Timers

[8-bit] [16-bit] [Register Overview] [Modes] [Examples]

The AVR has different Timer types. Not all AVRs have all Timers, so look at the datasheet of your AVR before trying to use a timer it doesn't have...This description is based on the AT90S2313.

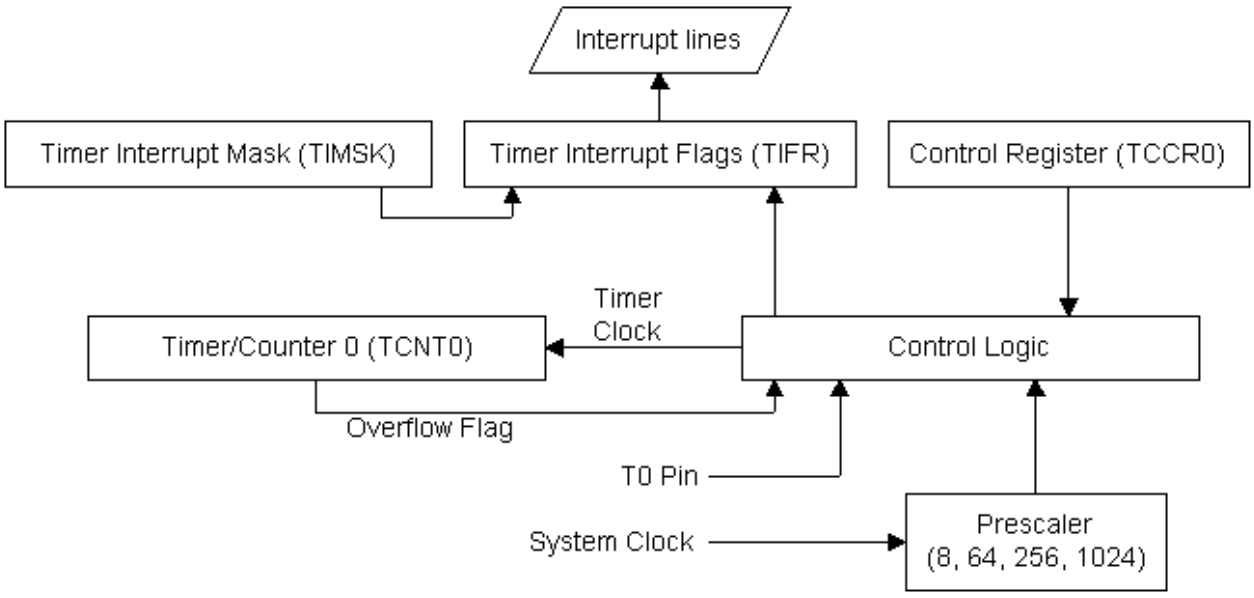
I will only describe the "simple" timer modes all timers have. Some AVRs have special timers which support many more modes than the ones described here, but they are also a bit more difficult to handle, and as this is a beginners' site, I will not explain them here.

The timers basically only count clock cycles. The timer clock can be equal to the system clock (from the crystal or whatever clocking option is used) or it can be slowed down by the prescaler first. When using the prescaler you can achieve greater timer values, while precision goes down.

The prescaler can be set to 8, 64, 256 or 1024 compared to the system clock. An AVR at 8 MHz and a timer prescaler can count (when using a 16-bit timer) (0xFFFF + 1) * 1024 clock cycles = 67108864 clock cycles which is 8.388608 seconds. As the prescaler increments the timer every 1024 clock cycles, the resolution is 1024 clock cycles as well: 1024 clock cycles = 0.000128 seconds compared to 0.125µs resolution and a range of 0.008192 seconds without prescaler. It's also possible to use an external pin for the timer clock or stop the timer via the prescaler.

The timers are realized as up-counters. Here's a diagram of the basic timer hardware. Don't panic, I'll explain the registers below.

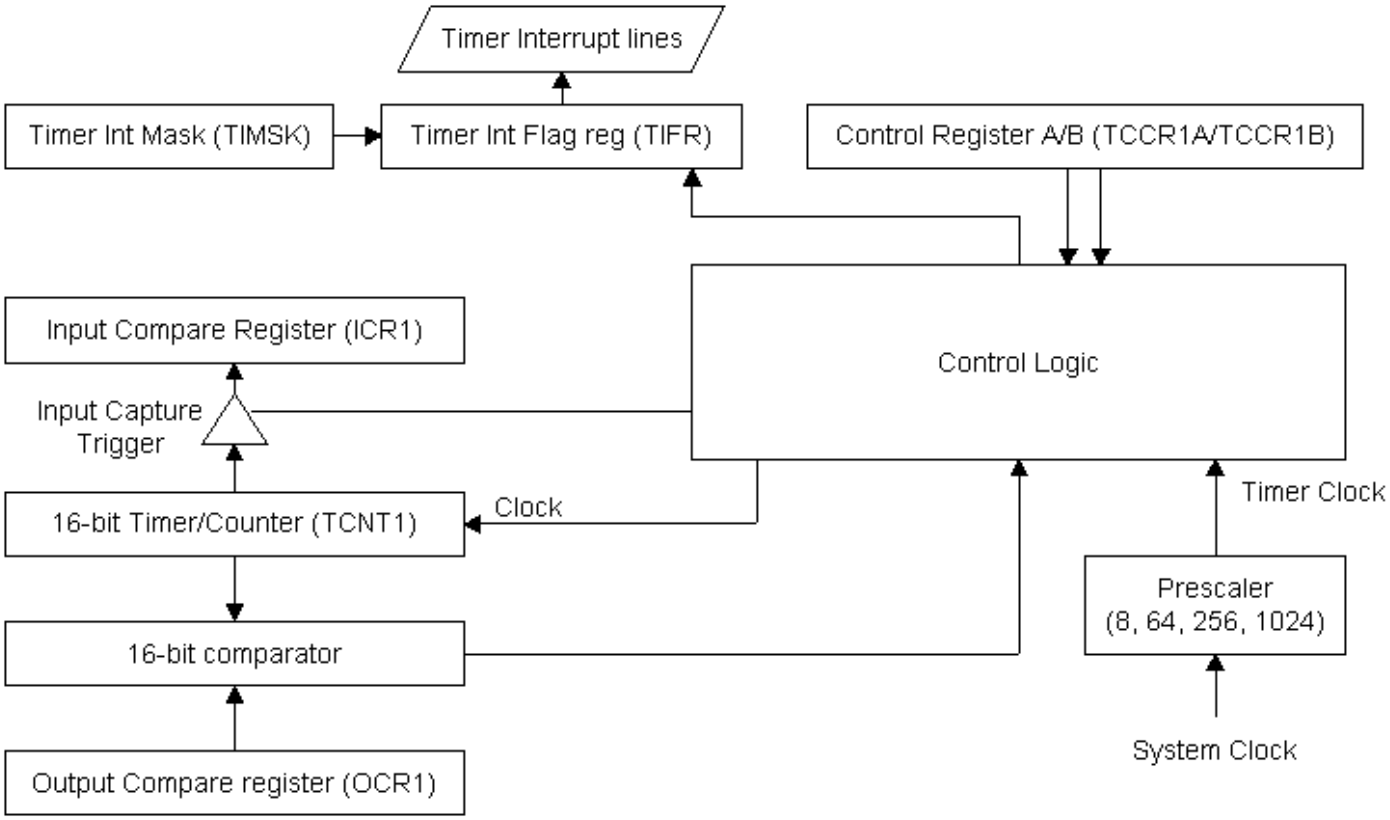
The 8-bit Timer:



The 8-bit timer is pretty simple: The timer clock (from System Clock, prescaled System Clock or External Pin T0) counts up the Timer/Counter Register (TCNT0). When it rolls over (0xFF -> 0x00) the Overflow Flag is set and the Timer/Counter 1 Overflow Interrupt Flag is set. If the corresponding bit in TIMSK (Timer Interrupt Mask Register) is set (in

this case the bit is named "TOIE0") and global Interrupts are enabled, the micro will jump to the corresponding interrupt vector (in the 2313 this is vector number 7).

The 16-bit Timer
is a little more complex, as it has more modes of operation:



Register Overview:

[TCNT1] [TCCR1A / TCCR1B] [OCR1A] [ICR1] [TIMSK and TIFR]

The register names vary from timer to timer and from AVR to AVR! Have a look at the datasheet of the AVR you're using first.

Important note: All 16-bit registers have can only be accessed one byte at a time. To ensure precise timing, a 16-bit temporary register is used when accessing the timer registers.

Write:
When writing the high byte (e.g. TCNT1H), the data is placed in the TEMP register. When the low byte is written, the data is transferred to the actual registers simultaneously. So the high byte must be written first to perform a true 16-bit write.

Read:
When reading the low byte, the high byte is read from TCNT1 simultaneously and can be read afterwards. So when reading, access the low byte first for a true 16-bit read.

TCNT

Most important is the Timer/Counter Register (TCNT1) itself. This is what all timer modes base on. It counts System Clock ticks, prescaled system clock or from the external pin.

TCCR

The Timer/Counter Control register is used to set the timer mode, prescaler and other options.

TCCR1A:

Bit 7							Bit 0
COM1A1	COM1A0	---	---	---	---	PWM11	PWM10

Here's what the individual bits do:

COM1A1/COM1A0: Compare Output Mode bits 1/0; These bits control if and how the Compare Output pin is connected to Timer1.

COM1A1	COM1A0	Compare Output Mode
0	0	Disconnect Pin OC1 from Timer/Counter 1
0	1	Toggle OC1 on compare match
1	0	Clear OC1 on compare match
1	1	Set OC1 on compare match

With these bit you can connect the OC1 Pin to the Timer and generate pulses based on the timer. It's further described below.

PWM11/PWM10: Pulse Width Modulator select bits; These bits select if Timer1 is a PWM and it's resolution from 8 to 10 bits:

PWM11	PWM10	PWM Mode
0	0	PWM operation disabled
0	1	Timer/Counter 1 is an 8-bit PWM
1	0	Timer/Counter 1 is a 9-bit PWM
1	1	Timer/Counter 1 is a 10-bit PWM

The PWM mode of Timer1 is dsecribed below.

TCCR1B:

Bit 7							Bit 0

ICNC1	ICES1	---	---	CTC1	CS12	CS11	CS10

ICNC1: Input Capture Noise Canceler; If set, the Noise Canceler on the ICP pin is activated. It will trigger the input capture after 4 equal samples. The edge to be triggered on is selected by the ICES1 bit.

ICES1: Input Capture Edge Select;

When cleared, the contents of TCNT1 are transferred to ICR (Input Capture Register) on the falling edge of the ICP pin.

If set, the contents of TCNT1 are transferred on the rising edge of the ICP pin.

CTC1: Clear Timer/Counter 1 on Compare Match; If set, the TCNT1 register is cleared on compare match. Use this bit to create repeated Interrupts after a certain time, e.g. to handle button debouncing or other frequently occuring events. Timer 1 is also used in normal mode, remember to clear this bit when leaving compare match mode if it was set. Otherwise the timer will never overflow and the timing is corrupted.

CS12..10: Clock Select bits; These three bits control the prescaler of timer/counter 1 and the connection to an external clock on Pin T1.

CS12	CS11	CS10	Mode Description
0	0	0	Stop Timer/Counter 1
0	0	1	No Prescaler (Timer Clock = System Clock)
0	1	0	divide clock by 8
0	1	1	divide clock by 64
1	0	0	divide clock by 256
1	0	1	divide clock by 1024
1	1	0	increment timer 1 on T1 Pin falling edge
1	1	1	increment timer 1 on T1 Pin rising edge

OCR1

The Output Compare register can be used to generate an Interrupt after the number of clock ticks written to it. It is permanently compared to TCNT1. When both match, the compare match interrupt is triggered. If the time between interrupts is supposed to be equal every time, the CTC bit has to be set (TCCR1B). It is a 16-bit register (see note at the beginning of the register section).

ICR1

The Input Capture register can be used to measure the time between pulses on the external ICP pin (Input Capture Pin). How this pin is connected to ICR is set with the ICNC and ICES bits in TCCR1A. When the edge selected is detected on the ICP, the contents of TCNT1 are transferred to the ICR and an interrupt is triggered.

TIMSK and TIFR

The Timer Interrupt Mask Register (TIMSK) and Timer Interrupt Flag (TIFR) Register are used to control which interrupts are "valid" by setting their bits in TIMSK and to determine which interrupts are currently pending (TIFR).

Bit 7							Bit 0
TOIE1	OCIE1A	---	---	TICIE1	---	TOIE0	---

TOIE1: Timer Overflow Interrupt Enable (Timer 1); If this bit is set and if global interrupts are enabled, the micro will jump to the Timer Overflow 1 interrupt vector upon Timer 1 Overflow.

OCIE1A: Output Compare Interrupt Enable 1 A; If set and if global Interrupts are enabled, the micro will jump to the Output Compare A Interrupt vetor upon compare match.

TICIE1: Timer 1 Input Capture Interrupt Enable; If set and if global Interrupts are enabled, the micro will jump to the Input Capture Interrupt vector upon an Input Capture event.

TOIE0: Timer Overflow Interrupt Enable (Timer 0); Same as TOIE1, but for the 8-bit Timer 0.

TIFR is not really necessary for controlling and using the timers. It holds the Timer Interrupt Flags corresponding to their enable bits in TIMSK. If an Interrupt is not enabled your code can check TIFR to determine whether an interrupt has occured and clear the interrupt flags. Clearing the interrupt flags is usually done by writing a logical 1 to them (see datasheet).

Timer Modes

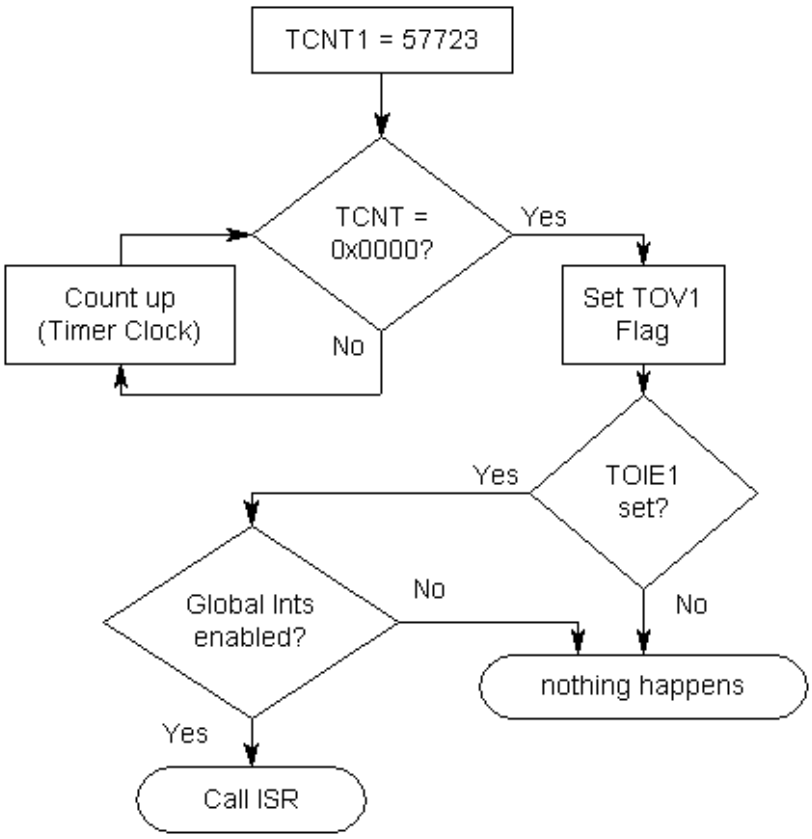
[Normal Mode] [Output Compare Mode] [Input Capture Mode] [PWM Mode]

Normal Mode:

In normal mode, TCNT1 counts up and triggers the Timer/Counter 1 Overflow interrupt when it rolls over from 0xFFFF to 0x0000. Quite often, beginners assume that they can just load the desired number of clock ticks into TCNT1 and wait for the interrupt (that's what I did...). This would be true if the timer counted downwards, but as it counts upwards, you have to load 0x0000 - (timer value) into TCNT1. Assuming a system clock of 8 MHz and a desired timer of 1 second, you need 8 Million System clock cycles. As this is too big for the 16-bit range of the timer, set the prescaler to 1024 (256 is possible as well).

8,000,000/1024 = 7812.5 ~ 7813
0x0000 - 7813 = 57723 <- Value for TCNT1 which will result in an overflow after 1 second (1.000064 seconds as we rounded up before)

So we now know the value we have to write to the TCNT1 register. So? What else? This is not enough to trigger the interrupt after one second. We also have to enable the corresponding interrupt and the global interrupt enable bit. Here's a flow chart of what happens:

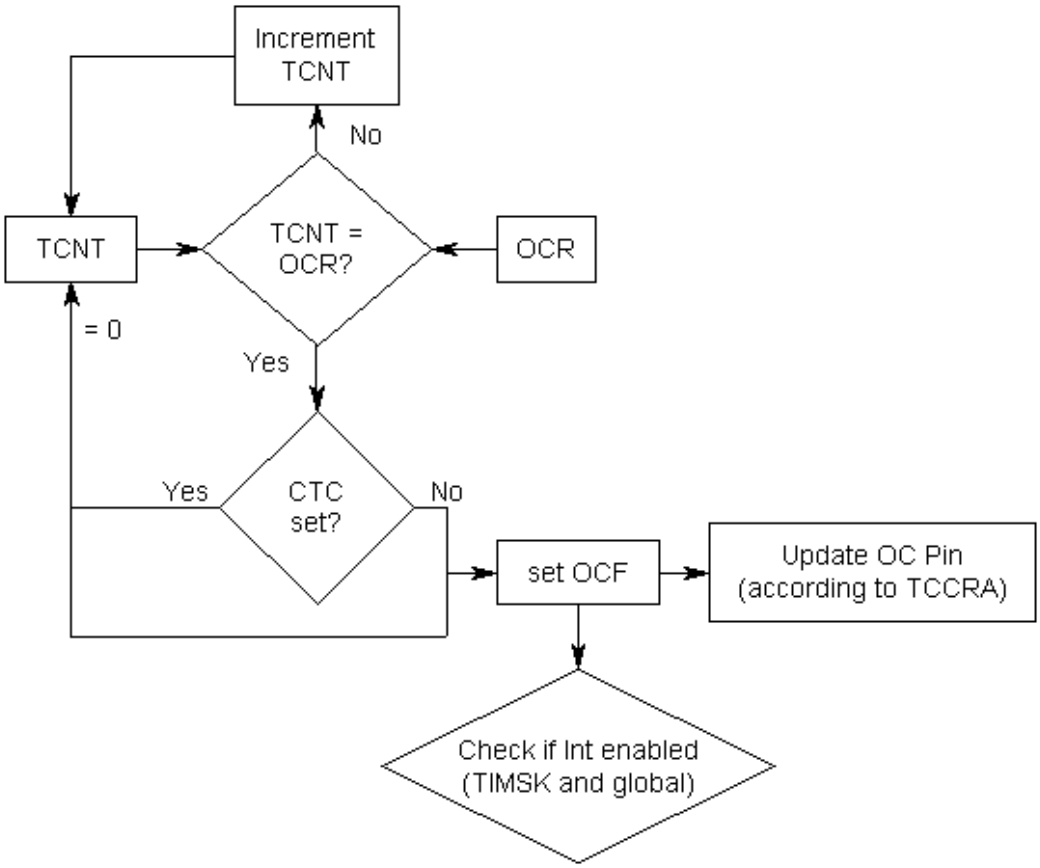


The only steps you have to do are:

- set prescaler to 1024 (set bits CS12 and CS10 in TCCR1B)
- write 57723 to TCNT1
- enable TOIE1 in TIMSK
- enable global interrupt bit in SREG
- wait. Or do anything else. All the counting and checking flags is done in hardware.

Output Compare Mode

The Output Compare mode is used to perform repeated timing. The value in TCNT1 (which is counting up if not stopped by the prescaler select bits) is permanently compared to the value in OCR1A. When these values are equal to each other, the Output Compare Interrupt Flag (OCF in TIFR) is set and an ISR can be called. By setting the CTC1 bit in TCCR1B, the timer can be automatically cleared upon compare match.



The flow chart should show an arrow from CTC1 set? "Yes" to set OCF instead of a line, but somehow the Flow Charting Program didn't think that was a good idea. Bad luck.

Let's discuss a small example:We want the Timer to fire an int every 10ms. At 8 MHz that's 80,000 clock cycles, so we need a prescaler (out of 16-bit range).

In this case, 8 is enough. Don't use more, as that would just pull down accuracy. With a prescaler of 8, we need to count up to 10,000.As the value of TCNT1 is permanently compared to OCR1 and TCNT1 is up-counting, the value we need to write to OCR is acutally 10,000 and not 0x0000-10,000, as it would be when using the timer in normal mode.

Also, we need to set CTC1: If we didn't, the timer would keep on counting after reaching 10,000, roll over and then fire the next int when reaching 10,000, which would then occur after 0xFFFF*8 clock cycles. That's after 0.065536 seconds. Not after 10ms. If CTC1 is set, TCNT1 is cleared after compare match, so it will count from 0 to 10,000 again with out rolling over first.

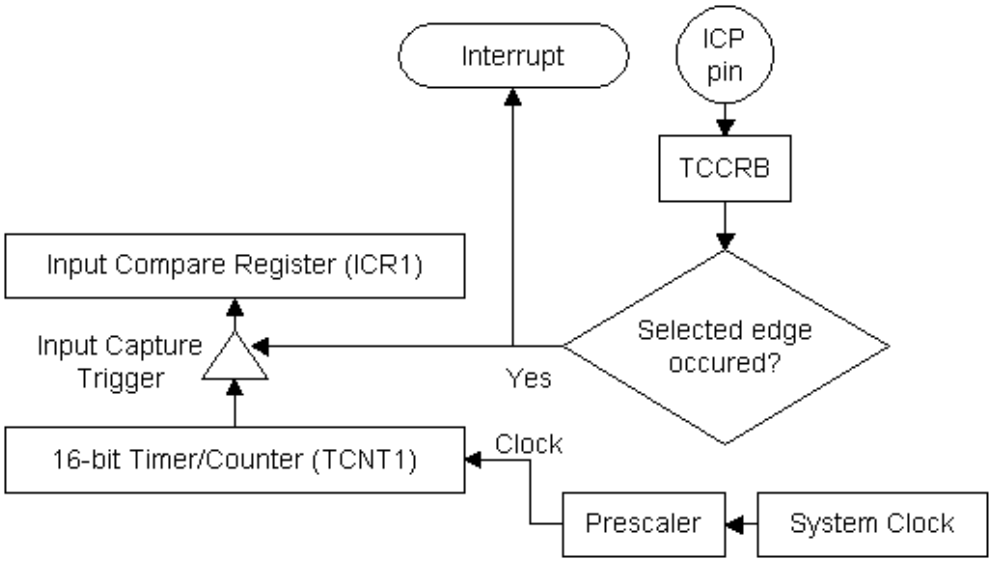
What is to be done when those 10ms Interrupts occur? That depends on the application. If code is to be executed, the corresponding interrupt enable bit has to be set, in this case it is OCIE1A in TIMSK. Also check that global interrupts are enabled.

If the OC1 (Output Compare 1) pin is to be used, specify the mode in TCCR1A. You can set, clear or toggle the pin. If you decide that you want to toggle it, think about your timing twice: If you want a normal pulse which occurs every 10ms, the timer cycle must be 5ms: 5ms -> toggle on -> 5ms -> toggle off. With the 10ms example above and OC1 set up to be toggled, the pulse would have a cycle time of 20ms.

Input Capture Mode

The Input Capture Mode can be used to measure the time between two edges on the ICP pin (Input Capture Pin). Some external circuits make pulses which can be used in just that way. Or you can measure the rpm of a motor with it. You can either set it up to measure time between rising or falling edges on the pin. So if you change this setting within the ISR you can measure the length of a pulse. Combine these two methods and you have completely analysed a pulse. How it works?

Here's a flow chart of its basic functionality:



You see that it's actually pretty simple. I left out the low level stuff such as Interrupt validation (enabled/global enable), as you should understand that by now. The contents of TCNT1 are transferred to ICR1 when the selected edge occurs on the Input Capture Pin and an ISR can be called in order to clear TCNT1 or set it to a specific value. The ISR can also change the egde which is used to generate the next interrupt.

You can measure the length of a pulse if you change the edge select bit from within the ISR. This can be done the following way:

Set the ICES (Input Capture Edge Select) bit to 1 (detect rising edge)

When the ISR occurs, set TCNT1 to zero and set ICES to 1 to detect negative egde

When the next ISR is called, the pin changed from high to low. The ICR1 now contains the number of (prescaled) cycles the pin was high. If the ISR again sets the edge to be detected to rising (ICES=1), the low pulse time is measured. Now we have the high time AND the low time: We can calculate the total cycle time and the duty cycle.

It's also possible to connect the Analog Comparator to the input capture trigger line. That means that you can use the Analog Comparator output to measure analog signal frequencys or other data sources which need an analog comparator for timing analysis. See the Analog Comparator page for more.

PWM Mode

The Pulse Width Modulator (PWM) Mode of the 16-bit timer is the most complex one of the timer modes available. That's why it's down here.

The PWM can be set up to have a resolution of either 8, 9 or 10 bits. The resolution has a direct effect on the PWM frequency (The time between two PWM cycles) and is selected via the PWM11 and PWM10 bits in TCCR1A. Here's a table showing how the resolution select bits act. Right now the TOP value might disturb you but you'll see what it's there for. The PWM frequency show the PWM frequency in relation to the timer clock (which can be prescaled) and NOT the system clock.

PWM11	PWM10	Resolution	TOP-value	PWM Frequency
0	0	PWM function disabled		
0	1	8 bits	$00FF	fclock/510
1	0	9 bits	$01FF	fclock/1022
1	1	10 bits	$03FF	fclock/2046

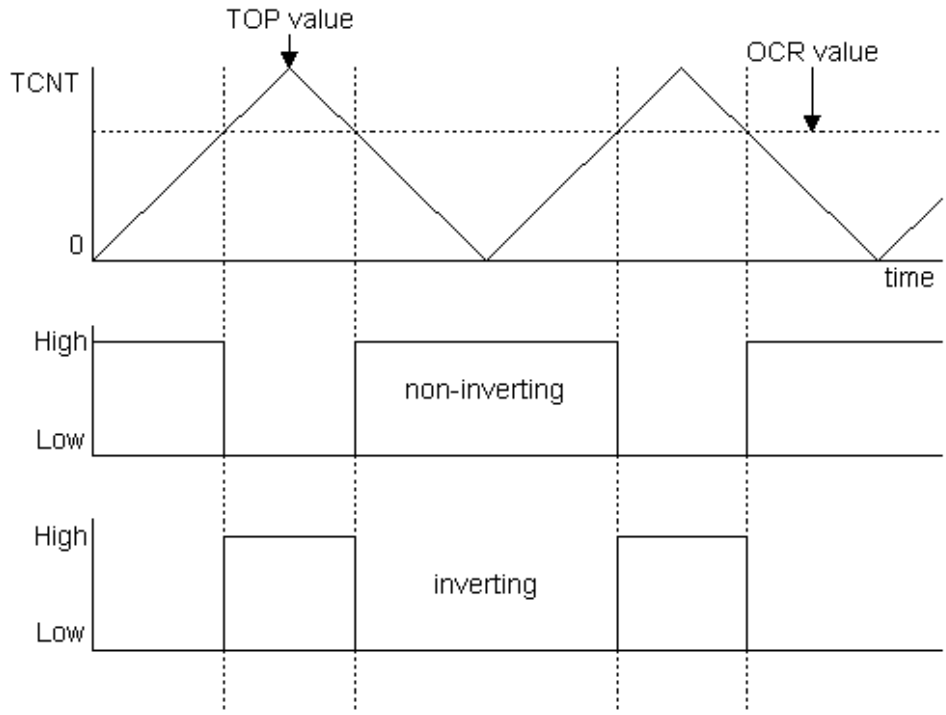
To understand the next possible PWM settings, I should explain how the PWM mode works. The PWM is an enhanced Output Compare Mode. In this mode, the timer can also count down, as opposed to the other modes which only use an up-counting timer. In PWM mode, the timer counts up until it reaches the TOP value (which is also the resolution of the timer and has effect on the frequency).

When the TCNT1 contents are equal to the OCR1 value, the corresponding output pin is set or cleared, depending on the selected PWM mode: You can select a normal and an inverted PWM. This is selected with the COM1A1 and COM1A0 bits (TCCR1A register). The possible settings are:

COM1A1	COM1A0	Effect:
0	0	PWM disabled
0	1	PWM disabled
1	0	Non-inverting PWM
1	1	inverting PWM

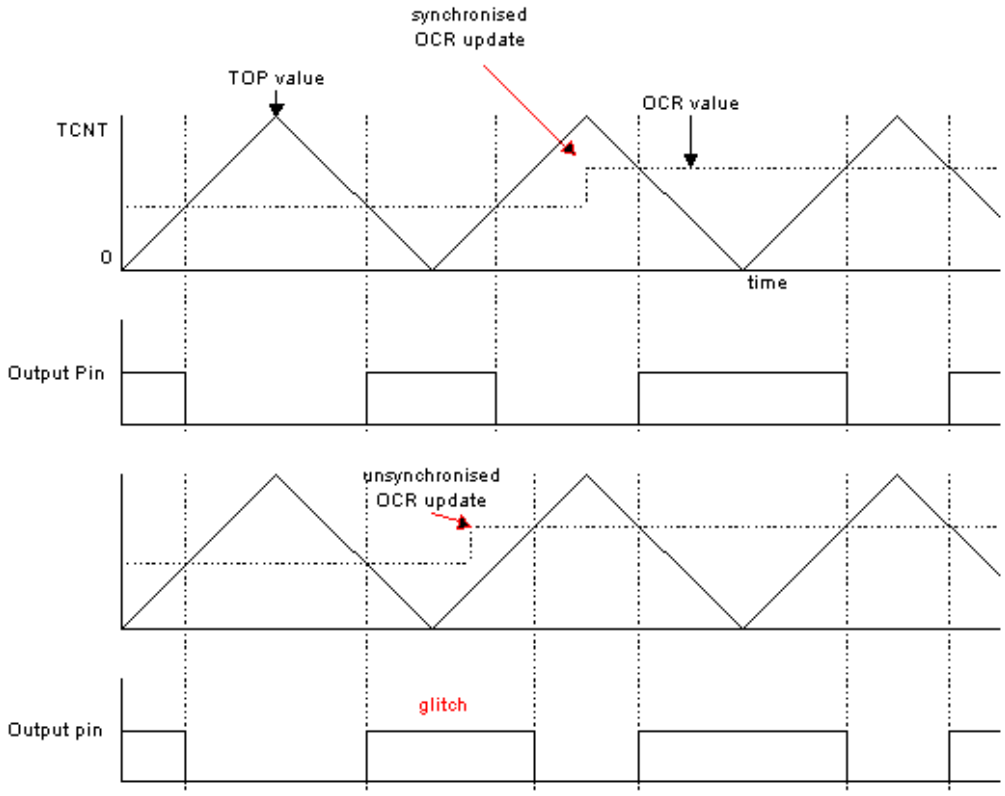
Non-inverted PWM means that the Output Compare Pin is CLEARED when the timer is up-counting and reaches the OCR1 value. When the timer reaches the TOP value, it switches to down-counting and the Output Compare Pin is SET when the timer value matches the OCR1 value.

Inverted PWM is, of course, the opposite: The Output Compare Pin is set upon an up-counting match and cleared when the down-couting timer matches the OCR1 value. Here are two diagrams showing what this looks like:



The reason why you can select between inverting and non-inverting pwm is that some external hardware might need an active-low pwm signal. Having the option to invert the PWM signal in hardware saves code space and processing time.

The PWM is also glitch-free. A glitch can occur when the OCR1 value is changed: Imagine the PWM counting down to 0. After the pin was set, the OCR1 value is changed to some other value. The next pulse has an undefined length because only the second half of the pulse had the specified new length. That's why the PWM automatically writes the new value of OCR1 upon reaching the TOP value and therefore prevents glitches.



Typical applications for the PWM are motor speed controlling, driving LEDs at variable brightness and so on. Make sure you have appropriate drivers and protection circuitry if you're using motors!

Some Examples...

Some simple examples can also be found here...

- Setting up a timer

- Flashing LED using the Timer Overflow Interrupt and the Output compare mode

- Pulse Width Modulated LED demo with two timers