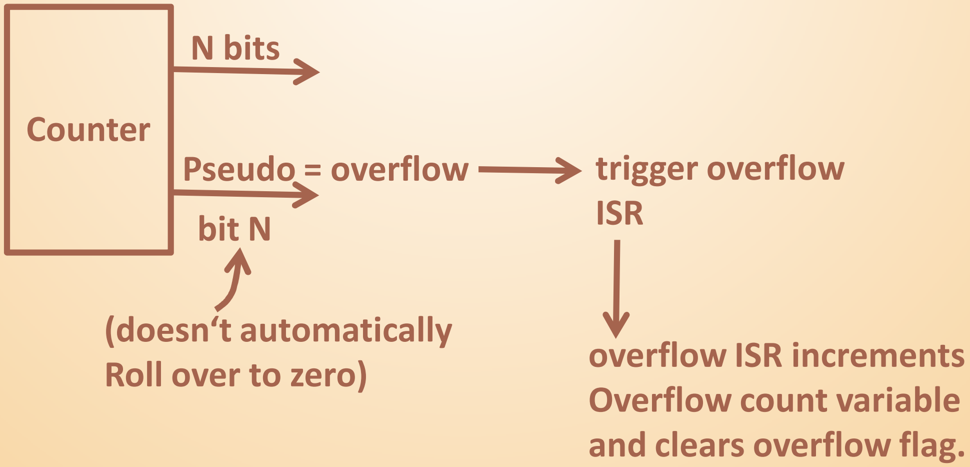
Extending Timer Through Software Using Overflow Bit

- The effecter timer/counter range may be extended using software.
- Timer/counters typically have overflow bits.
  - Can be used to double the range of the counter for one go-through, though it doesn't roll-over to zero automatically like other bits
  - Can be used with software that counts count timer/counter overflows
    - Total count = #overflows*(max count + 1) + counter register
    - If using interrupt, interrupt hardware must reset overflow flag before the next counter/timer register roller over
    - If using polling, software must check and clear overflow flag least once per timer/counter rollover period



Counter

N bits

Pseudo = overflow

bit N

(doesn't automatically Roll over to zero)

trigger overflow ISR

overflow ISR increments Overflow count variable and clears overflow flag.

Multi-register access problems – pt1

- There is an atomicity problem when using overflows in the manner described above.This problem occurs when an overflow occurs between accessing the timer register and accessing the overflow status, and it occurs no matter in what order the register and the status are accessed. The problem is that when a timer overflow occurs between the two accesses, the data read in those two accesses becomes inconsistent. One example of this is the following sequence:
  - Read timer register (SAY 255)
  - Timer overflows
  - Read overflow status (now 1) ->>> Interpreted value is 1_11111111
- An example of the opposite order is this:
  - Read overflow status SAY 0
  - Timer overflows
  - Read timer register SAY 2
- In both cases, the combined data of the overflow state and the timer register yield a corrupted value, much the same as the corrupted values seen in the discussion on interrupts. The timer overflow is not an interrupt (although it may generate one), but like an interrupt it results in an asynchronous modification of the data being accessed, unknown to the program.

- Pasted from http://www.scriptoriumdesigns.com/embedded/timers.php

Multi-register access problems – pt2

- Luckily, we can detect such an event and correct for it. The timer register value after an overflow is less than the value before the overflow, which is not the case in the normal act of reading the timer register two times in succession. Similarly, the overflow status after an overflow is different than the status before an overflow. We can use these facts to check if the potentially corrupting situation has occurred, and if we detect this, we can immediately access the data again. This second access of the data, assuming it is done in a timely fashion, will always be valid because the timer cannot overflow again so soon after a previous overflow. Note that what we are detecting is not a corruption of data –we cannot detect that with certainty. All we can detect is that an overflow happened at some point in the sequence of accesses. The timing of the overflow may have resulted in corruption, or there may have been no corruption, but since we cannot know that the data accesses were uncorrupted, we perform the accesses again to obtain a guarantee of uncorrupted data. This is a technique worth remembering in any case where you are accessing multiple pieces of data that can be changing as they are being accessed.
  - Read timer register
  - Read overflow status (timer may have overflowed after reading timer register)
  - Compare current timer register with value from #1
  - If current value is less than first value, an overflow occurred sometime between 1st and 2nd timer register reads
  - Read overflow status again and use in conjunction with 2nd timer register value
- The same technique works if the order of data accesses is reversed
  - Read overflow status
  - Read timer register (timer may have overflowed after reading overflow status)
  - Compare current overflow status with first overflow status value
  - If current overflow status is not the same as first overflow status, an overflow occurred sometime between 1st and 2nd overflow status reads
  - Read timer register again and use in conjunction with 2nd overflow status value

- Pasted from http://www.scriptoriumdesigns.com/embedded/timers.php

Hardware Solutions for Multi-word Software Writes

- This is a general problem when multiple-word-access is involved when accessing hardware that can change while being accessed.
- Consider reading a 16-bit timer like this:

Timer is at 0x00FF

Upper Byte is read

Meanwhile timer increments to 0x0100

Lower Byte is read

(Result is 0x0000)

- Consider writing a 16-bit countdown timer with value 0x0102 like this:

Timer is at 0x0200

Write 0x01 to upper

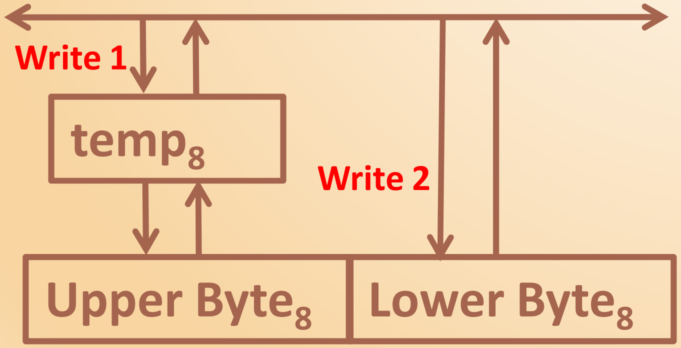
Timer decrements to 0x00FF

Write 0x02 to lower

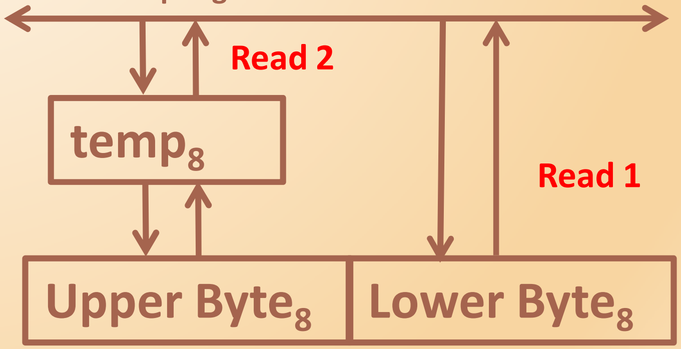
Timer is now at 0x0002

- Luckily, hardware solution is typically available for this, only requiring the software access the registers in a specified order. In 8-bit AVR C, the compiler takes care of reading/writing the bytes of a 16-bit register in the correct order.

Only write to lower triggers write to registers, upper byte is buffered to a temp reg and should be done first.



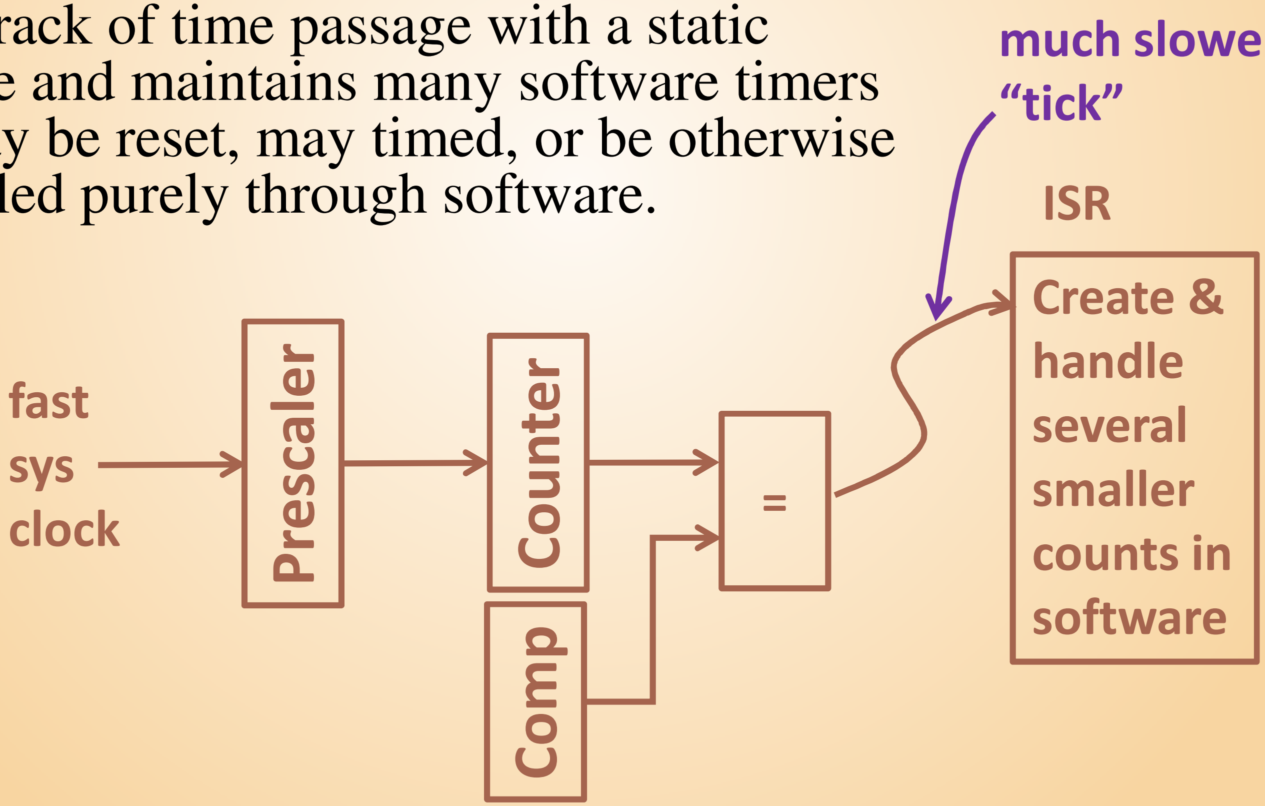
Reading lower byte triggers upper-byte to be copied to temp registers. Second read of upper byte is actually from the temp register.



System Tick and Software Timers

- In a complex-software application, several events may require timing but may not require much precision (perhaps on the order of 1 ms or 10s of ms). A common technique is to have an ISR triggered at some interval by the compare-match or timer overflow events that keeps track of time passage with a static variable and maintains many software timers that may be reset, may timed, or be otherwise controlled purely through software.

Using one slow hardware clock (taking advantage of precision and size of timer registers) with multiple derived software timers



Create & handle several smaller counts in software

System Tick and Software Timers

- To summarize the following: Use actual timer values, not the one you wanted….

- If you try to create a 3000 Hz clock but instead get 99 Hz through your configuration, use 99 for all derived calculations instead of 100. For instance to wait 2 seconds , you should wait 2994*2 cycles not 6000

```
# F_CPU = (1000000.0)
#define F_DESIRED = 3000
#define PRESCALER = 2                               // typically a power of two
#define F_PRESCALE = (F_CPU/PRESCALER)              // gives value 500000
#define TIMER_COUNT = ceil((F_PRESCALE)/F_DESIRED)  // find integer count:
                                                    //   ceil -> slightly faster error
                                                    //   floor -> slightly slower error
#define TIMER_MAX_COUNT_TO_SET = (TIMER_COUNT-1)    // define max count to set
#define F_ACTUAL = F_CPU/(PRESCALER *(TIMER_COUNT)) // The point of this slide is to do
                                                    //   this (find 2994.012 Hz)
# define TWO_SECONDS = F_ACTUAL * 2                 // and this
```

- Had the TWO_SECONDS been define with F_DESIRED, the time would overshoot and have been 2* (3000/2994.012) seconds

- If floor was used and TWO_SECONDS was defined with F_DESIRED, the time would undershoot and have been 2* (3000/3012.0482) seconds

Slip Adjustment

- If, due to hardware considerations, your system tick ends up not being a nice "round number" and want to remove long-term drift (say e.g. you want to maintain an elapsed time for logging events), you can "slip" your system tick software accumulator to remove any long-term drift.

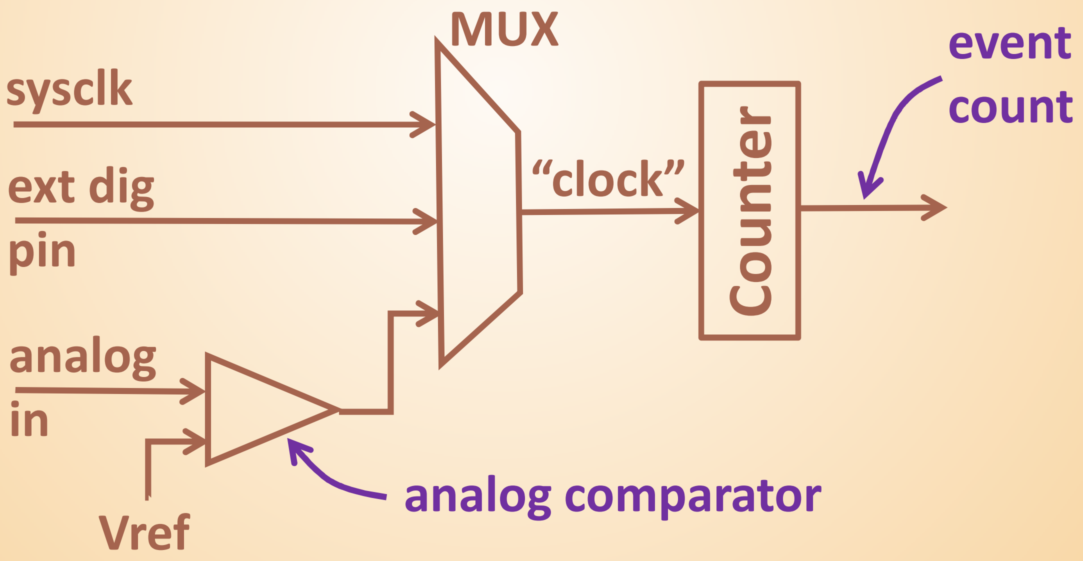
- For some leap ticks, the count will be modified, just as a day is added every leap year.

- In the previous example, if 3012.0482 was used instead of 3000, and could be corrected for by waiting an extra clock period every 20 ticks.

Event Counters

- Though we have mostly discussed the use of the counters as timers, the hardware counters are also useful for counting rapid events, like tracking the number of rotations of a high RPM motor.

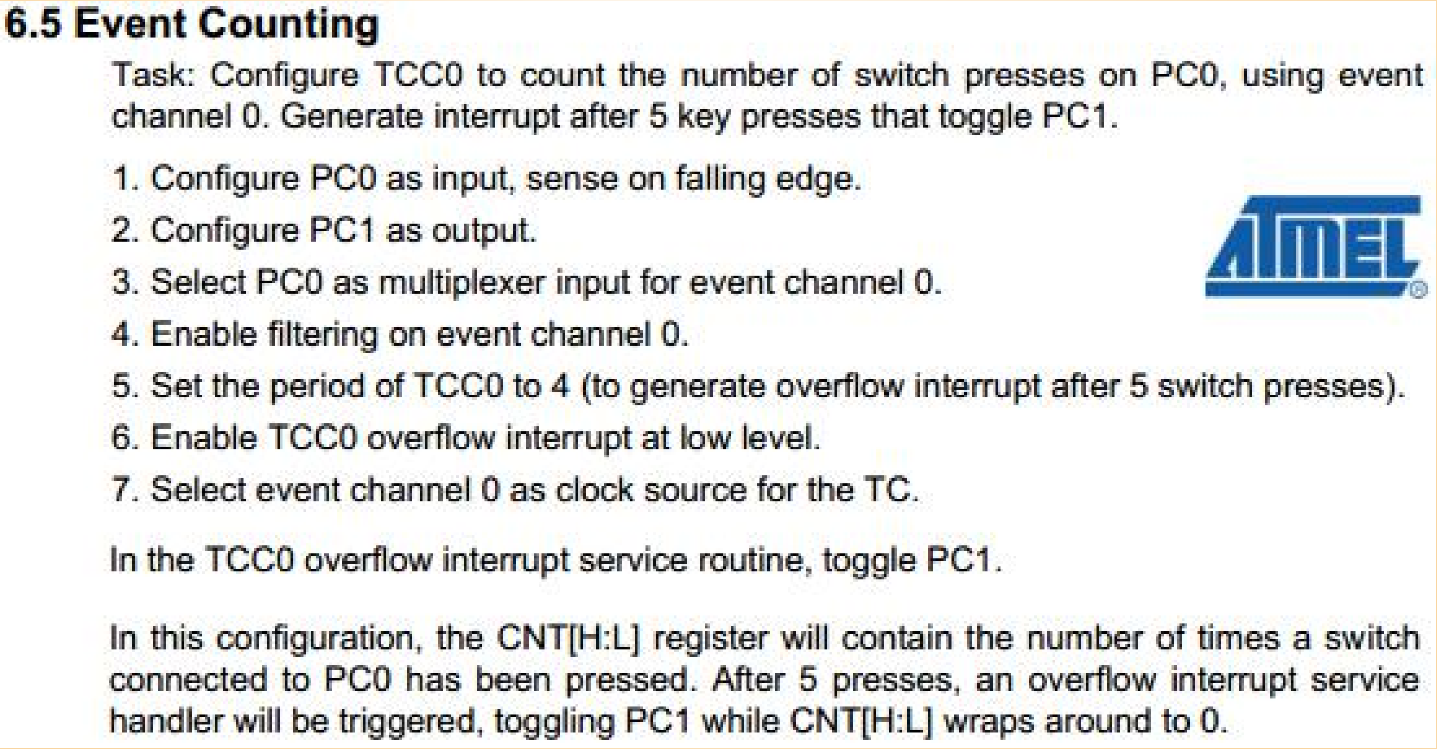
- Note, this can be used to count the number of bounces on a mechanical switch[1]. This is done by selecting a trigger input other than a regular clock. If an external pin is used, and it is connected to a clock, it is still a timer of course.



- [1] http://www.scriptoriumdesigns.com/embedded/timers.php

Timer/Counter Application Note

- We will now get into the details of AVR, but first, it should be noted that there is an application note describing many use cases for timers, which modes to use, and what bits should be set. It is a bit friendlier read as compared to the data sheet:
- AVR1306: Using the XMEGA Timer/Counter App Note
  http://www.atmel.com/dyn/resources/prod_documents/doc8045.pdf



6.5 Event Counting

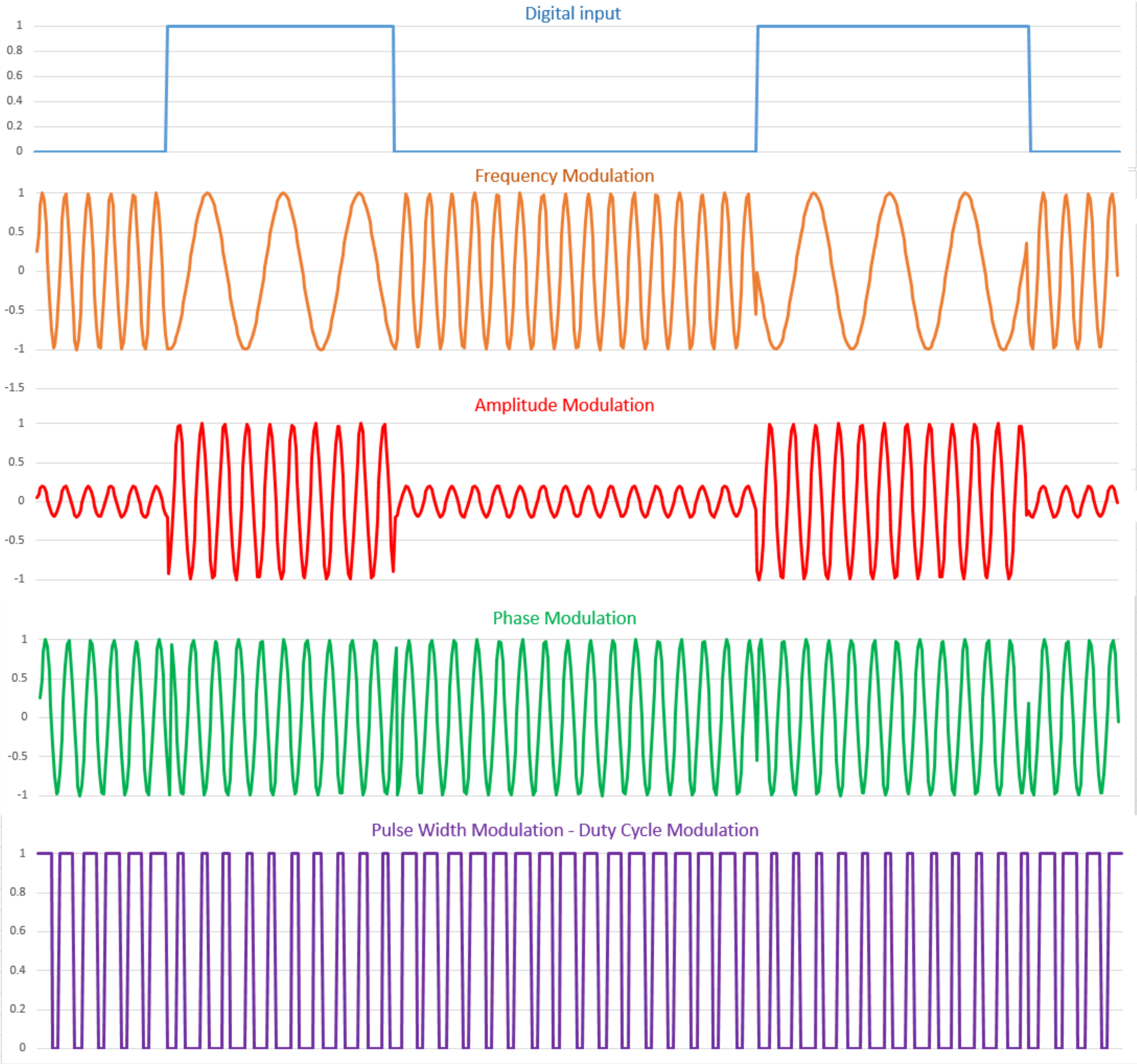
Task: Configure TCC0 to count the number of switch presses on PC0, using event channel 0. Generate interrupt after 5 key presses that toggle PC1.

1. Configure PC0 as input, sense on falling edge.
2. Configure PC1 as output.
3. Select PC0 as multiplexer input for event channel 0.
4. Enable filtering on event channel 0.
5. Set the period of TCC0 to 4 (to generate overflow interrupt after 5 switch presses).
6. Enable TCC0 overflow interrupt at low level.
7. Select event channel 0 as clock source for the TC.

In the TCC0 overflow interrupt service routine, toggle PC1.

In this configuration, the CNT[H:L] register will contain the number of times a switch connected to PC0 has been pressed. After 5 presses, an overflow interrupt service handler will be triggered, toggling PC1 while CNT[H:L] wraps around to 0.

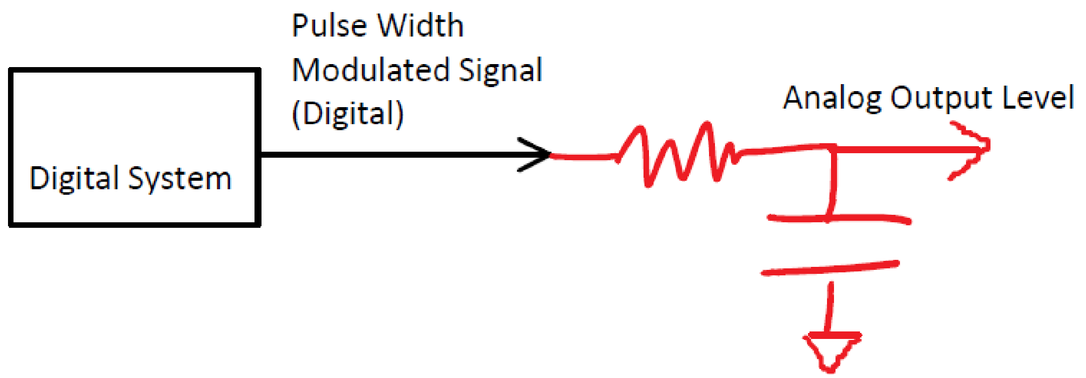
Pulse Width Modulation and Frequency Modulation



Counter Modes

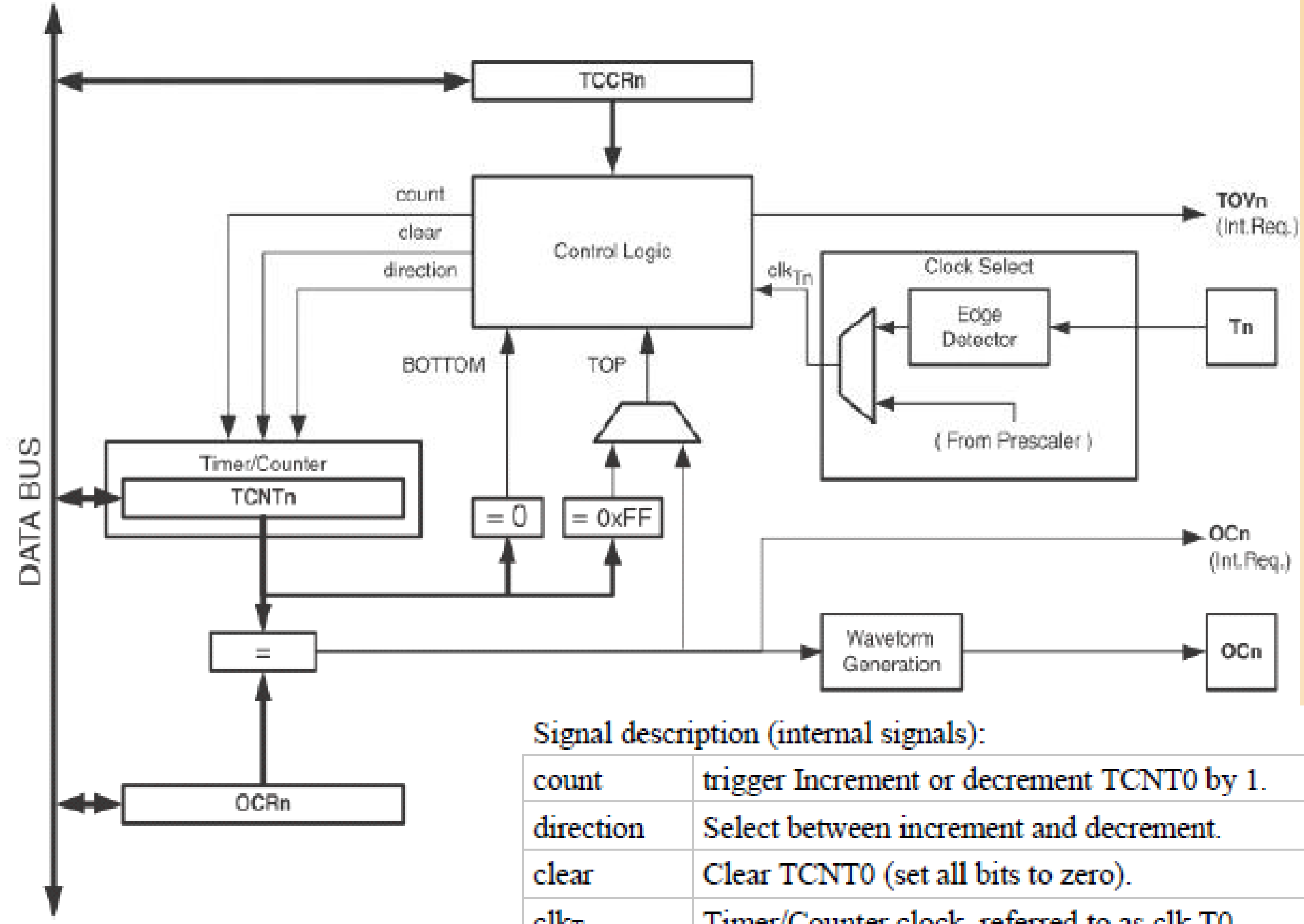
- AVR timers have a few important modes of operation:
  - Non-PWM Modes
    - Normal Mode-count and reset at overflow, setting overflow flag
    - Compare Timer Clear (CTC) Mode-reset upon reaching comparison value
  - PWM Modes
    - Fast PWM-beginning of pulses are regularly spaced
    - Phase Correct PWM-center of pulses regularly spaced
- Note, (explicit) electronic filtering is not always necessary. When we see a fast flashing LED we only perceive the recent average intensity, which is related to the duty cycle. A DC motor will respond the recent average supply level. PWM is also the basis of switching power supplies used in computers. By monitoring the output level using an ADC, the digital system can respond to load changes modifying the duty cycle.

A simple Digital to Analog Converter



Timer/Counter (T/C) 0 Overview

Figure 14-1. 8-bit Timer/Counter Block Diagram

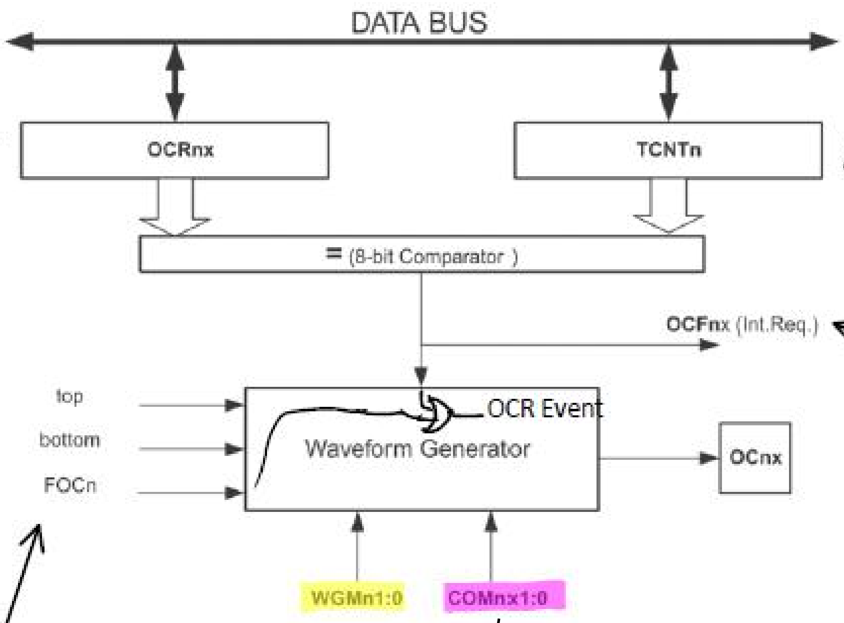


Signal description (internal signals):

count	trigger Increment or decrement TCNT0 by 1.
direction	Select between increment and decrement.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk T0 in the following.
top	Signalize that TCNT0 has reached maximum value...determined by 0xFF or match to compare register OCRn
bottom	Signalize that TCNT0 has reached minimum value (zero).

Waveform Generation

This is double-buffered in PWM mode. Writes to OCRnx are not propagated until the next cycle when top/bottom is reached to avoid interrupting a cycle



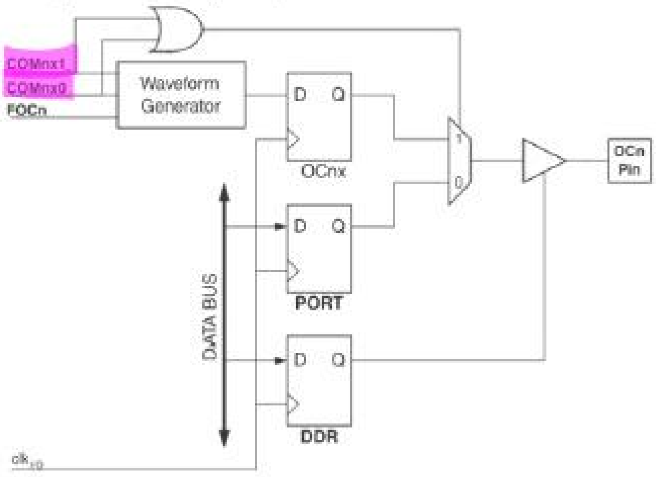
Note: writing TCNTn to the same value as OCRnx is handled as a special case where immediate compare events are blocked

The Flag is set when counter reaches the value in compare register. It is cleared automatically if interrupts are enabled and interrupt routine is called.

Force Output Compare Event:
Can be used by software as a software trigger for OCR event, resulting behavior still determined by the mode bits WGNn and COMnx .

COMnx bits control configuration including the behavior of the output

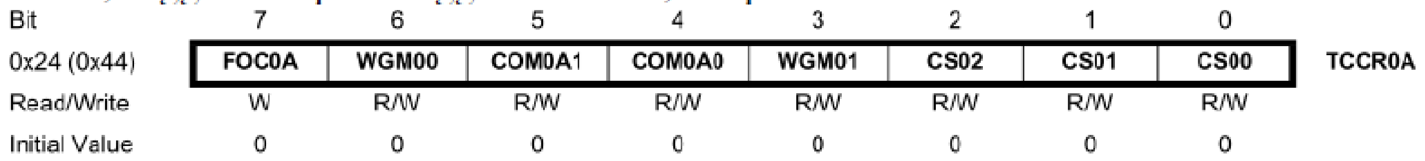
Figure 14-4. Compare Match Output Unit, Schematic



T/C 0 Register Overview

All are 8 bits

TCCR0A – Timer/Counter Control Register A - bits to set waveform generation mode and compare modes, trigger compare-triggered events, clk prescaler selected

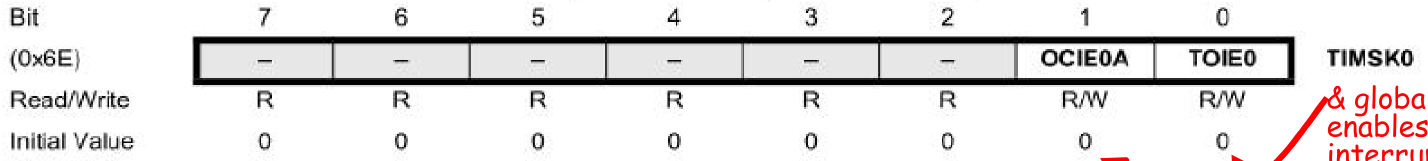


Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	FOC0A	WGM00	COM0A1	COM0A0	WGM01	CS02	CS01	CS00	TCCR0A
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNT0 – Timer/Counter Register - counter value, can read and write...note compare event blocking when TCNT0 is set to same value as compare register

OCR0A – Output Compare Register A, can read or write

TIMSK0 – Timer/Counter 0 Interrupt Mask Register interrupt enable bits

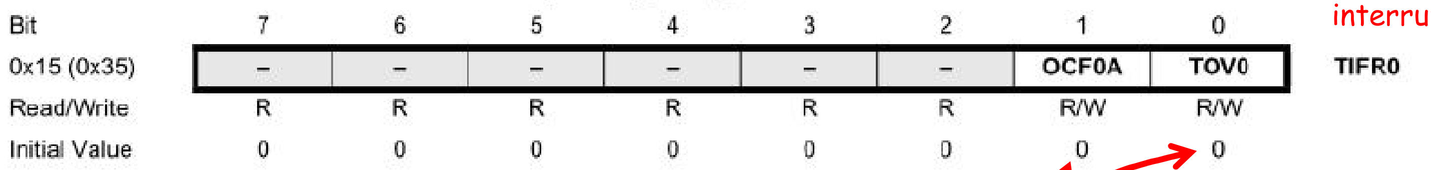


Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	–	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

& global interrupt enable bit enables counter overflow interrupt

& global interrupt enable bit enables output compare interrupt

TIFR0 – Timer/Counter 0 Interrupt Flag Register

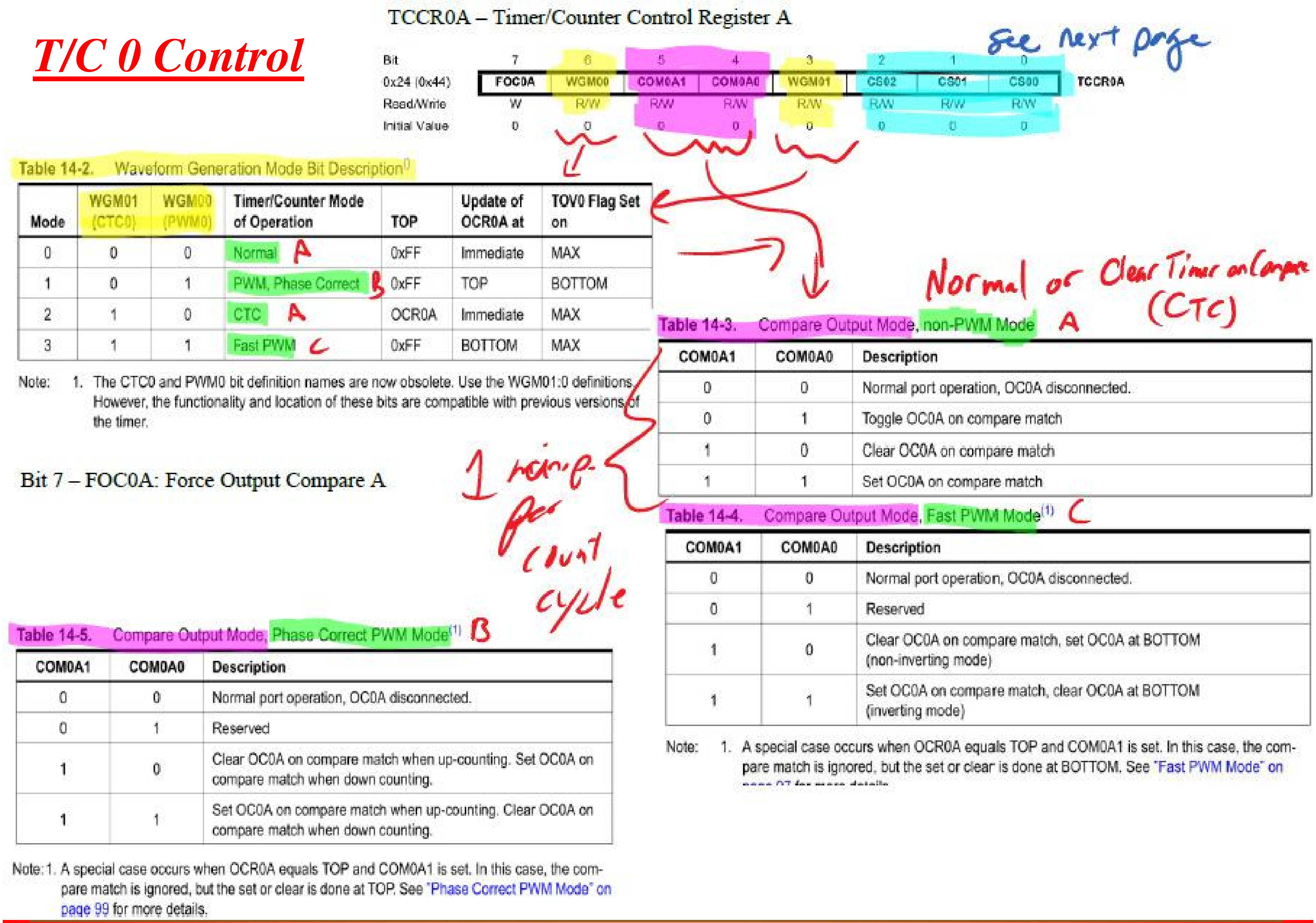


Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	–	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Overflow flag, clears when interrupt service routine is called or when a 1 is written

Compare flag, clears when interrupt service routine is called or when a 1 is written

T/C 0 Control



TCCR0A – Timer/Counter Control Register A

See next page

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	FOC0A	WGM00	COM0A1	COM0A0	WGM01	CS02	CS01	CS00	TCCR0A
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-2. Waveform Generation Mode Bit Description[1]

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation		TOP	Update of OCR0A at	TOV0 Flag Set on
0	0	0	Normal	A	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	B	0xFF	TOP	BOTTOM
2	1	0	CTC	A	OCR0A	Immediate	MAX
3	1	1	Fast PWM	C	0xFF	BOTTOM	MAX

Note: 1. The CTC0 and PWM0 bit definition names are now obsolete. Use the WGM01:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Bit 7 – FOC0A: Force Output Compare A

1 time per count cycle

Normal or Clear Timer on Compare (CTC)

Table 14-3. Compare Output Mode, non-PWM Mode A

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on compare match
1	0	Clear OC0A on compare match
1	1	Set OC0A on compare match

Table 14-4. Compare Output Mode, Fast PWM Mode[1] C

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Reserved
1	0	Clear OC0A on compare match, set OC0A at BOTTOM (non-inverting mode)
1	1	Set OC0A on compare match, clear OC0A at BOTTOM (inverting mode)

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at BOTTOM. See "Fast PWM Mode" on page 97 for more details.

Table 14-5. Compare Output Mode, Phase Correct PWM Mode[1] B

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Reserved
1	0	Clear OC0A on compare match when up-counting. Set OC0A on compare match when down counting.
1	1	Set OC0A on compare match when up-counting. Clear OC0A on compare match when down counting.

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See "Phase Correct PWM Mode" on page 99 for more details.

Clock Select Bits

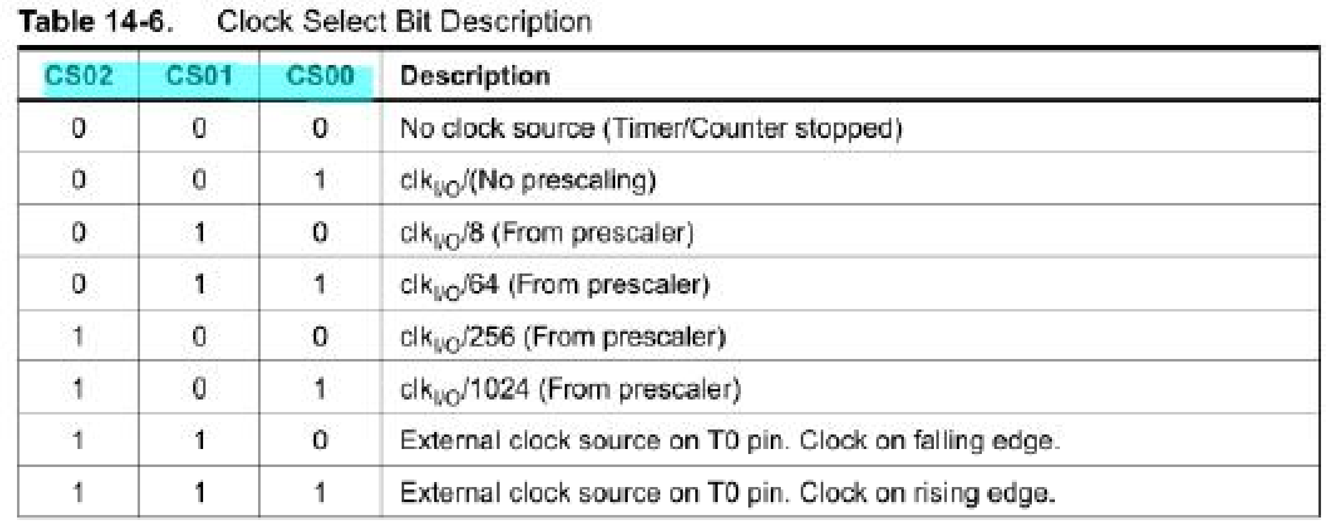


Table 14-6. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

- To implement a software clock, you may select the external clock source on the T0 pin **and configure the pin as an output pin.**

Normal Mode

- Always count up
- Non counter clear, just rollover from 0x00 to 0xFF
- T/C overflow flag (TOV0) set the same cycle that TCNT0 becomes 0
  - Can treat TOV0 like 9th bit on first count run, but it is not cleared upon second counter rollover (0x1FF -> 0x100)
  - Can use TOV0 with software to extend the count Either
    - Increment software counter when TOV0 becomes 1 and clear T0V0 before next rollover (write 1 to clear any I.F. to 0)
    - Use ISR to increment software counter, TOV0 is cleared automatically in hardware
- By setting the configuration bits COM0A[1:0], a designated output pin can be modified when the counter matches the value of the Output Compare Register (OCR0A)

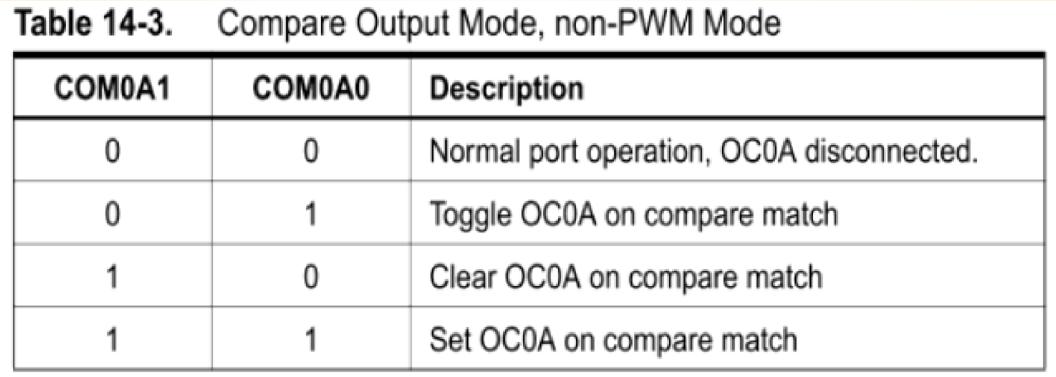


Table 14-3. Compare Output Mode, non-PWM Mode

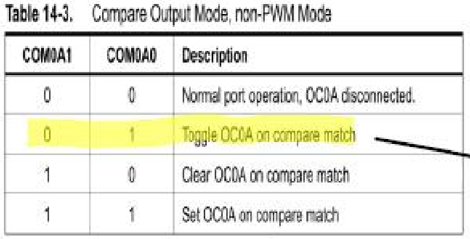
COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on compare match
1	0	Clear OC0A on compare match
1	1	Set OC0A on compare match

- The normal mode is limited to count periods of 2^N, where N is the number counter bits. To create a variable count period, the CTC modes can be used.

Clear Timer on Compater Match (CTC) Mode

- In the CTC mode, when the counter reaches the value of the Output Compare Register (OCR0A), the event is referred to as an Output Compare Event
- Upon an Output Compare event, the next value of the counter (TCNT0) is 0 and the Output Compare Flag (OCF0A) is set
- By changing the output compare registers, one manipulates the counter max value and thus controls the count cycle freq. and the freq. of derived waveforms or interrupt calls.
- There is a designated output pin that can be automatically modified on Output Compare Events according to configuration bits COM0A[1:0]
- If the Output Compare Interrupt Flag (OCIE) is set and the Global Interrupt Enable Flag is set in SREG, an interrupt is triggered and the Output Compare Flag is cleared automatically.

Table 14-3. Compare Output Mode, non-PWM Mode

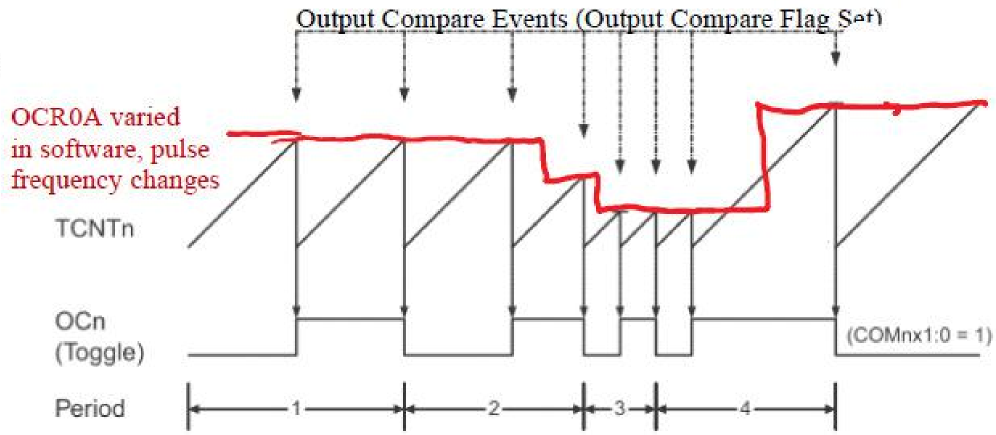


COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on compare match
1	0	Clear OC0A on compare match
1	1	Set OC0A on compare match

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

↑
prescaler
1/8/64/256/1024

Figure 14-5. CTC Mode, Timing Diagram **For COM0A[1:0]== 1**


Output Compare Events (Output Compare Flag Set)
OCR0A varied in software, pulse frequency changes
TCNTn
OCn (Toggle)
(COMnx1:0 = 1)
Period
1
2
3
4

- Caution: if computing a new OCR0A value in software after the Compare Flag is set, if the value is too small and the softwareis too slow, the counter will pass the value before it is set and the comparison event not happen on that cycle. The counter will count to 0xFF rolloever, and the match will occur on the next count cycle and may result in an irregular waveform. This also why timers (TCNT0) should be setto 0 initially.

Fast PWM

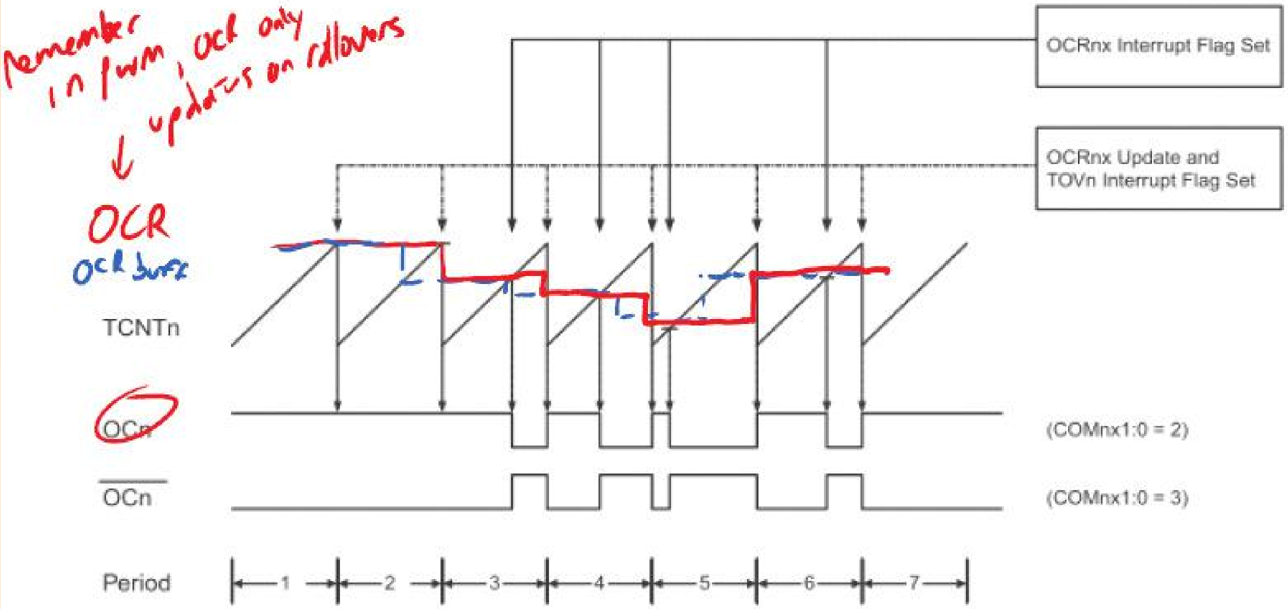
Table 14-4. Compare Output Mode, Fast PWM Mode[1]

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Reserved
1	0	Clear OC0A on compare match, set OC0A at BOTTOM (non-inverting mode)
1	1	Set OC0A on compare match, clear OC0A at BOTTOM (inverting mode)

really just one mode, and its inverse

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at BOTTOM. See "Fast PWM Mode" on page 97 for more details.

Figure 14-6. Fast PWM Mode, Timing Diagram



remember in pwm, ocr only updates on rollovers

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

Special cases:

OCR = MAX 0xFF
give const. waveform

→ dep. of mode

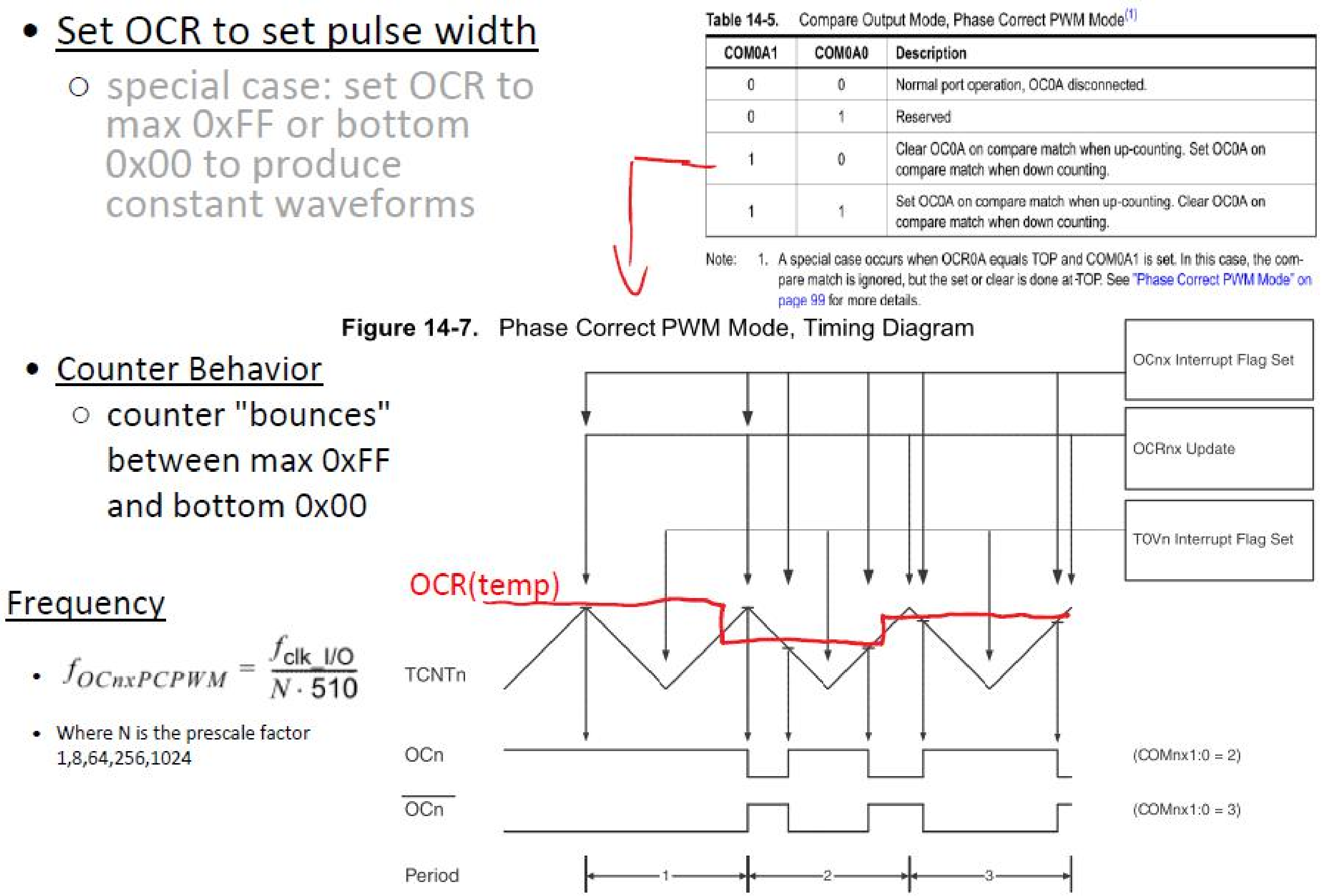
OCR = MIN 0x00
gives 1 cycle pulse

Special Fast PWM mode

- A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0A to toggle its logical level on each compare match (COM0A1:0 = 1). The waveform generated will have a maximum frequency of:
- $f_{OC0} = f_{clk\_I/O}/2$
- when OCR0A is set to zero. This feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

Phase Correct PWM mode

- Phase Correct PWM modes are like standard PWM modes, but the center of pulse spacing is doesn't change when the pulse width changes.
- This is achieved by a special "dual-slope" counter behavior
  - "The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM."



- Set OCR to set pulse width
  - special case: set OCR to max 0xFF or bottom 0x00 to produce constant waveforms

Table 14-5. Compare Output Mode, Phase Correct PWM Mode[1]

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Reserved
1	0	Clear OC0A on compare match when up-counting. Set OC0A on compare match when down counting.
1	1	Set OC0A on compare match when up-counting. Clear OC0A on compare match when down counting.

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See "Phase Correct PWM Mode" on page 99 for more details.

Figure 14-7. Phase Correct PWM Mode, Timing Diagram

- Counter Behavior
  - counter "bounces" between max 0xFF and bottom 0x00

Frequency

- $f_{OCnxPCPWM} = \dfrac{f_{clk\_I/O}}{N \cdot 510}$
- Where N is the prescale factor 1,8,64,256,1024

Special Notes

- At the very start of period 2 in Figure 14-7 on page 99 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.
- OCR0A changes its value from MAX, like in Figure 14-7 on page 99. When the OCR0A value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.
- The timer starts counting from a value higher than the one in OCR0A, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

16 Bit Timers

- Similar to 8 bit timers see documentation for details
  - Note all 16-bit reads and writes involve a high-byte buffer.
  - If using 8-bit access, Read Low Byte First and Write it last
  - If using 16-bit register names, C compiler take care of the order

Figure 15-5. Compare Match Output Unit, Schematic

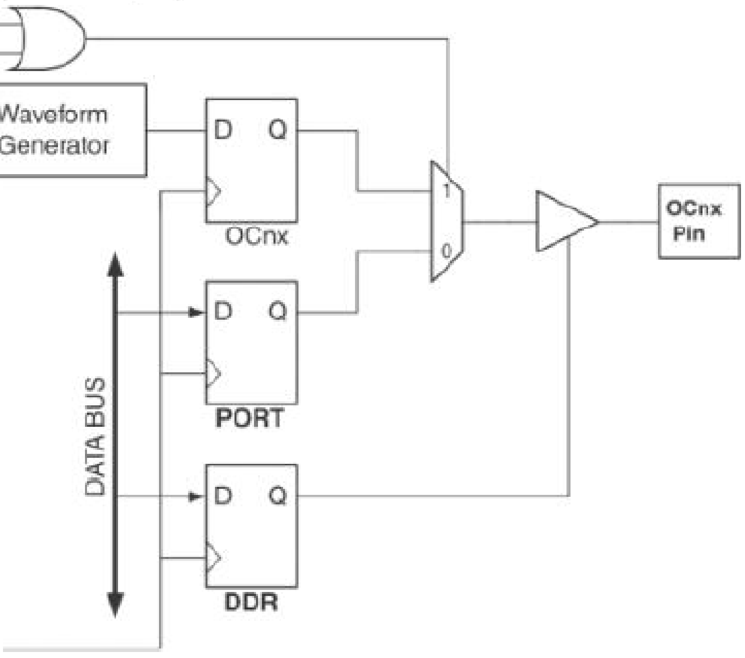
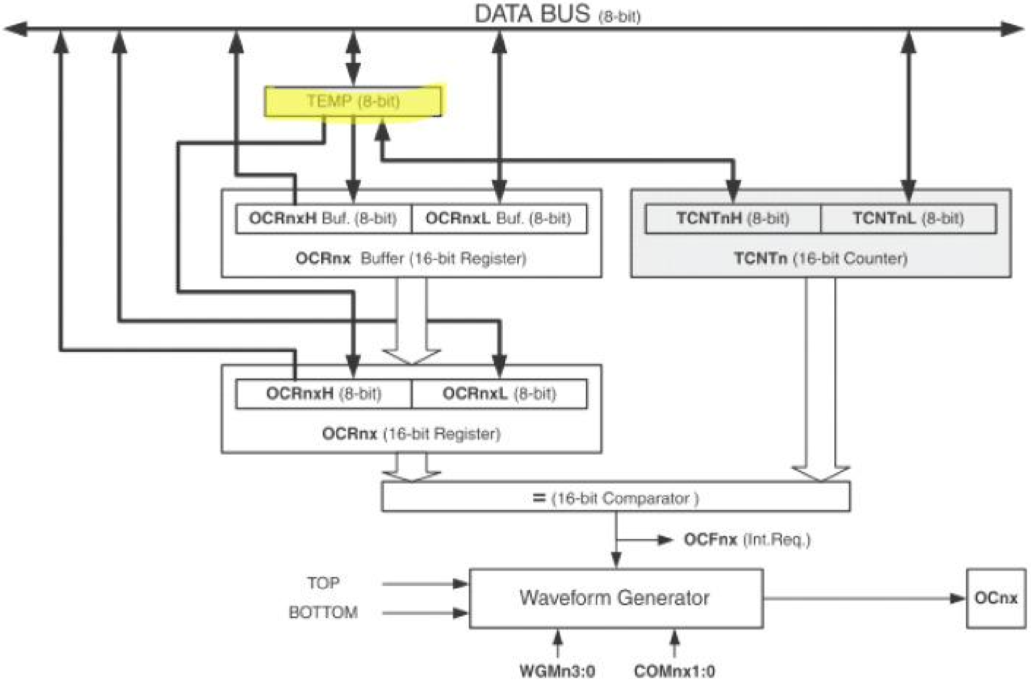


Figure 15-4. Output Compare Unit, Block Diagram



T/C 0 and T/C 1 Prescalar unit and external clock sync

- They share the prescaler unit but have separate selection

Figure 16-2. Prescaler for Timer/Counter0 and Timer/Counter1[(1)]

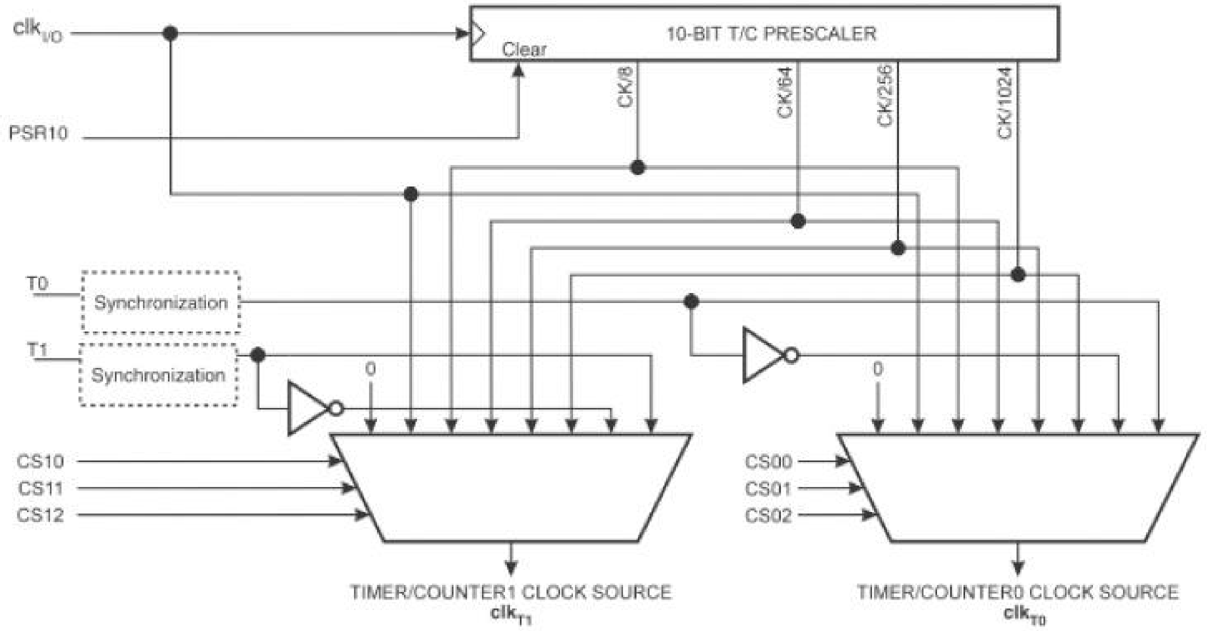
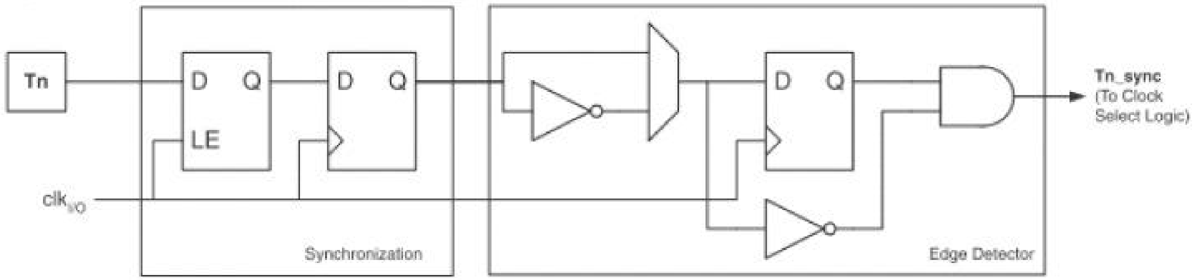


Figure 16-1. T1/T0 Pin Sampling

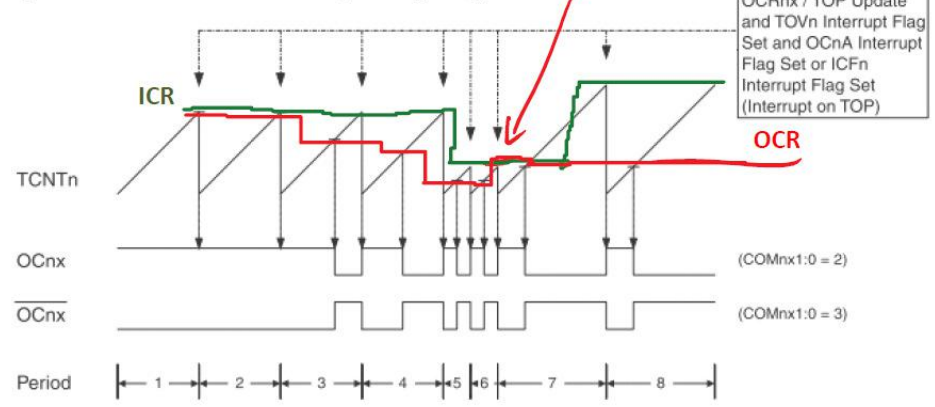


- Synchronizer --minimizes chance of metastable states (where input is captured around the time of a transition, violating setup/hold requirements causing invalid operation)

T/C 1 Modes

- Modes are similar to 8-bit T/C 0, but use dual-purpose register Input Capture Register / Count Top Register (ICR) . In PWM modes ,
  - ICR controls frequency
  - OCR and ICR together control the pulse width

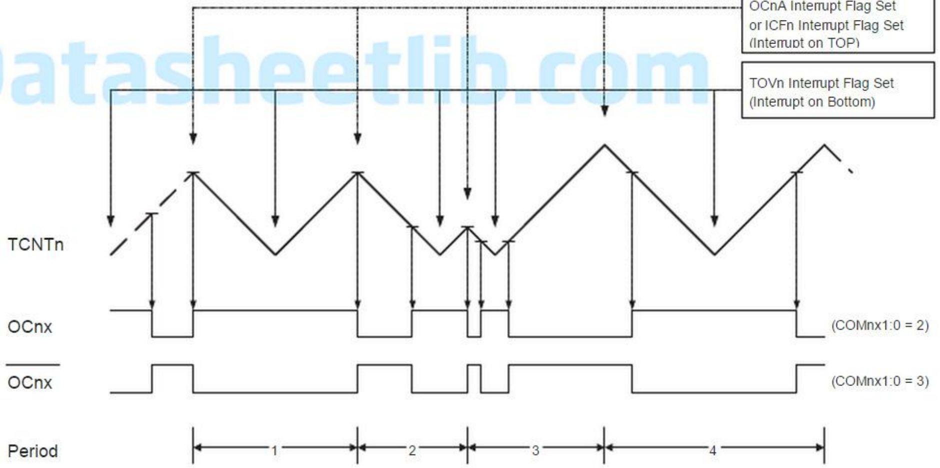
Figure 15-7. Fast PWM Mode, Timing Diagram



OCRnx / TOP Update and TOVn Interrupt Flag Set and OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)

- In this final mode, updates to the OCR and top registers are buffered and not propagated until the counter reaches the bottom so that values are not changed in the middle of a pulse.
- It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static
- TOP value there are practically no differences between the two modes of operation.

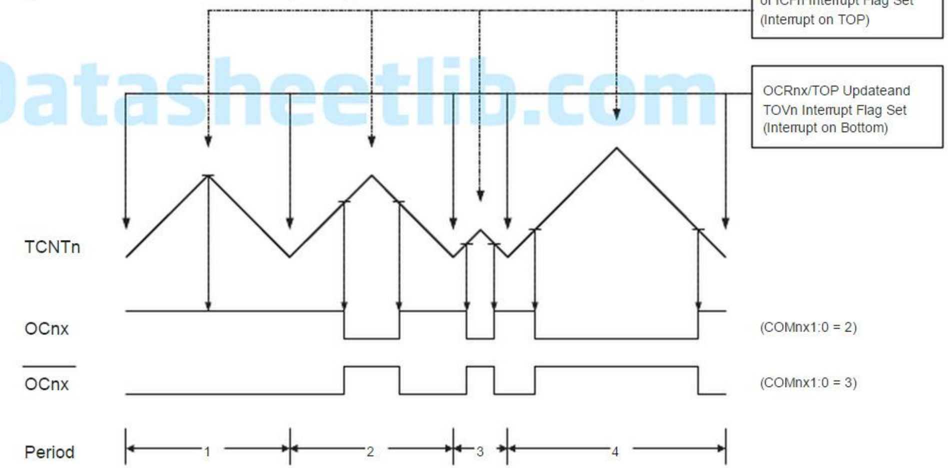
Figure 15-8. Phase Correct PWM Mode, Timing Diagram



OCRnx/TOP Update and OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)

TOVn Interrupt Flag Set (Interrupt on Bottom)

Figure 15-9. Phase and Frequency Correct PWM Mode, Timing Diagram



OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)

OCRnx/TOP Updateand TOVn Interrupt Flag Set (Interrupt on Bottom)

Accessing 16-bit Registers

- TCNT1, OCR1A/B, and ICR1 are 16-bit registers
  - When the low byte of a 16-bit register is written by the CPU, the high byte stored in the temporary register, and the low byte written are both copied into the 16-bit register in the same clock cycle.
  - When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read.

- ASM

```
...
;Set TCNT1 to 0x01FF
   ldir17,0x01
   ldir16,0xFF
   outTCNT1H,r17
   outTCNT1L,r16
;Read TCNT1 into r17:r16
   in r16,TCNT1L
   in r17,TCNT1H
```

- In C, the compiler takes care of the byte order

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
...
```

- But each access is still multiple clock cycles. To avoid interaction with other code accessing the timer registers in the middle of the multi-bit write, atomic access should be implemented.

Atomic access to 16-bit registers

- Atomic access (uninterrupted access) requires disabling interrupts

- Atomic Read:
  - ASM

```
TIM16_ReadTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
; Restore global interrupt flag
outSREG,r18
ret
```

  - C

```
unsigned int TIM16_ReadTCNT1( void ) {
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  __disable_interrupt();
  /* Read TCNT1 into i */
  i = TCNT1;
  /* Restore global interrupt flag */
  SREG = sreg;
  return i;
}
```

- Atomic Write:
  - ASM

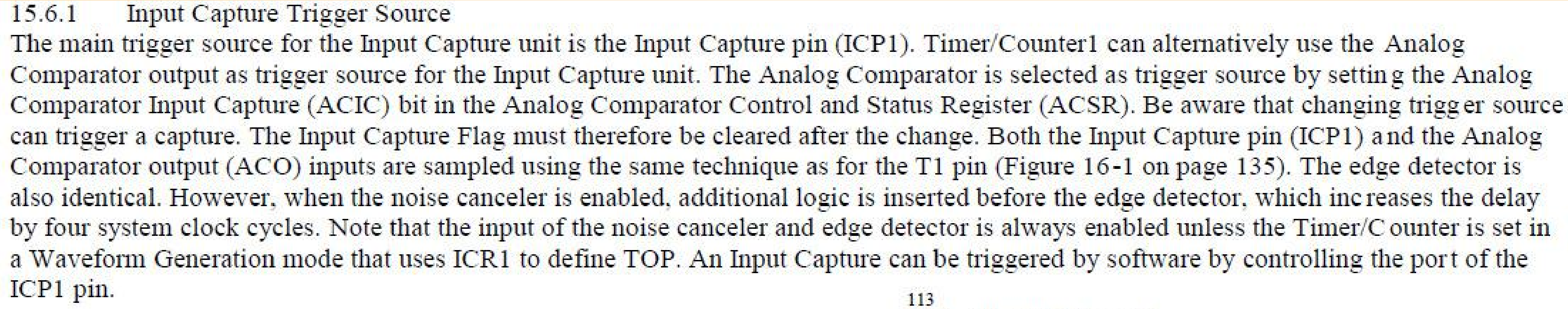
```
TIM16_WriteTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Set TCNT1 to r17:r16
outTCNT1H,r17
outTCNT1L,r16
; Restore global interrupt flag
outSREG,r18
ret
```

  - C

```
void TIM16_WriteTCNT1( unsigned int i ) {
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  __disable_interrupt();
  /* Set TCNT1 to i */
  TCNT1 = i;
  /* Restore global interrupt flag */
  SREG = sreg;
}
```

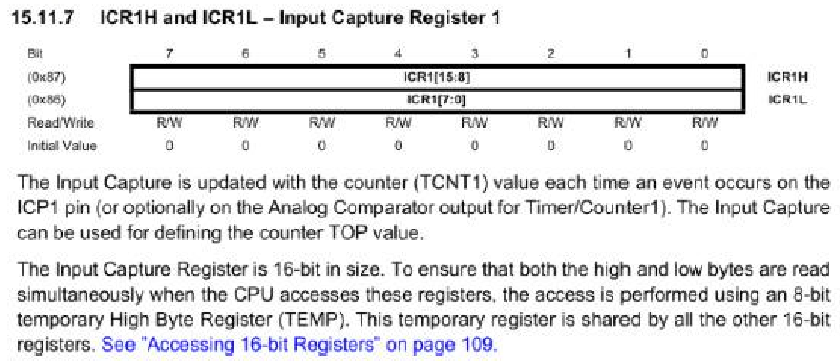
T/C 1Capture Unit

- As discussed, a capture unit and register (ISR) is provided to save the counter value precisely at the time of some external event, the follow details are provided for your reference.



15.6.1 Input Capture Trigger Source

The main trigger source for the Input Capture unit is the Input Capture pin (ICP1). Timer/Counter1 can alternatively use the Analog Comparator output as trigger source for the Input Capture unit. The Analog Comparator is selected as trigger source by setting the Analog Comparator Input Capture (ACIC) bit in the Analog Comparator Control and Status Register (ACSR). Be aware that changing trigger source can trigger a capture. The Input Capture Flag must therefore be cleared after the change. Both the Input Capture pin (ICP1) and the Analog Comparator output (ACO) inputs are sampled using the same technique as for the T1 pin (Figure 16-1 on page 135). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a Waveform Generation mode that uses ICR1 to define TOP. An Input Capture can be triggered by software by controlling the port of the ICP1 pin.



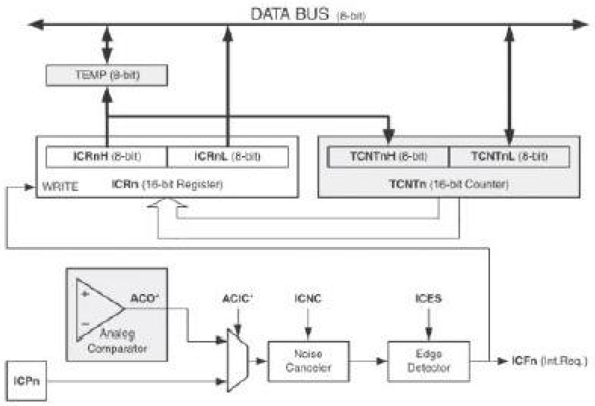
15.11.7 ICR1H and ICR1L – Input Capture Register 1

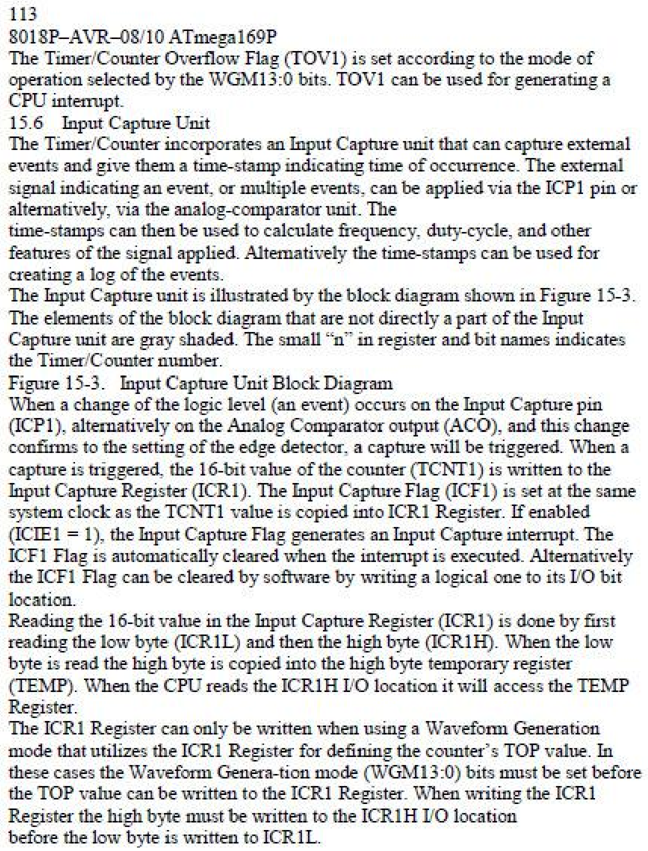
Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Input Capture is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 109.

Figure 15-3. Input Capture Unit Block Diagram



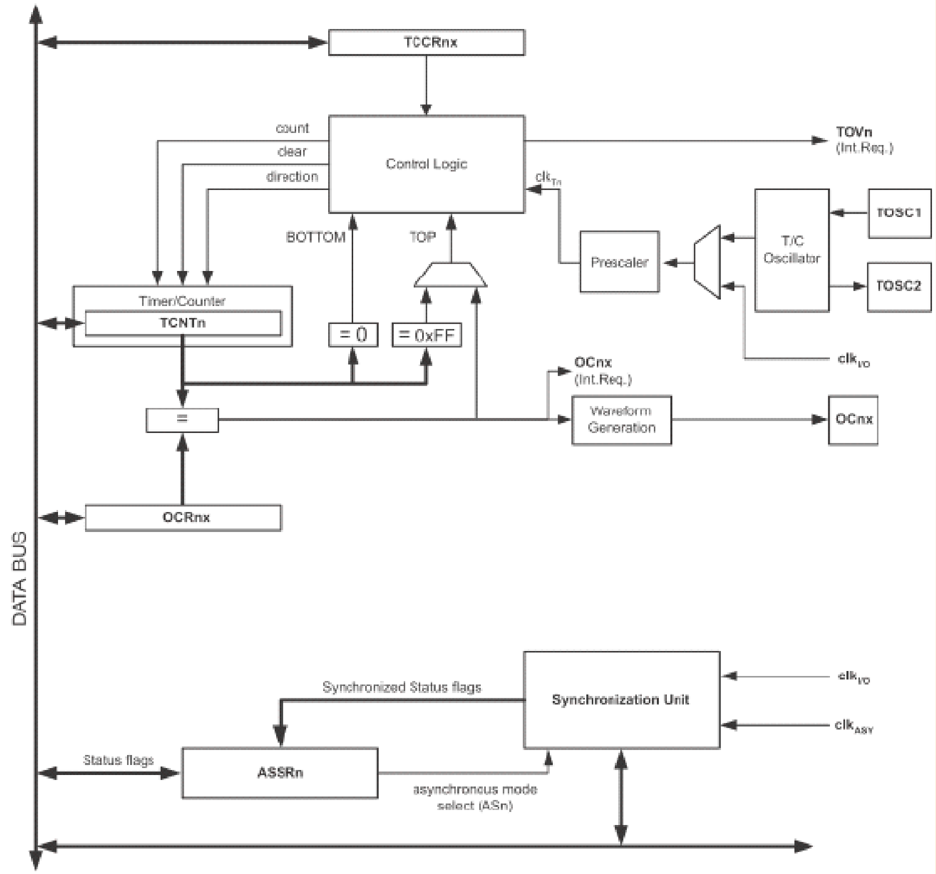


113
8018P–AVR–08/10 ATmega169P
The Timer/Counter Overflow Flag (TOV1) is set according to the mode of operation selected by the WGM13:0 bits. TOV1 can be used for generating a CPU interrupt.
15.6 Input Capture Unit
The Timer/Counter incorporates an Input Capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the analog-comparator unit. The
time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.
The Input Capture unit is illustrated by the block diagram shown in Figure 15-3. The elements of the block diagram that are not directly a part of the Input Capture unit are gray shaded. The small "n" in register and bit names indicates the Timer/Counter number.
Figure 15-3. Input Capture Unit Block Diagram
When a change of the logic level (an event) occurs on the Input Capture pin (ICP1), alternatively on the Analog Comparator output (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the Input Capture Register (ICR1). The Input Capture Flag (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 Register. If enabled (ICIE1 = 1), the Input Capture Flag generates an Input Capture interrupt. The ICF1 Flag is automatically cleared when the interrupt is executed. Alternatively the ICF1 Flag can be cleared by software by writing a logical one to its I/O bit location.
Reading the 16-bit value in the Input Capture Register (ICR1) is done by first reading the low byte (ICR1L) and then the high byte (ICR1H). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP Register.
The ICR1 Register can only be written when using a Waveform Generation mode that utilizes the ICR1 Register for defining the counter's TOP value. In these cases the Waveform Genera-tion mode (WGM13:0) bits must be set before the TOP value can be written to the ICR1 Register. When writing the ICR1 Register the high byte must be written to the ICR1H I/O location before the low byte is written to ICR1L.

Timer/Counter (T/C) 2

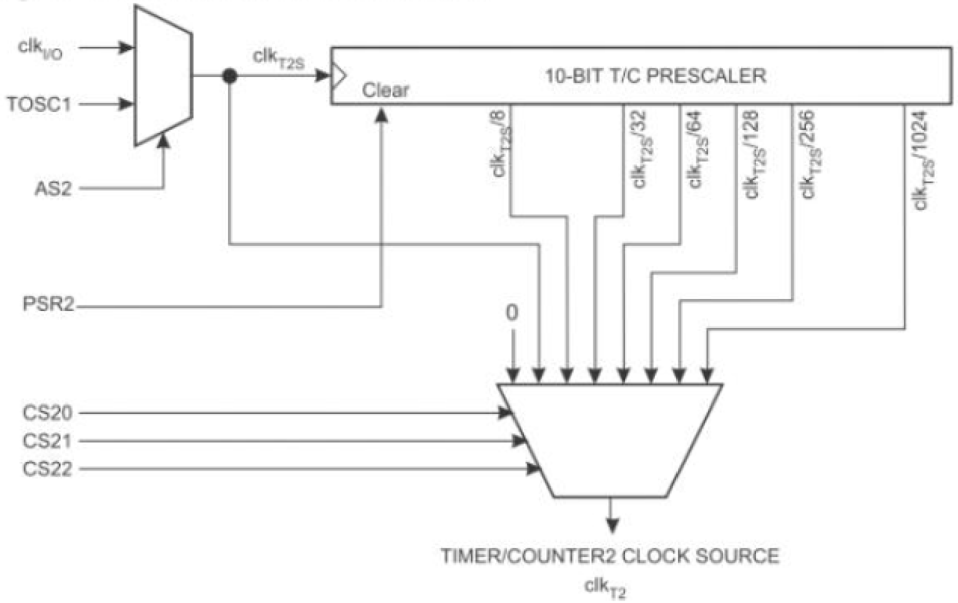
- Only significant thing to note here is that T/C2 provides the option use an external 32kHz crystal, an accurate clock independent of the system clock. The following details are provided for reference.

Figure 17-1. 8-bit Timer/Counter Block Diagram



- The Oscillator is optimized for use with a 32.768 kHz crystal. If applying an external clock on TOSC1, the EXCLK bit in ASSR must be set.

Figure 17-12. Prescaler for Timer/Counter2


clkI/O
TOSC1
AS2
clkT2S
Clear
10-BIT T/C PRESCALER
clkT2S/8
clkT2S/32
clkT2S/64
clkT2S/128
clkT2S/256
clkT2S/1024
PSR2
0
CS20
CS21
CS22
TIMER/COUNTER2 CLOCK SOURCE
clkT2