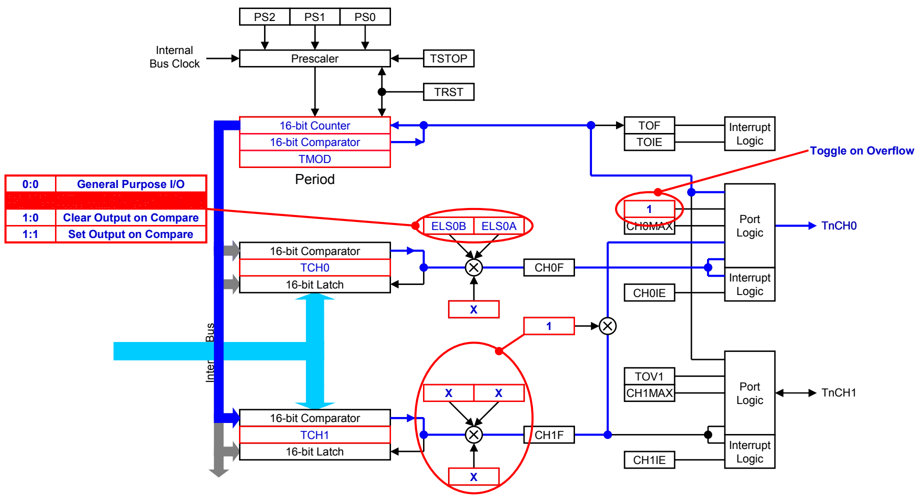
In addition, setting MS0B to one enables ELS0A & ELS0B to be used to program the pin operation, just as we've seen before.

Since the TxCH0 pin activity is controlled by ELS0A & ELS0B, the other timer channel control bits, ELS1A, ELS1B and MS1A, are "don't cares".

As in Unbuffered PWM mode, the value stored in the timer channel register, TCH0, determines the duty cycle.

And the value in TMOD dictates the PWM signal period, with the Counter being constantly compared to the value in TCH0.

Timer Interface Module
Buffered Pulse Width Modulation



When the value in the 16-bit Counter equals that of TCH0, the programmed output signal transition (ie., “Clear Output on Compare”) is forced on the signal pin TxCH0.

Once again, as in Unbuffered PWM, the TOV0 (or “Toggle on Overflow”) bit must be programmed. Setting TOV0 to one will toggle the output every time the 16-bit Counter overflows.

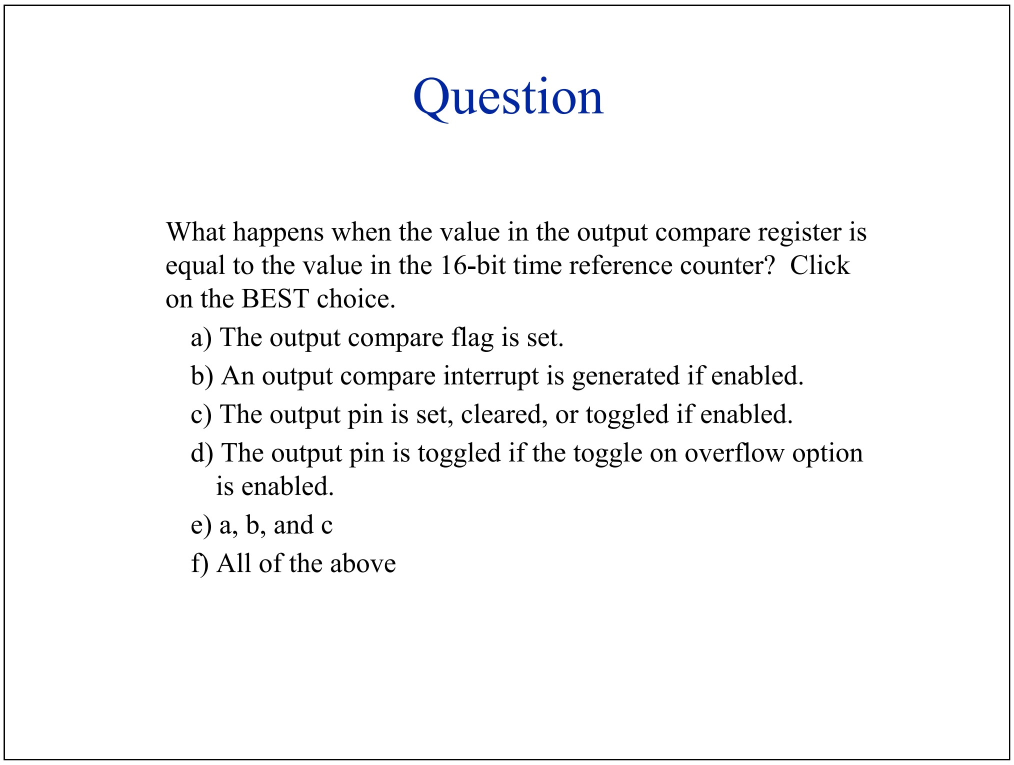
When the value in the 16-bit Counter equals that of TMOD, an overflow is forced, resetting the Counter to zero, and, since TOV=1, the output pin is toggled.

Next, to change the duty cycle, the software must program the next value into alternate timer channel registers. If TCH0 is currently active, then any change in PWM duty cycle is written into TCH1, and vice versa.

Duty cycle changes are synchronized with the next Counter Overflow, as determined by TMOD. Since, in our example, TCH0 is currently active and we’ve written a new value into TCH1, the logic will automatically begin using TCH1 for the next duty cycle.

The next change to the duty cycle will require a write to TCH0. User software will need to keep track of which Timer Channel register was written to last in order to determine which channel is to be written to next. But, since duty cycle changes are now automatically synchronized with the timer, this software no longer needs to reside within a timer interrupt service routine.

So, when the value in the 16-bit Counter equals that of TCH1, the programmed output signal transition (ie., “Clear Output on Compare”) is forced on the signal pin TxCH0. The desired PWM will be generated until a new duty cycle is programmed in the TCH1 or TCH0 registers.



Question

What happens when the value in the output compare register is equal to the value in the 16-bit time reference counter? Click on the BEST choice.

a) The output compare flag is set.

b) An output compare interrupt is generated if enabled.

c) The output pin is set, cleared, or toggled if enabled.

d) The output pin is toggled if the toggle on overflow option is enabled.

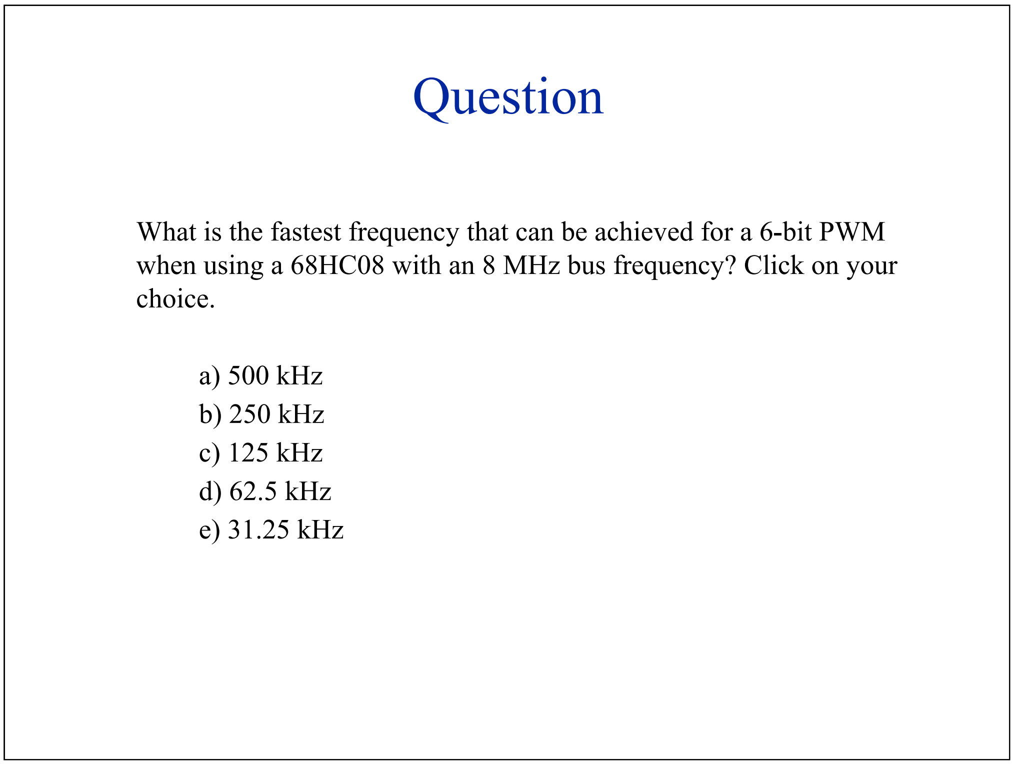
e) a, b, and c

f) All of the above

Let's complete this tutorial with a number of questions to check your understanding of the material.

What happens when the value in the output compare register is equal to the value in the 16-bit time reference counter? Click on your choice.

Answer: Actions specified in choices a, b, and c can occur when the output compare register value equals the reference counter value.



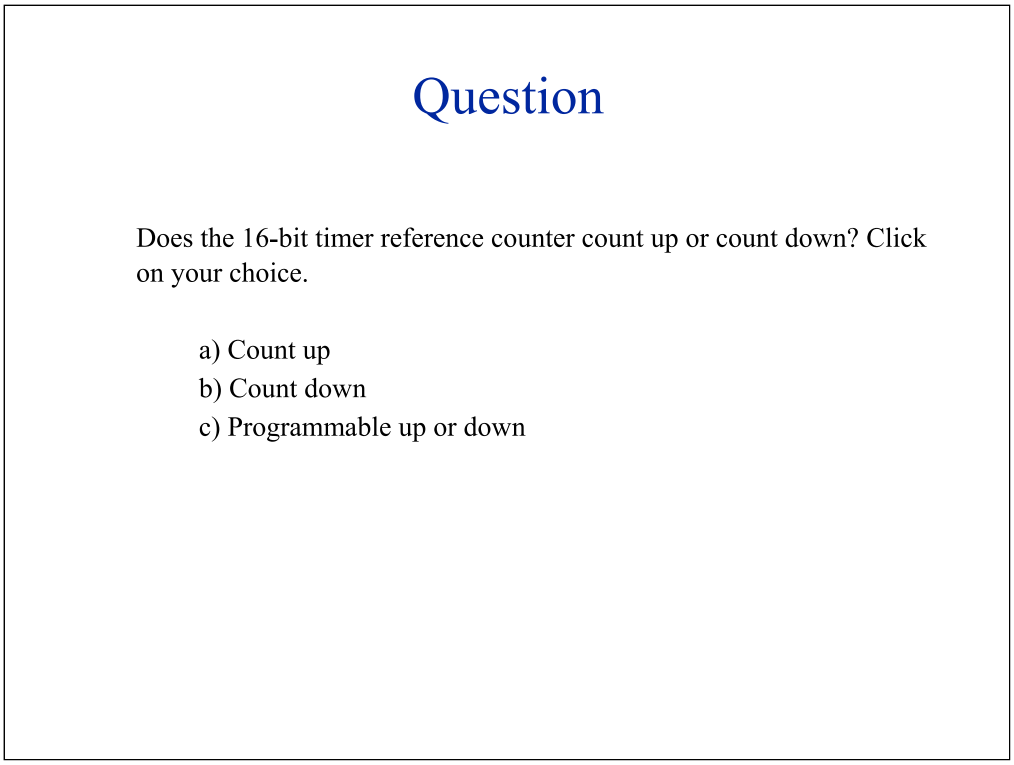
Question

What is the fastest frequency that can be achieved for a 6-bit PWM when using a 68HC08 with an 8 MHz bus frequency? Click on your choice.

a) 500 kHz
b) 250 kHz
c) 125 kHz
d) 62.5 kHz
e) 31.25 kHz

What is the fastest frequency that can be achieved for a 6-bit PWM when using a 68HC08 with an 8 MHz bus frequency? Click on your choice.

Answer:The frequency is 125 kHz, calculated as: bus clock/(2^6) = 8 MHz/64 = 125 kHz. A 6-bit PWM requires the counter to count at least 6 binary bits (2^6), or 64 cycles, otherwise you would not be able to get 6-bit resolution in setting the duty cycle.



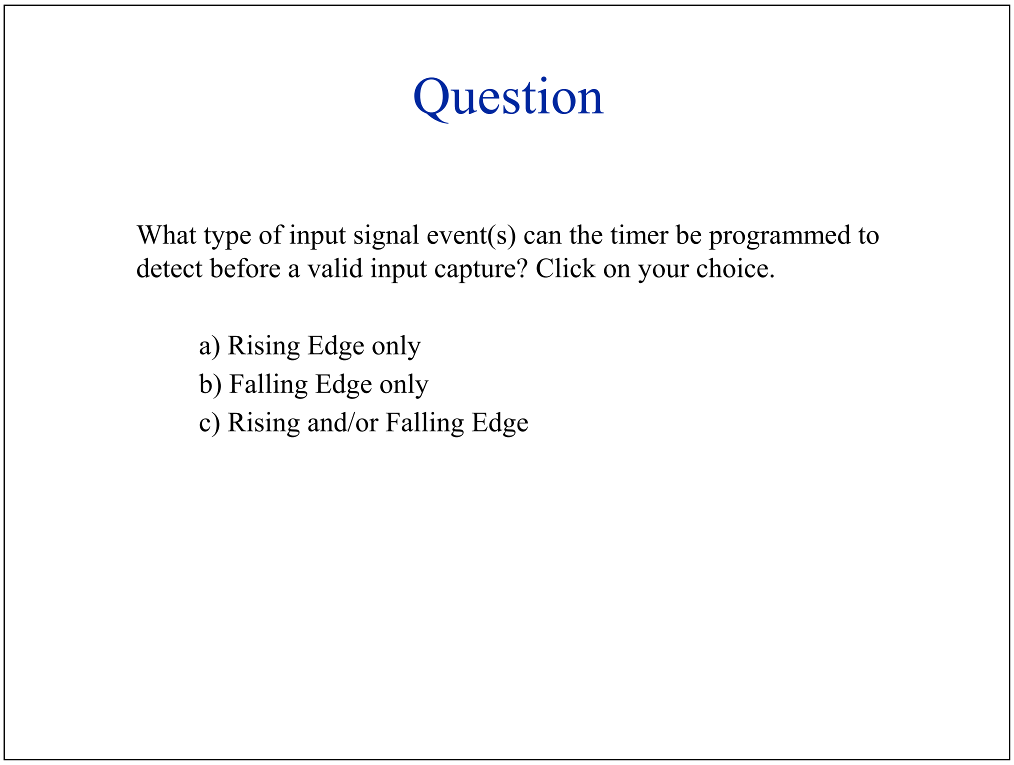
Question

Does the 16-bit timer reference counter count up or count down? Click on your choice.

a) Count up
b) Count down
c) Programmable up or down

Does the 16-bit timer reference counter count up or count down? Click on your choice.

Answer:The 16-bit timer counts up and only to a value loaded in the modulo comparator register and then it resets and restarts another count-up sequence.



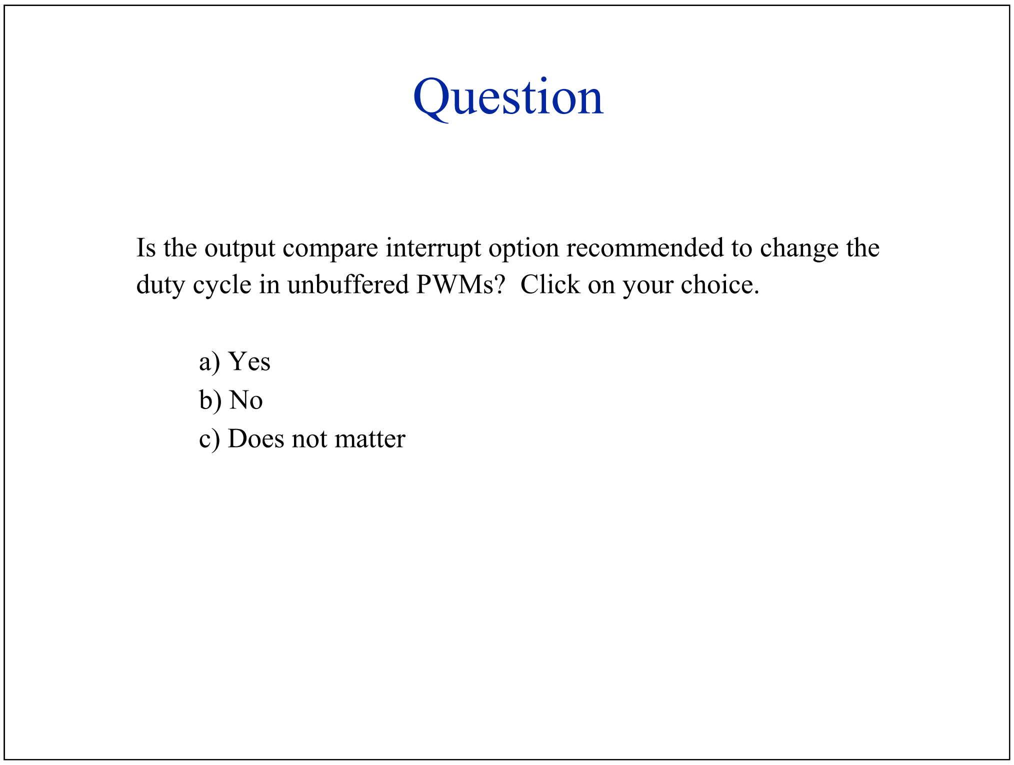
Question

What type of input signal event(s) can the timer be programmed to detect before a valid input capture? Click on your choice.

a) Rising Edge only
b) Falling Edge only
c) Rising and/or Falling Edge

What type of input signal event(s) can the timer be programmed to detect before a valid input capture? Click on your choice.

Answer: Either type of edge can be detected before a valid input capture.



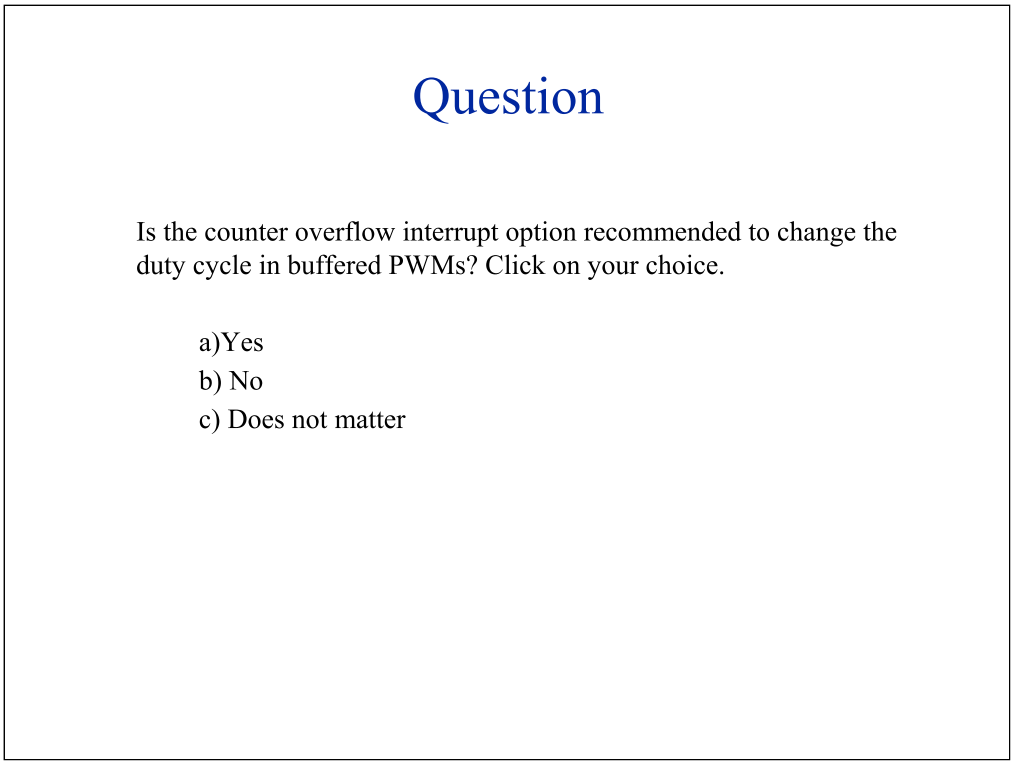
Question

Is the output compare interrupt option recommended to change the duty cycle in unbuffered PWMs? Click on your choice.

a) Yes
b) No
c) Does not matter

Is the output compare interrupt option recommended to change the duty cycle in unbuffered PWMs? Click on your choice.

Answer: Using the output compare interrupt gives more time than the counter overflow interrupt to set up the timer channel for the next output compare operation. This approach will work correctly except in cases where you need to move directly from a very high duty cycle to a very low duty cycle. In this case, you'd use buffered PWMs.



Question

Is the counter overflow interrupt option recommended to change the duty cycle in buffered PWMs? Click on your choice.

a)Yes
b) No
c) Does not matter

Is the counter overflow interrupt option recommended to change the duty cycle in buffered PWMs? Click on your choice.

Answer:The buffered PWM mode utilizes two timer channels and automatically synchronizes new duty cycle values at the end of the PWM period if the new duty cycle is written to the non-active output compare channel. Therefore, as long as the user writes new duty cycle values to alternate output compare channels, no interrupt service routine is required.

Tutorial Completion

- TIM Configuration
- Output Compare
- Input Capture
- Unbuffered PWM
- Buffered PWM

In this tutorial, you’ve had an opportunity to work with the 68HC08 Timer Interface Module. You’ve learned how to set up and perform output compare and input capture functions, as well as a few unbuffered and buffered pulse width modulation functions.