

# Timers and Counters

Hardware Timing, Counters

A decorative graphic consisting of several horizontal lines of varying lengths and colors (red, white, and light blue) extending from the right side of the slide.

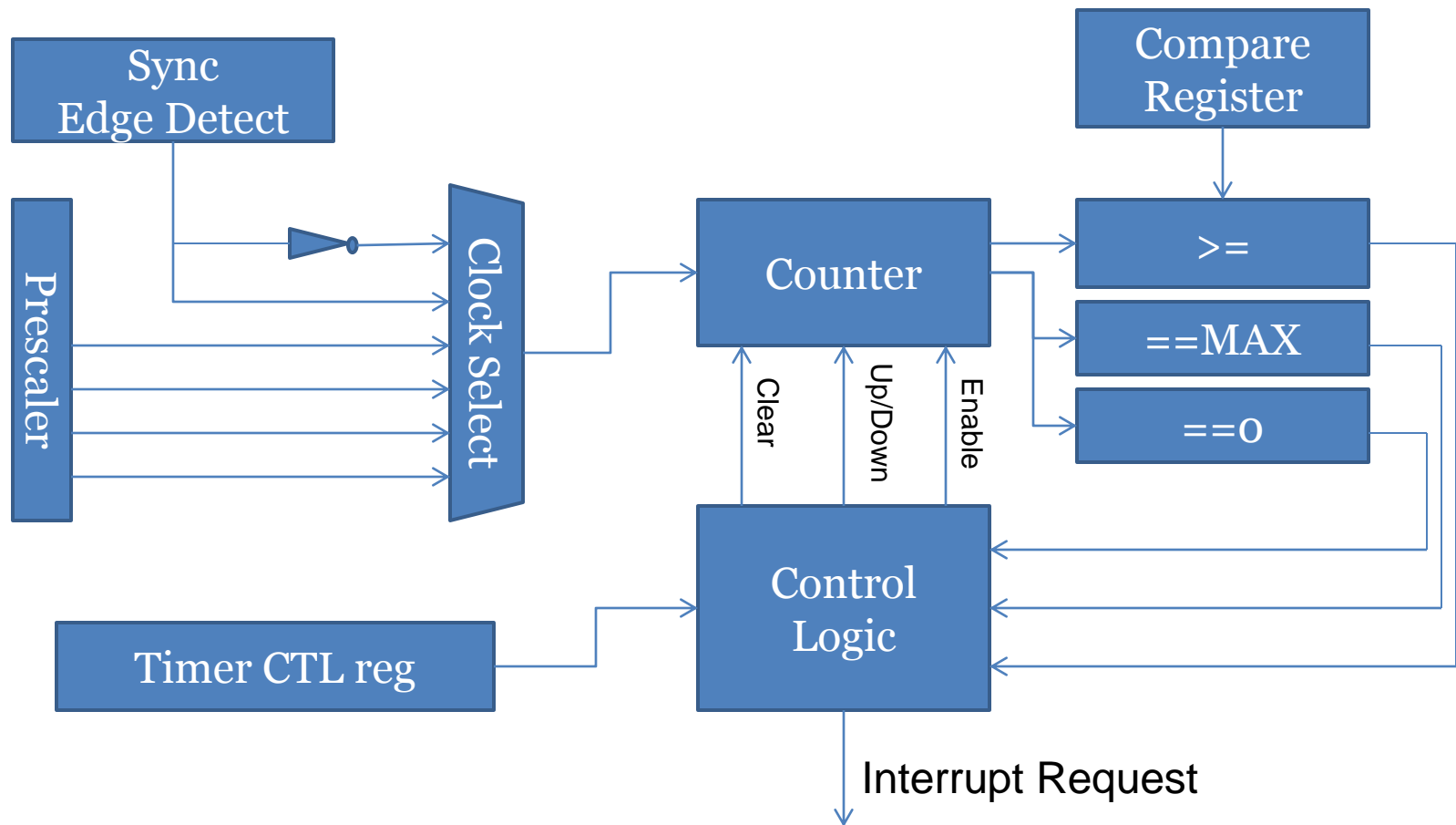
# What are Timers/Counters

- Hardware provided module which counts
  - Timers count clock cycles
  - Counters count arbitrary signal change
- Counts up to threshold then sets interrupt flag
  - CPU can poll for interrupt flag – software polling
  - CPU can be interrupted by flag – interrupt driven

# Timer/Counter Components

- Clock Prescaler (divider)
- Clock source selection
  - In the presence of multiple clocks
- N-bit timer/counter – Actual Count value
- Compare registers
- Capture registers – for precise capture of time of event
- Control Logic
- Status/Control Registers – Defines behavior

# Timer Hardware



# Clock Prescaler

- The clock prescaler modifies the rate at which a timer changes
  - *Allows timer/counter register to cover a larger range*
- For a timer,  $\log_2(\text{Range} * \text{Clock Freq})$  is a lower bound on the number of counter bits required
  - *Assuming no prescaling*
- Range AND precision define the # of bits required
  - *If you don't need as high of precision, you may prescale the clock to achieve a higher range from same # of bits*

# Capture Registers

- Dedicated hardware registers for determining the time of an event
  - Output of timer/counter register is connected to storage register that is triggered to save by an event signal
- What would be the software equivalent?
  - How would this compare in accuracy to the hardware version?

# Compare/Match Registers

- Additional Custom hardware to monitor the value of the timer/counter and signal another event to occur
- Can trigger:
  - Flag to be set for software to detect through polling
  - An interrupt
  - A pin to change
  - A reset of the timer/counter register

# Timer/Counter Polling

- `while(COUNTER_REG < VALUE);`
  - This would wait until the counter reg is at least value. Use at least to avoid missing the counter value. Can still overflow counter reg
- Timer polling for delay
  - Get easily predictable accuracy
    - Reset timer, wait for value to be reached or exceeded
- Timer polling for delay with other stuff to do
  - Useful when response precision isn't high
    - Reset Timer, do other stuff, check if value is reached



# Polling for a Timer flag

- In our previous examples, we can potentially miss values
  - If the counter overflows, checking if register is less than will not give the value we want
- Use flags which must be explicitly cleared
- Overflow Flag
  - Raised when counter hits max value
- Compare-Match flag
  - Raised when counter hits compare-register value
- Must explicitly clear these flags after event

# Timer Actions

- A timer can be set up to regularly do one or more of the following:
  - Trigger an interrupt
  - Cause the hardware to make some pin value change
  - Reset the timer itself
- On one of the following events
  - Overflow
  - Compare-match events

# Timer Actions

- This hardware-based ISR triggering and self-resetting allows for precise timing and very regular triggering of ISR calls
  - Does not rely on any software timing or intermittent and often irregular polling
    - (software doesn't always take the same time in every loop iteration if conditional statements exist).
  - Note that when an interrupt is enabled the corresponding flag is automatically cleared in hardware.
- Automatic Pin Changes allow for precise and flexible digital waveform generation.

# Getting the Desired Timing

- Takes some thought to get frequency you want
  - **Resulting Frequency = input clock/prescaler/(maxcount+1)**
    - Max count may be defined by a compare register or maximum value of the counter
    - Clock division = prescaler \*( maxcount+1)
    - You must choose the prescaler and max count to get the clock scaling you want
    - Prescaler is typically limited to a few select values

# Example

- Assume 1MHz input clock, need 3 kHz period from 8-bit timer
  - We want division factor of  $(1\text{M}/3\text{k})=333.3$ 
    - Setting compare register to 333 can't be done on 8-bit counter
  - We can set prescaler to 2 and compare register to  $(333.33/2-1) = 167-1$
  - We get a resulting period of  $(1\text{M}/167) = 2994\text{kHz}$
  - Alternatively you may use prescaler of 4 and max count of  $83-1$  to get  $3012\text{kHz}$

# Simultaneous Timers

- If multiple irregular (perhaps simultaneous) timing events are needed
- Leapfrogging
  - Moves compare value in software instead of resetting the counter in hardware
- Useful if multiple timers are to be implemented with limited set of hardware timers
- Ex: timing event and preserve total time
  - Set compare register for first trigger event and set up ISR
  - Reset and start counter
  - Upon call of ISR, determine next event value and set compare register
  - Repeat

# Extending Timer (software+overflow)

- The effective timer/counter range can be extended using software and the overflow bit
  - Can double range of counter for one go-through
- Example: count  $10.5 * \text{max counter}$ 
  - Start counter, value = 0
  - Counter overflows (overflow bit set)
  - Software detects overflow, increments value resets overflow flag
  - Value reaches 10, ISR set on compare register
  - Counter hits  $\frac{1}{2}$  max, interrupt occurs

# Multi-Register Access Problems

- Consider trying to read a 16-bit timer
  - Read lower value – 0x00(FF)
  - Read upper value – 0x(01)00
  - Would yield 0x01FF, which is incorrect
- Consider trying to write to a 16-bit timer
  - Write upper value – 0x(00)FF
  - Write lower value – 0x01(00)
  - Would yield 0x0100



# AVR Multi-Register Access solution

- Write to upper byte loads into temp register
- Write to lower byte triggers write to register from temp register to top byte
- Read from lower byte triggers upper byte to load into temp register
- Read from upper byte reads from temp register
- Always write top then lower
- Always read lower then top

# System Tick and Software Timers

- Can implement several low precision ( $\sim 1\text{ms}$ - $20\text{ms}$ ) timers using system tick and software compare events
- System “tick” would be a slower timer (perhaps  $0.1\text{ms}$  –  $1\text{ms}$ ) that triggers ISR to increment tick value

# Slip Adjustment

- If your system tick ends up not being a nice “round number” and you want to remove long-term drift
  - Slip your system tick software accumulator to remove drift
- For some leap ticks, count is modified, just as a day is added every leap year

# Event Counters

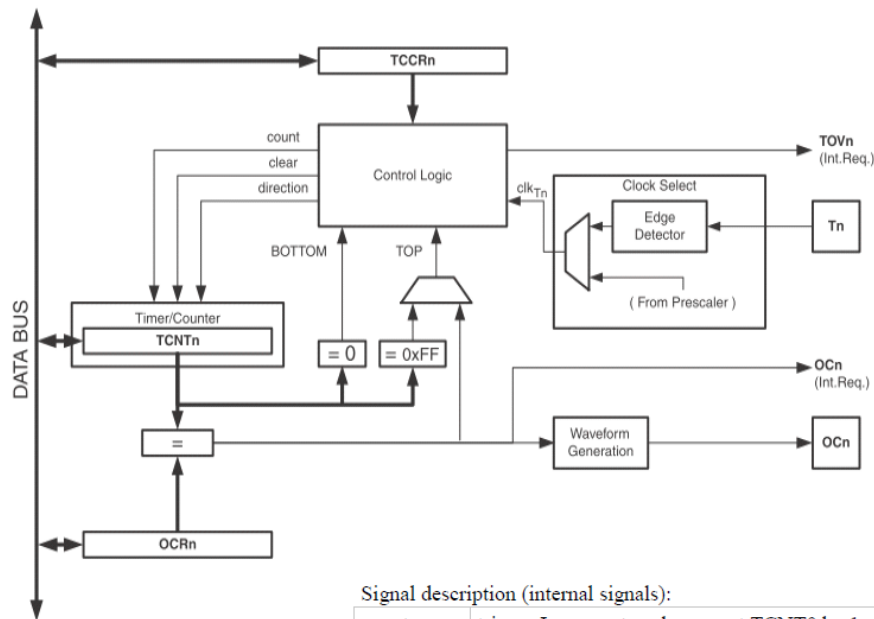
- Counters can be configured to count a certain number of events (usually for rapid events)
  - Select a trigger input rather than regular clock
  - Use external pin

# Counter Modes

- AVR timers have a few important modes of operation
- Non-PWM Modes
  - Normal Mode – count and reset at overflow – set 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

# AVR Counter Hardware

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 <sub>Tn</sub>	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).

# 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 9<sup>th</sup> bit on first count run
  - Can use TOV0 to extend counter
- Setting configuration bits COM0A[1:0] a designated output pin can be modified when counter matches the value of output compare register

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

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

# Clear Timer on Compare Match (CTC) Mode

- When counter reaches value of Output Compare Register (OCR0A) event is referred to as Output Compare Event
- Upon event, next value of counter is 0 and Output Compare Flag is set
- By changing output compare registers, one manipulates the counter max value and thus controls the count cycle frequency and frequency of derived waveforms or interrupt calls
- If output compare interrupt flag is set and the global interrupt enable flag is set, an interrupt is triggered and output compare flag is cleared automatically



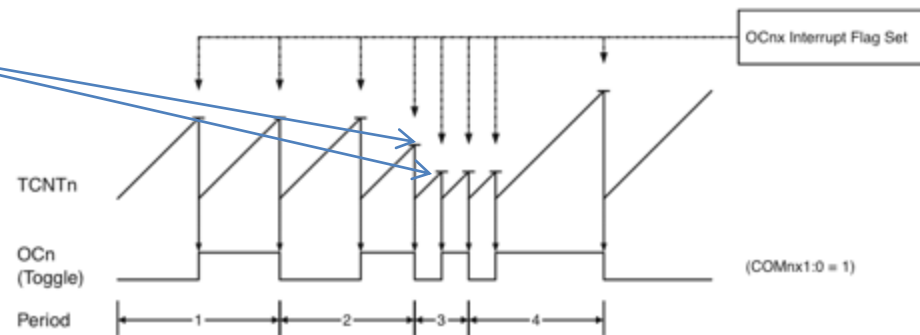
# CTC Mode Example

- Use CTC mode to toggle output pin on compare match
- Vary counter compare register to achieve different frequency

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

Figure 14-5. CTC Mode, Timing Diagram



Caution: if computing a new OCR0A value in software after the Compare Flag is set, if the value is too small and the software is 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 roll over, and the match will occur on the next count cycle and may result in an irregular waveform. This also why timers (TCNT0) should be set to 0 initially.

# Fast PWM

- Clears/sets value when hits compare register
- Sets/clears value when overflow to bottom

Figure 14-6. Fast PWM Mode, Timing Diagram

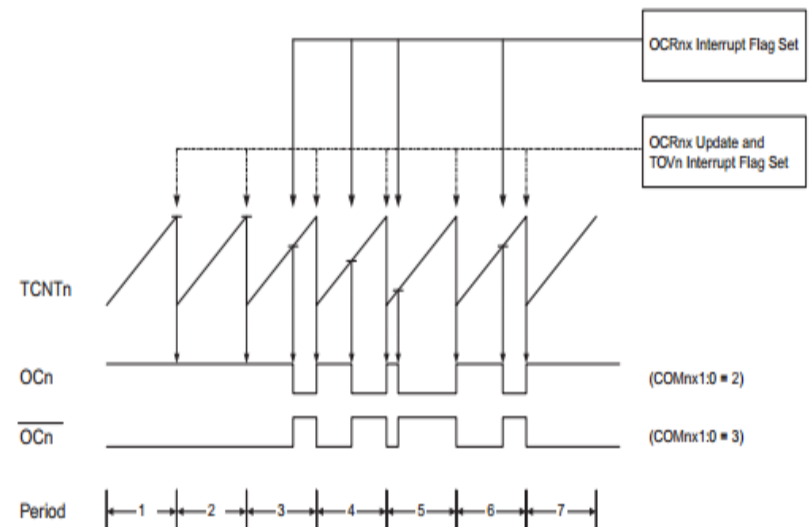


Table 14-4. Compare Output Mode, Fast PWM Mode<sup>(1)</sup>

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)

# Phase Correct PWM Mode

- Phase correct PWM modes are like standard PWM modes but the center of the pulse does not change spacing when 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

# Phase Correct PWM Mode

- Clears/sets on compare match on up slope
- Sets/clears on compare match on down slope

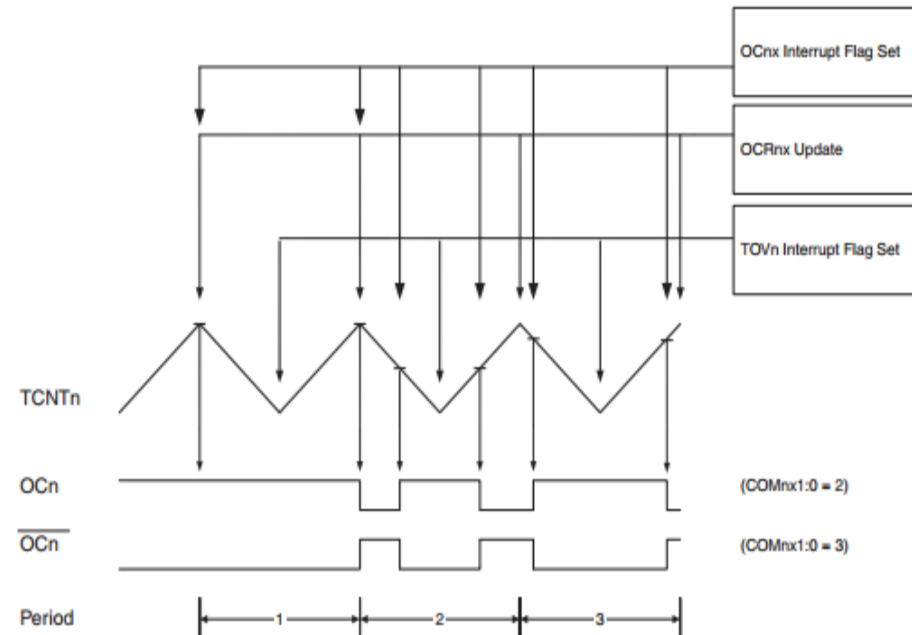


Table 14-5. Compare Output Mode, Phase Correct PWM Mode<sup>(1)</sup>

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.

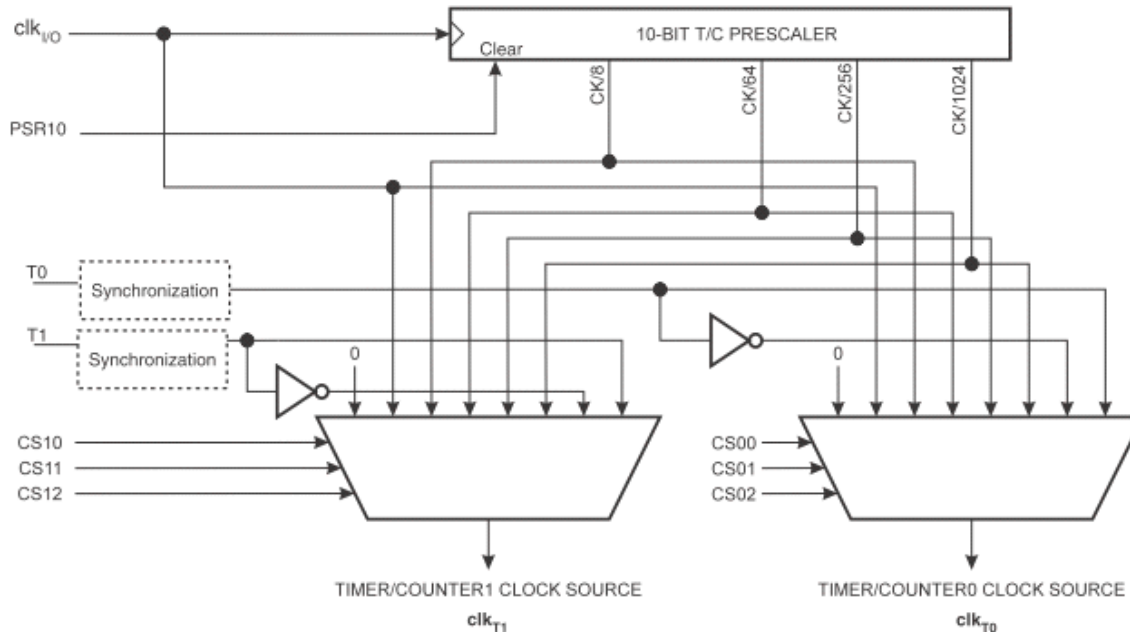
# 16-Bit Timers

- Similar to 8-bit timers
  - 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 will take care of the order

# Timer Counter 0 and 1

- Share the same prescaler but have separate selection

Figure 16-2. Prescaler for Timer/Counter0 and Timer/Counter1<sup>(1)</sup>



# Timer/Counter 1 Modes

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

Figure 15-7. Fast PWM Mode, Timing Diagram

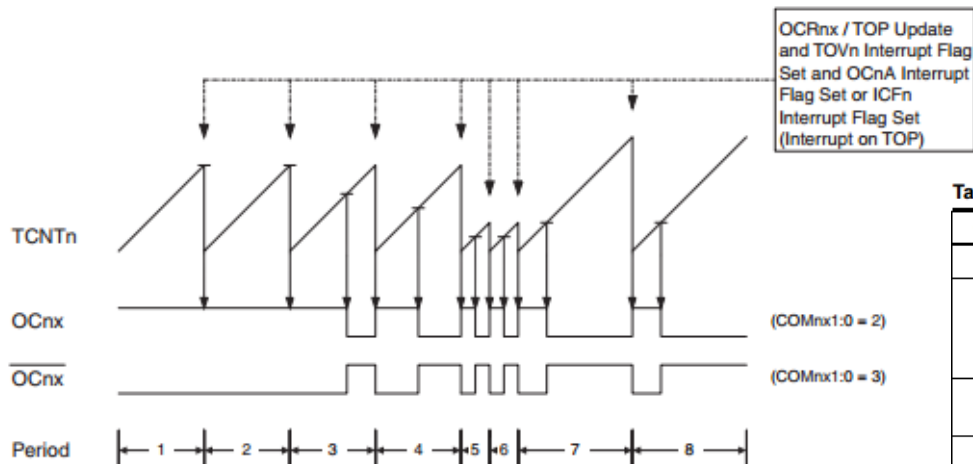


Table 15-2. Compare Output Mode, Fast PWM<sup>(1)</sup>

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)