TIMERS/COUNTERS

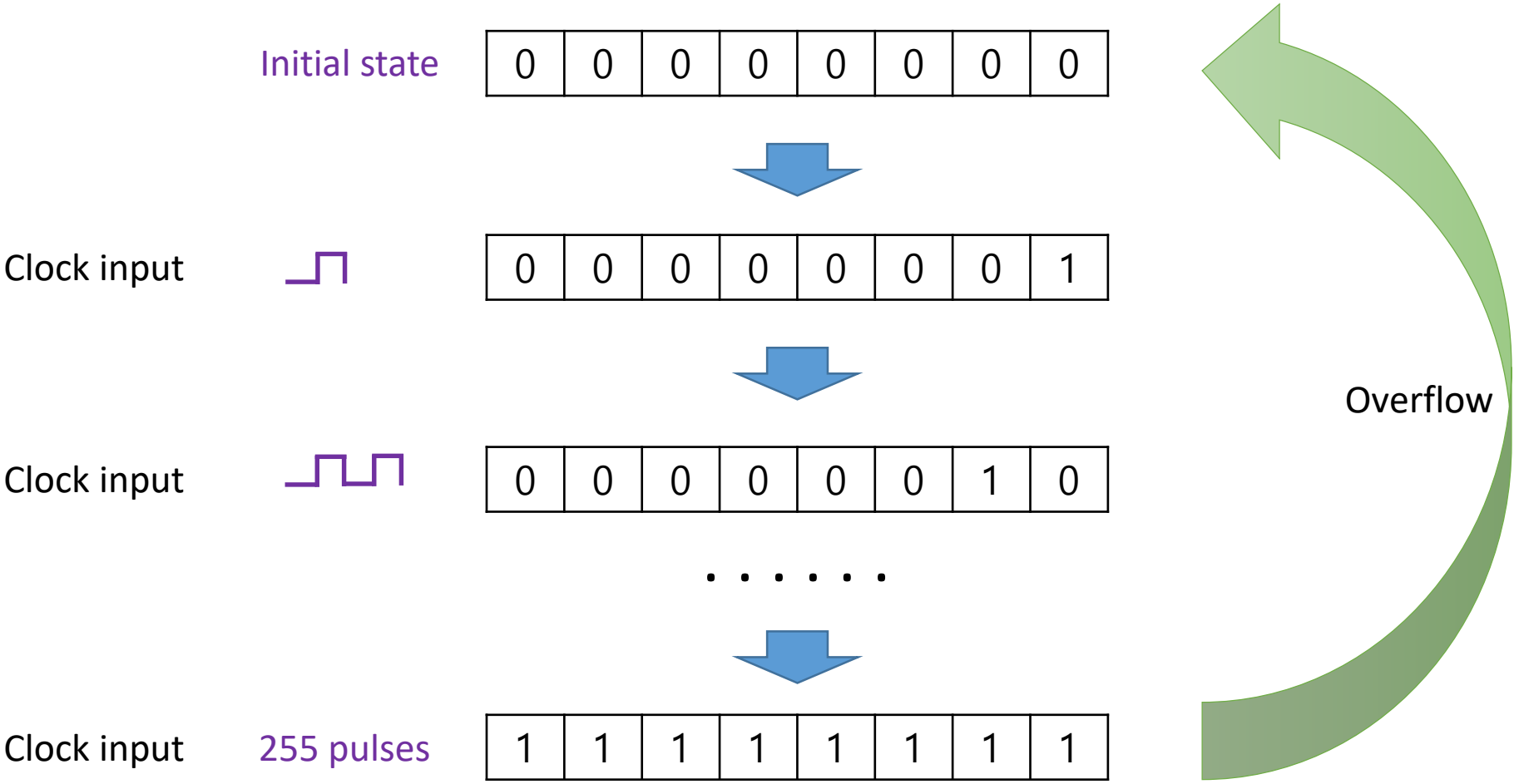
ATmega328P(B) Timers/Counters

	Width (bits)	ATmega328P	ATmega328PB
Timer/Counter0	8	○	○
Timer/Counter1	16	○	○
Timer/Counter2	8	○	○
Timer/Counter3	16	X	○
Timer/Counter4	16	X	○

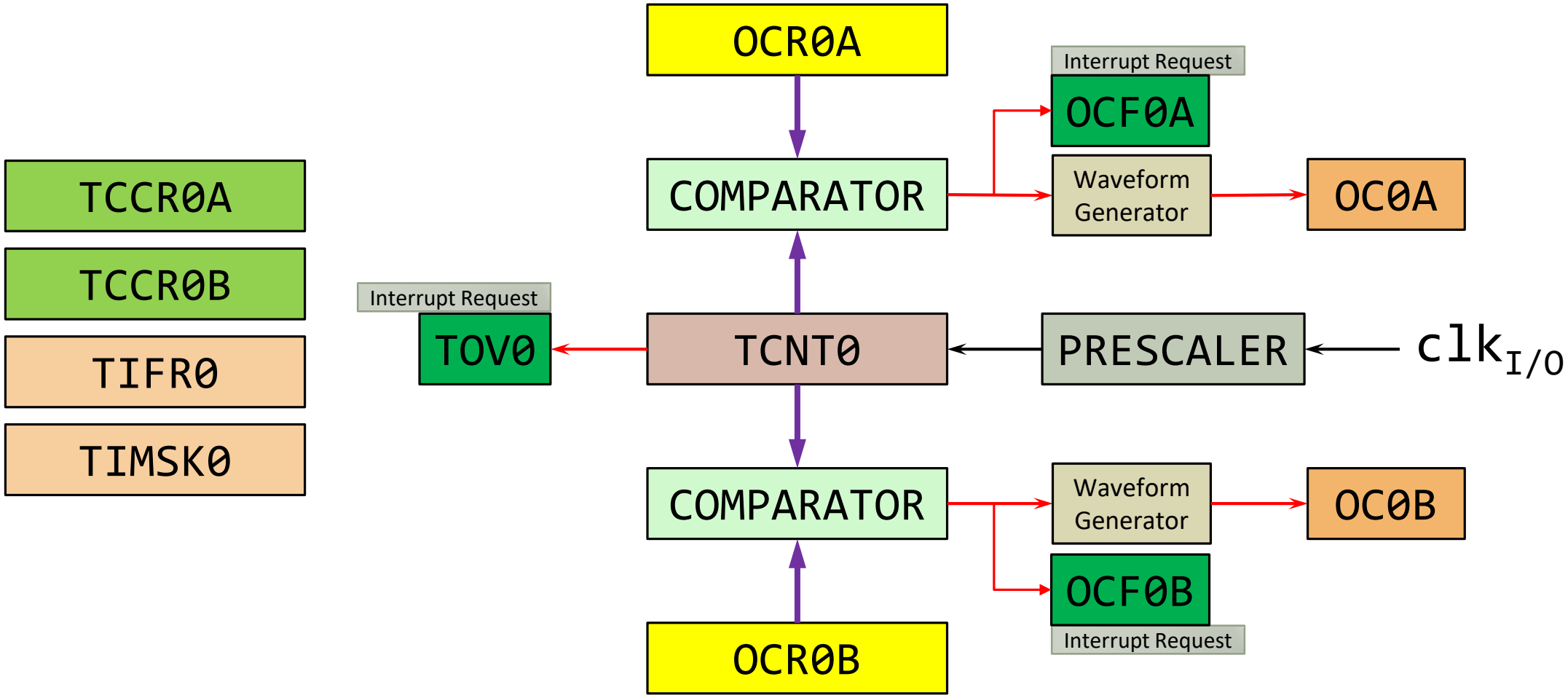
Timer/Counter Applications

- To do some works periodically. (CTC mode)
- To control load current/voltage. (PWM)
- To count external event. (Normal)

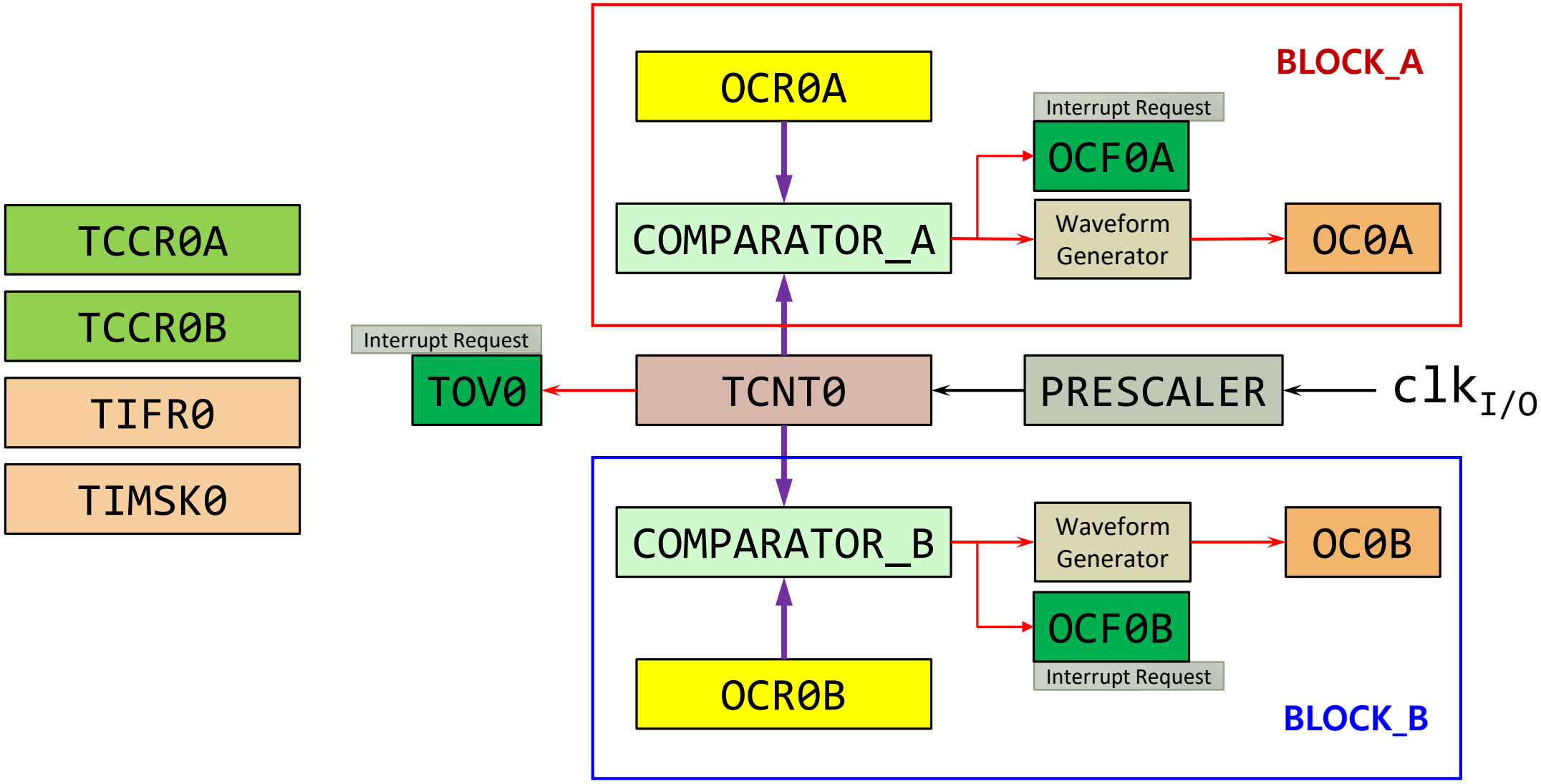
8-Bit Timer/Counter Basics



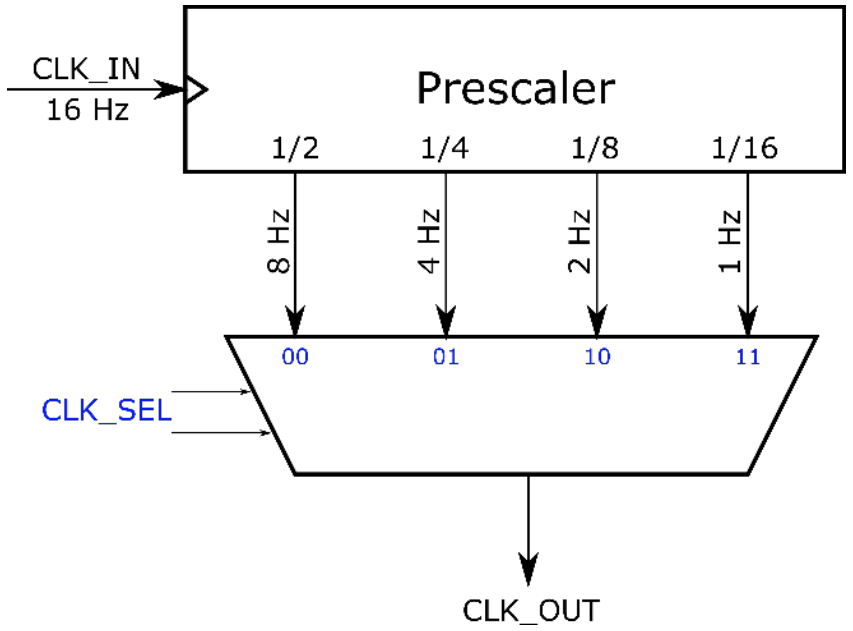
ATmega328PB Timer/Counter0 Block Diagram

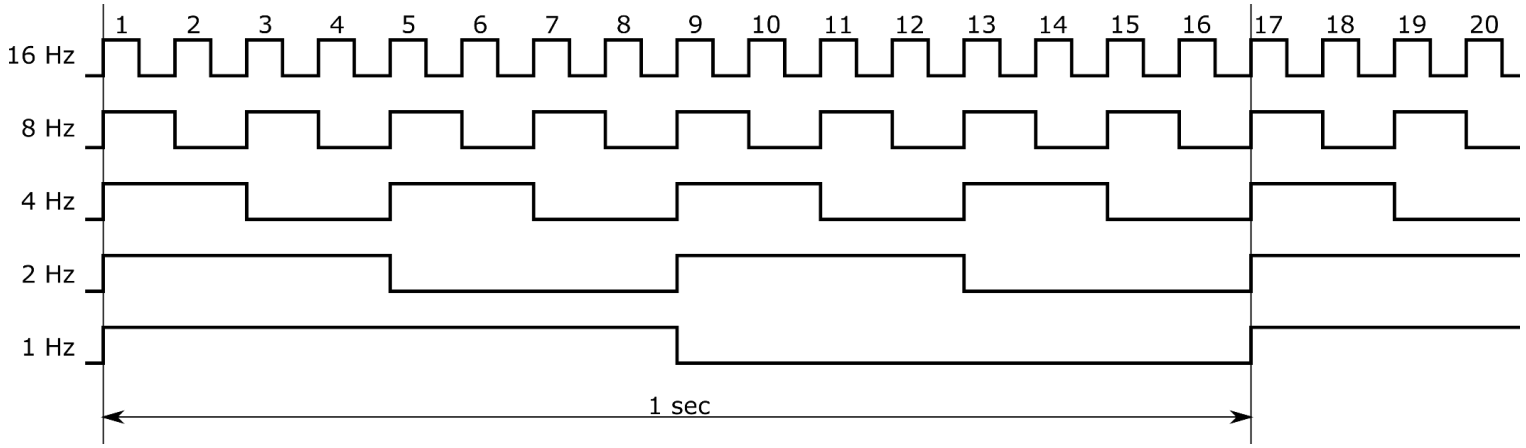


ATmega328PB Timer/Counter0 Block Diagram

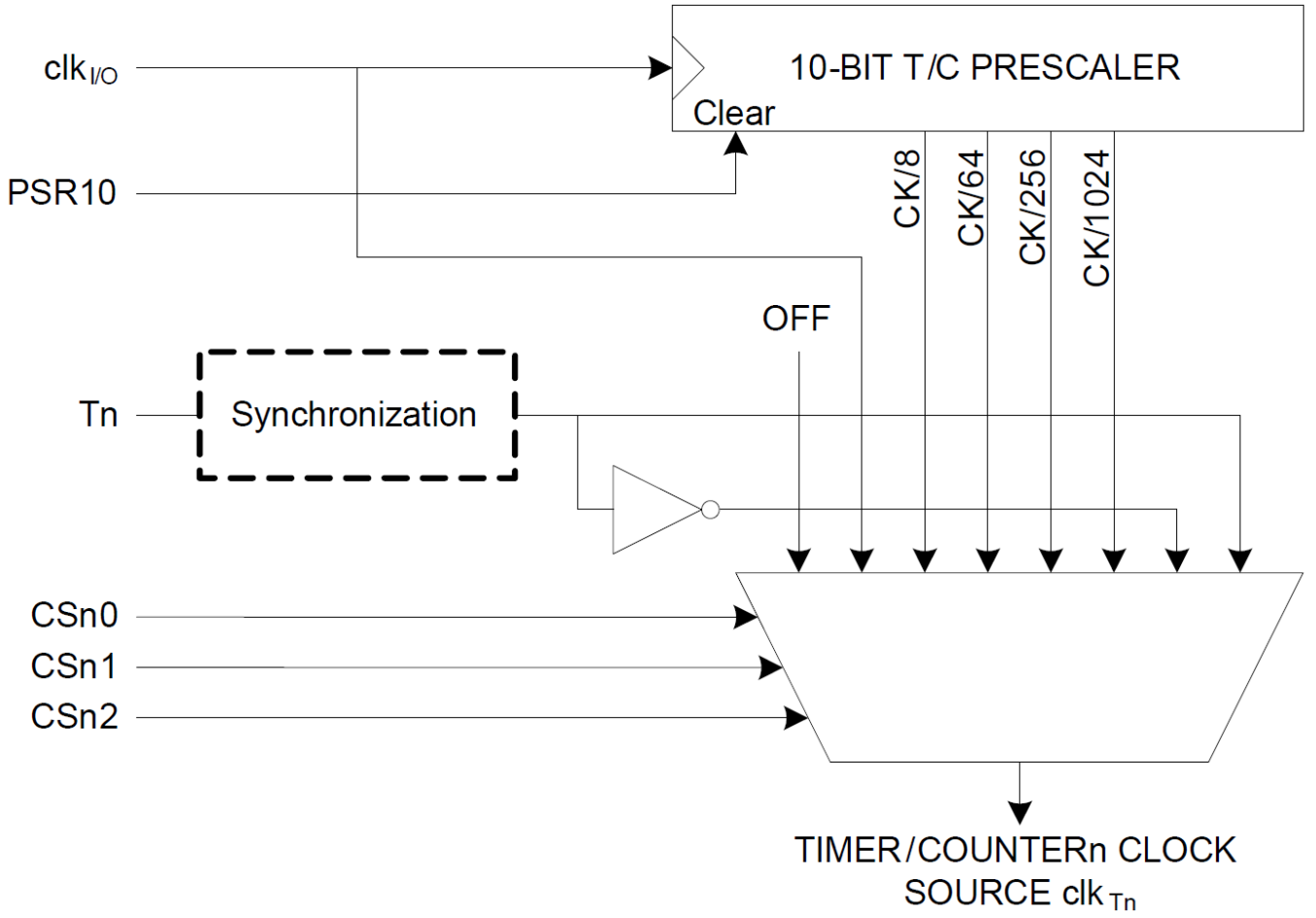


What is a Prescaler?





Prescaler for Timer/Counter0, 1, 3, 4



8-bit Timer/Counter0

Mode of Operation

- Determines the behavior of the Timer/Counter and the Output Compare pins.
- Defined by WGM0[2:0] in TCCR0B[3] and TCCR0A[1:0] registers.

MAX = 0xFF
BOTTOM = 0x00

Mode	WGM0[2]	WGM0[1]	WGM0[0]	Mode of Operation	TOP	Update of OCR0x at	TOV flag set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	Phase Correct PWM	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCR0A	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	Phase Correct PWM	OCR0A	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCR0A	BOTTOM	TOP

Timer/Counter0
Normal mode

Normal Mode

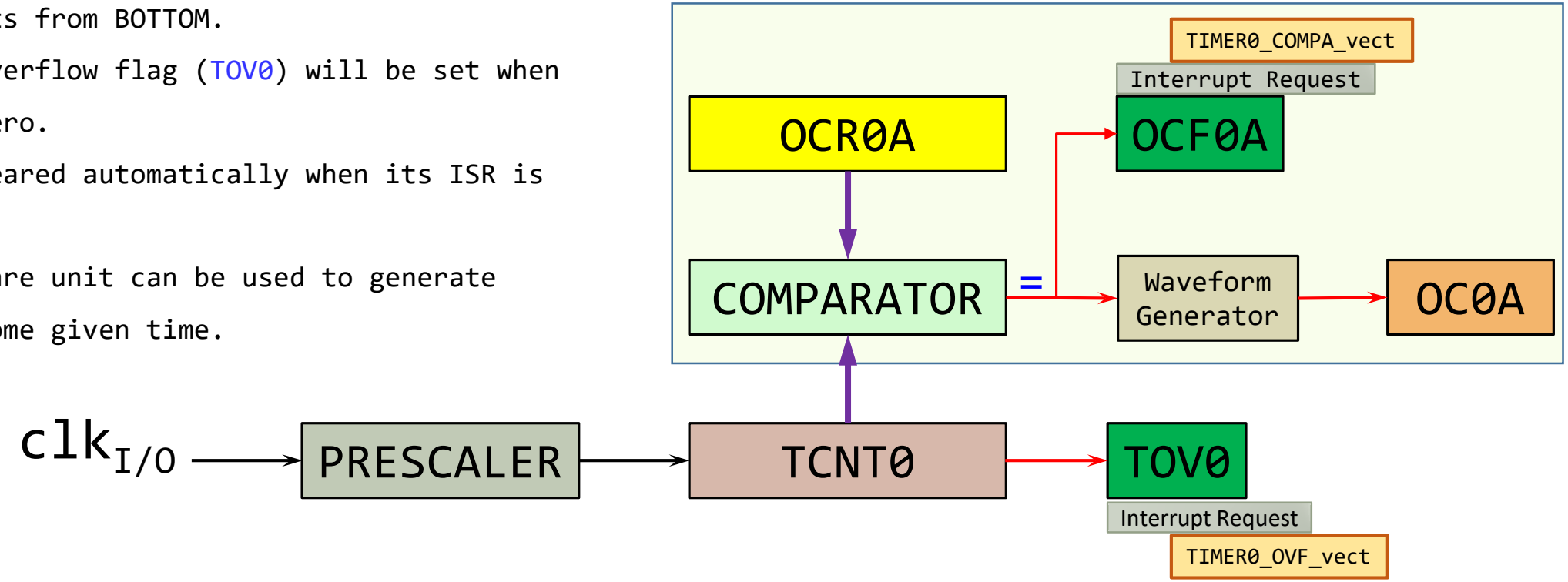
WGM0[2:0] = 0b000 in TCCR0B[3] and TCCR0A[1:0] registers.

MAX = 0xFF
BOTTOM = 0x00

Mode	WGM0[2]	WGM0[1]	WGM0[0]	Mode of Operation	TOP	Update of OCR0x at	TOV flag set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	Phase Correct PWM	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCR0A	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	Phase Correct PWM	OCR0A	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCR0A	BOTTOM	TOP

Normal Mode: Simple Counter

- WGM0[2:0] = 0b000
- TCNT0 always counts from BOTTOM (0x00) to TOP (0xFF) and then restarts from BOTTOM.
- Timer/Counter Overflow flag (TOV0) will be set when TCNT0 becomes zero.
- TOV0 flag is cleared automatically when its ISR is executed.
- The output compare unit can be used to generate interrupts at some given time.



Timer/Counter0

CTC mode

(Clear Timer on Compare Match mode)

CTC Mode

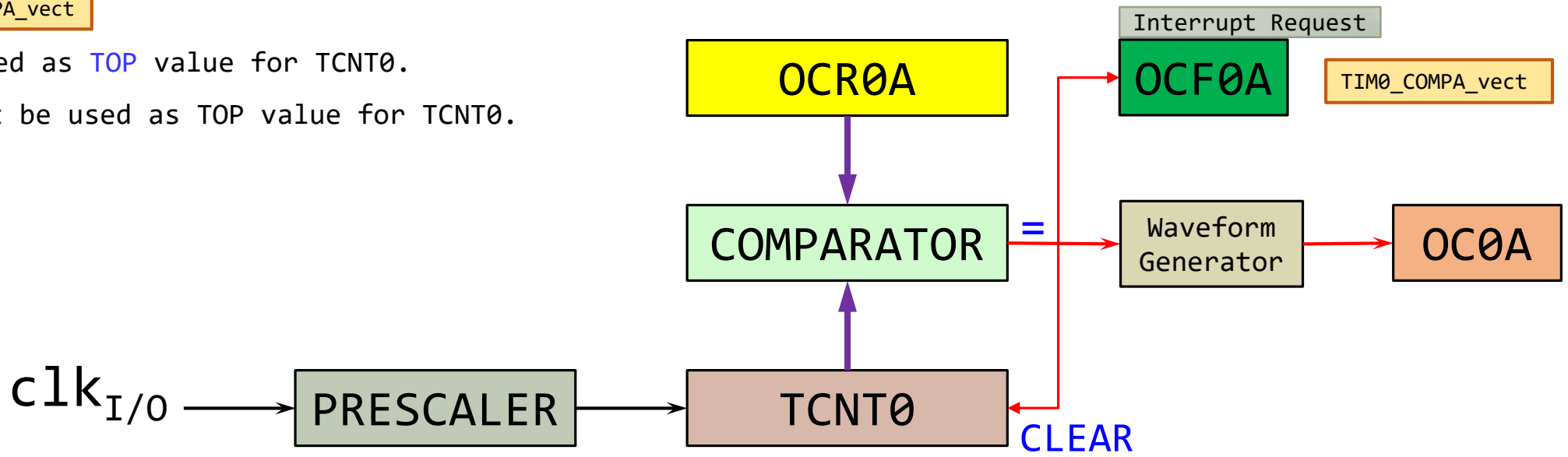
WGM0[2:0] = 0b010 in TCCR0B[3] and TCCR0A[1:0] registers.

MAX = 0xFF
BOTTOM = 0x00

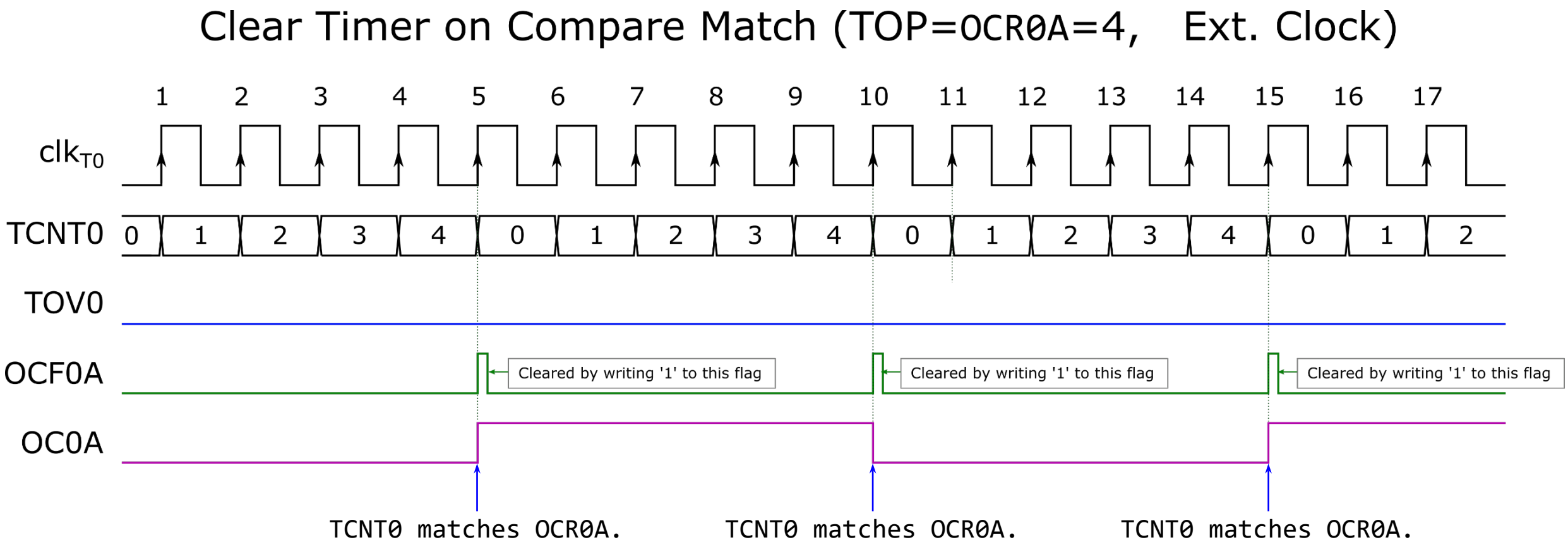
Mode	WGM0[2]	WGM0[1]	WGM0[0]	Mode of Operation	TOP	Update of OCR0x at	TOV flag set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	Phase Correct PWM	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCR0A	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	Phase Correct PWM	OCR0A	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCR0A	BOTTOM	TOP

CTC Mode: Clear Timer on Compare Match (1)

- WGM0[2:0] = 0b010
- TCNT0 repeats counting from 0 to the value of OCR0A register.
- When TCNT0 value matches OCR0A value,
  - ✓ OC0A output can be set to toggle. (when COM0A[1:0]=0b01)
  - ✓ OCF0A flag is set and an interrupt can be generated.
    - ➢ TIM0_COMPA_vect
- Only OCR0A is used as TOP value for TCNT0.
  - ✓ OCR0B cannot be used as TOP value for TCNT0.

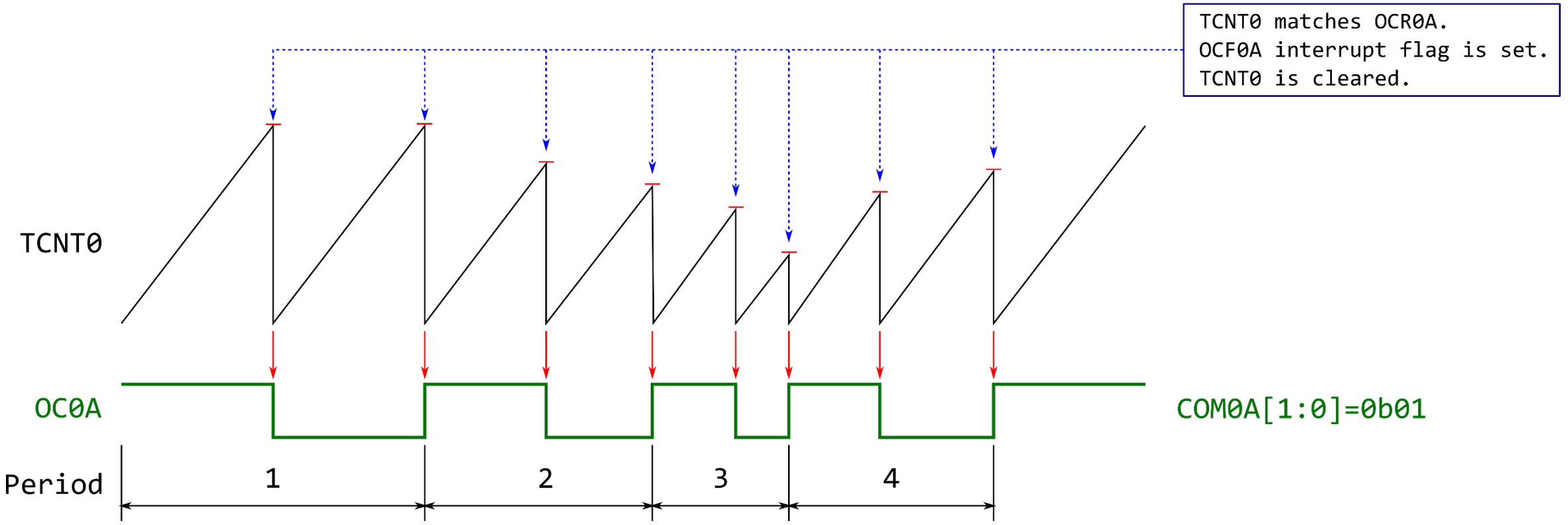


CTC Mode: Clear Timer on Compare Match (2)



CTC Mode: Clear Timer on Compare Match (3)

- WGM0[2:0] = 0b010
- TCNT0 repeats counting from 0 to the value of OCR0A register.
- When TCNT0 value matches OCR0A value,
  - ✓ OC0A output can be set to toggle. (when COM0A[1:0]=0b01)
  - ✓ OCF0A interrupt flag is set and interrupt can be generated.

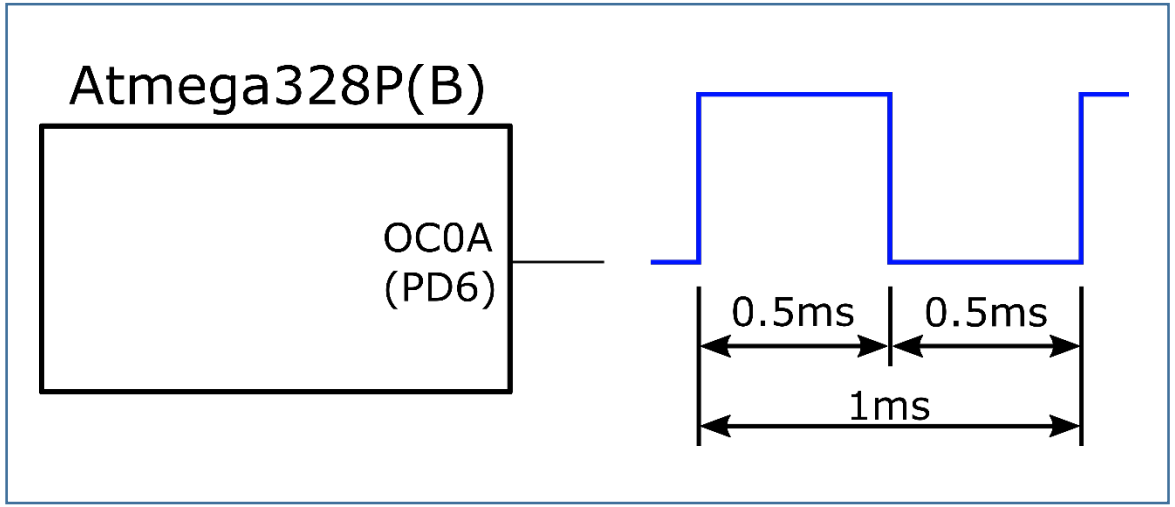


Timer/Counter0
CTC Example 1

Timer/Counter0 CTC Example 1 (1)

Waveform Generation

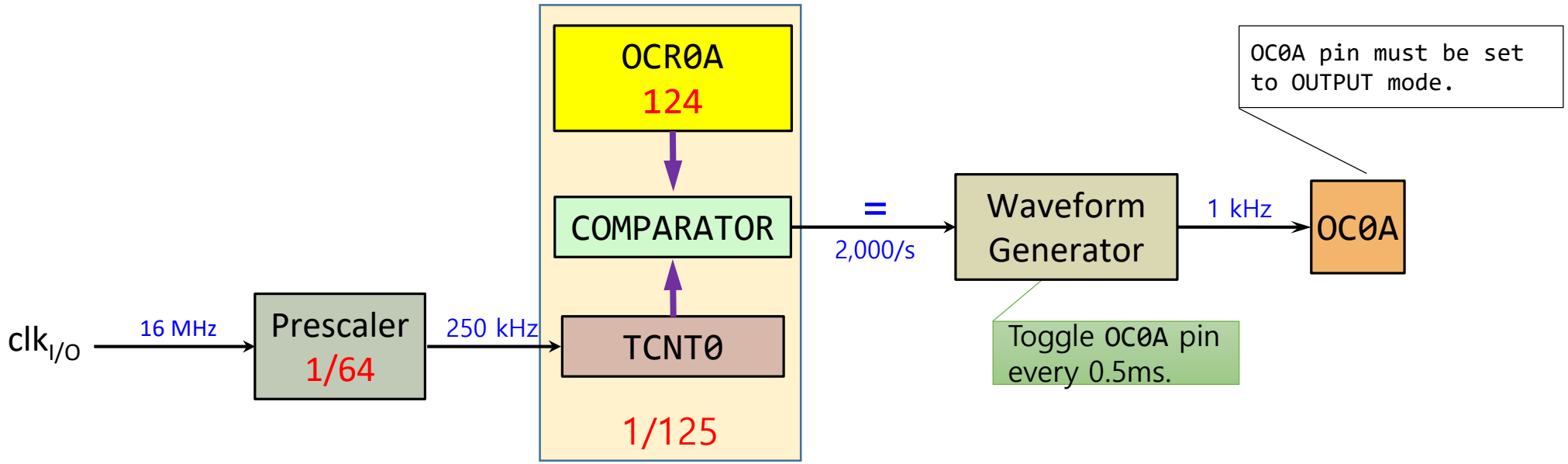
- Make an application which generates 1 kHz rectangular wave on OC0A (PD6).
- Use Timer/Counter0 CTC mode.
- Assume that system clock frequency is 16 MHz.



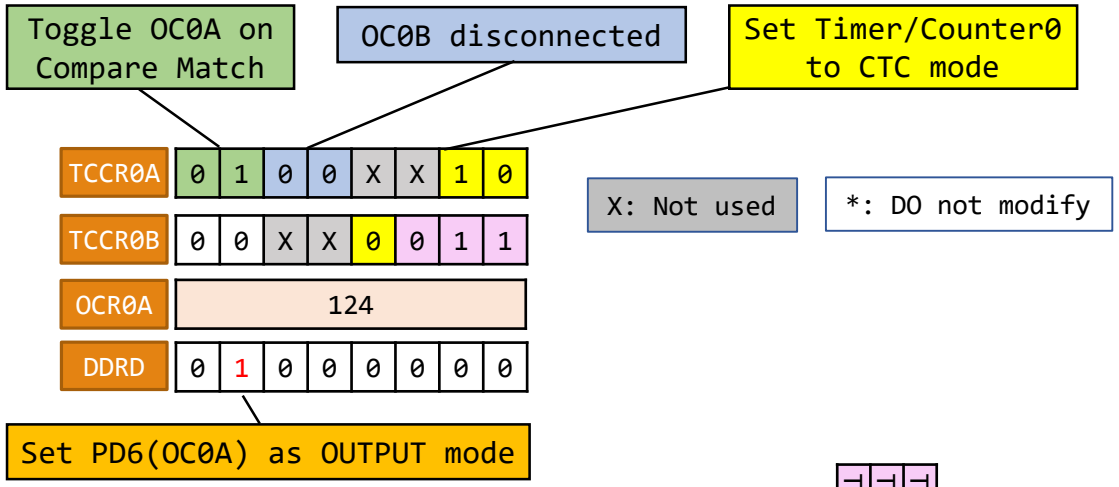
Timer/Counter0 CTC Example 1 (2)

Waveform Generation

- Make an application which generates 1 kHz rectangular wave on OC0A (PD6).
- Use Timer/Counter0 CTC mode. (Assume that system clock frequency is 16 MHz)

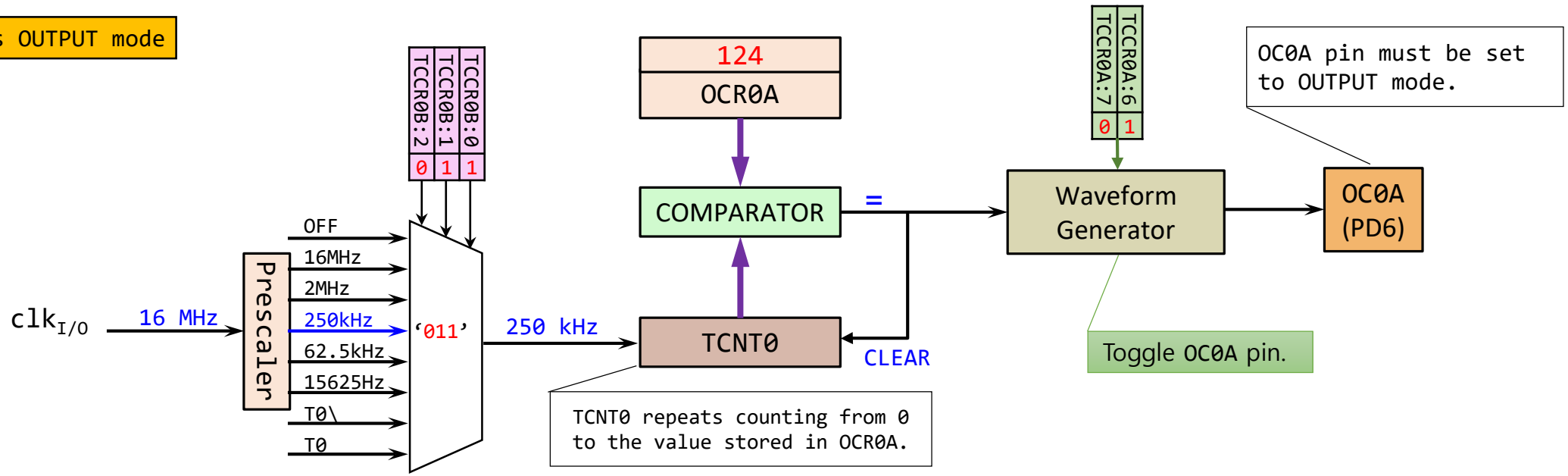


Timer/Counter0 CTC Example 1 (3)



Waveform Generation

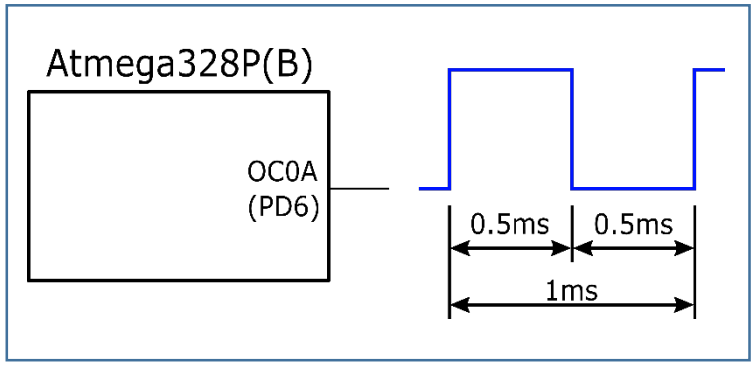
- Make an application which generates 1 kHz rectangular wave on OC0A (PD6).
- Use Timer/Counter0 CTC mode.
- Assume that system clock frequency is 16 MHz.



Timer/Counter0 CTC Example 1 (4)

Make an application which generates 1 kHz rectangular wave on OC0A (PD6).

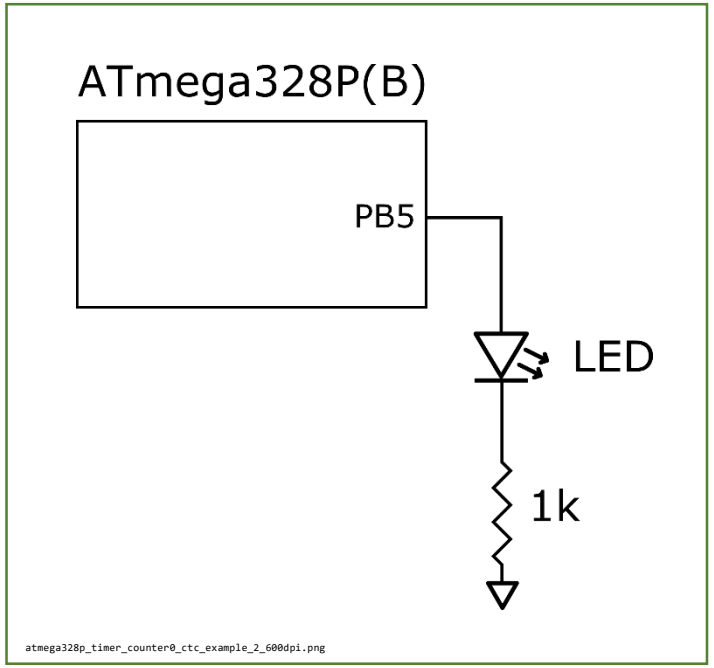
```
/* timer0_ctc_oc0a_1khz.c */

#include <avr/io.h>

int main(void)
{
    DDRD   = (1 << 6);          // Set PD6 (OC0A) to OUTPUT mode
    TCCR0A = 0b01000010;        // Set Timer/Counter0 to CTC mode. Toggle OC0A.
    TCCR0B = 0b00000011;        // select Prescaler division ratio. 1/64
    OCR0A  = 124;

    while (1)
    { /* do nothing */ }
}
```



Timer/Counter0
CTC Example 2

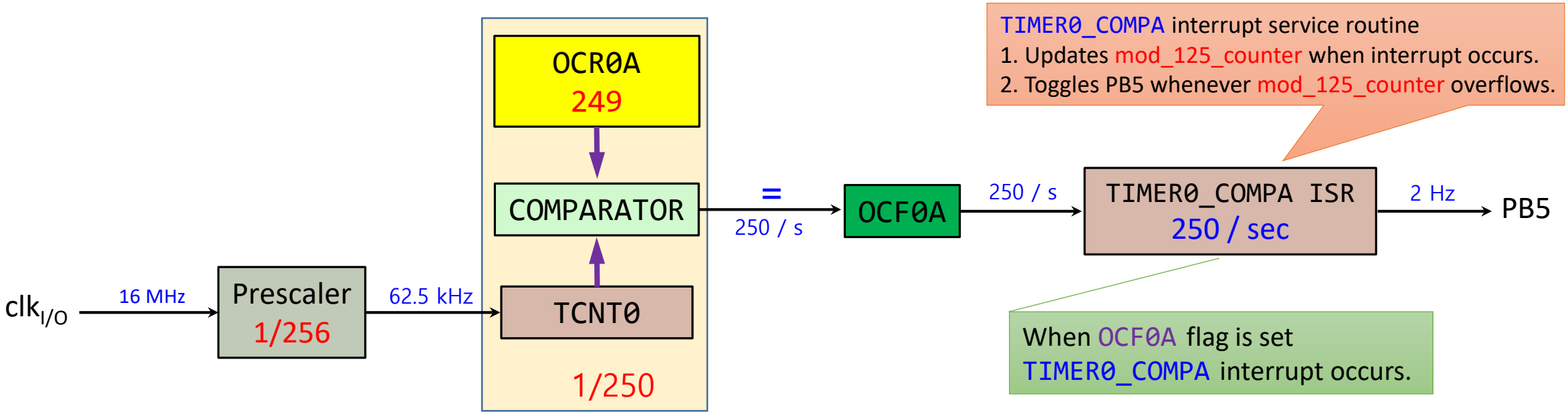
Timer/Counter0 CTC Example 2 (1)

- Make an application which toggles the LED connected to PB5 every 500 msec using Timer/Counter0.
- Use Timer/Counter0 Output Compare Match A Interrupt.
- Assume that system clock frequency is 16 MHz.

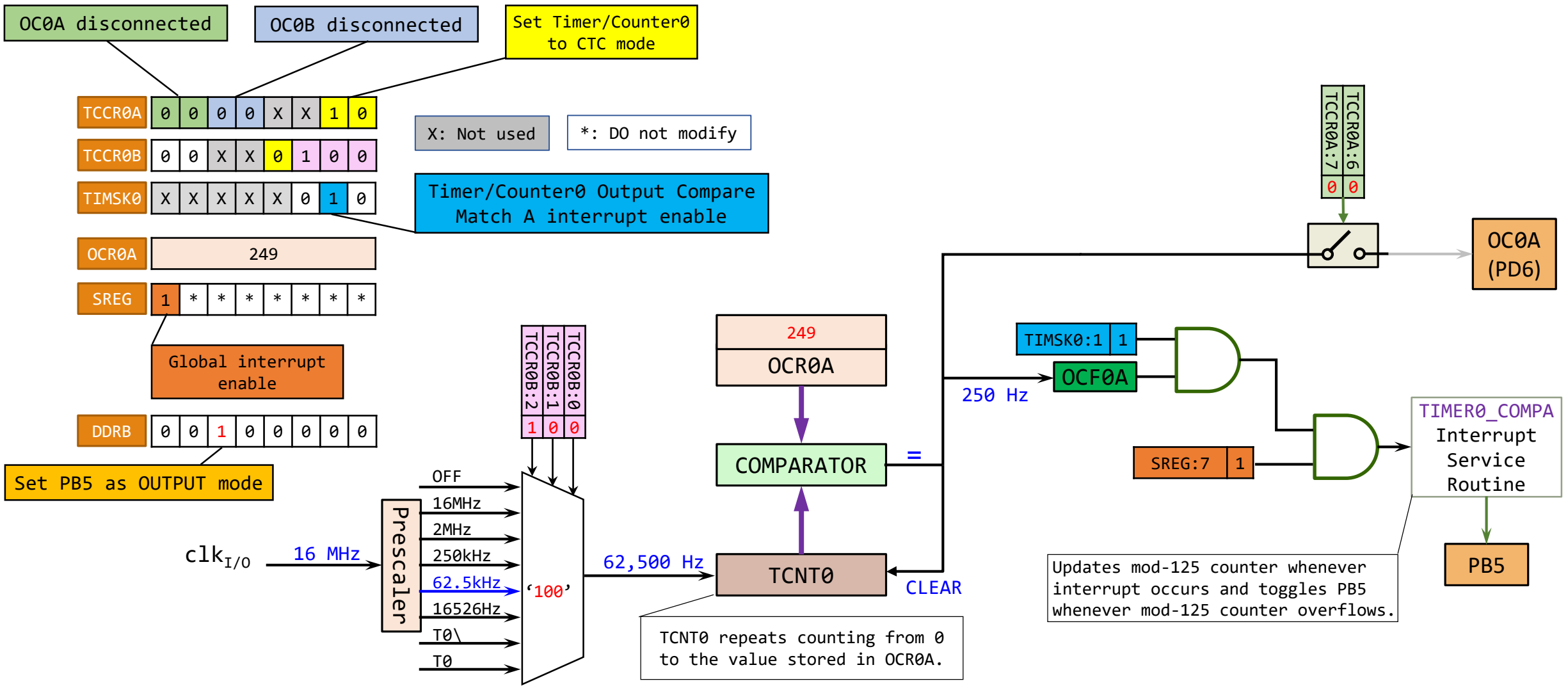


Timer/Counter0 CTC Example 2 (2)

CTC: Clear Timer on Compare Match.



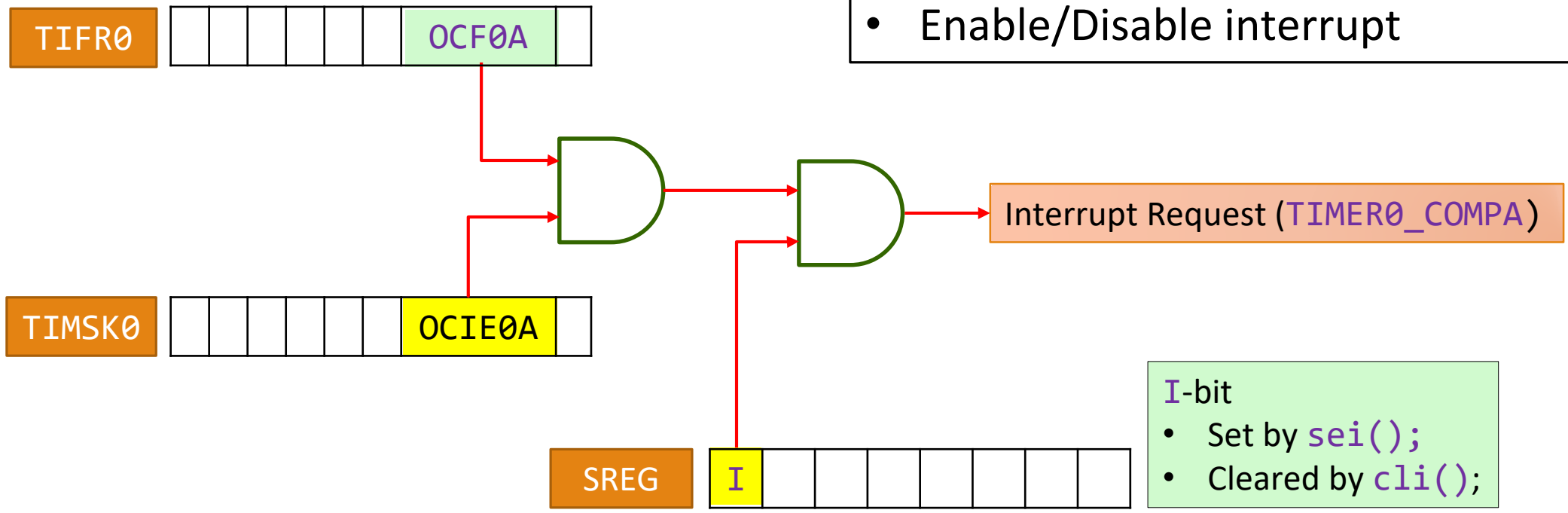
Timer/Counter0 CTC Example 2 (3)



Timer/Counter0 Output Compare Match A Interrupt

OCF0A flag is
- Set when TCNT0 and OCR0A matches.
- Cleared when executing ISR.

- Interrupt
- Interrupt Vector
- Interrupt Service Routine (ISR)
- Enable/Disable interrupt

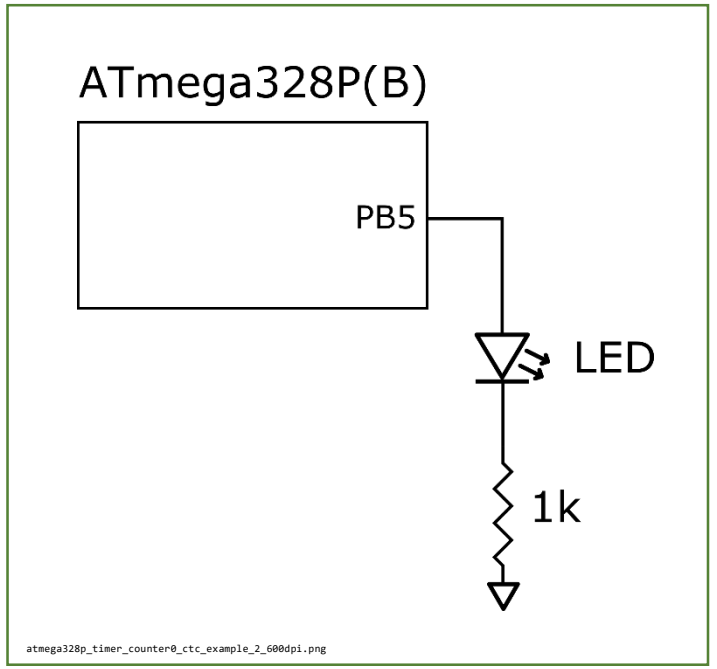


I-bit
- Set by sei();
- Cleared by cli();

Timer/Counter0 CTC Example 2 (4)

Make an application which toggles the LED every 500 ms using Timer/Counter0.

```c
/* timer0_ctc_toggle_pb5_500ms.c */

#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    DDRB   = (1 << DDB5);     // Set PB5 to OUTPUT mode
    TCCR0A = 0b00000010;      // Set Timer/Counter0 to CTC mode. Disconnect OC0A and OC0B
    TCCR0B = 0b00000100;      // select Prescaler division ratio. 1/256
    OCR0A  = 249;             // set Time/Counter0 TOP value.

    TIMSK0 = 0b00000010;      // Timer/Counter0 Output Compare Match A interrupt enable.
    sei();                    // Enable global interrupt

    while (1)
    { /* do nothing */ }
}

ISR(TIMER0_COMPA_vect)        // Timer/Counter0 Output Compare Match A interrupt service routine
{
    static unsigned char mod_125_counter = 0;

    mod_125_counter++;                // Update mod-125 counter
    if (mod_125_counter == 125)       // If mod-125 counter reaches 125
    {
        PINB |= (1 << PINB5);         // Toggle PB5
        mod_125_counter = 0;          // Clear mod-125 counter
    }
}
```

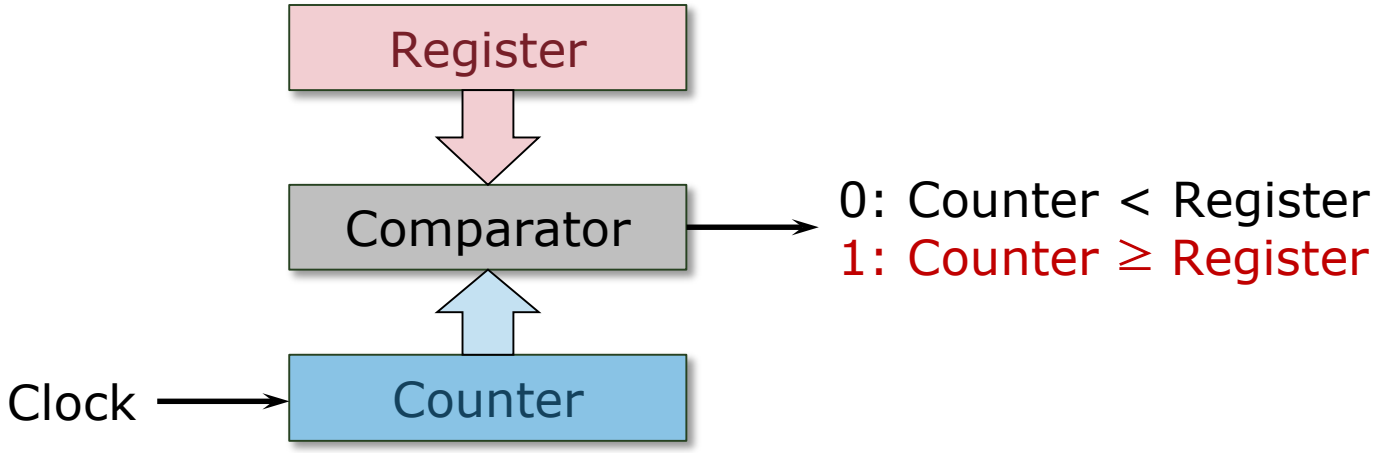


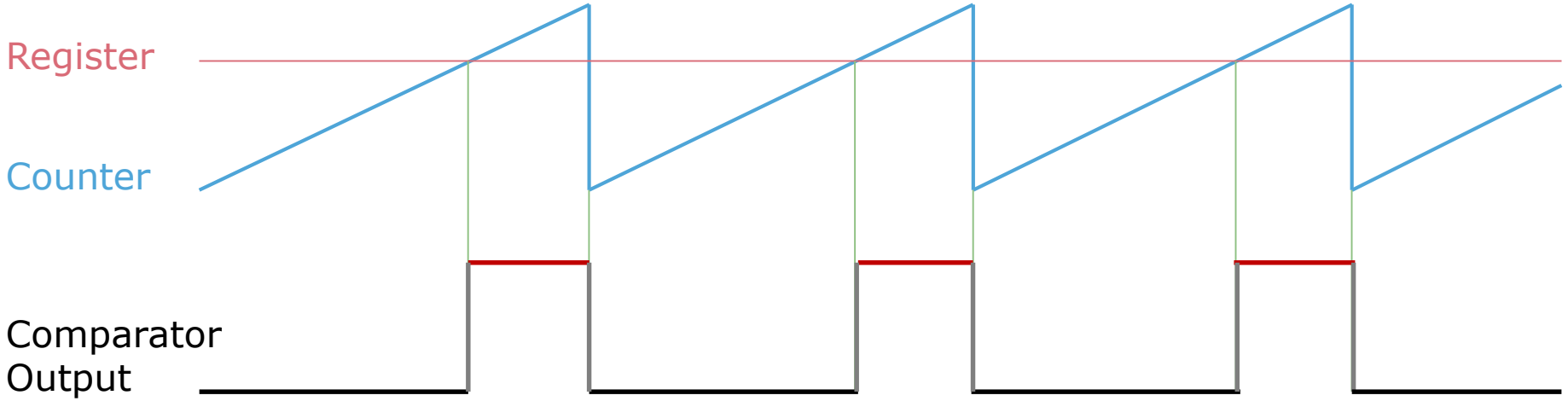
Timer/Counter0

Fast PWM mode

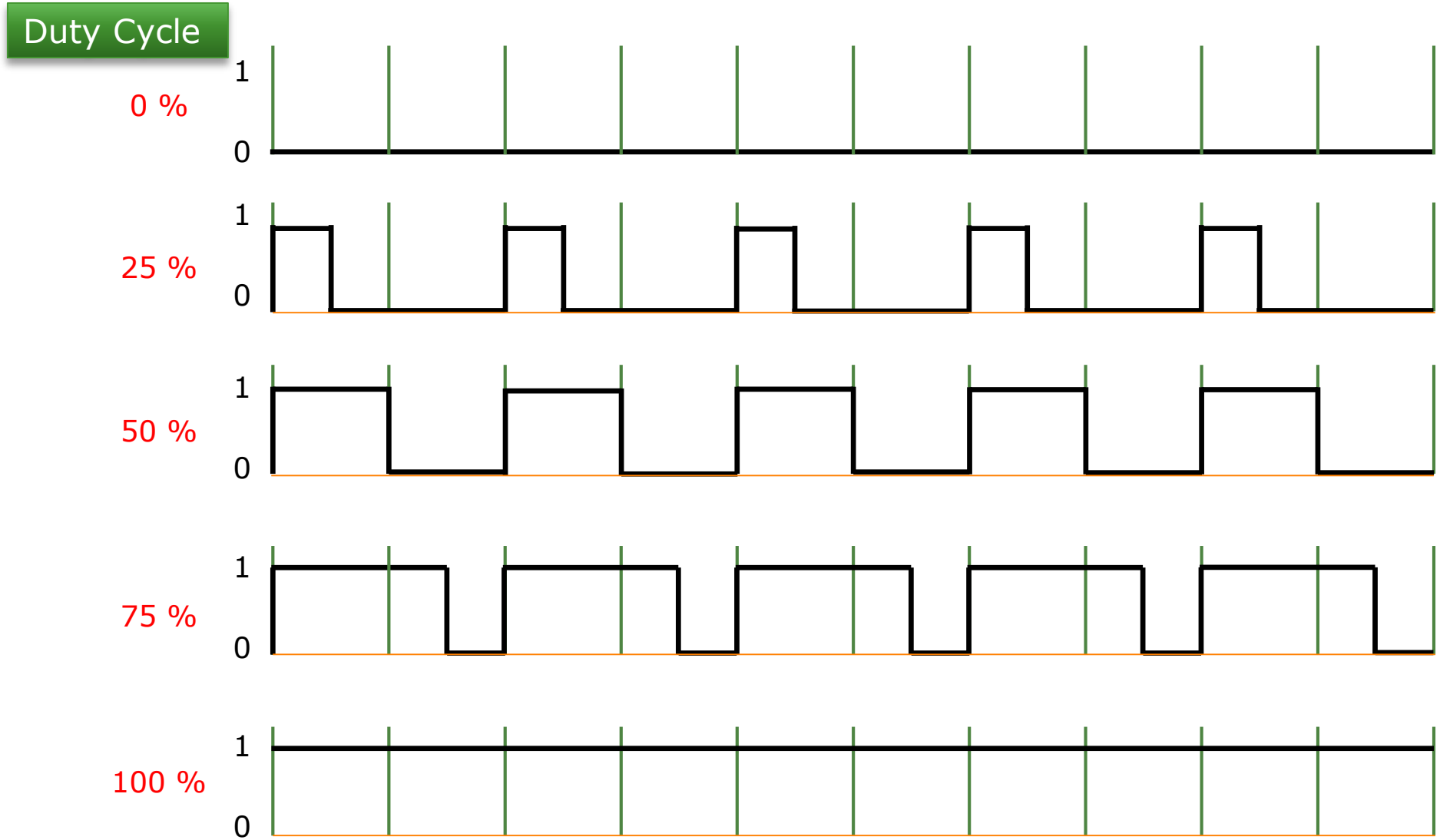
(Fast Pulse Width Modulation mode)

Pulse Width Modulation (PWM)

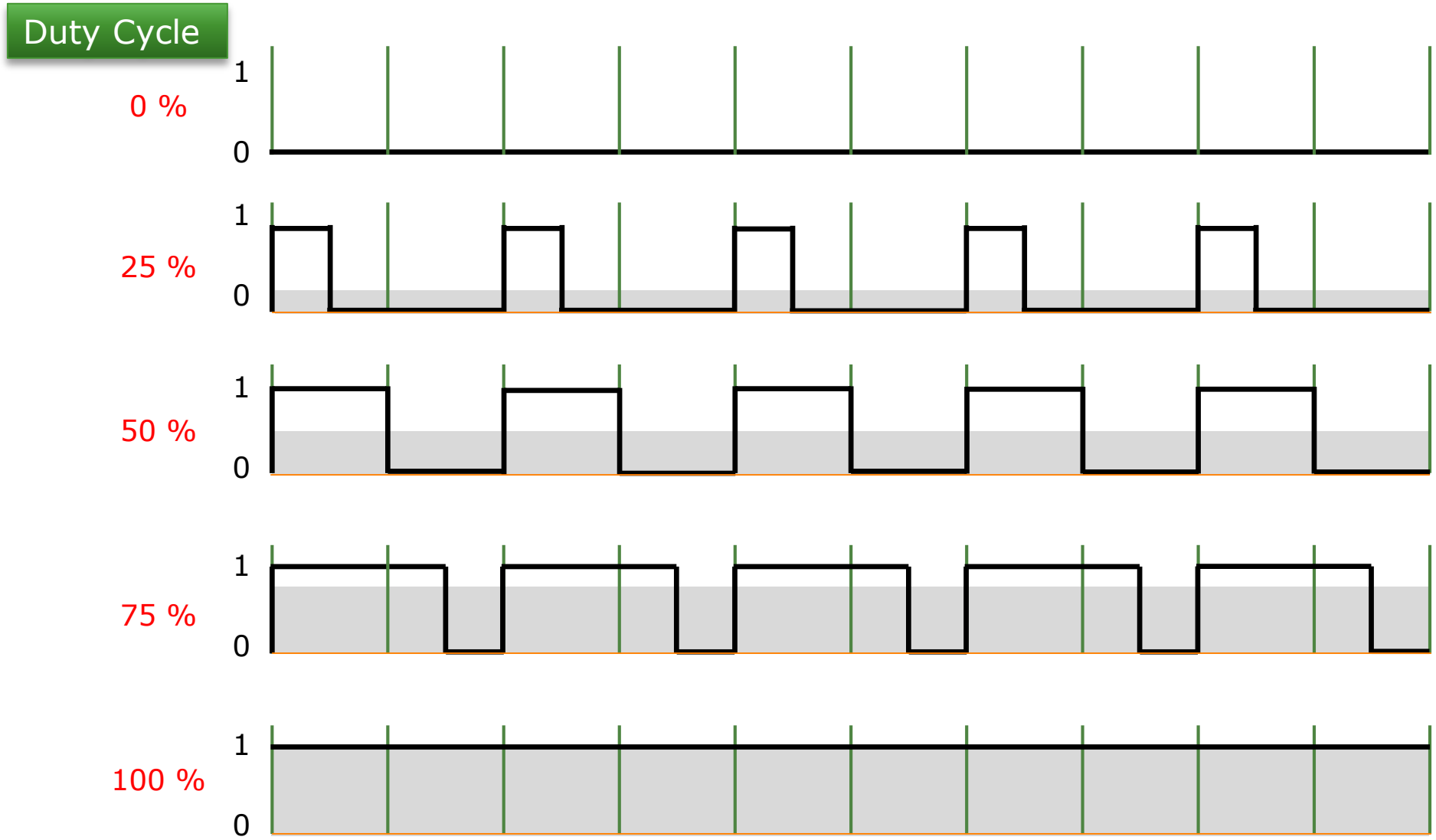




Pulse Width Modulation (PWM)



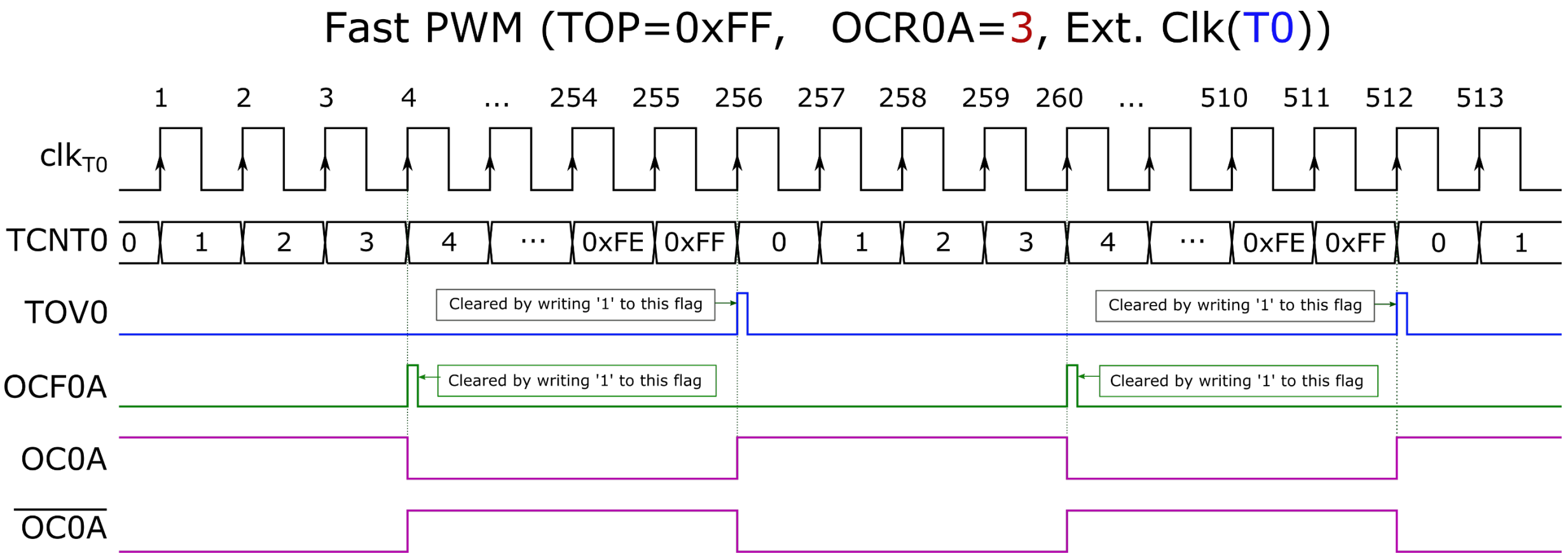
Pulse Width Modulation (PWM)



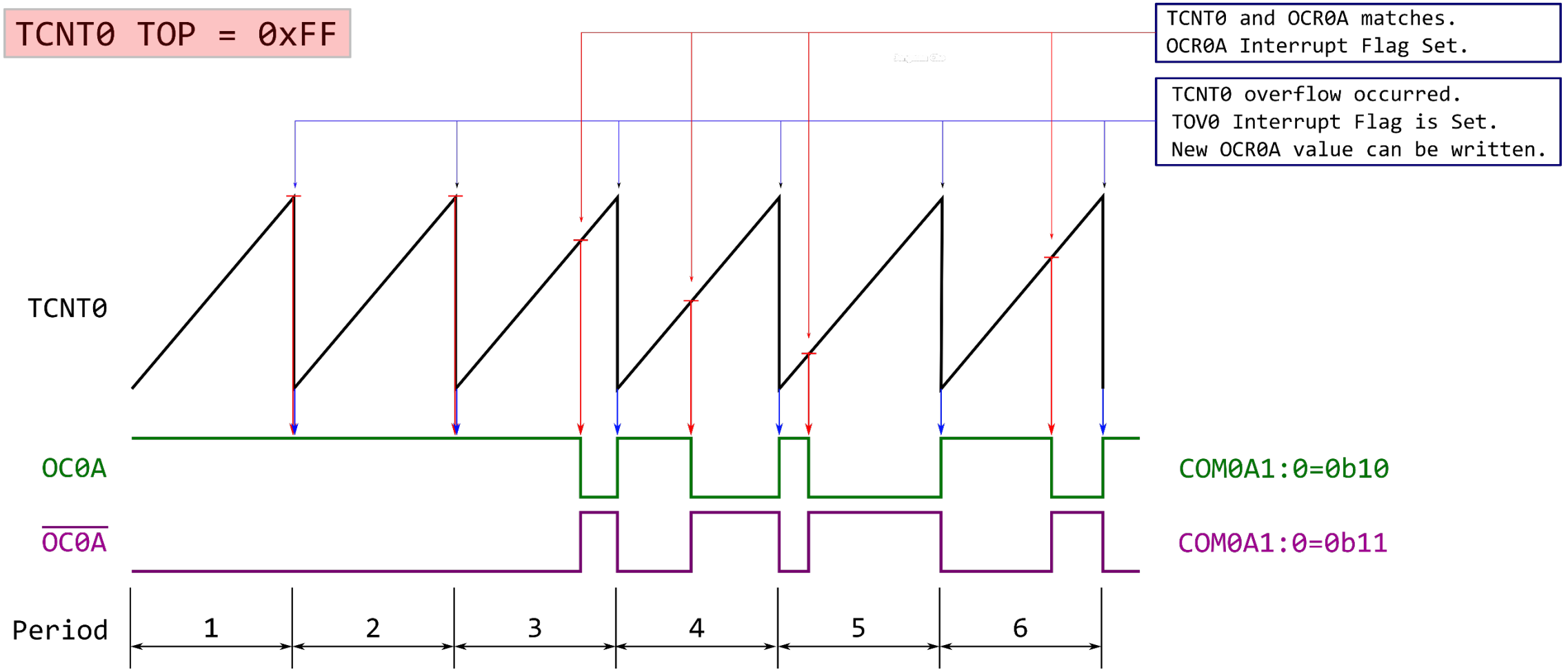
Timer/Counter0 Fast PWM Mode (1)

- WGM0[2:0]=0b011 or WGM0[2:0]=0b111
- The counter(TCNT0) counts from BOTTOM to TOP, then restarts from BOTTOM.
  - ✓ BOTTOM is always 0.
  - ✓ TOP is
    - 0xFF when WGM[02:00]=0b011.
    - OCR0A when WGM[02:00]=0b111.
- The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches TOP.
  - ✓ If the interrupt is enabled, the interrupt handler routine(TIMER0_OVF_vect) can be used for updating the compare value.
- Two types of PWM signals can be available
  - ✓ Invert: COM0A[1:0]=0b11
  - ✓ Non-invert: COM0A[1:0]=0b10

Timer/Counter0 Fast PWM Mode (2)



Timer/Counter0 Fast PWM Mode (3)



Timer/Counter0

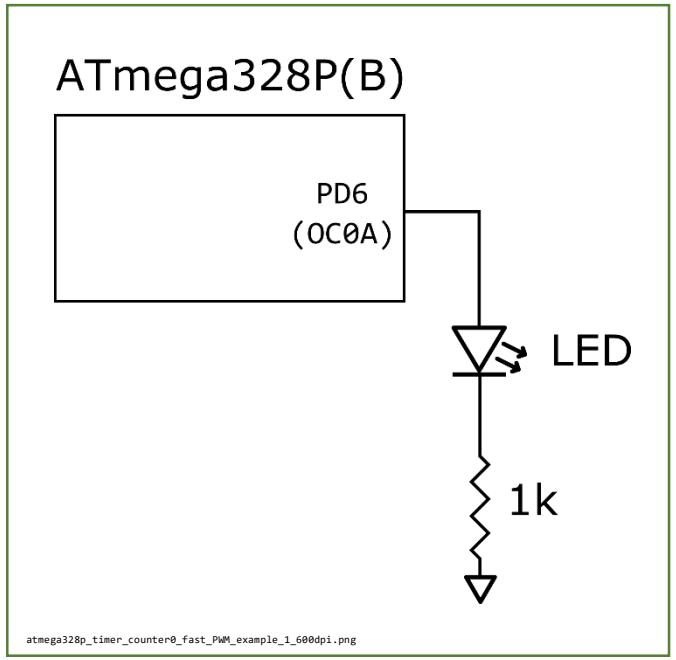
Fast PWM Example 1

(Single Channel / Fixed Duty Cycle)

Timer/Counter0 Fast PWM Example 1 (1)

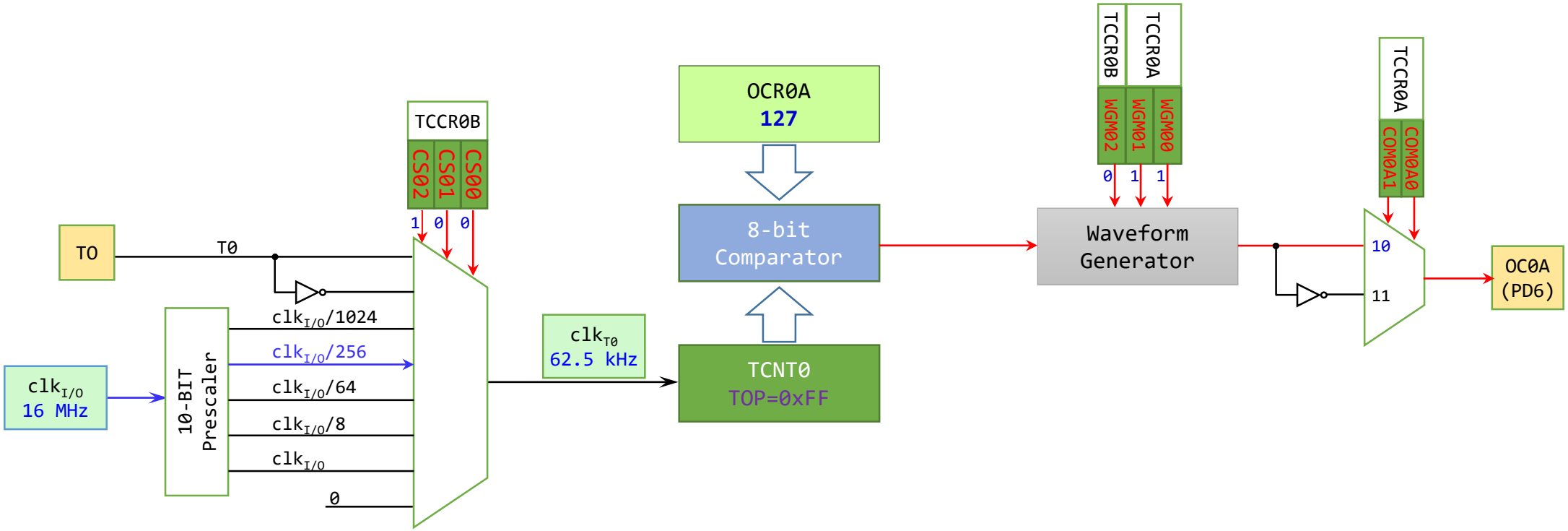
Make an application which generates single PWM signal.

- PWM signal drives an LED connected to OC0A (PD6).
- Duty cycle: 50% (fixed).
- PWM frequency: 244.14Hz (fixed)



Timer/Counter0 Fast PWM Example 1 (2)

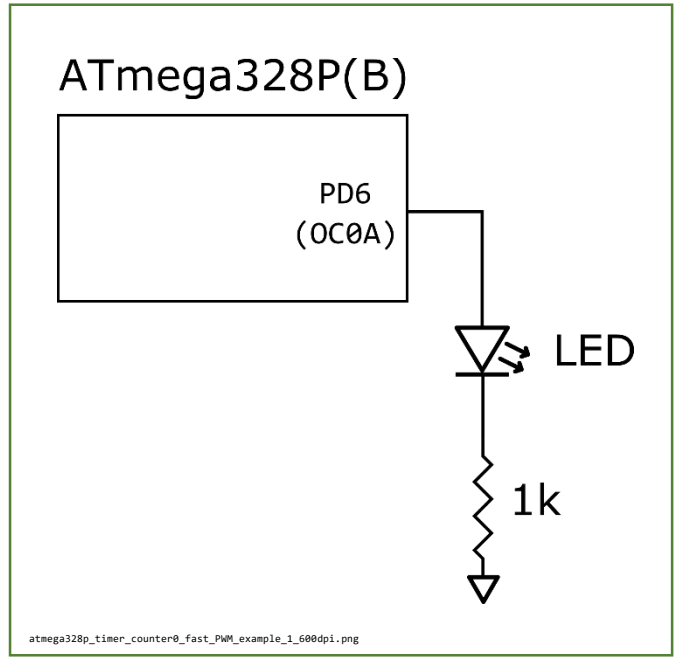
TCNT0 TOP = 0xFF
PWM frequency = 62.5kHz/256 = 244.14Hz
PWM duty cycle = 50% ⇨ OCR0A ← 127 ($256 \times 0.5 = 128$)



Timer/Counter0 Fast PWM Example 1 (3)

Make an application which generates fixed duty cycle PWM signal.

```c
/* timer0_fastPWM_fixed_duty.c */

#include <avr/io.h>

int main(void)
{
    DDRD   = 1 << DDD6;      // Set OC0A (PD6) to OUTPUT
    TCCR0A = 0b10000011;     // Fast PWM (TOP=0xFF) mode.
                             //   OC0A:non-inverting PWM signal.
                             //   OC0B:disconnected
    TCCR0B = 0b00000100;     // Prescaler: div by 256.
                             //   PWM freq: (16,000,000Hz/256/256) = 244.14Hz
    OCR0A  = 127;            // Duty cycle: 50% (fixed)

    while (1)
    { /* do nothing */ }
}
```



Timer/Counter0

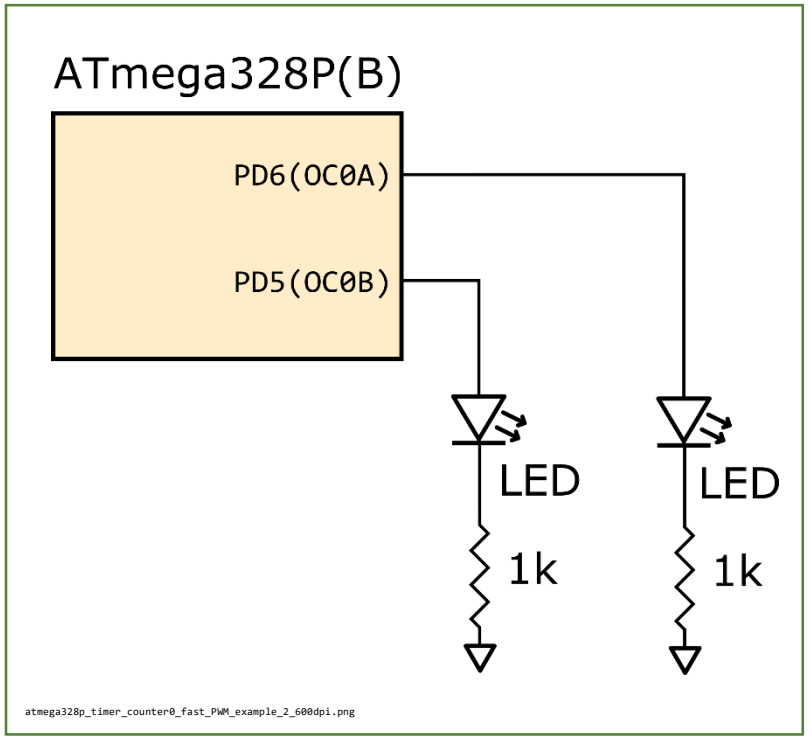
Fast PWM Example 2

(Dual Channel / Fixed Duty Cycle)

Timer/Counter0 Fast PWM Example 2 (1)

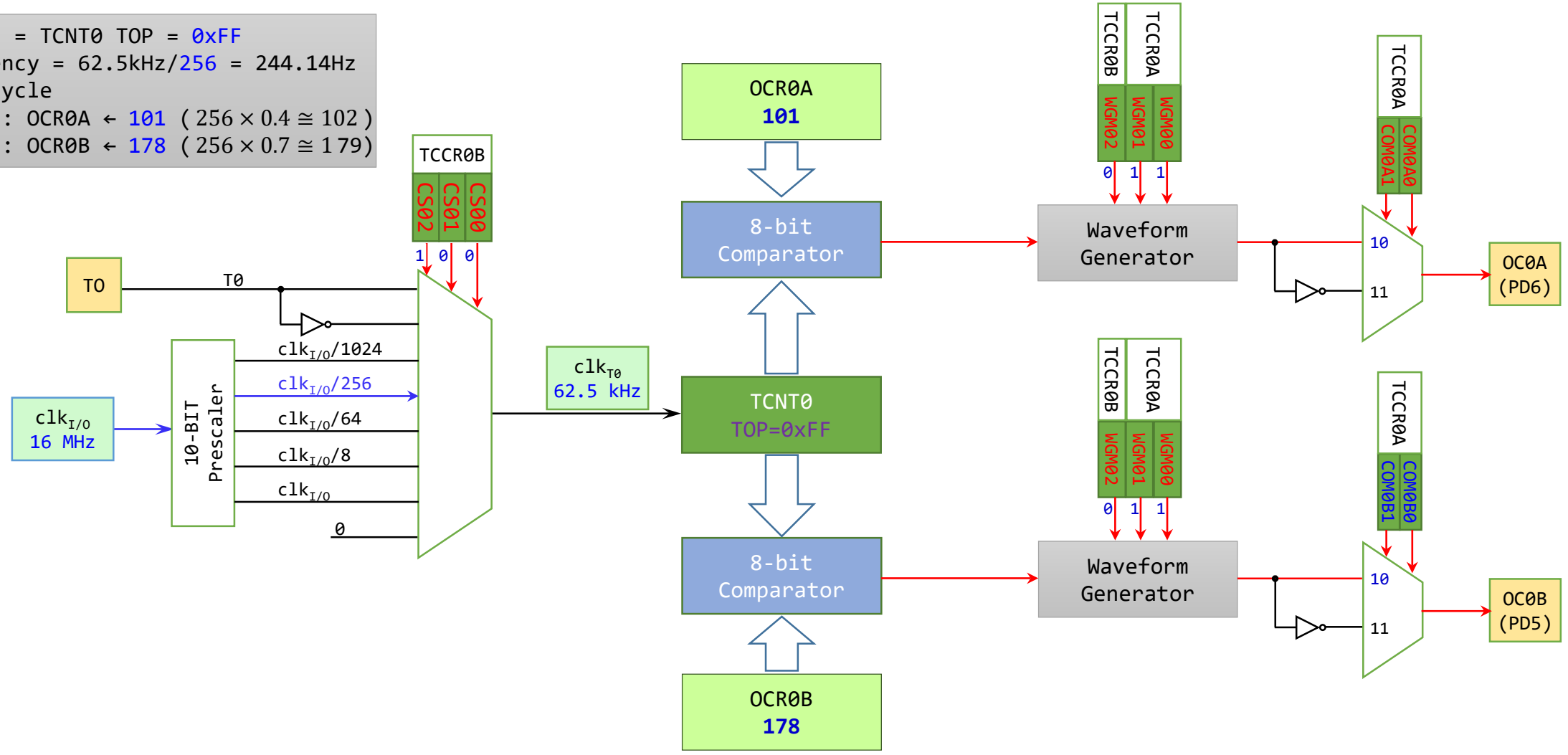
Make an application which generates dual PWM signals.

- PWM signals are available at OC0A (PD6) and OC0B (PD5).
- Duty cycle: 40% (OC0A) and 70% (OC0B).
- PWM frequency: 244.14Hz (fixed)



Timer/Counter0 Fast PWM Example 2 (2)

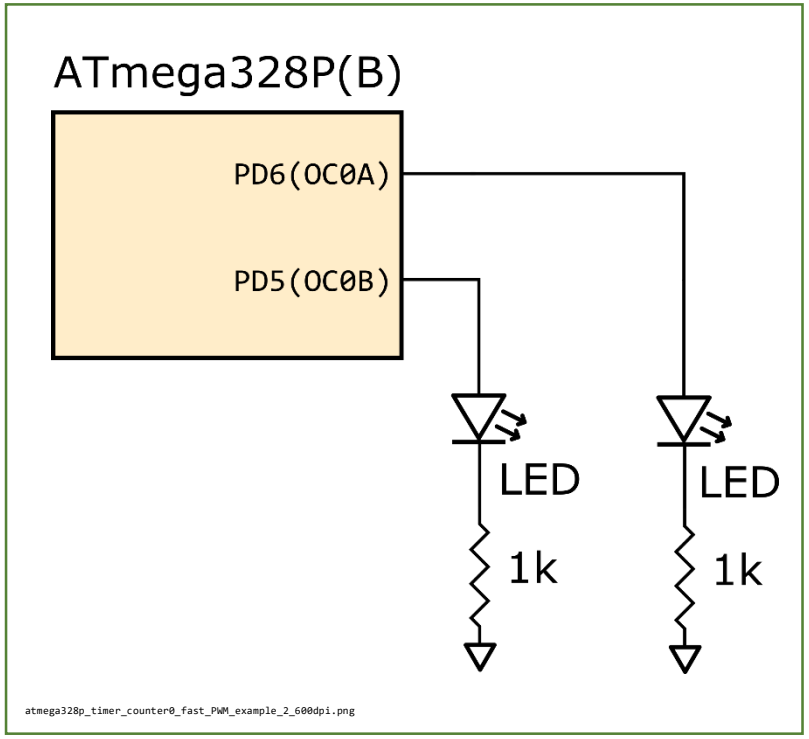
PWM Period = TCNT0 TOP = 0xFF
PWM frequency = 62.5kHz/256 = 244.14Hz
PWM duty cycle
 OC0A(40%): OCR0A ← 101 ($256 \times 0.4 \cong 102$)
 OC0B(70%): OCR0B ← 178 ($256 \times 0.7 \cong 179$)



Timer/Counter0 Fast PWM Example 2 (3)

Make an application which generates dual fixed duty cycle PWM signals.

```c
/* timer0_fastPWM_fixed_duty_dual_outputs.c */
/* PWM signal drives two LEDs connected to OC0A (PD6) and OC0B (PD5). */
/* Duty cycle: 40% at OC0A and 70% at OC0B. */

#include <avr/io.h>

int main(void)
{
    DDRD   = (1 << DDD6) | (1 << DDD5);    // Set OC0A (PD6) and OC0B (PD5) to OUTPUT
    TCCR0A = 0b10100011;                   // Fast PWM (TOP=0xFF) mode.
                                           // OC0A, OC0B:non-inverting PWM signal.
    TCCR0B = 0b00000100;                   // Prescaler: div by 256.
                                           // PWM freq: (16,000,000Hz/256/256)=244.14Hz
    OCR0A  = 101;                              // OC0A duty cycle: 40% (256*0.4=102)
    OCR0B  = 178;                              // OC0B duty cycle: 70% (256*0.7=179)

    while (1)
    { /* do nothing */ }
}
```



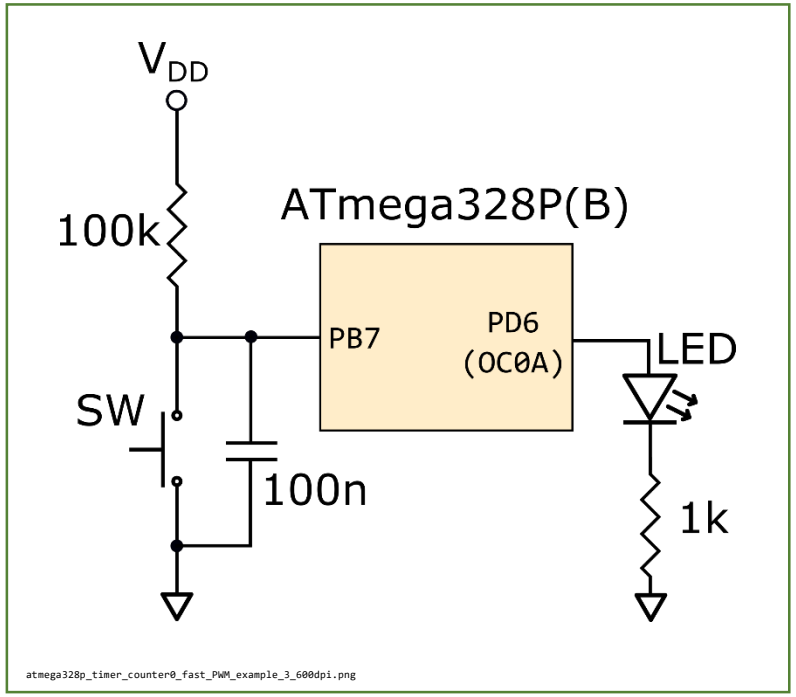
Timer/Counter0

Fast PWM Example 3

(Single Channel / Variable Duty Cycle)

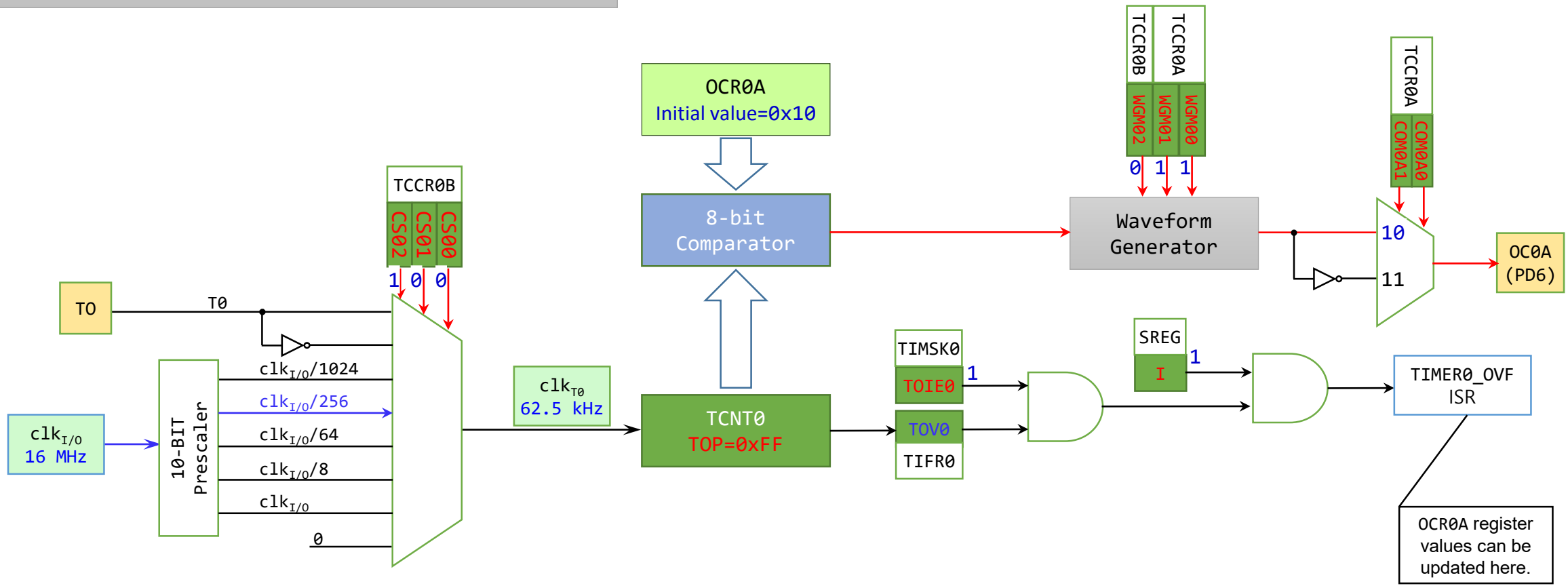
Timer/Counter0 Fast PWM Example 3 (1)

Make an application which controls LED brightness by SWITCH press, i.e. LED brightness is getting higher whenever the SWITCH is pressed.



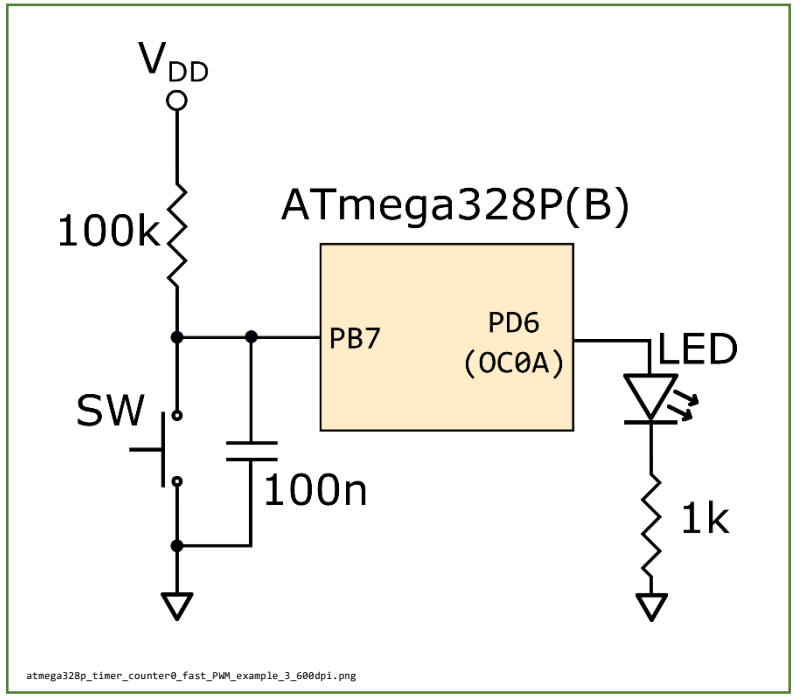
Timer/Counter0 Fast PWM Example 3 (2)

TCNT0 TOP = 0xFF
PWM frequency =62.5kHz/256=244.14Hz



Timer/Counter0 Fast PWM Example 3 (3)

```c
/* timer0_fastPWM_variable_duty_single_output.c */
 * PWM signal at OC0A(PD6) changes whenever SWITCH at PB7 is pressed.
 * PWM frequency: 244.14Hz (fixed) */

#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char duty_cycle = 0x10;

int main(void)
{
    DDRB  &= ~(1 << DDB7);    // Set PB7 to INPUT mode for switch input
    DDRD   = 1 << DDD6;       // Set OC0A (PD6) to OUTPUT mode
    TCCR0A = 0b10000011;      // Fast PWM (TOP=0xFF) mode. OC0A:non-inverting PWM signal.
    TCCR0B = 0b00000100;      // Prescaler: div by 256. PWM freq: (16,000,000Hz/256/256) = 244.14Hz
    OCR0A  = duty_cycle;      // Initial duty cycle: 16/256 = 6.25%

    TIMSK0 |= (1 << TOIE0);   // Enable Timer/Counter0 overflow interrupt.
    sei();                    // Enable global interrupt

    while (1)
    {
        if ((PINB & (1 << PINB7)) == 0)             // switch pressed
        {
            duty_cycle += 0x10;                     // update duty cycle
            while ((PINB & (1 << PINB7)) == 0);     // wait for switch release
        }
    }
}

ISR(TIMER0_OVF_vect)
{
    OCR0A = duty_cycle;               // Apply new duty value
}
```



Timer/Counter0 Phase Correct PWM Mode (1)

- WGM0[2:0]=0b001 or WGM0[2:0]=0b101
- The counter(TCNT0) is incremented until the counter value matches TOP.
- When the counter reaches TOP, it changes the count direction.
- The TCNT0 value will be equal to TOP for one timer clock cycle.
- The counter is decremented until the counter value matches BOTTOM.
  - ✓ BOTTOM is 0
  - ✓ TOP is
    - 0xFF when WGM[02:00]=0x1.
    - OCR0A when WGM[02:00]=0x5.
- Two types of PWM signals can be available
  - ✓ Non-invert: COM0A[1:0]=0x2
    - OC0A bit is cleared on compare match while up-counting.
    - OC0A bit is set on the compare match while down-counting.
  - ✓ Invert: COM0A[1:0]=0x3
    - Operation is inverted.

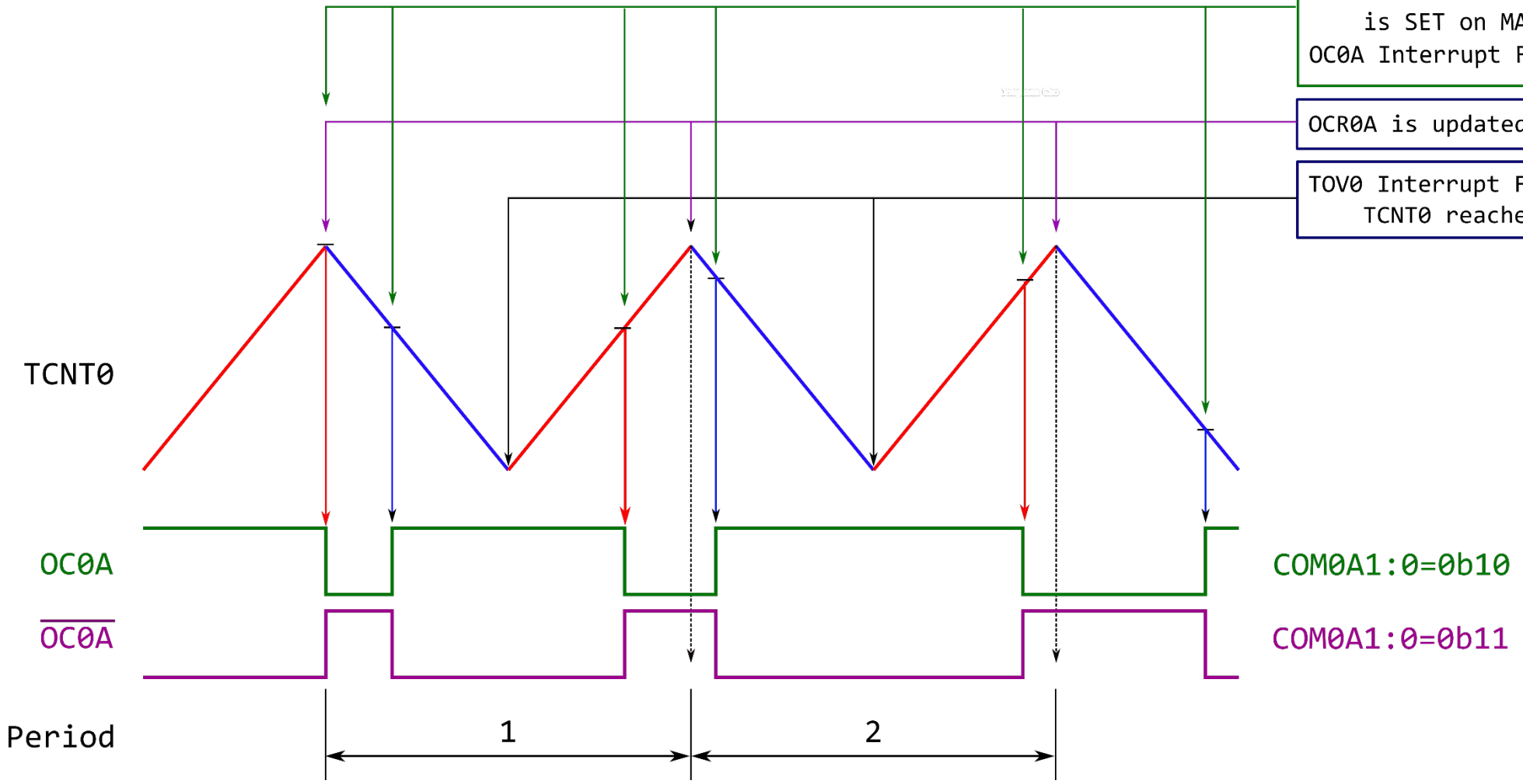
Timer/Counter0 Phase Correct PWM Mode (2)

TCNT0 TOP == 0xFF

IN Non-Inverting mode, OC0A
    is CLEARED on MATCH while UP-COUNTING
    is SET on MATCH while DOWN-COUNTING.
OC0A Interrupt Flag is SET on MATCH.

OCR0A is updated when TCNT0 reaches TOP.

TOV0 Interrupt Flag is SET when
    TCNT0 reaches BOTTOM.

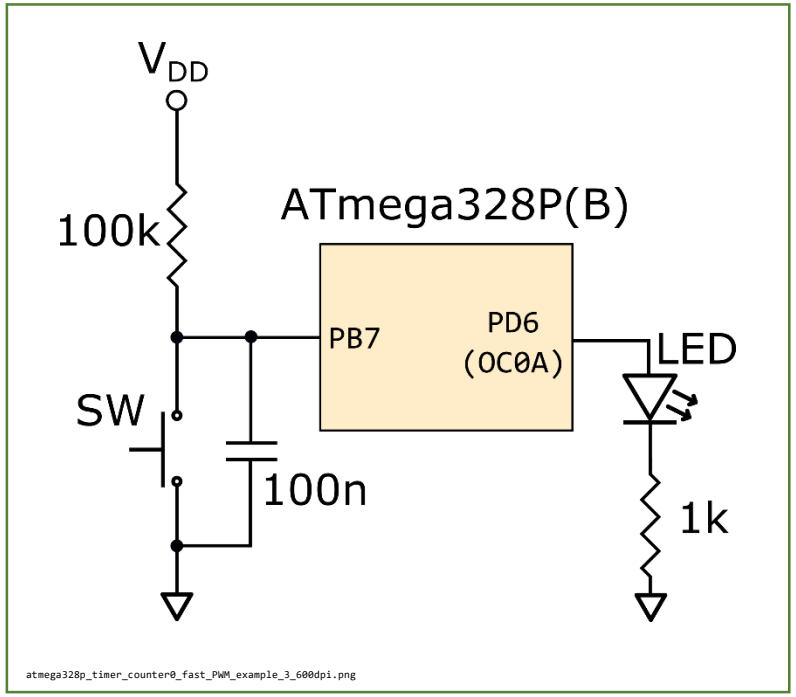


Timer/Counter0
Phase Correct PWM Example 1

(Single Channel / Variable Duty Cycle)

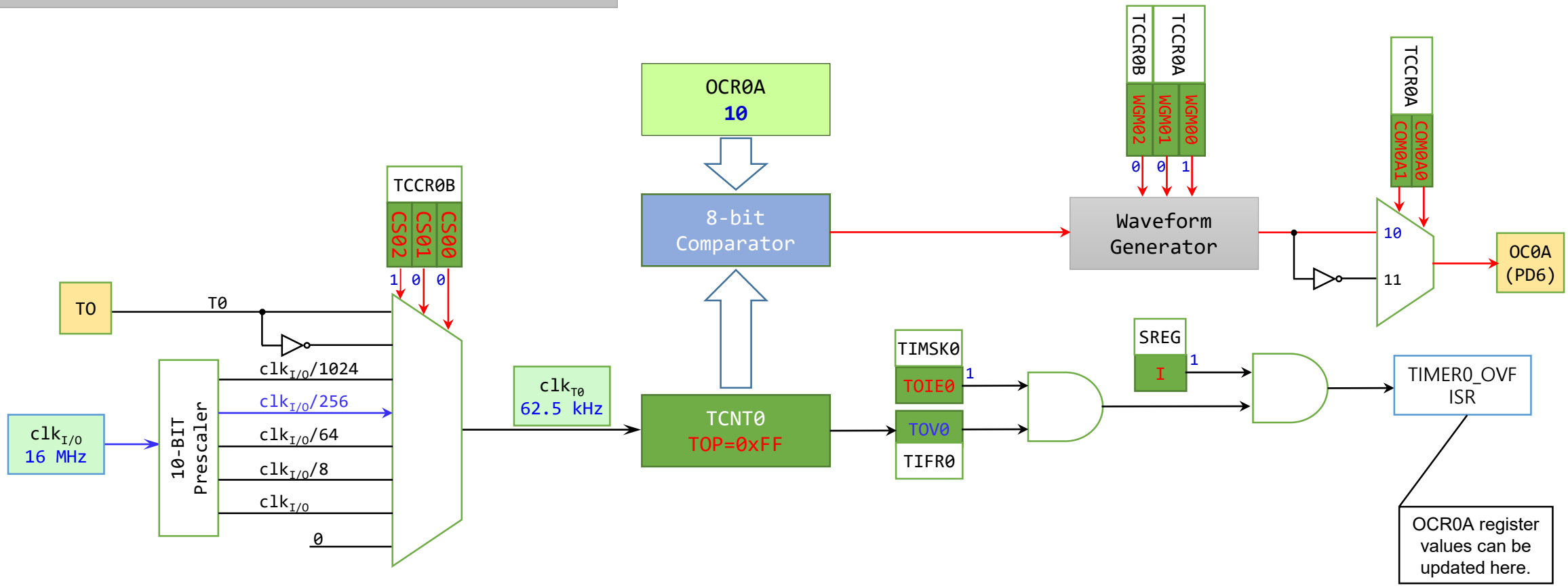
Timer/Counter0 Phase Correct PWM Example (1)

Make an application which controls LED brightness connected to PD6(OC0A) by SWITCH press, i.e. LED brightness is getting higher whenever the SWITCH is pressed.



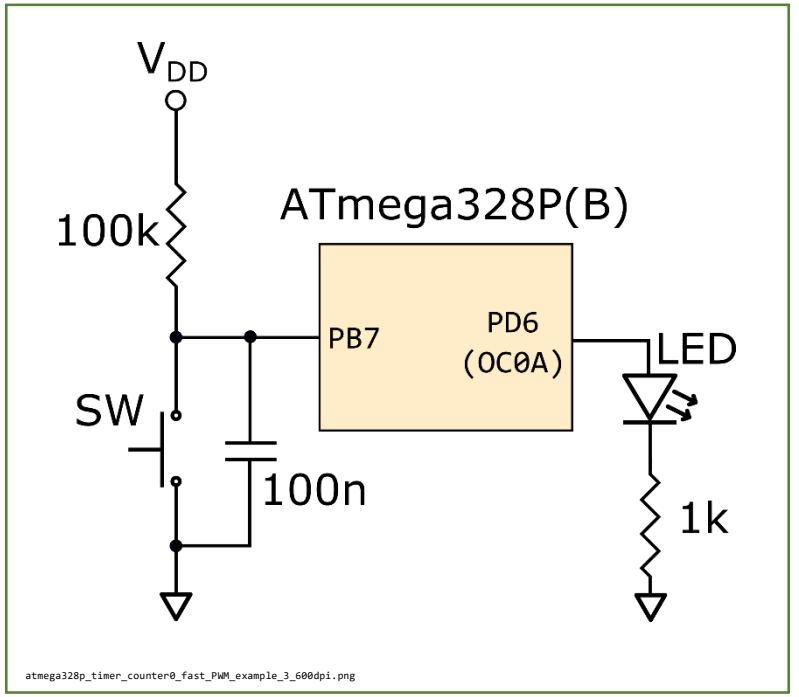
Timer/Counter0 Phase Correct PWM Example (2)

TCNT0 TOP = 0xFF
PWM frequency=62.5kHz/510=122.55Hz



Timer/Counter0 Phase Correct PWM Example (3)

```c
/* timer0_phase_correct_PWM_variable_duty_single_output.c */
 * PWM signal at OC0A(PD6) changes whenever SWITCH at PB7 is pressed.
 * PWM frequency: 62.5kHz/510=122.55Hz (fixed) */

#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char duty_cycle = 0x10;

int main(void)
{
    DDRB  &= ~(1 << DDB7);     // Set PB7 to INPUT mode for switch input
    DDRD   = 1 << DDD6;        // Set OC0A (PD6) to OUTPUT
    TCCR0A = 0b1000 0001;      // Phase Correct PWM (TOP=0xFF) mode. OC0A:non-inverting PWM signal.
    TCCR0B = 0b00000100;       // Prescaler: div by 256. PWM freq: (16,000,000Hz/256/510) = 122.55Hz
    OCR0A  = 0x10;             // Initial duty cycle: 10/256 = 4%

    TIMSK0 |= (1 << TOIE0);    // Enable Timer/Counter0 overflow interrupt.
    sei();                     // Enable global interrupt

    while (1)
    {
        if ((PINB & (1 << PINB7)) == 0)             // switch pressed
        {
            duty_cycle += 0x10;                     // update duty cycle
            while ((PINB & (1 << PINB7)) == 0);     // wait for switch release
        }
    }
}

ISR(TIMER0_OVF_vect)
{
    OCR0A = duty_cycle;                // Apply new duty value
}
```



Timer/Counter0
END

16-bit Timer/Counter TC1, TC3, TC4

Timer/Counter1,3,4 Block Diagram

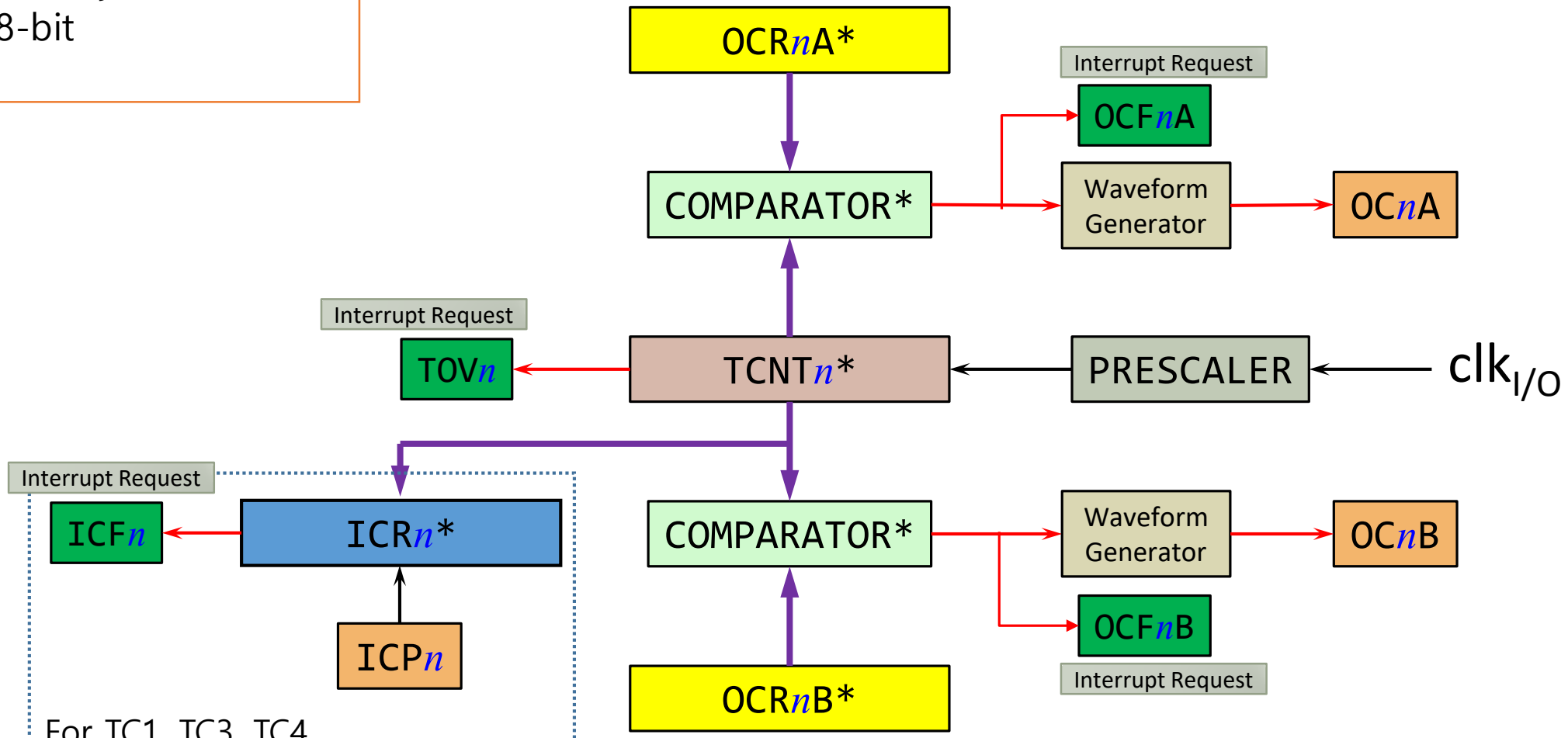
TCNT*n*, OCR*n*A, OCR*n*B, ICR*n*: 16-bit
Other Registers: 8-bit
n: 1, 3 or 4

TCCR*n*A

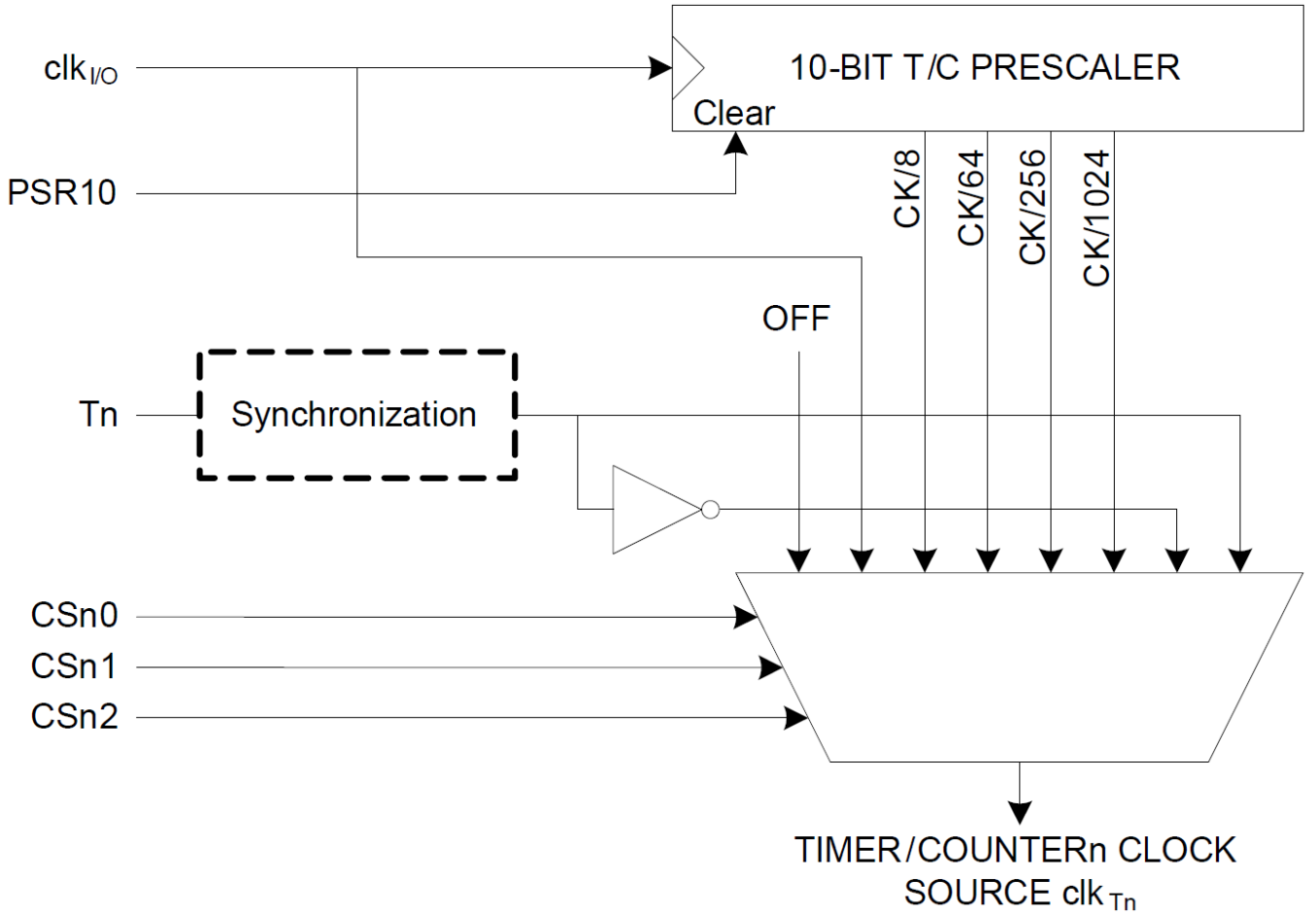
TCCR*n*B

TIFR*n*

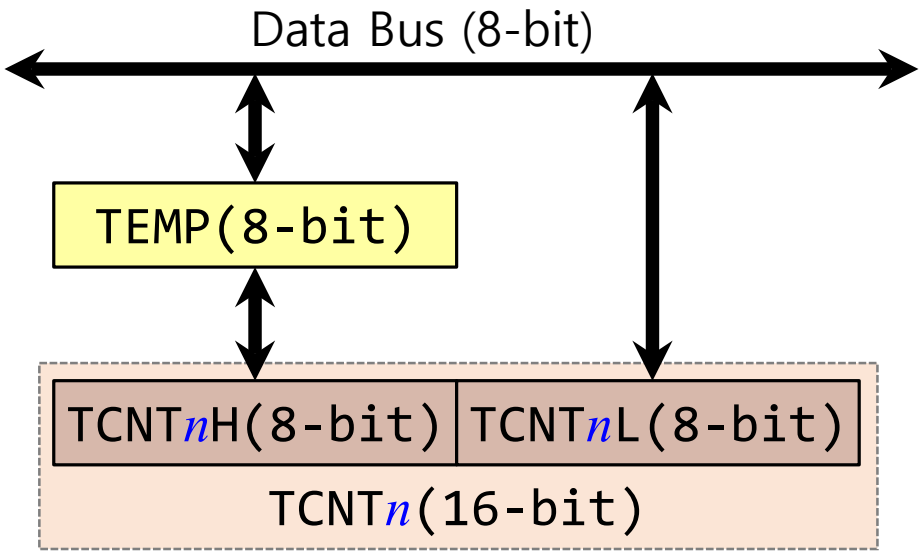
TIMSK*n*



Prescaler for Timer/Counter0,1,3,4



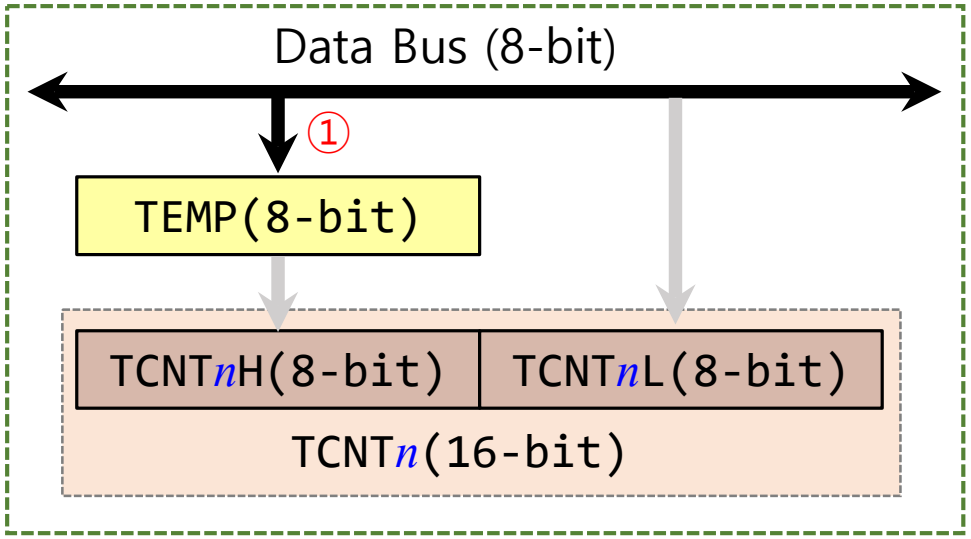
Accessing 16-bit Registers for T/C0,1,3,4(1)



- TCNT*n*, OCR*n*A, OCR*n*B, ICR*n*: 16-bit registers.

  (*n*: 1, 3 or 4)
- These 16-bit register must be accessed byte-wise, using two read or write operations.
- Each 16-bit timer has a single 8-bit TEMP register for temporary storing of the high byte of the 16-bit access.
- The same temporary register is shared between all 16-bit registers within each 16-bit timer.

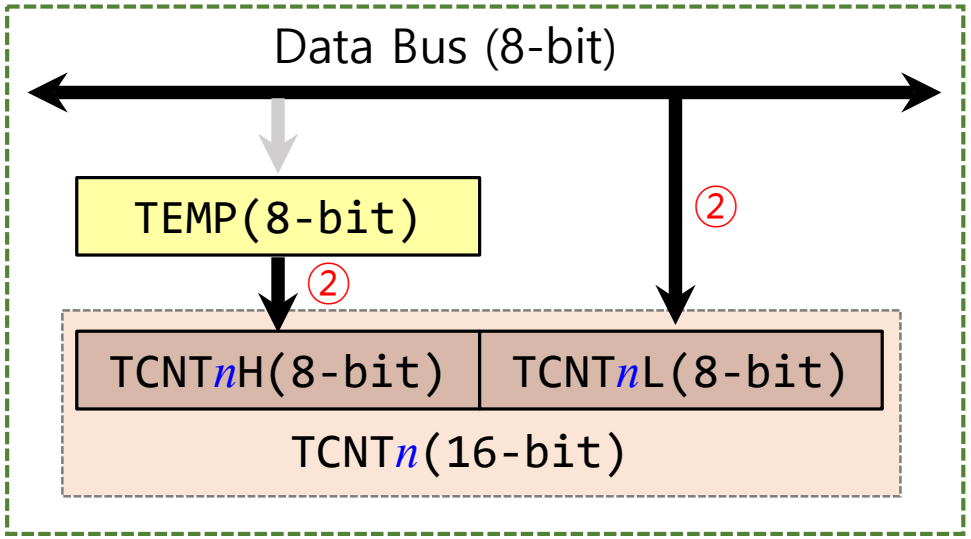
Accessing 16-bit Registers for T/C0,1,3,4(2)



Writing the 16-bit TCNT*n* Register (1)

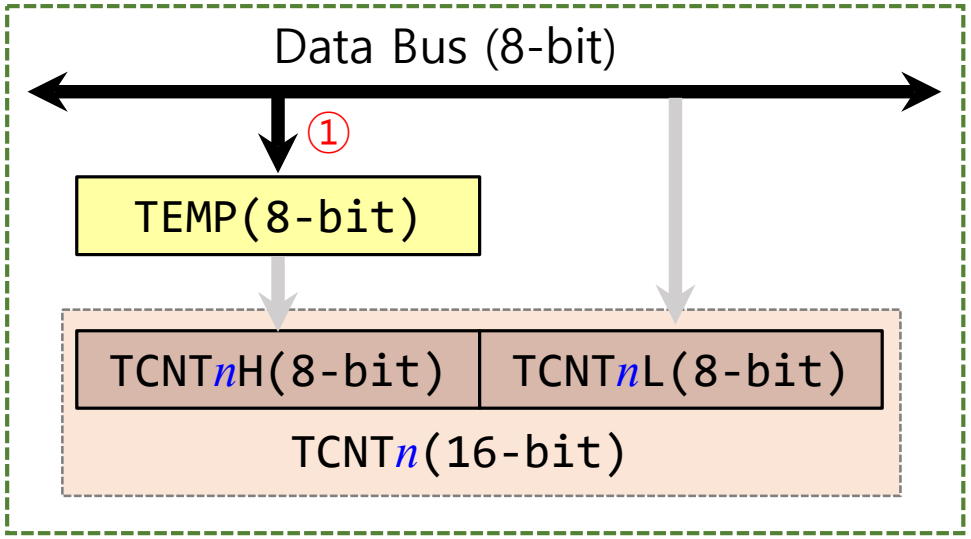
1. Write the high byte of the 16-bit value to the TCNT*n*H register.

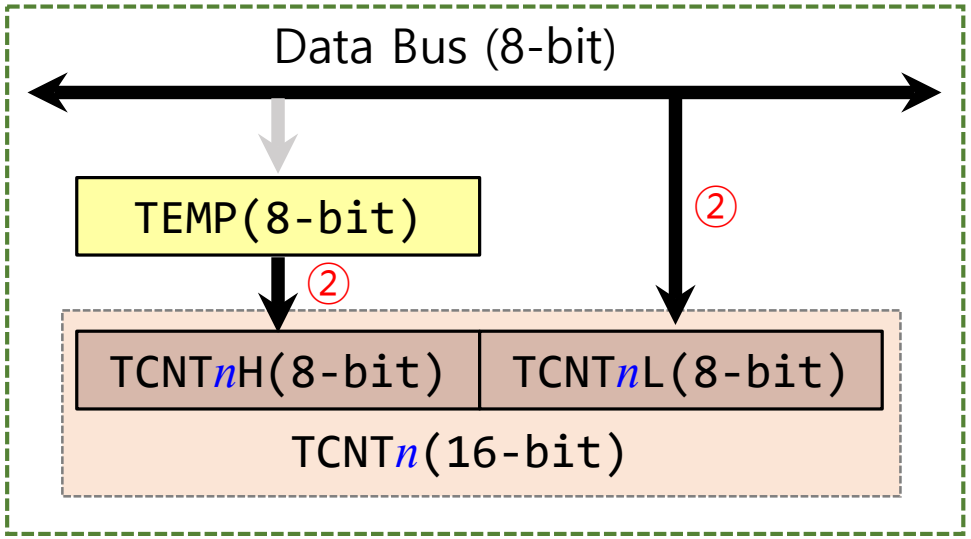
   This value is written to the TEMP register.



2. Write the low byte of the 16-bit value to the TCNT*n*L register.
   - This value is written to the TCNT*n*L register.
   - The value in the TEMP register is written to TCNT*n*H register in the same clock cycle.

Accessing 16-bit Registers for T/C0,1,3,4(3)



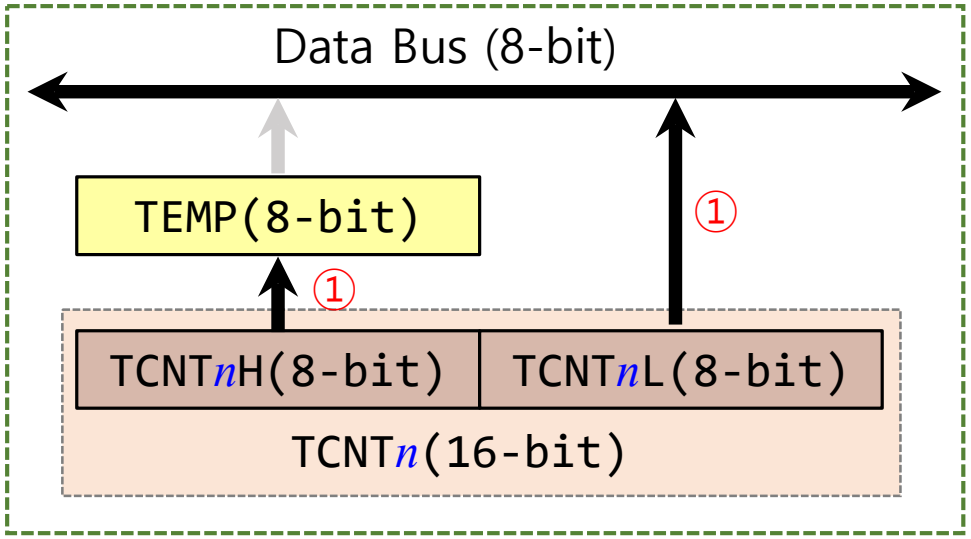


Writing the 16-bit TCNT*n* Register (2)

```
    IN  R24,SREG        ; save global interrupt flag
    CLI                 ; disable global interrupt

    LDI R18,0x34        ; load lower byte
    LDI R19,0x12        ; load higher byte
    STS TCNT1H,R19      ; write higher byte first
    STS TCNT1L,R18      ; write lower byte later

    OUT SREG,R24        ; restore global interrupt flag
```

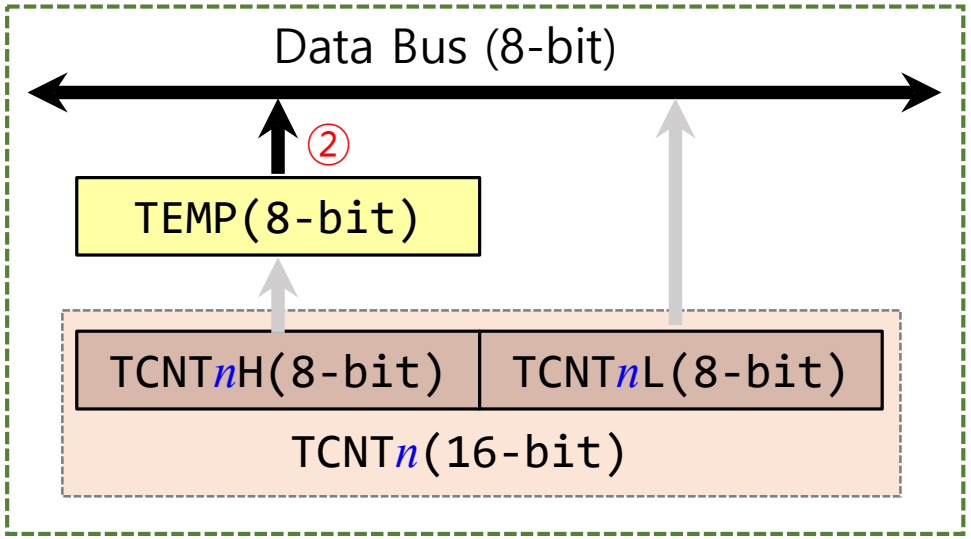
```
unsigned char sreg;

sreg = SREG;            /* Save interrupt flag */
cli();                  /* Disable interrupts */
TCNT1 = 0x1234;         /* Write value to TCNT1
SREG = sreg;            /* Restore interrupt flag */
```

Accessing 16-bit Registers for T/C0,1,3,4(4)



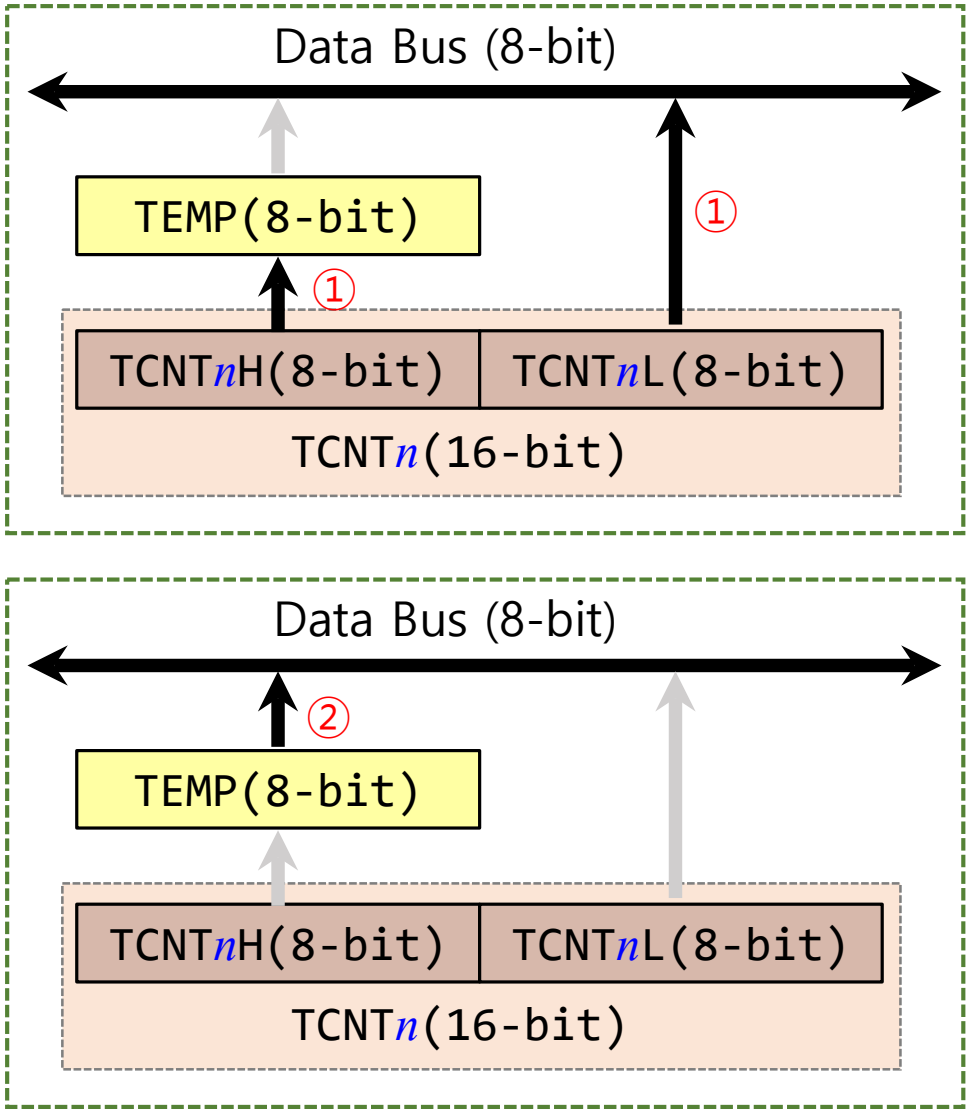
Reading the 16-bit TCNT*n* Register (1)

1. Read the low byte register ,TCNT*n*L, first.
   - The value of the TCNT*n*H register is copied to the TEMP register in the same clock cycle.



2. Read the high byte register, TCNT*n*H.
   - This operation reads the value in the TEMP register.

Accessing 16-bit Registers for T/C0,1,3,4(5)



Reading the 16-bit TCNT*n* Register (2)

```
    IN  R24,SREG        ; save global interrupt flag
    CLI                 ; disable global interrupt

    LDS R18,TCNT1L      ; read lower byte first
    LDS R19,TCNT1H      ; read higher byte later

    OUT SREG,R24        ; restore global interrupt flag
```

```
unsigned char sreg;
unsigned int timer_value;

sreg = SREG;            /* Save interrupt flag */
cli();                  /* Disable interrupts */
timer_value = TCNT1;    /* Read TCNT1
SREG = sreg;            /* Restore interrupt flag */
```

Timer/Counter1,3,4

CTC mode

(Clear Timer on Compare Match mode)

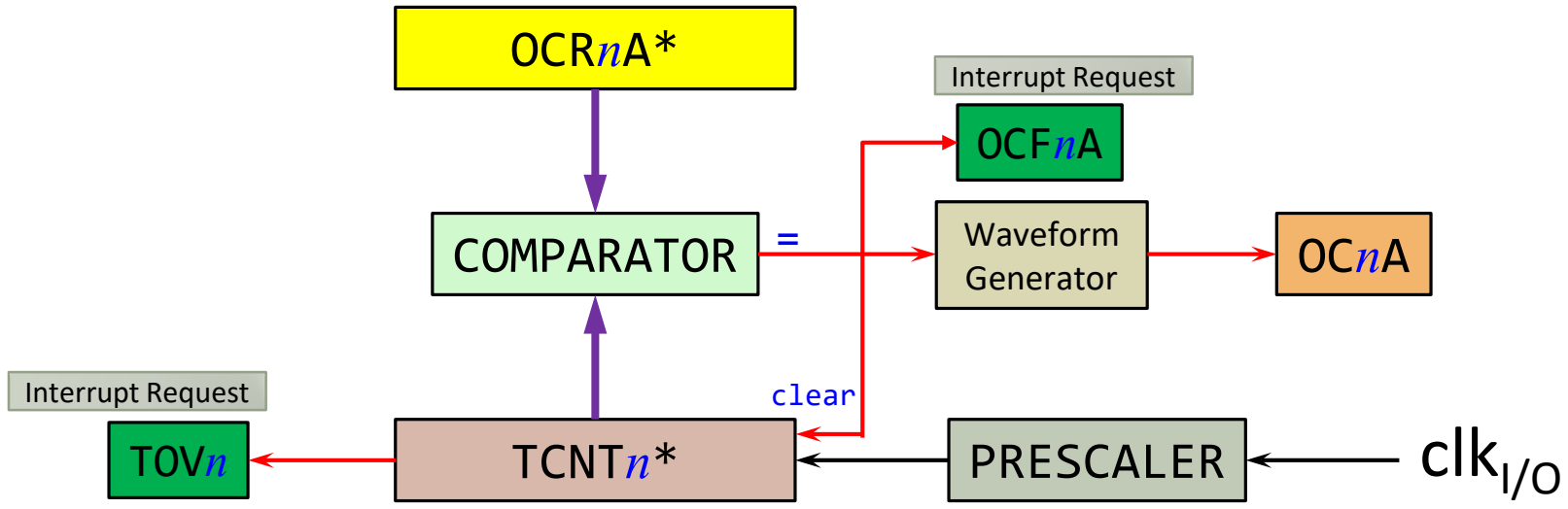
CTC Mode: Clear Timer on Compare Match(1) (TC1, TC3, TC4)

TCNT*n*, OCR*n*A, OCR*n*B, ICR*n*: 16-bit
Other Registers: 8-bit
n: 1, 3 or 4

❖ When WGM[3:0] = 0b0100,

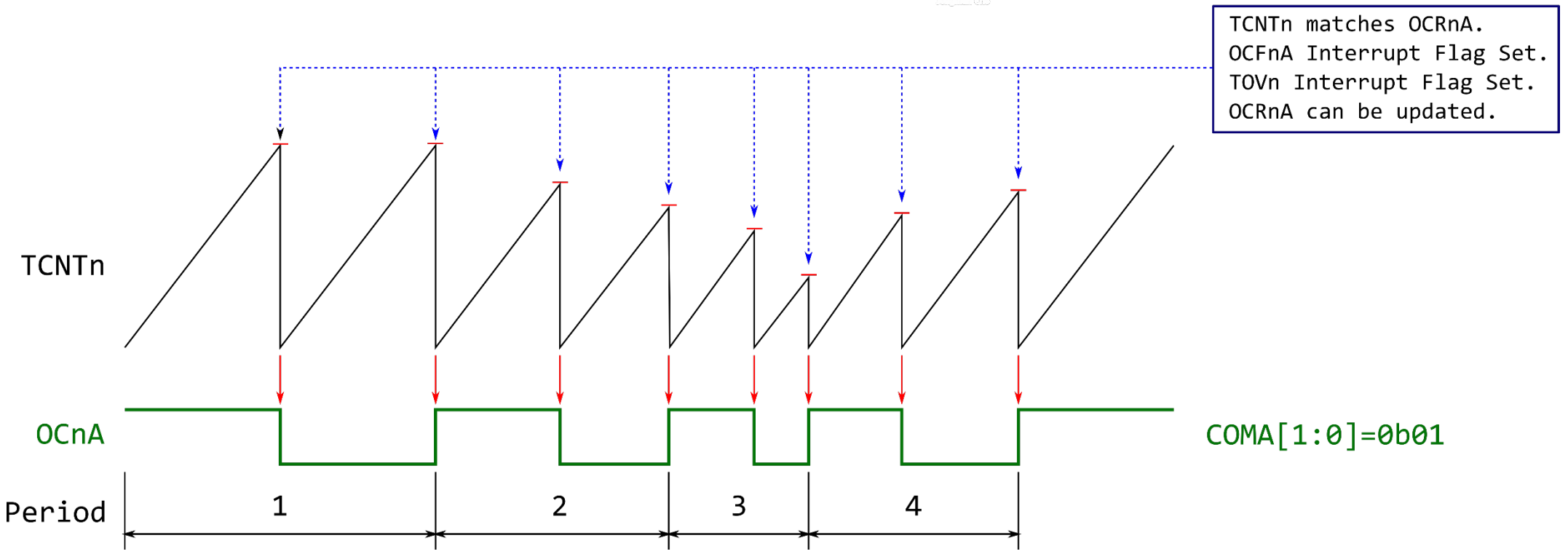
- OCR*n*A is used as TOP value for TCNT*n*.
- TCNT*n* repeats counting from 0 to the value of OCR*n*A register.
- When TCNT*n* value matches OCR*n*A value,
  - ✓ OC*n*A output can be set to toggle. (waveform generation)
  - ✓ OCF*n*A flag is set and an interrupt can be generated.

TIM*n*_COMPA_vect



CTC Mode: Clear Timer on Compare Match(2) (TC1, TC3, TC4)

- When WGM[3:0] = 0b0100,
- OCR*n*A is used as TOP value for TCNT*n*.
- TCNT*n* repeats counting from 0 to the value of OCRnA register.
- When TCNT*n* value matches OCRnA value,
  - ✓ OCnA output can be set to toggle. (waveform generation)
  - ✓ OCFnA flag is set and an interrupt can be generated.



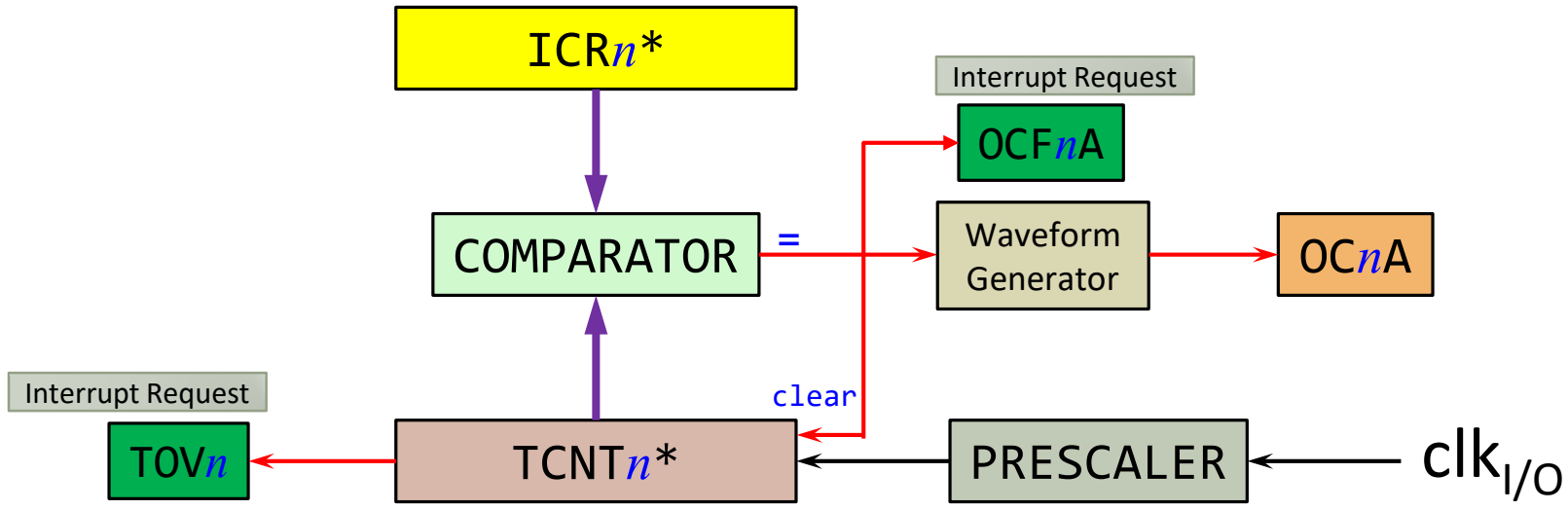
CTC Mode: Clear Timer on Compare Match(3) (TC1, TC3, TC4)

TCNT*n*, OCR*n*A, OCR*n*B, ICR*n*: 16-bit
Other Registers: 8-bit
n: 1, 3 or 4

❖ When WGM[3:0] = 0b1100,

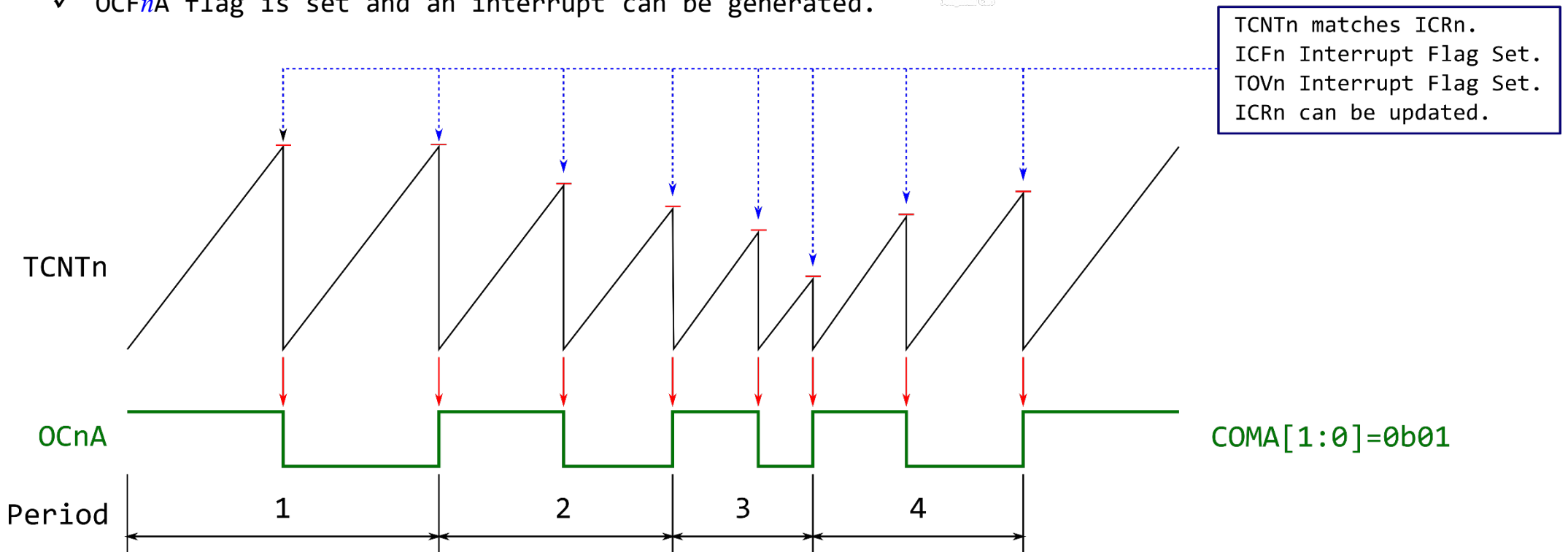
- ICR*n* is used as TOP value for TCNT*n*.
- TCNT*n* repeats counting from 0 to the value of ICR*n* register.
- When TCNT*n* value matches ICR*n* value,
  - ✓ OC*n*A output can be set to toggle. (waveform generation)
  - ✓ OCF*n*A flag is set and an interrupt can be generated.

TIM*n*_COMPA_vect



CTC Mode: Clear Timer on Compare Match(4) (TC1, TC3, TC4)

- When WGM[3:0] = 0b1100,
  - ICR*n* is used as TOP value for TCNT*n*.
  - TCNT*n* repeats counting from 0 to the value of ICR*n* register.
  - When TCNT*n* value matches ICR*n* value,
    - ✓ OC*n*A output can be set to toggle. (waveform generation)
    - ✓ OCF*n*A flag is set and an interrupt can be generated.

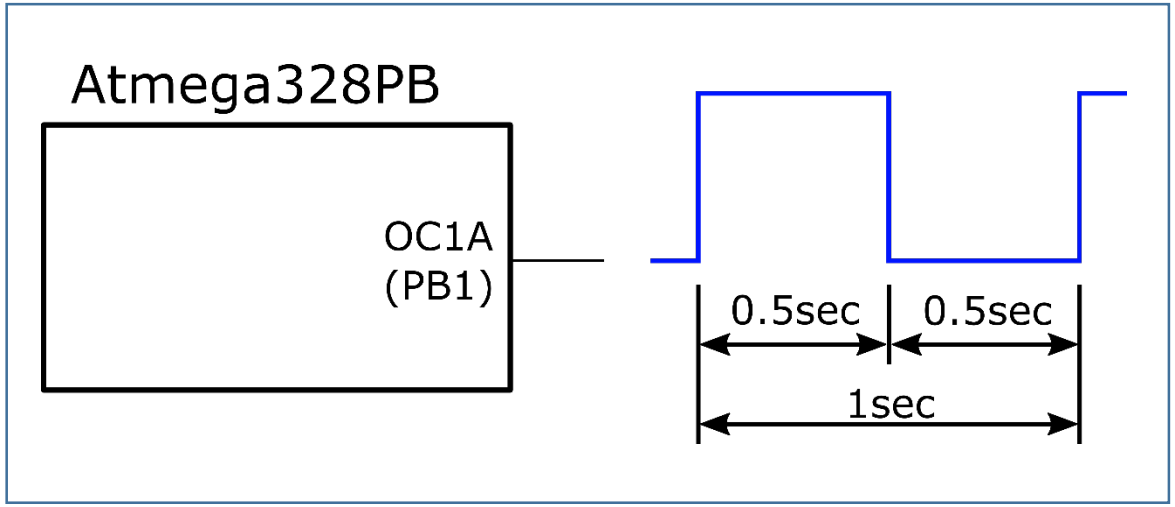


Timer/Counter1
CTC Example
(OCR1A as TOP)

Timer/Counter1 CTC Example (OCR1A as TOP) (1)

Waveform Generation

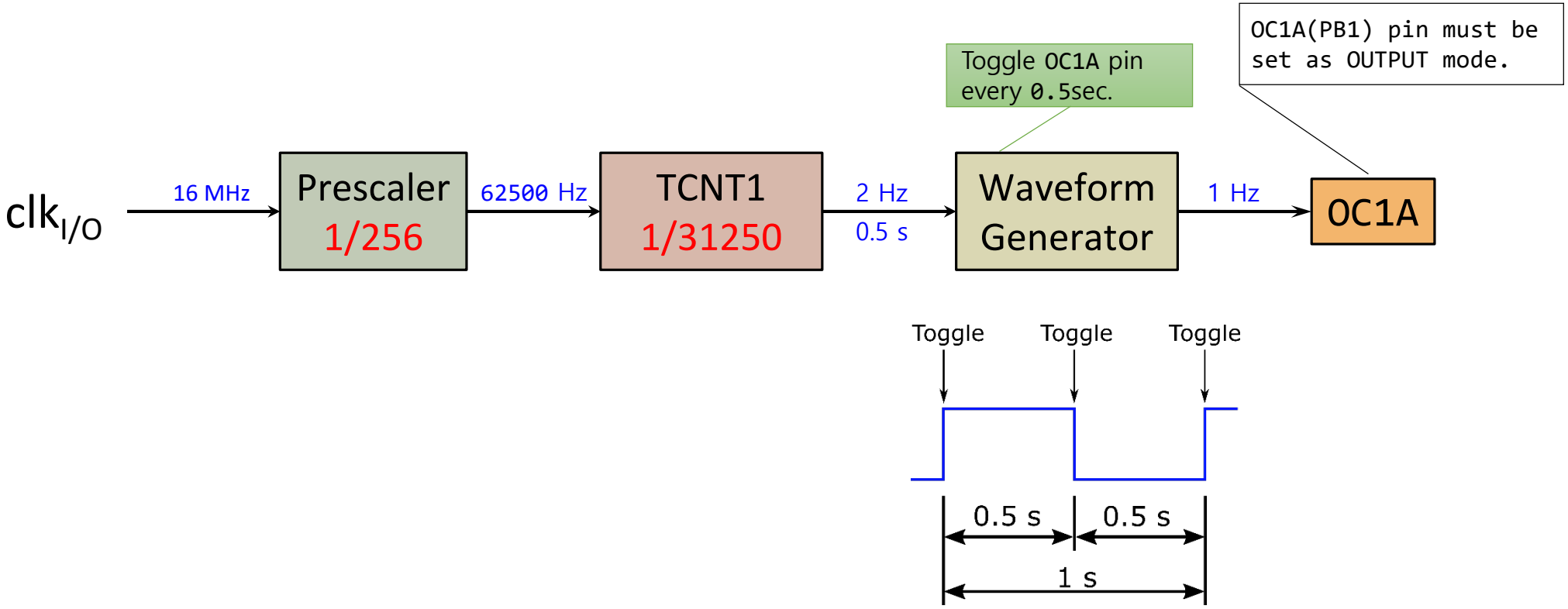
- Make an application which generates 1 Hz rectangular wave on OC1A (PB1).
- Use Timer/Counter1 CTC mode.



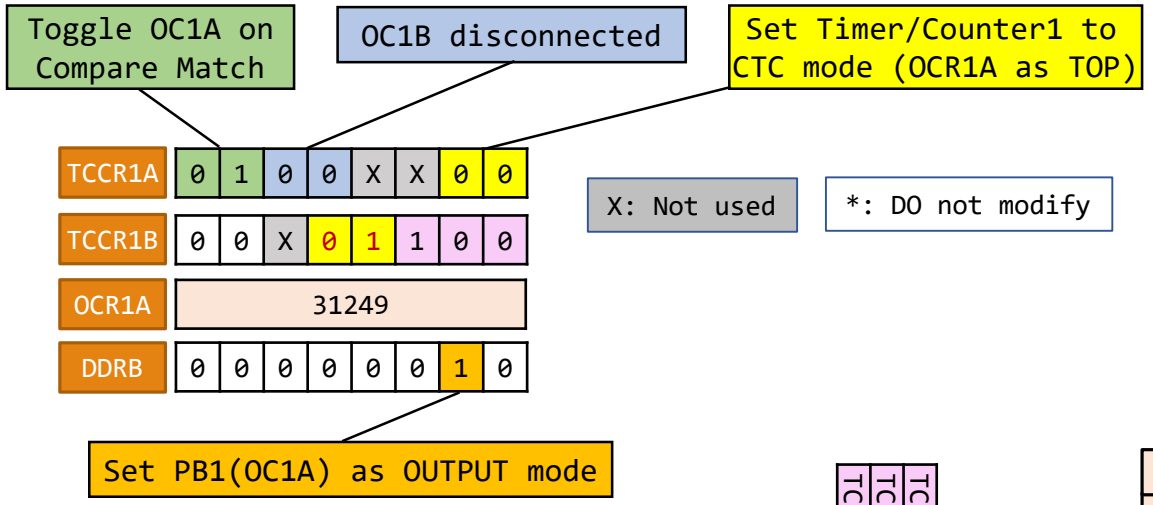
Timer/Counter1 CTC Example (OCR1A as TOP) (2)

Waveform Generation

- Make an application which generates 1 Hz rectangular wave on OC1A (PB1).
- Use Timer/Counter1 CTC mode.

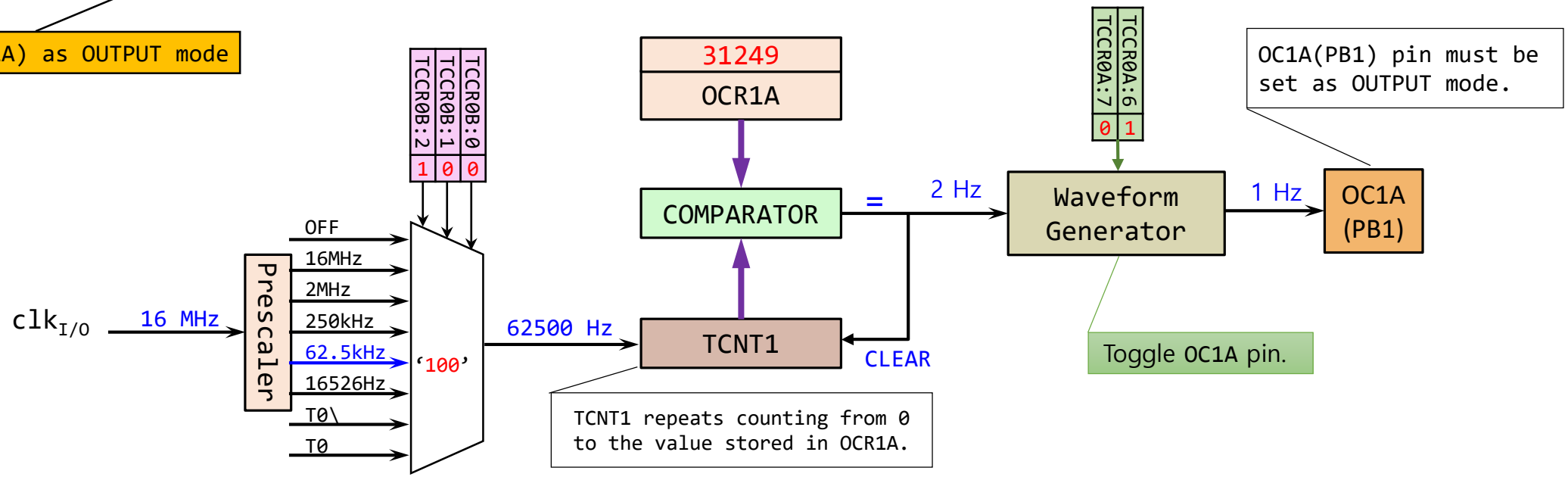


Timer/Counter1 CTC Example (OCR1A as TOP) (3)



Waveform Generation

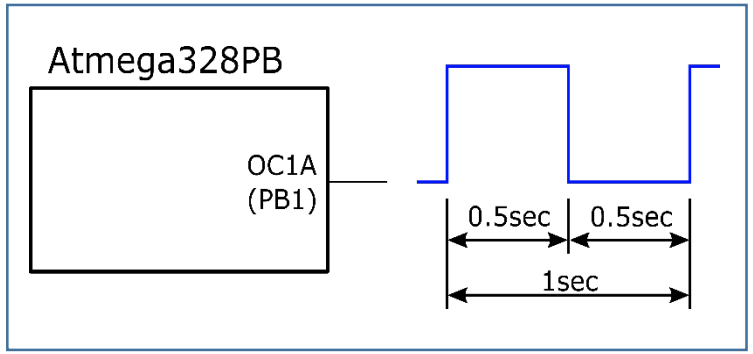
- Make an application which generates 1 Hz rectangular wave on OC1A (PB1).
- Use Timer/Counter1 CTC mode.



Timer/Counter1 CTC Example (OCR1A as TOP) (4)

Make an application which generates 1 Hz rectangular wave on OC1A (PB1).

```
/* tc1_ctc_OCR1A_1hz.c */

#include <avr/io.h>

int main(void)
{
    DDRB   = (1 << DDB1);  // Set PB1 (OC1A) as OUTPUT mode
    TCCR1A = 0b01000000;   // Set T/C1 to CTC mode(OCR1A as TOP).
                           //   Toggle OC1A.
    TCCR1B = 0b00001100;   // Select Prescaler division ratio.
                           //   16MHz/256=62,500Hz
    OCR1A  = 31249;        // Match Rate is 2Hz. 62500Hz/31250=2Hz

    while (1)
    { /* do nothing */ }
}
```

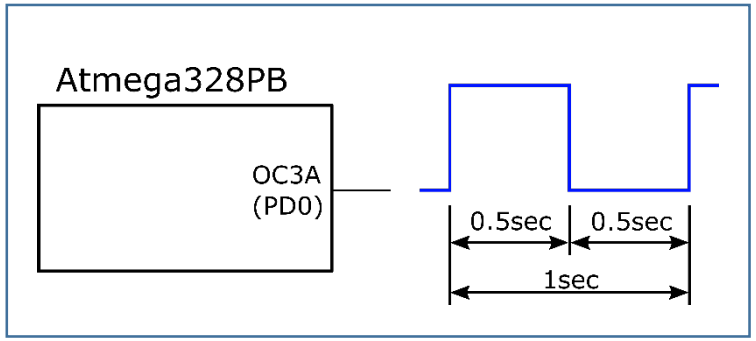


Timer/Counter3
CTC Example
(OCR3A as TOP)

Timer/Counter3 CTC Example (OCR3A as TOP) (4)

Make an application which generates 1 Hz rectangular wave on OC3A (PD0).

```
/* tc3_ctc_OCR3A_1hz.c */

#include <avr/io.h>

int main(void)
{
    DDRD   = 1 << 0;        // Set PD0 (OC3A) as OUTPUT mode
    TCCR3A = 0b01000000;    // Set T/C3 to CTC mode(OCR3A as TOP).
                            //   Toggle OC3A.
    TCCR3B = 0b00001100;    // Select Prescaler division ratio.
                            //   16MHz/256=62,500Hz
    OCR3A  = 31249;         // Match Rate is 2Hz. 62500Hz/31250=2Hz

    while (1)
    { /* do nothing */ }
}
```

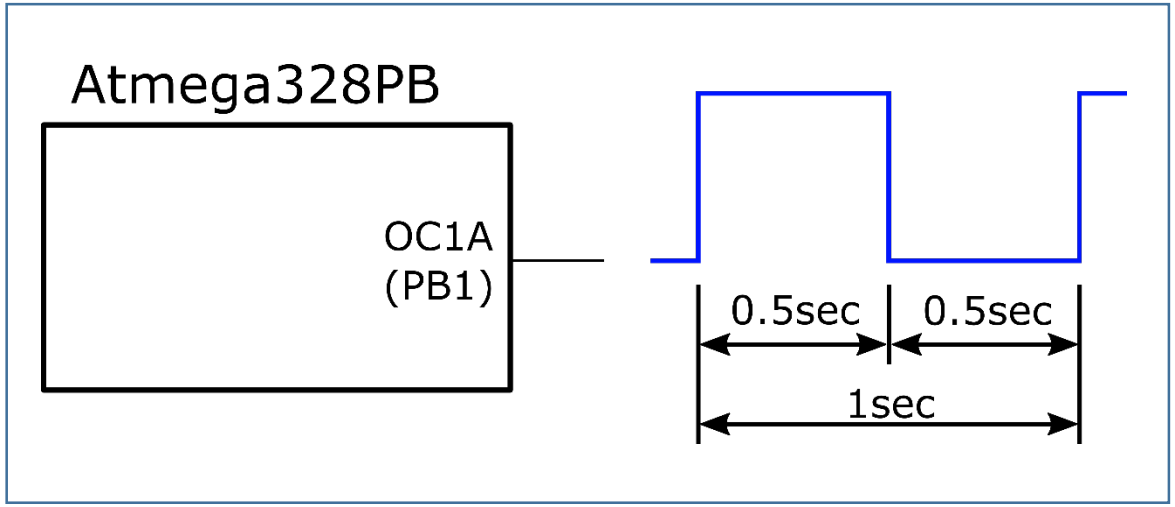


Timer/Counter1
CTC Example
(ICR1 as TOP)

Timer/Counter1 CTC Example (ICR1 as TOP) (1)

Waveform Generation

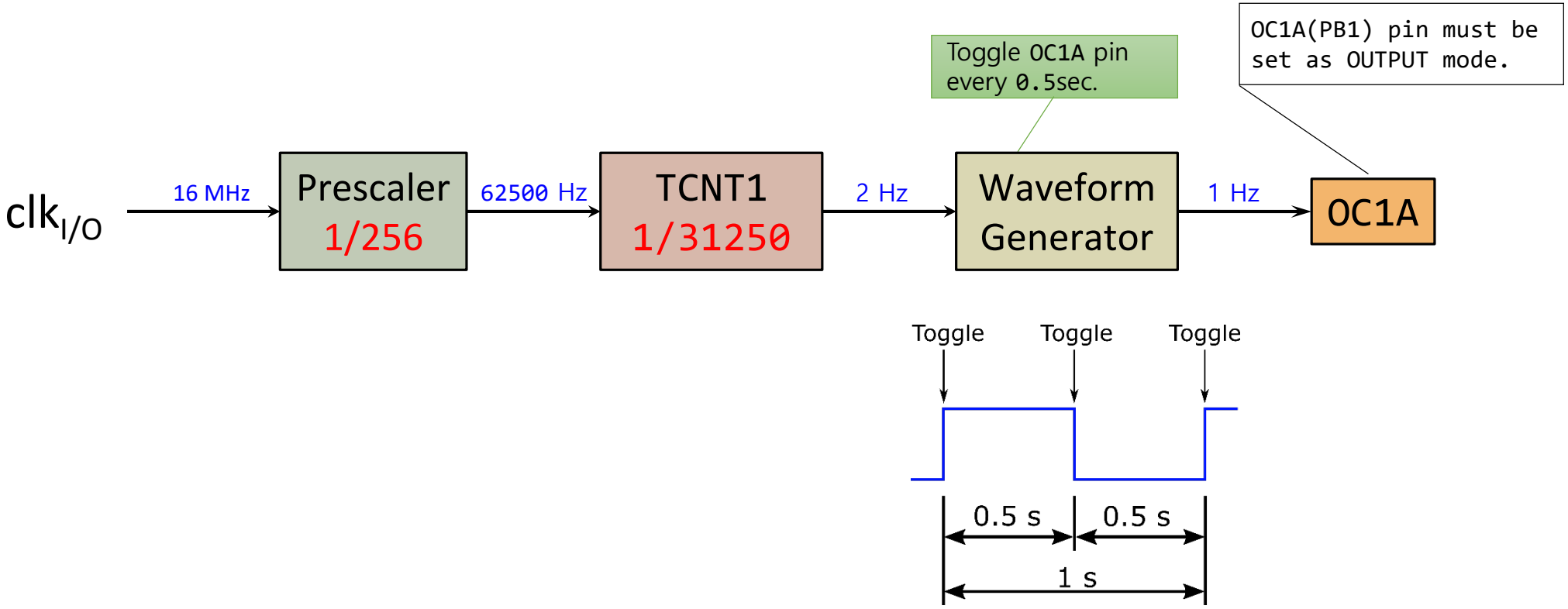
- Make an application which generates 1 Hz rectangular wave on OC1A (PB1).
- Use Timer/Counter1 CTC mode.



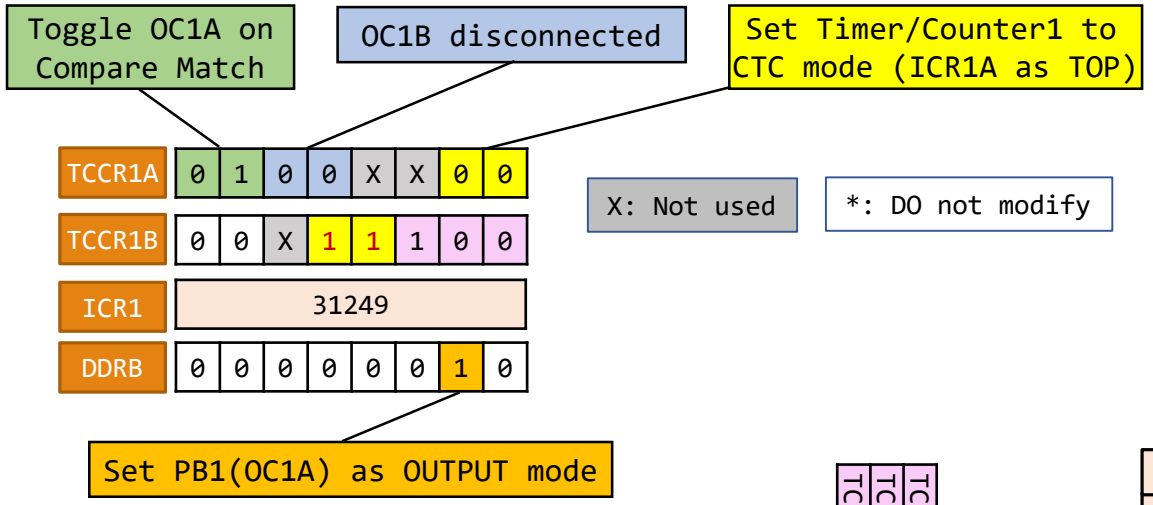
Timer/Counter1 CTC Example (ICR1 as TOP) (2)

Waveform Generation

- Make an application which generates 1 Hz rectangular wave on OC1A (PB1).
- Use Timer/Counter1 CTC mode.

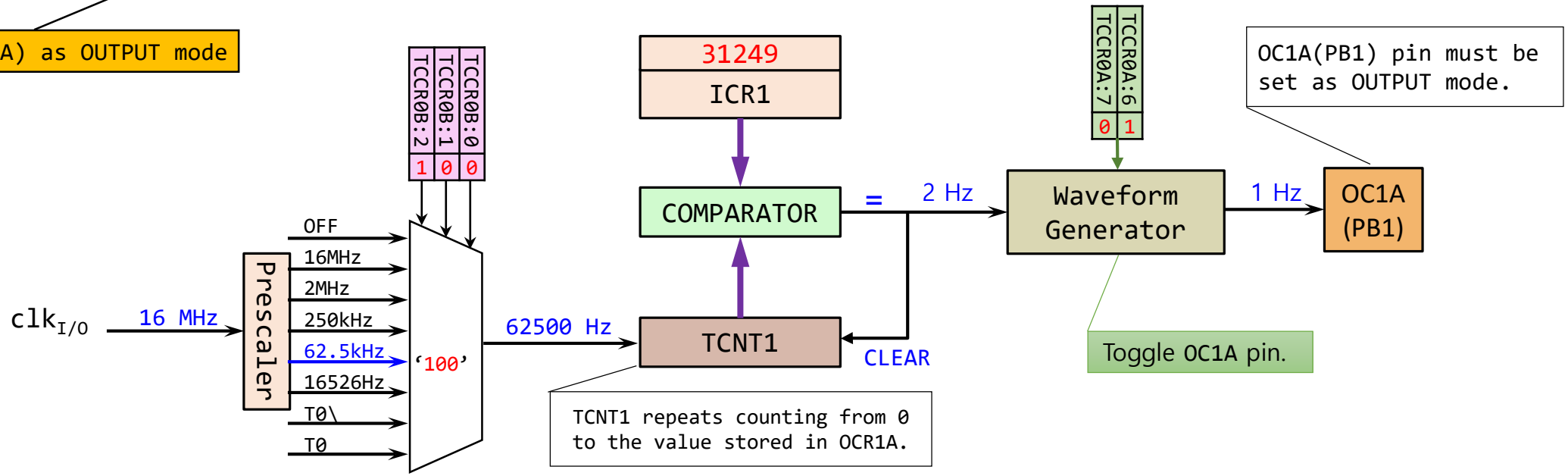


Timer/Counter1 CTC Example (ICR1 as TOP) (3)



Waveform Generation

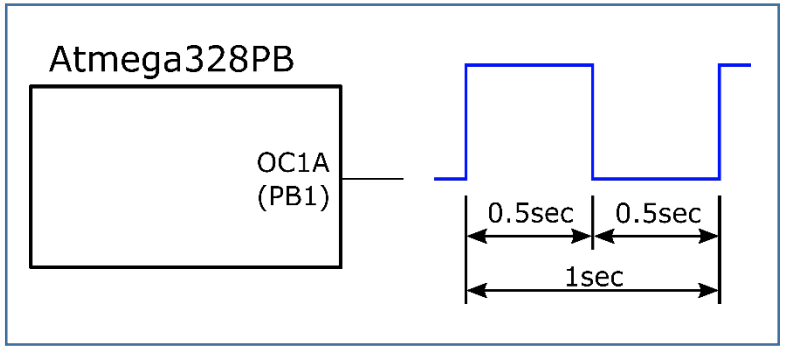
- Make an application which generates 1 Hz rectangular wave on OC1A (PB1).
- Use Timer/Counter1 CTC mode.



Timer/Counter1 CTC Example (ICR1 as TOP) (4)

Make an application which generates 1 Hz rectangular wave on OC1A (PB1).

```
/* tc1_ctc_ICR1_1hz.c */

#include <avr/io.h>

int main(void)
{
    DDRB   = (1 << DDB1);  // Set PB1 (OC1A) as OUTPUT mode
    TCCR1A = 0b01000000;   // Set T/C1 to CTC mode(ICR1 as TOP).
                           //   Toggle OC1A.
    TCCR1B = 0b00011100;   // Select Prescaler division ratio.
                           //   16MHz/256=62,500Hz
    ICR1A  = 31249;        // Match Rate is 2Hz. 62500Hz/31250=2Hz

    while (1)
    { /* do nothing */ }
}
```

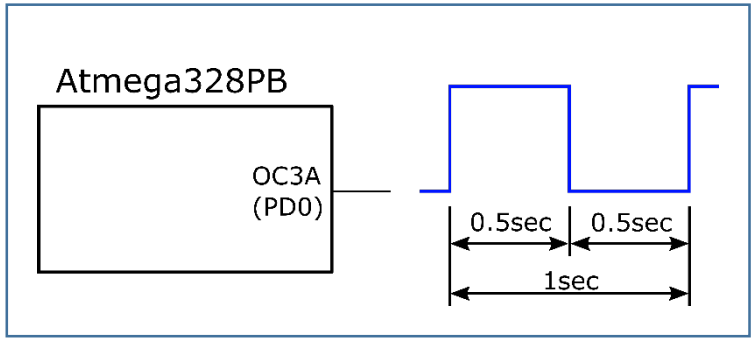


Timer/Counter3
CTC Example
(ICR3 as TOP)

Timer/Counter3 CTC Example (ICR3 as TOP)

Make an application which generates 1 Hz rectangular wave on OC3A (PD0).

```
/* tc3_ctc_ICR3_1hz.c */

#include <avr/io.h>

int main(void)
{
    DDRD   = (1 << DDD0);   // Set PD0 (OC3A) as OUTPUT mode
    TCCR3A = 0b01000000;    // Set T/C3 to CTC mode(ICR3 as TOP).
                            //   Toggle OC3A.
    TCCR3B = 0b00011100;    // Select Prescaler division ratio.
                            //   16MHz/256=62,500Hz
    ICR3A  = 31249;         // Match Rate is 2Hz. 62500Hz/31250=2Hz

    while (1)
    { /* do nothing */ }
}
```

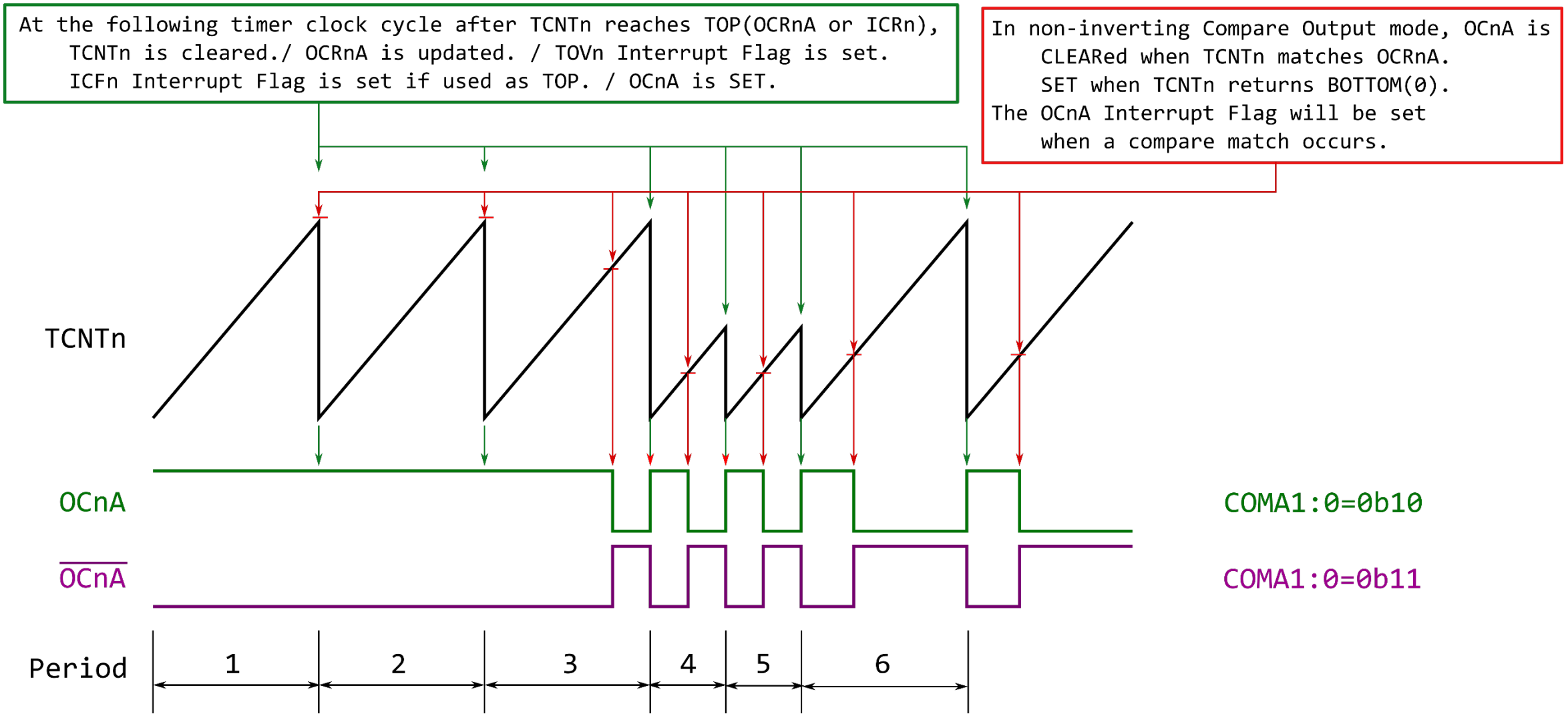


Timer/Counter1,3,4
Fast PWM mode
(Fast Pulse Width Modulation mode)

Timer/Counter1,3,4 Fast PWM Mode (1)

- WGM[3:0]=0b0101, WGM[3:0]=0b0110, WGM[3:0]=0b0111, WGM[3:0]=0b1110, WGM[3:0]=0b1111
- The counter(TCNTn) counts from BOTTOM to TOP, then restarts from BOTTOM.
  - ✓ BOTTOM is 0
  - ✓ TOP is
    - 0x00FF when WGM[3:0]=0101.
    - 0x01FF when WGM[3:0]=0110.
    - 0x03FF when WGM[3:0]=0111.
    - ICRn when WGM[3:0]=1110.
    - OCRnA when WGM[3:0]=1111.
- The Timer/Counter Overflow Flag (TOVn) is set each time the counter reaches TOP.
  - ✓ If the interrupt is enabled, the interrupt handler routine(TIMERn_OVF_vect) can be used for updating the compare value.
- Two types of PWM signals can be available
  - ✓ Invert: COMA[1:0]=0b11
  - ✓ Non-invert: COMA[1:0]=0b10

Timer/Counter1,3,4 Fast PWM Mode (2)

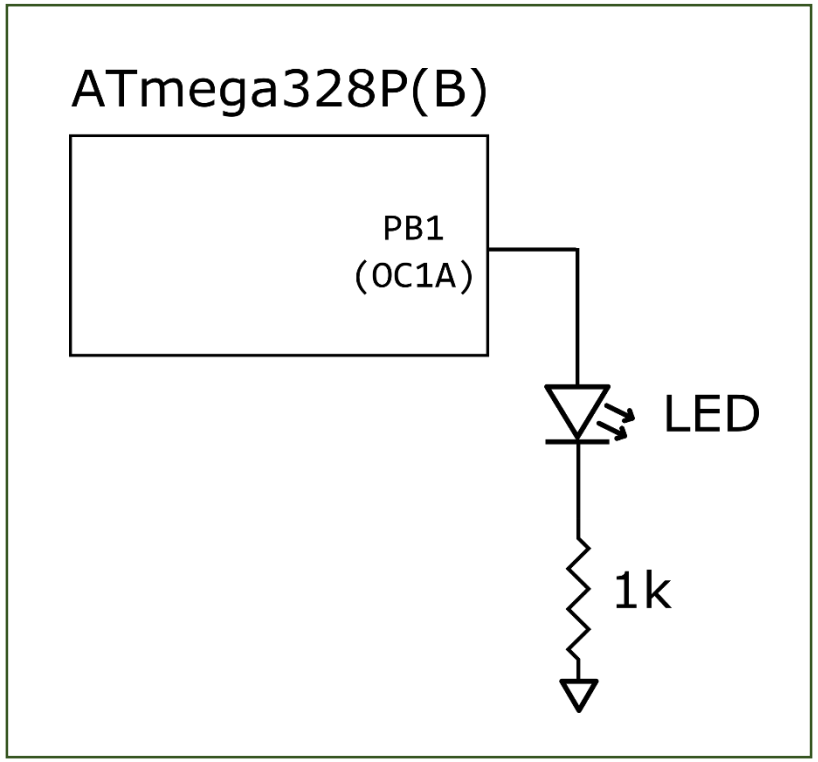


Timer/Counter1,3,4
Fast PWM Example 1
(Single Channel / Fixed Duty Cycle)

Timer/Counter1,3,4 Fast PWM Example 1 (1)

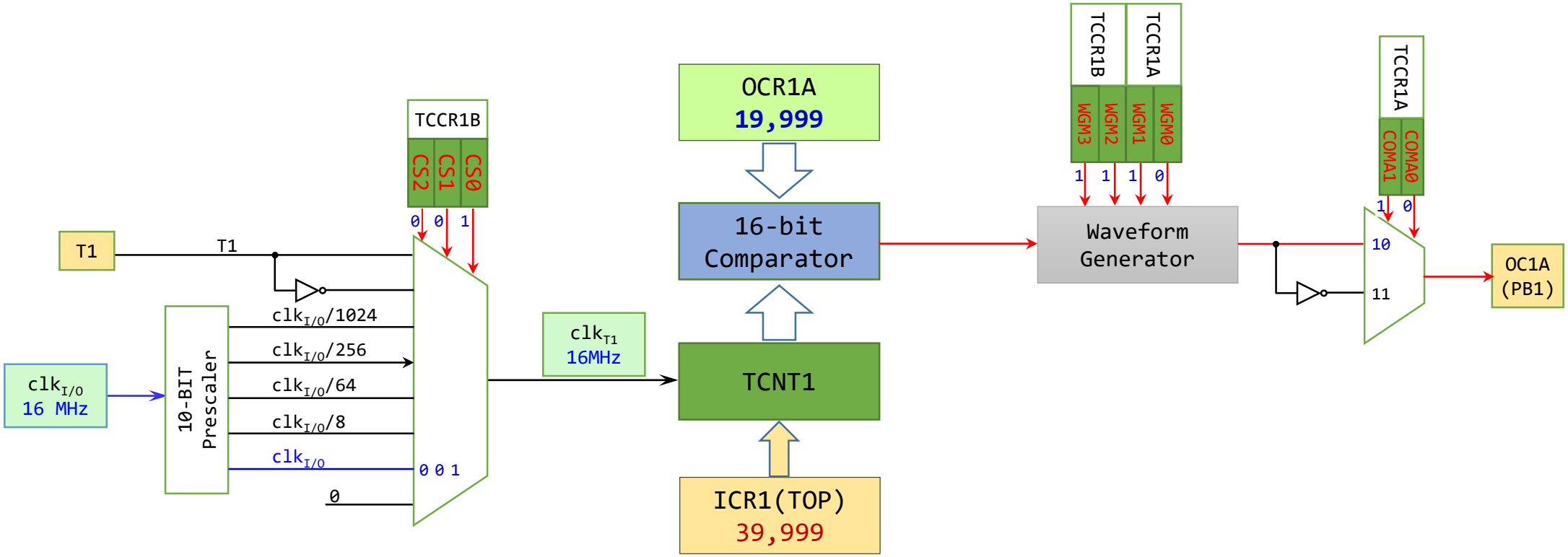
Make an application which generates single PWM signal.

- PWM signal drives an LED connected to OC1A (PB1).
- Duty cycle: 50% (fixed).
- PWM frequency: 400Hz (fixed)
- Use ICR1 as TOP (WGM[3:0]=1110)



Timer/Counter1,3,4 Fast PWM Example 1 (2)

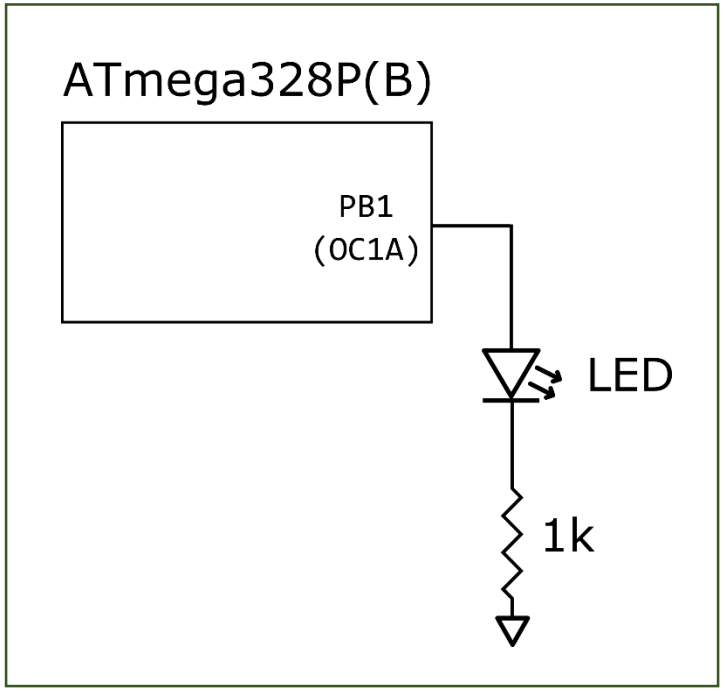
TCNT0 TOP = ICR1 ← 39,999
PWM frequency = 16MHz/40,000 = 400Hz
PWM duty cycle = 50% ⇨ OCR1A ← 19,999



Timer/Counter1,3,4 Fast PWM Example 1 (3)

Make an application which generates fixed duty cycle PWM signal.

```
/* tc1_fast_pwm_ICR1_top_OCR1A.c */

#include <avr/io.h>

int main(void)
{
    DDRB   = 0b00000010;    // Set OC1A (PB1) to OUTPUT
    TCCR1A = 0b10000010;    // OC1A:Non-Inverting PWM, OC1B:Disconnect
                            // Fast PWM, TOP=ICR1
    TCCR1B = 0b00011001;    // Fast PWM, TOP=ICR1
                            // Prescaler: div by 1,
                            // CLKt1=16MHz/1=16MHz
    ICR1 = 39999;           // TCNT1 TOP. PWM freq = 16MHz/40,000=400Hz
    OCR1A = 19999;          // Duty cycle=50% (fixed)

    while (1)
    {  /* do nothing */  }
}
```

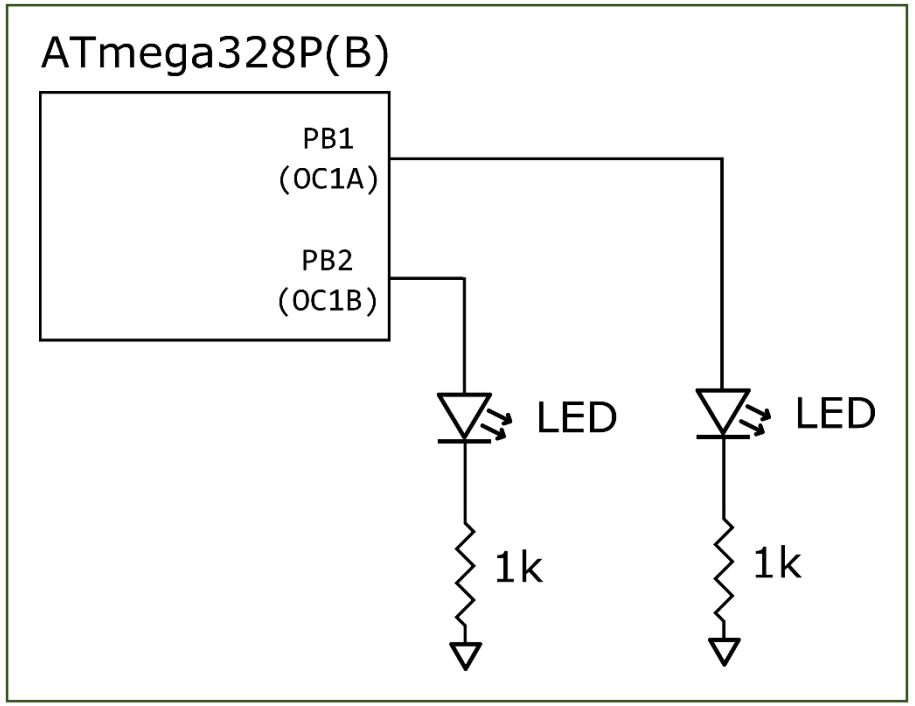


Timer/Counter1,3,4
Fast PWM Example 2
(Dual Channel / Fixed Duty Cycle)

Timer/Counter1,3,4 Fast PWM Example 2 (1)

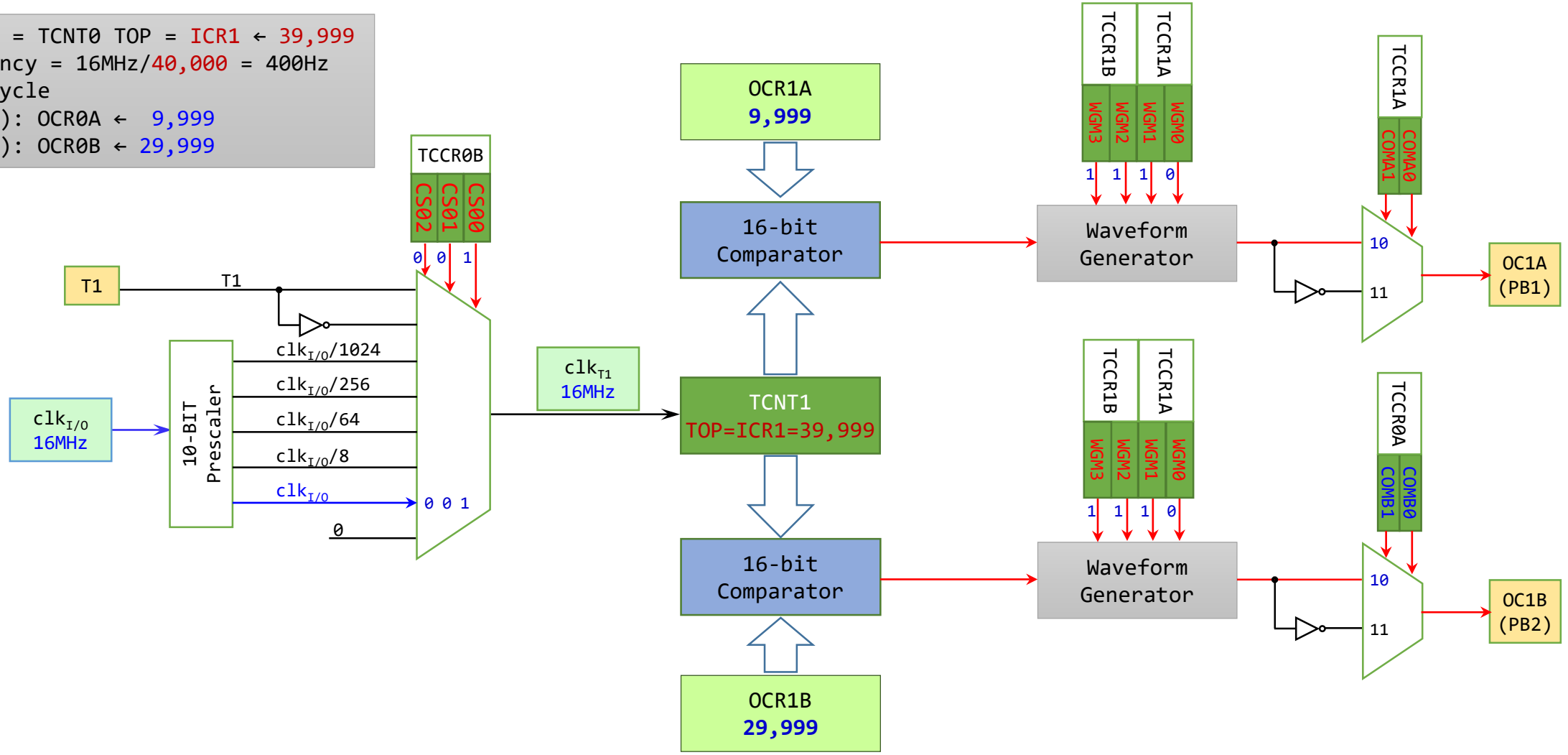
Make an application which generates dual PWM signals.

- PWM signals are available at OC1A (PB1) and OC1B (PB2).
- Duty cycle: 25% (OC1A) and 75% (OC1B).
- PWM frequency: 400Hz (fixed).
- Use ICR1 as TOP (WGM[3:0]=1110)



Timer/Counter1,3,4 Fast PWM Example 2 (2)

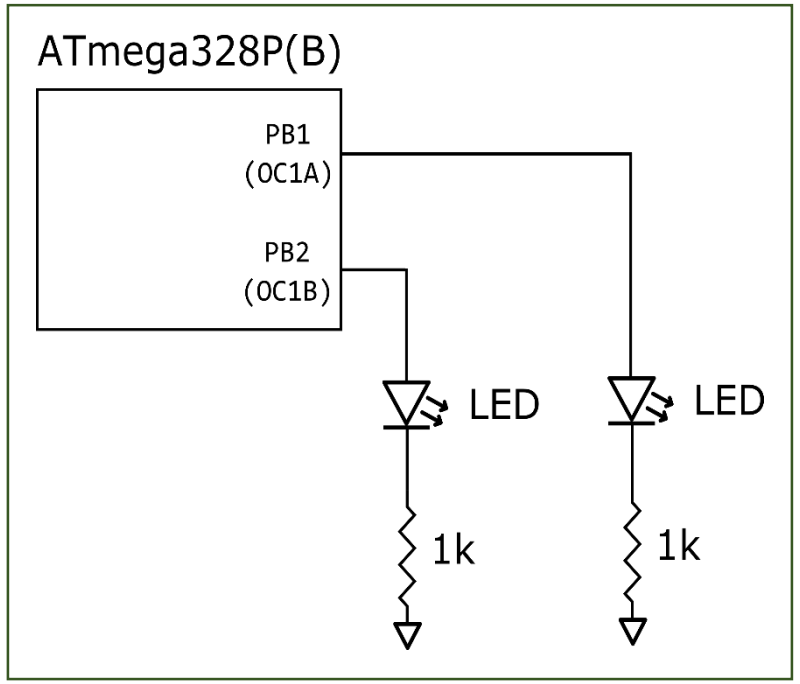
```
PWM Period = TCNT0 TOP = ICR1 ← 39,999
PWM frequency = 16MHz/40,000 = 400Hz
PWM duty cycle
  OC0A(25%): OCR0A ←  9,999
  OC0B(75%): OCR0B ← 29,999
```



Timer/Counter1,3,4 Fast PWM Example 2 (3)

Make an application which generates dual fixed duty cycle PWM signals.

```
/* tc1_fast_pwm_ICR1_top_dual.c */
/* PWM signal drives two LEDs connected to OC1A(PB1) and OC1B(PB2). */
/* Duty cycle: 25% at OC1A and 75% at OC1B. */

#include <avr/io.h>

int main(void)
{
    DDRB   = 0b00000110;    // Set OC1A(PB1) and OC1B(PB2) to OUTPUT
    TCCR1A = 0b10100010;    // OC1A and OC1B:Non-Inverting PWM
                            // Fast PWM, TOP=ICR1
    TCCR1B = 0b00011001;    // Fast PWM, TOP=ICR1
                            // Prescaler: div by 1, CLKt1=16MHz/1=16MHz
    ICR1   = 39999;         // TCNT1 TOP. PWM freq = 16MHz/40,000=400Hz
    OCR1A =  9999;          // Duty cycle on OC1A=25% (fixed)
    OCR1B = 29999;          // Duty cycle on OC1B=75% (fixed)

    while (1)
    { /* do nothing */ }
}
```

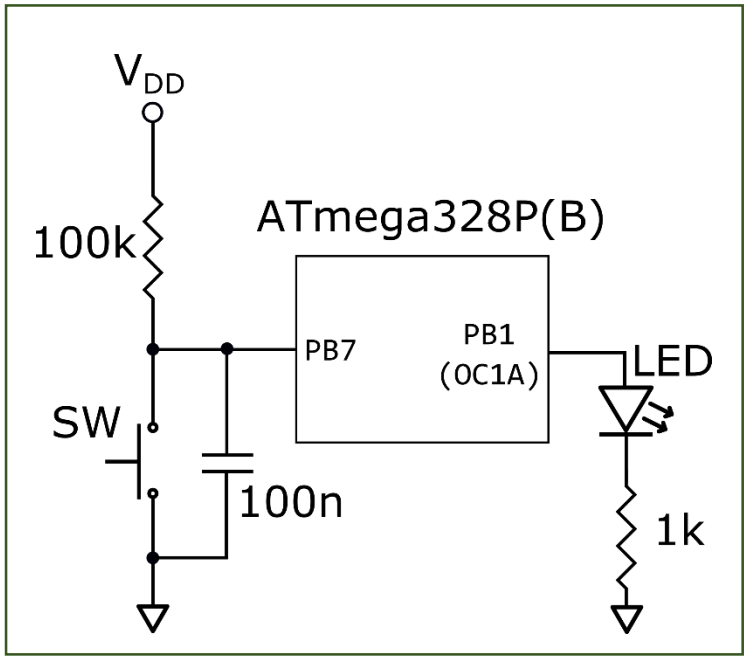


Timer/Counter1,3,4
Fast PWM Example 3
(Single Channel / Variable Duty Cycle)

Timer/Counter1,3,4 Fast PWM Example 3 (1)

Problem

Make an application which controls LED brightness by SWITCH press, i.e. LED brightness is getting higher whenever the SWITCH is pressed.

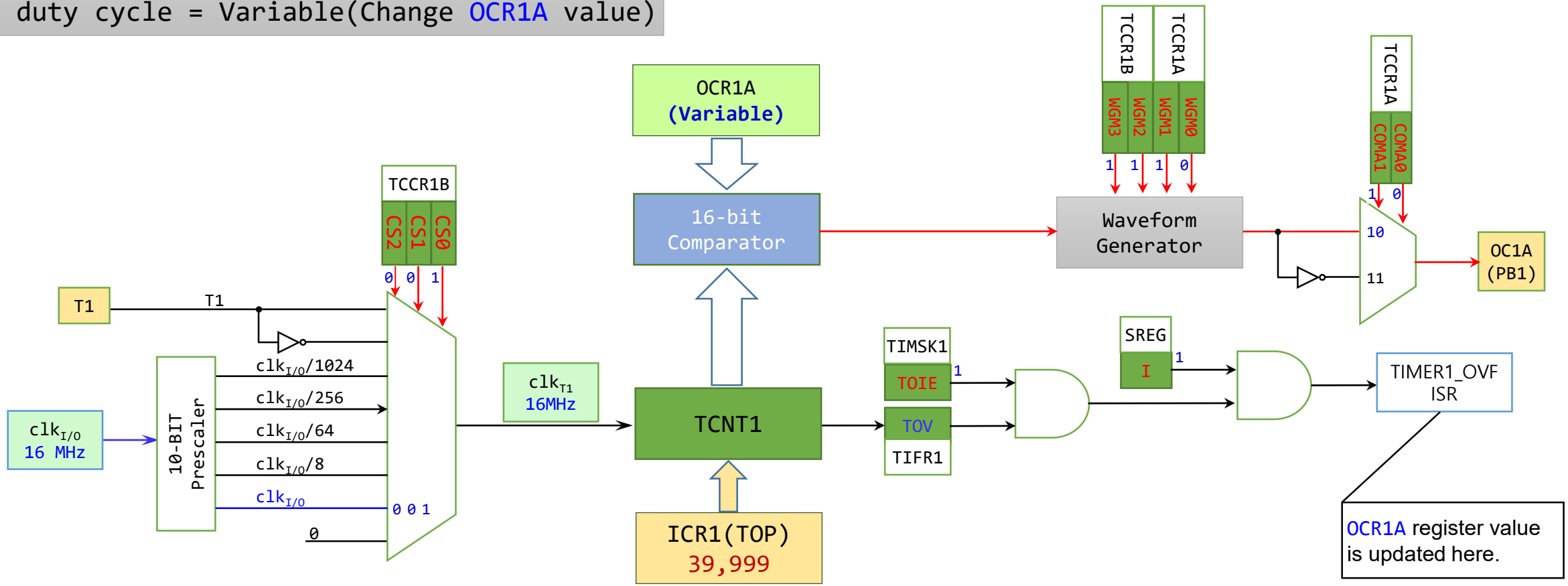


Solution

- Use `Timer/Counter1` Fast PWM mode
- Use `ICR1` as TOP (`WGM[3:0]=1110`)
- PWM signal drives an LED connected to OC1A (PB1).
- Duty cycle: variable.
- PWM frequency: 400Hz (fixed)

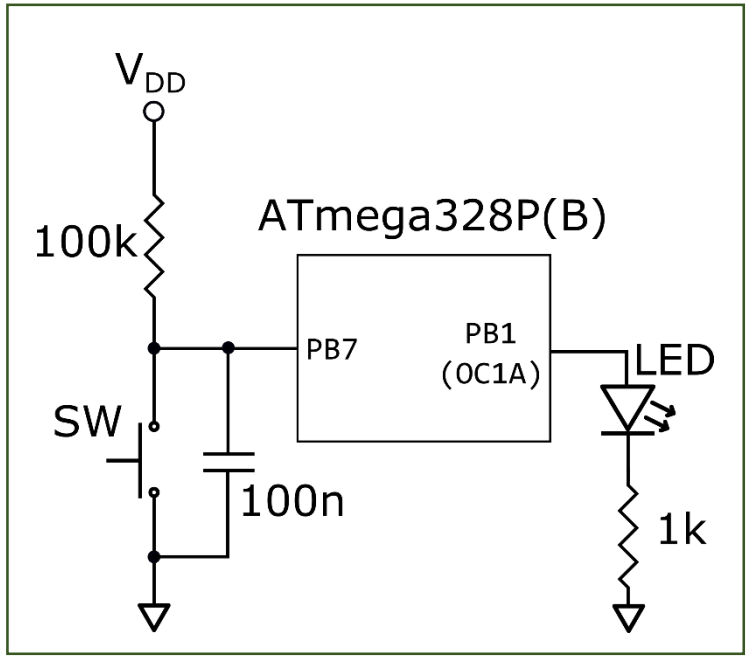
Timer/Counter1,3,4 Fast PWM Example 3 (2)

TCNT0 TOP = ICR1 ← 39,999
PWM frequency = 16MHz/40,000 = 400Hz
PWM duty cycle = Variable(Change OCR1A value)



Timer/Counter1,3,4 Fast PWM Example 3 (3)

```
/* <tc1_fast_PWM_ex3.c>  PWM signal at OC1A(PB1) changes whenever SWITCH at PB7 is pressed. */

#include <avr/io.h>
#include <avr/interrupt.h>
#define TOP  39999

unsigned int duty_cycle = 0;

int main(void)
{
    DDRB   = 0b00000010;  // Set OC1A (PB1) to OUTPUT
    TCCR1A = 0b10000010;  // OC1A:Non-Inverting PWM, OC1B:Disconnect, Fast PWM, TOP=ICR1
    TCCR1B = 0b00011001;  // Fast PWM, TOP=ICR1, Prescaler: div by 1, CLKt1=16MHz/1=16MHz
    ICR1   = TOP;         // TCNT1 TOP. PWM freq = 16MHz/40,000=400Hz
    OCR1A  = duty_cycle;  // Duty cycle=variable (Initial value:0)
    TIMSK1 = 0b00000001;  // Enable Timer/Count1 Overflow Interrupt
    sei();

    while (1)
    {
        if ((PINB & (1 << PINB7)) == 0)     // switch pressed?
        {
             duty_cycle += 2000;            // update duty cycle
             if (duty_cycle > TOP)
                 duty_cycle = 0;
             while ((PINB & (1 << PINB7)) == 0);     // wait for switch release
        }
    }
}

ISR(TIMER1_OVF_vect)
{
    OCR1A = duty_cycle;   // Apply new duty value
}
```



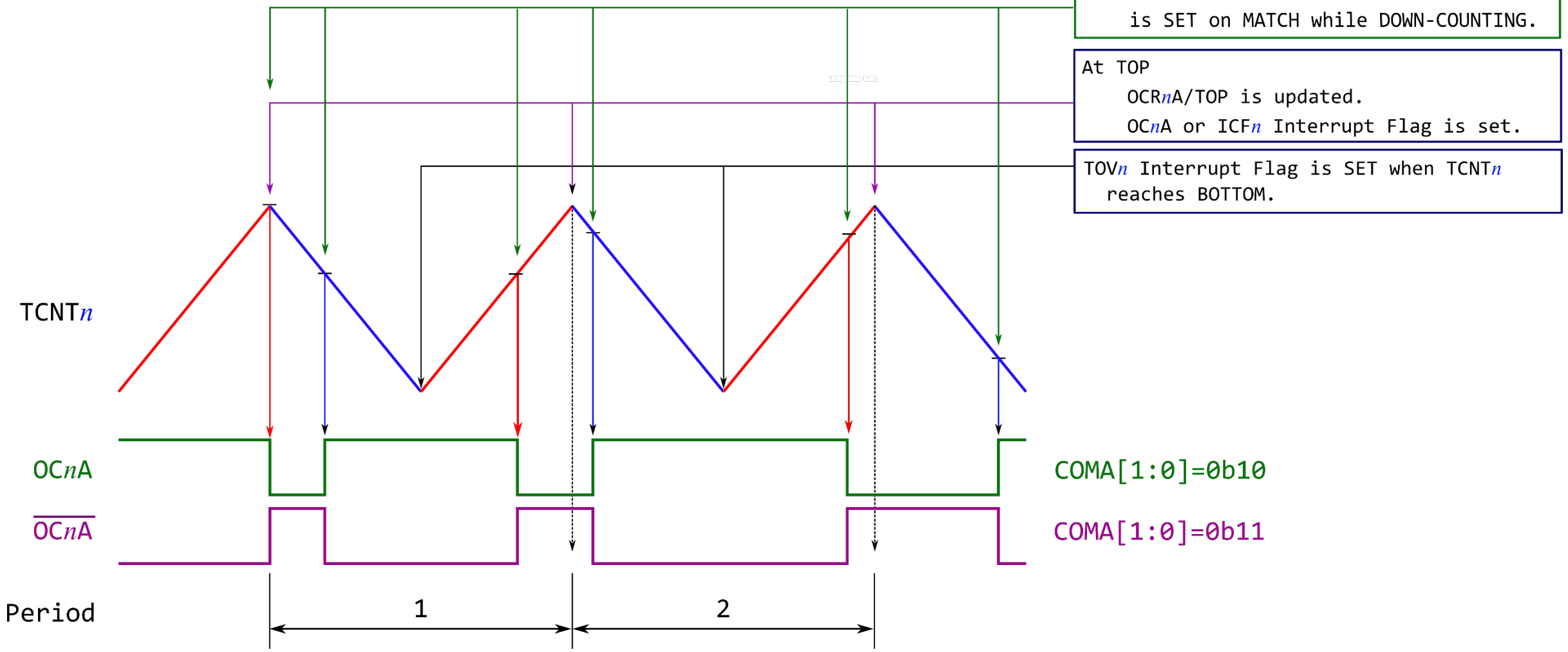
Timer/Counter1,3,4
Phase Correct PWM mode

Timer/Counter1,3,4 Phase Correct PWM Mode (1)

- WGM[3:0]=0b0001, WGM[3:0]=0b0010, WGM[3:0]=0b0011, WGM[3:0]=0b1001, WGM[3:0]=0b1011
- The counter(TCNT*n*) is incremented until the counter value matches TOP.
- When the counter reaches TOP, it changes the count direction.
- The TCNT*n* value will be equal to TOP for one timer clock cycle.
- The counter is decremented until the counter value matches BOTTOM.
  - ✓ BOTTOM is 0
  - ✓ TOP is
    - 0x00FF when WGM[3:0]=0001.
    - 0x01FF when WGM[3:0]=0010.
    - 0x03FF when WGM[3:0]=0011.
    - ICRn when WGM[3:0]=1010.
    - OCRnA when WGM[3:0]=1011.
- Two types of PWM signals can be available
  - ✓ Non-invert: COMA[1:0]=0b10
    - OC*n*A bit is cleared on compare match while up-counting.
    - OC*n*A is set on the compare match while down-counting.
  - ✓ Invert: COMA[1:0]=0b11
    - Operation is inverted.

Timer/Counter1,3,4 Phase Correct PWM Mode (2)

TCNT*n* TOP == OCR*n*A or ICR*n*



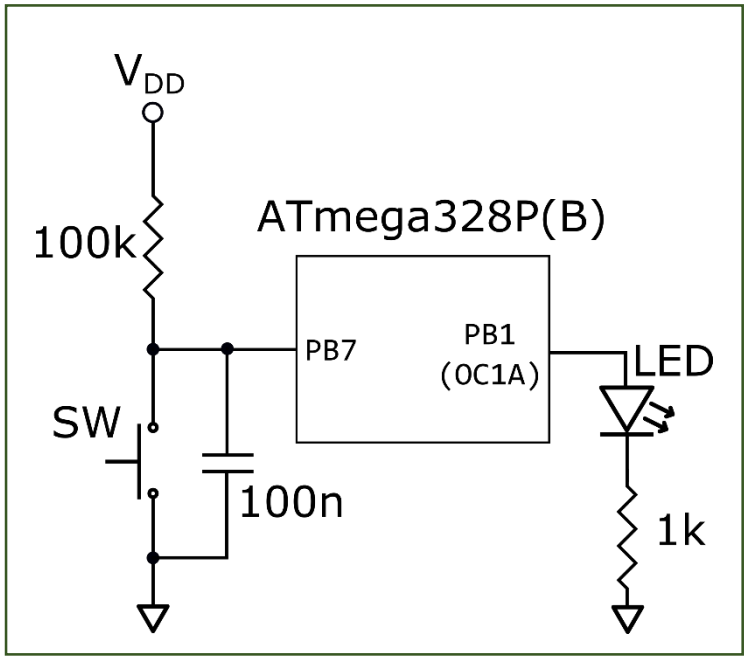
Timer/Counter1,3,4
Phase Correct PWM Example 1

(Single Channel / Variable Duty Cycle)

Timer/Counter1,3,4 Phase Correct PWM Example (1)

Problem

Make an application which controls LED brightness by SWITCH press, i.e. LED brightness is getting higher whenever the SWITCH is pressed.

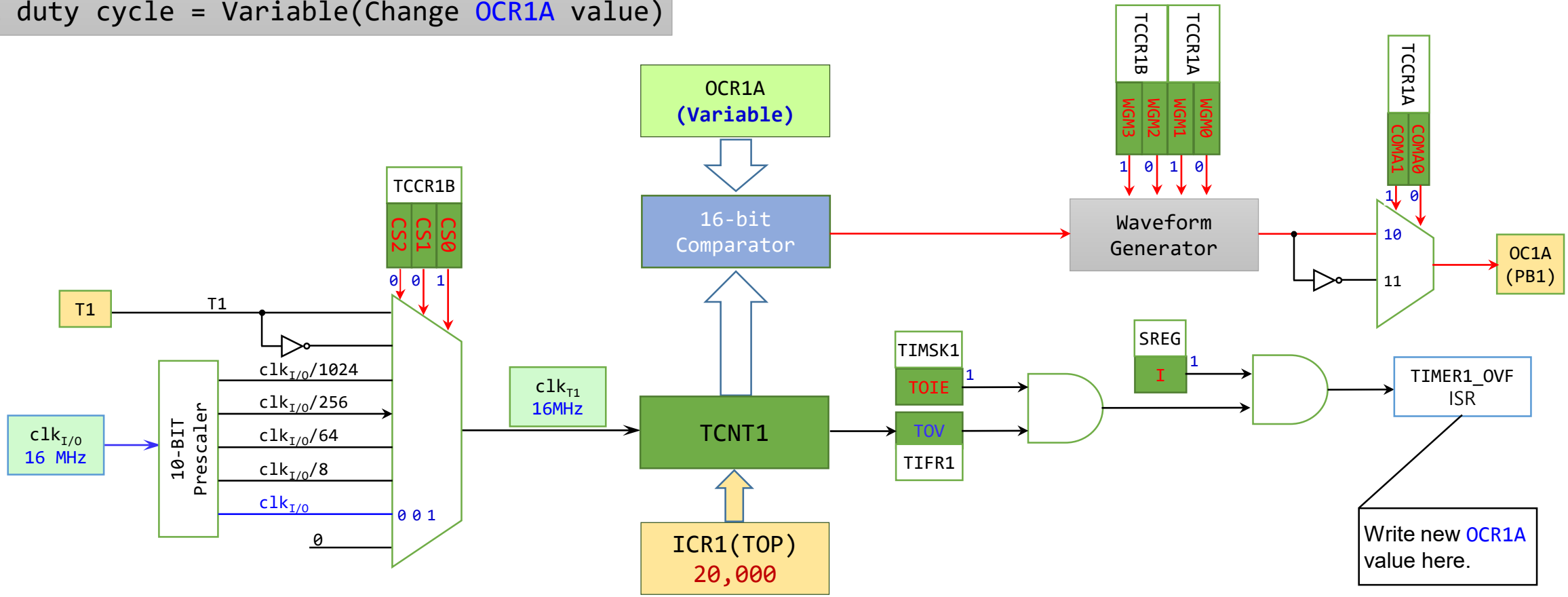


Solution

- Use `Timer/Counter1` Phase Correct PWM mode
- Use `ICR1` as TOP (`WGM[3:0]=1010`)
- PWM signal drives an LED connected to OC1A (PB1).
- Duty cycle: variable.
- PWM frequency: 400Hz (fixed)

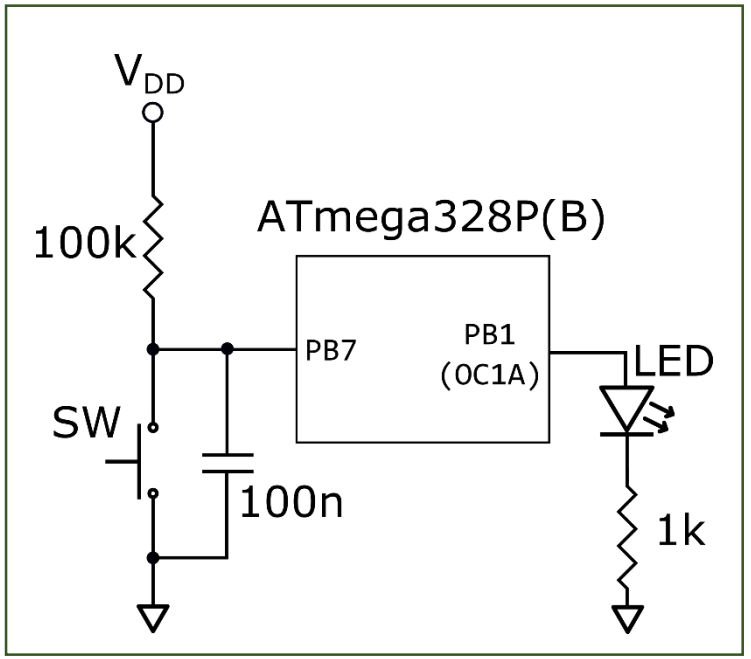
Timer/Counter1,3,4 Phase Correct PWM Example (2)

TCNT0 TOP = ICR1 ← 20,000
PWM frequency = 16MHz/40,000 = 400Hz
PWM duty cycle = Variable(Change OCR1A value)



Timer/Counter1,3,4 Phase Correct PWM Example (3)

```c
/* tc1_phase_correct_pwm_example.c
 * PWM signal at OC1A(PB1) changes whenever SWITCH at PB7 is pressed. */

#include <avr/io.h>
#include <avr/interrupt.h>

#define TOP    20000

unsigned int duty_cycle = 0;

int main(void)
{
    DDRB   = 0b00000010;     // Set OC1A (PB1) to OUTPUT
    TCCR1A = 0b10000010;     // OC1A:Non-Inverting PWM, OC1B:Disconnect, Phase Correct PWM, TOP=ICR1
    TCCR1B = 0b00010001;     // Phase Correct PWM, TOP=ICR1, Prescaler: div by 1, CLKt1=16MHz/1=16MHz
    ICR1 = TOP;              // TCNT1 TOP. PWM freq = 16MHz/40,000=400Hz
    OCR1A = duty_cycle;      // Duty cycle=variable (Initial value:0)
    TIMSK1 = 0b00000001;     // Enable Timer/Count1 Overflow Interrupt
    sei();

    while (1)
    {
        if ((PINB & (1 << PINB7)) == 0)     // switch pressed?
        {
            duty_cycle += 2000;             // update duty cycle
            if (duty_cycle > TOP)           // OCR1A value must be smaller than TOP
                duty_cycle = 0;
            while ((PINB & (1 << PINB7)) == 0);          // wait for switch release
        }
    }
}

ISR(TIMER1_OVF_vect)
{
    OCR1A = duty_cycle;      // Apply new duty value
}
```

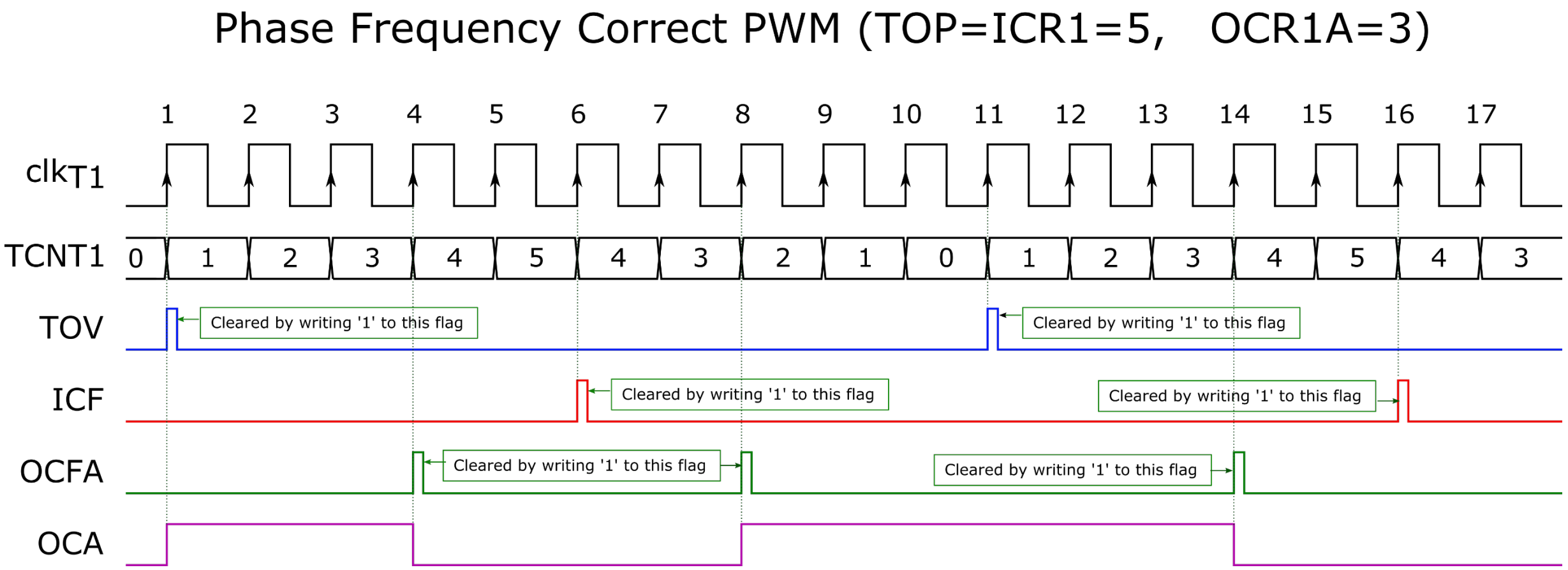


Timer/Counter1,3,4
Phase Frequency Correct PWM mode

Timer/Counter1,3,4 Phase Frequency Correct PWM Mode (1)

- WGM[3:0]=0b1000, WGM[3:0]=0b1001
- The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCRnx Register is updated by the OCRnx Buffer Register.
- The counter(TCNT*n*) is incremented until the counter value matches TOP.
- When the counter reaches TOP, it changes the count direction.
- The TCNT*n* value will be equal to TOP for one timer clock cycle.
- The counter is decremented until the counter value matches BOTTOM.
  - ✓ BOTTOM is 0
  - ✓ TOP is
    - ICRn when WGM[3:0]=0b1000.
    - OCRnA when WGM[3:0]=0b1001.
- Two types of PWM signals can be available
  - ✓ Non-invert: COMA[1:0]=0b10
    - OC*n*A bit is cleared on compare match while up-counting.
    - OC*n*A is set on the compare match while down-counting.
  - ✓ Invert: COMA[1:0]=0b11
    - Operation is inverted.

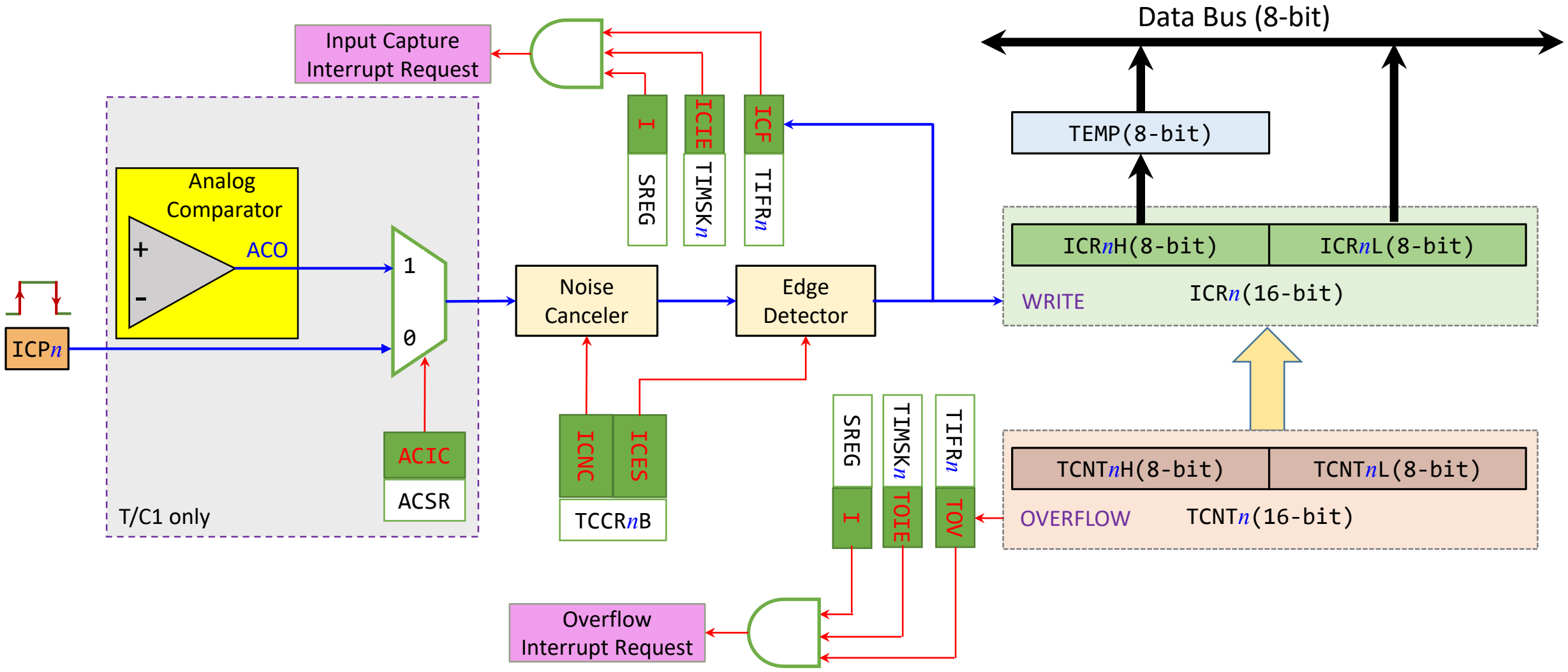
Timer/Counter1,3,4 Phase Frequency Correct PWM Mode (2)



Timer/Counter1,3,4

Input Capture Unit

T/C1,3,4 Input Capture Unit Block Diagram



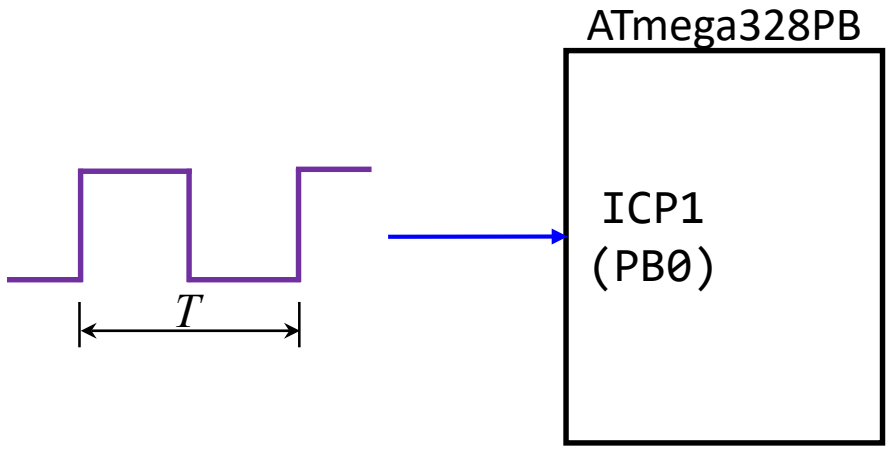
Timer/Counter1
Input Capture Example

Timer/Counter1 Input Capture Example (1)

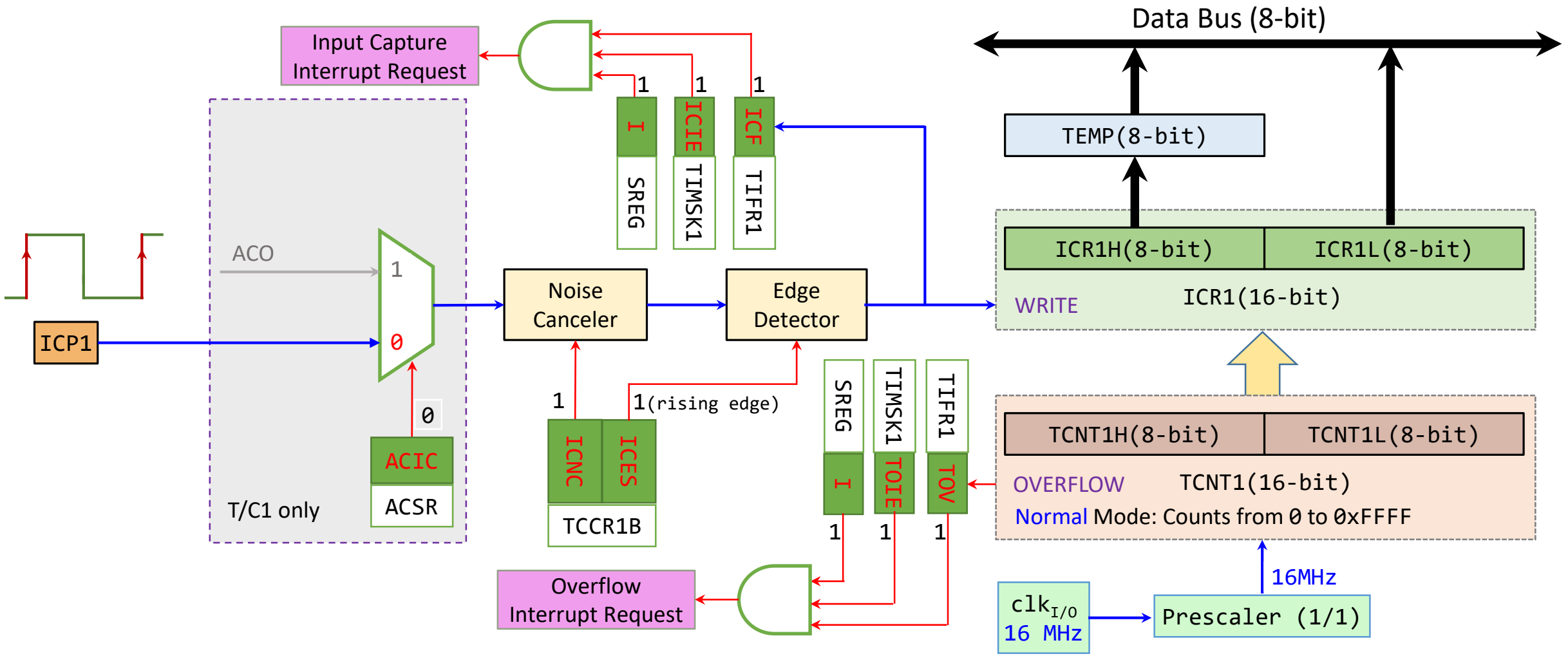
Frequency Measurement of A Rectangular Waveform

- Make an application which measures frequency of a rectangular wave on ICP1 (PB0).
- Use Timer/Counter1 Input Capture Unit.
- Use Noise Canceler.

Frequency $f = 1/T$

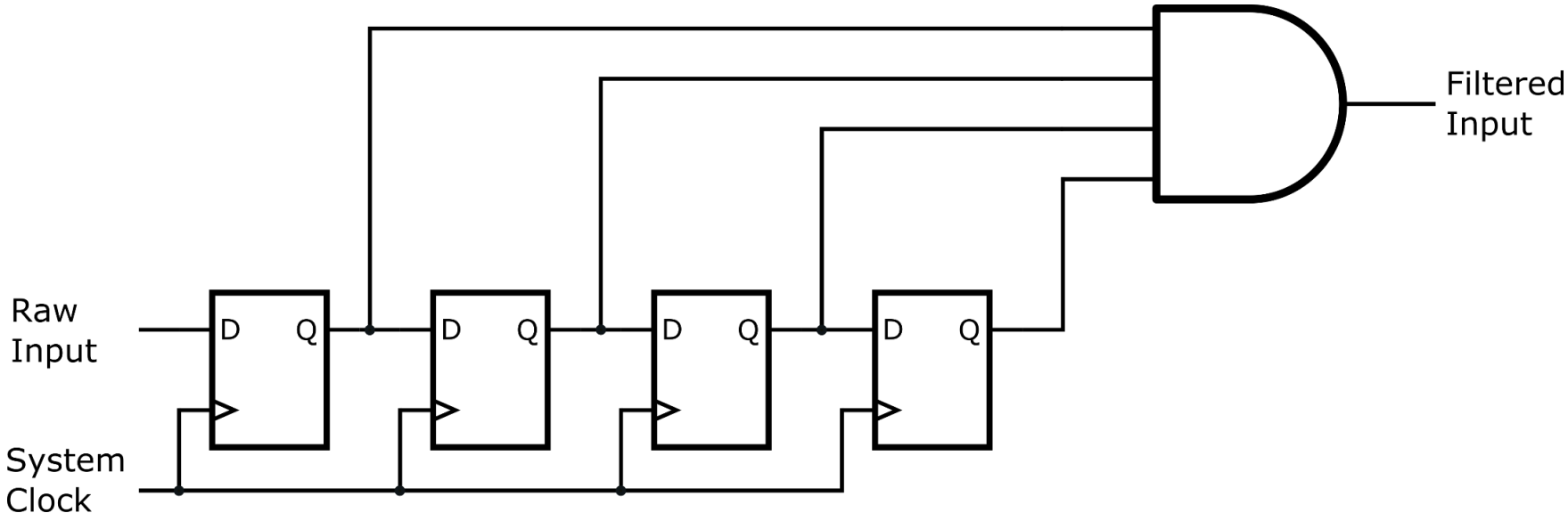


Timer/Counter1 Input Capture Example (2)

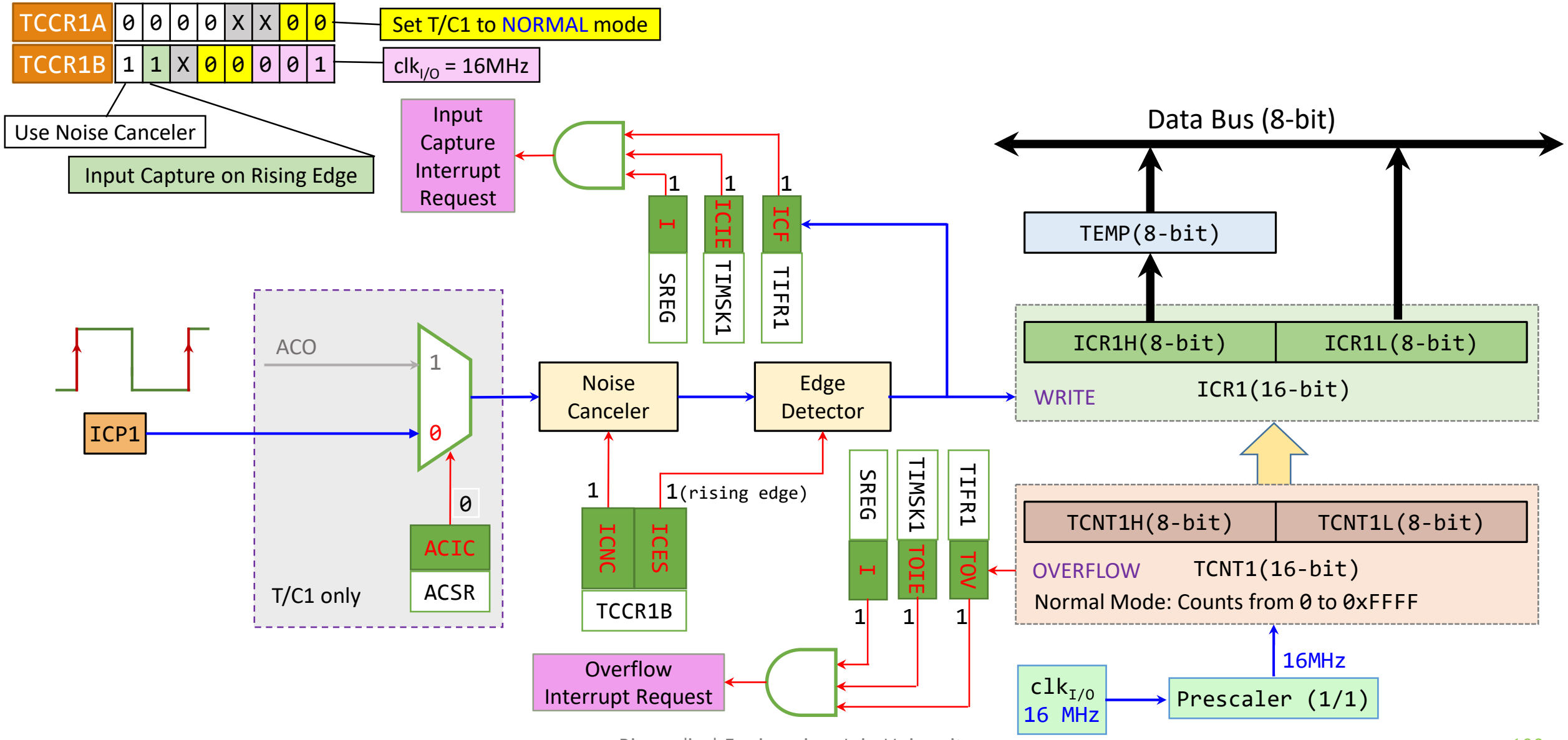


Noise Canceler

- The noise canceler improves noise immunity by using a simple digital filtering scheme.
- The noise canceler input is monitored over four samples.
- All four must be equal for changing the output that in turn is used by the edge detector.
- The noise canceler is enabled by setting the `Input Capture Noise Canceler` bit in the `TCCRnB.ICNC`.
- When enabled, the noise canceler introduces an additional delay of four system clock cycles between a change applied to the input and the update of the `ICRn` Register.



Timer/Counter1 Input Capture Example (3)



Timer/Counter1 Input Capture Example (4)

```
#define F_CPU         16000000UL
#define F_TIMER1      F_CPU
#define BAUD_RATE     1000000UL// 1 Mbps

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>

void uart_init(uint32_t baudrate);

volatile uint8_t capture_flag;
uint32_t overflow_count;
uint32_t num_of_count;

int main(void)
{
    uint32_t input_count;
    float    input_freq;

    uart_init(BAUD_RATE);

    TCCR1A = 0x00;
    TCCR1B = (1 << ICNC1)              // Enable Noise Canceler
           | (1 << ICES1)              // Capture on Positive Edge
           | (0b001 << CS10);          // Prescaler: divide by 1
    ACSR  &= ~(1<<ACIC );              // Select ICP1 as Capture Trigger Input
    TIFR1  |= (1 << ICF1) | (1 << TOV1); // Clear ICF1 and TOV1
    TIMSK1 = (1<< ICIE1)               // Enable Input Capture Interrupt
           | (1 << TOIE1);             // Enable Overflow Interrupt
    sei();
```

```
    while (1)
    {
        if (capture_flag == 1)    // new captured value is available
        {
            input_count = num_of_count;
            input_freq = (float)F_TIMER1 / (float)input_count;
            printf("Freq=%.1f Hz\n", input_freq);

            capture_flag = 0;
         }
    }
}

ISR(TIMER1_OVF_vect)      // T/C1 Overflow ISR
{
    overflow_count++;
}

ISR(TIMER1_CAPT_vect)     // T/C1 Capture ISR
{
    static uint32_t  prev_count = 0;
    uint32_t         curr_count;
    uint32_t         curr_icr_value;

    curr_icr_value = ICR1;
    curr_count = overflow_count * 65536L + curr_icr_value;
    num_of_count = curr_count - prev_count;

    prev_count = curr_count;
    capture_flag = 1;
}
```

Timer/Counter1 Input Capture Example (5)

```c
/* uart.c */

#define F_CPU 16000000UL

#include <avr/io.h>
#include <stdio.h>

int uart_putchar(char ch, FILE *stream);

FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_RW);

void uart_init(uint32_t baudrate)
{
    UCSR0B |= (1 << RXEN0) | (1 << TXEN0);        // enable TX and RX
    UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);      // 8-bit word
    UBRR0 = (((F_CPU / (baudrate * 16UL))) - 1);
    stdout = &uart_str;
}

int uart_putchar(char ch, FILE *stream)
{
    while ((UCSR0A & (1 << UDRE0)) == 0);
    UDR0 = ch;
    return 0;
}
```

16-bit Timer/Counter
TC1, TC3, TC4

END

What's next?


THE TWO STATES OF EVERY PROGRAMMER
I AM A GOD.
I HAVE NO IDEA WHAT I'M DOING.