ATmega328P Timer/Counter 1

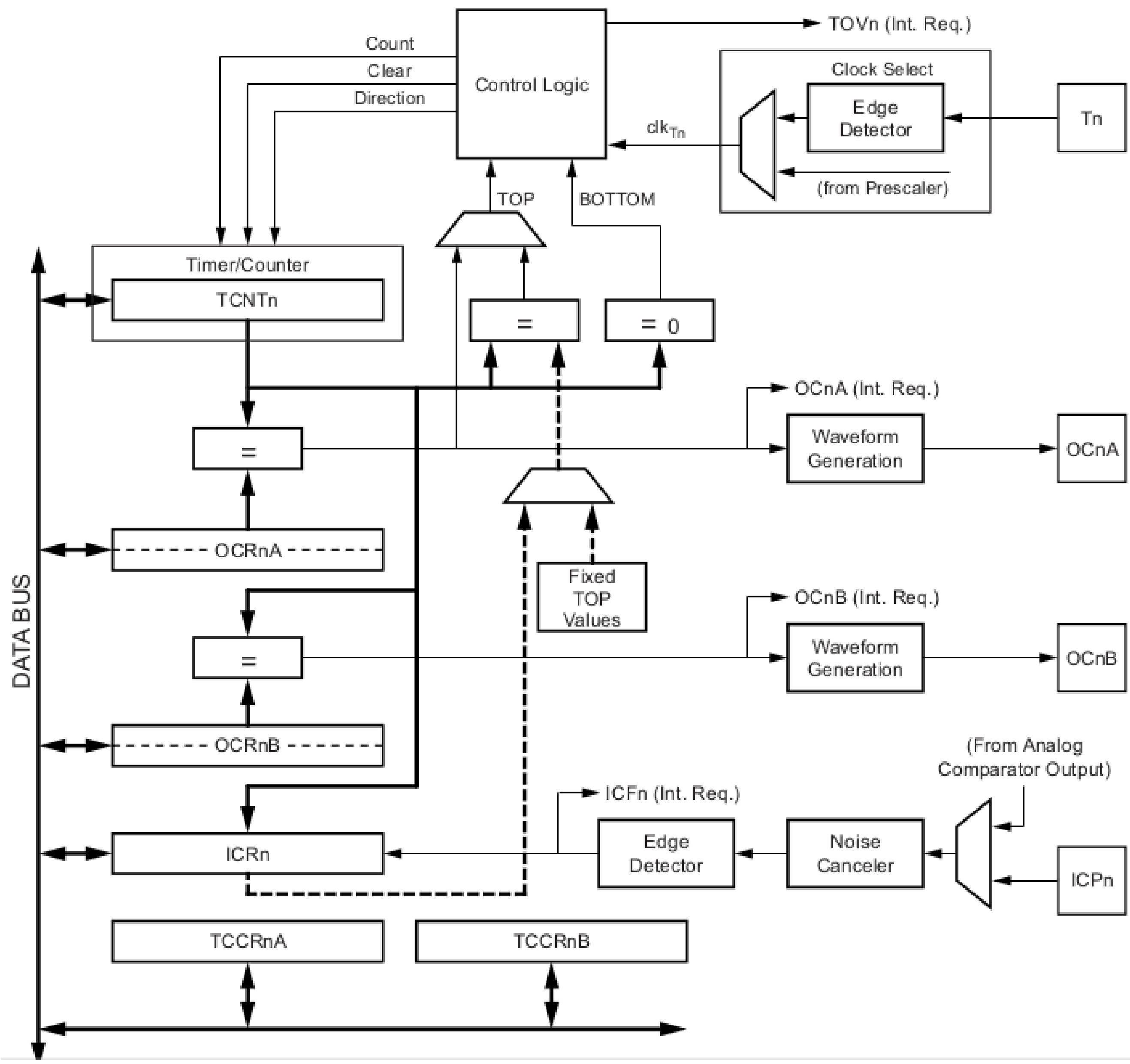
Narendiran S

July 28, 2020

1 Features

- General purpose 16-bit PWM/Counter module.
- Two independent output compare units and One input capture unit
- Variable PWM.
- Four independent interrupt sources (TOV1, OCF0A, OCF1B and ICF1).
- Clear timer on compare match (auto reload)

2 Block Diagram



3 Terminologies and Registers

Parameter	Description
BOTTOM	counter reaches 0x0000
MAX	ounter reaches 0xFFFF
TOP	counter reaches highest value (depends on mode of operation can be 0xFF, 0x1FF, 0x3FF, OCR1A, ICR1)

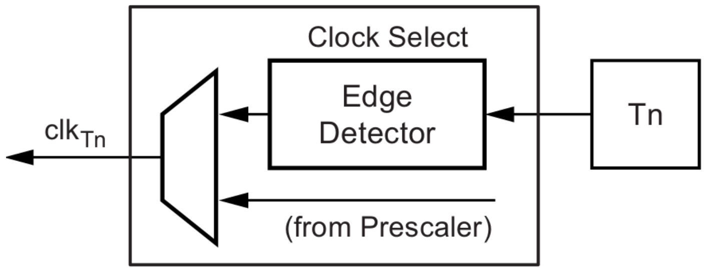
Register - 16 bit	Name
TCN10	Timer/Counter1count value
TCCR1A	Timer/Coutner1 Control Register A
TCCR1B	Timer/Coutner1 Control Register B
OCBR1A	Output compare register A
OCBR1B	Output compare register B
TIFR1	Timer Interrupt Flag Register
TIMSK1	Timer interrupt Mask Register
ICR1	Input Capture Register

Note:

- The **CNT1, OCR1A/B, ICR1** are 16-bit registers that can be accessed by the CPU via the 8-bit data bus.
- **For 16-bit write, the high byte must be written before the low byte.**
- **For 16-bit read, the low byte must be read before the high byte.**

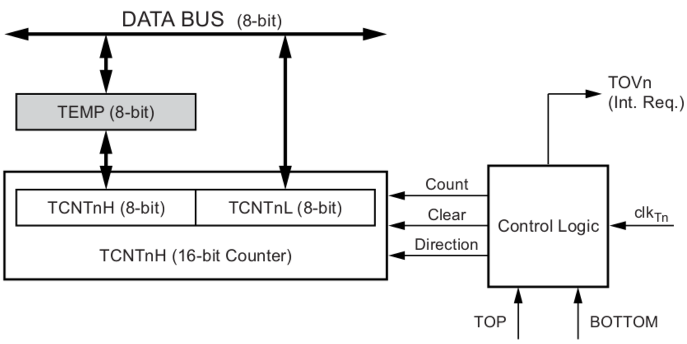
4 Timer/Counter1 Units

4.1 Clock Source/Select Unit



- The source for the Timer/Counter0 can be external or internal.
- External clock source is from *T1* pin.
- While Internal Clock source can be clocked via a prescalar.
- The output of this unit is the timer clock (clk_{T1}).
- It uses ***CS1[2:0]*** bits in **TCCR1B** register to select the source.

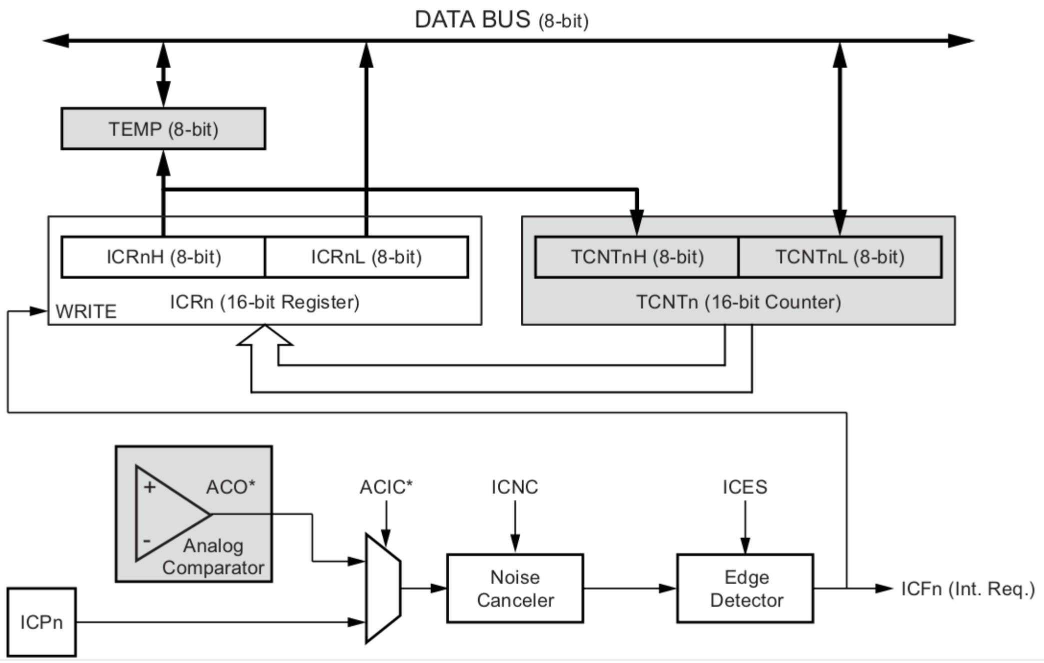
4.2 Counter Unit



Signal	Description
count	Increment or decrement **TCNT1** by 1
direction	Select between increment or decrement
clear	Clears **TCNT1** to 0x0000
clk_{T1}	Timer/Coutner1 clock
top	Signalize that **TCNT1** has maximum value
bottom	Signalize that **TCNT1** has minimum value(0x0000)

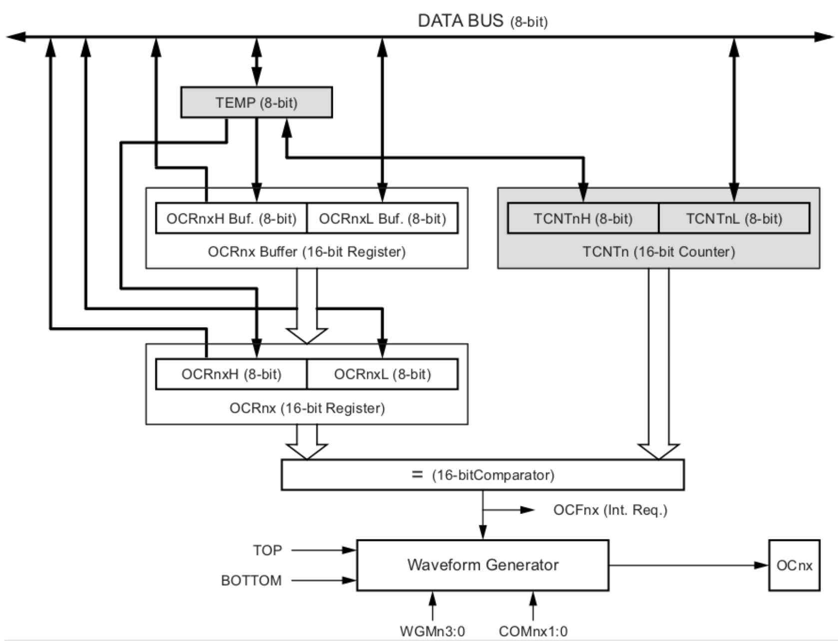
- The main part of the 16-bit Timer/Counter is the programmable bi-directional counter.
- Counter high (**TCNT1H**) containing the upper eight bits of the counter, and counter low (**TCNT1L**) containing the lower eight bits.
- Depending the mode of operation the counter is cleared, incremented, or decremented at each timer clock (clk_{T1}).
- Counting sequence is determined by ***WGM1[3:0]*** bits of **TCCR1A** -Timer/Counter1 Control register A and **TCCR1B** - Timer/Counter1 Control register B.
- The Timer/Counter1 Overflow flag ***(TOV1)*** is set and can generate interrupt according to the mode.

4.3 Input Capture Unit



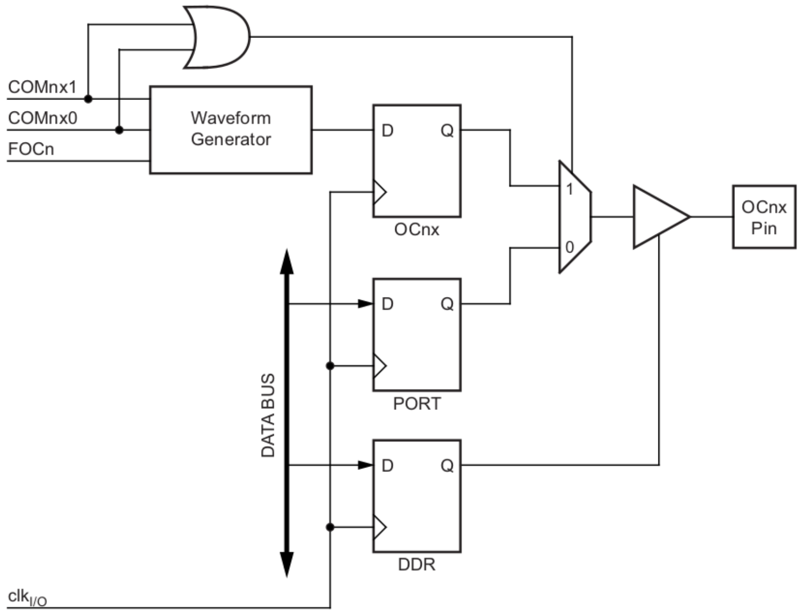
- Can capture external events and give them time-stamp indicating time of occurance.
- External signal can be from *ICP1* pin or analog-comparator unit.
- Usage : calculate frequency, duty-cycle, log of the signal
- When a change of the logic level (an event) occurs on the input capture pin (*ICP1*), or 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.
- ***ICF1*** flag is automatically cleared when the interrupt is executed and by writing on to i.
- An input capture can be triggered by software by controlling the port of the *ICP1* pin.

4.4 Output Compare Unit



- 16-bit comparator continuously compares **TCNT1** with both **OCR1A** and **OCR1B**.
- When **TCNT1** equals **OCR1A** or **OCR1B**, the comparator signals a match which will set the output compare flag at the next timer clock cycle.
- If interrupts are enabled, then output compare interrupt is generated.
- The waveform generator uses the match signal to generate an output according to operating mode set by the ***WGM1[3:0]*** bits and compare output mode ***COM0x[1:0]*** bits.

4.5 Compare Match Output Unit



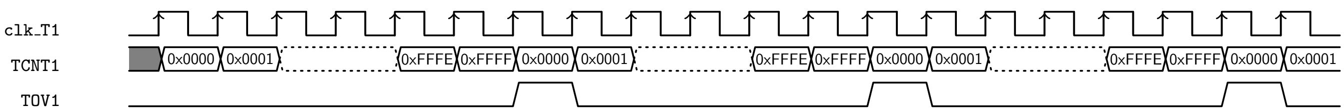
- This unit is used for changing the state of *OC1A* and *OC1B* pins by configuring the ***COM1x[1:0]*** bits.
- But, general I/O port function is overriiden by DDR reigster.

5 Modes of Operation

- The mode of operation can be defined by combination of waveform generation mode (***WGM1[3:0]***) and compare output mode(***COM1[1:0]***) bits.
- The waveform generation mode (***WGM1[3:0]***) bits affect the counting sequence.
- For non-PWM mode, ***COM1[1:0]*** bits control if the output should be set, cleared or toggled at a compare match.
- For PWM mode, ***COM1[1:0]*** bits control if the PWM generated should be inverted or non-inverted.

5.1 Normal Mode - Non-PWM Mode

- ***WGM1[3:0]*** $-->$ 000.
- Counter counts up and no counter clear.
- Overruns TOP(0XFFFF) and restarts from BOTTOM(0X0000).
- ***TOV1*** Flag is only set when overrun.
- We have to clear ***TOV1*** flag inorder to have next running.
- But, if we use interrupt we don't need to clear it as interrupt automatically clear the ***TOV1*** flag.
- The input capture unit can be used to capture events at *ICP1* pin or *ACO* pin.
- The timing can be seen below.



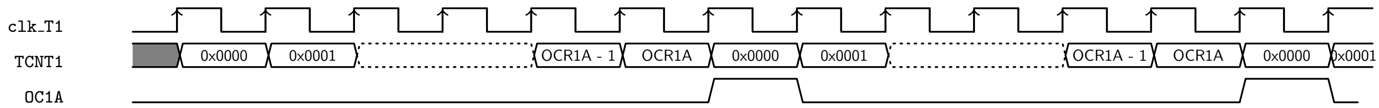
5.2 Clear Timer on Compare Match(CTC) Mode - Non-PWM Mode

- ***WGM1[3:0]*** $-->$ 0100 or 1100.
  - Counter value clears when **TCNT1** reaches **OCR1A** if ***WGM1[3:0]*** is 0100.
  - Counter value clears when **TCNT1** reaches **ICR1** if ***WGM1[3:0]*** is 1100.
- Interrupt can be generated each time **TCNT1** reaches **OCR1A** register value by ***OCF1A*** flag.
- Interrupt can be generated each time **TCNT1** reaches **ICR1** register value by ***ICF1*** flag.
- When ***COM1A[1:0]*** == 01, the *OC1A* pin output can be set to toggle its match between **TCNT1** and **OCR1A** or **ICR1** register to generate waveform.
- The frequency of the waveform its

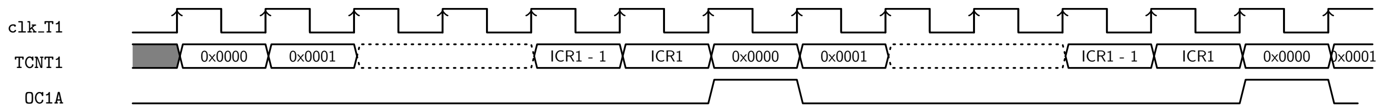
$$f_{OC1A} = \frac{f_{clkT1}}{2*N*(1+OCR1A)}$$

- Here N is prescalar factor and can be (1, 8, 64, 256, or 1024).

5.2.1 WGM1[3:0] == 0100



5.2.2 WGM1[3:0] == 1100

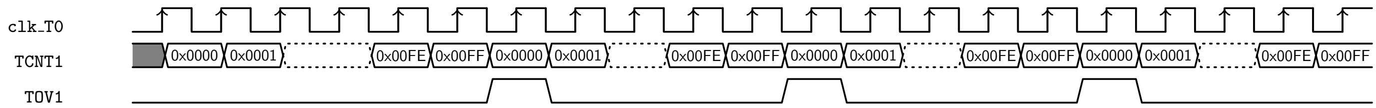


5.3 Fast PWM Mode

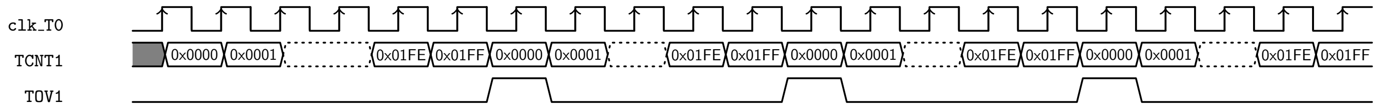
- ***WGM1[3:0]*** $-->$ 0101 or 0110 or 0111 or 1110 or 1111.
- Power Regulation, Rectification, DAC applications.
- Single slope operations causing high frequency PWM waveform.
- Counter starts from BOTTOM to TOP and then restarts from BOTTOM.
- TOP is defined by
  - TOP == 0x00FF if ***WGM1[3:0]*** $-->$ 0101
  - TOP == 0x01FF if ***WGM1[3:0]*** $-->$ 0110
  - TOP == 0x03FF if ***WGM1[3:0]*** $-->$ 0111
  - TOP == **ICR1** if ***WGM1[3:0]*** $-->$ 1110
  - TOP == **OCR1A** if ***WGM1[3:0]*** $-->$ 1111
- When ***COM1A[1:0]*** == 01, the *OC1A* pin output can be set to toggle its match between **TCNT1** and TOP to generate waveform.
  - The above is possible only when ***WGM12*** bit is set.
  - And only on *OC1A* pin and not on *OC1B* pin.
- In Inverting Compare Mode ***COM1A[1:0]*** == 10 , the *OC1A* or *OC1B* pins is made 1 on compare match between **TCNT1** and TOP and made 0 on reaching BOTTOM.
- In Non-Inverting Compare Mode ***COM1A[1:0]*** == 11 , the *OC1A* or *OC1B* pins is made 0 on compare match between **TCNT1** and TOP and 1 made on reaching BOTTOM.
- The Timer/Counter overflow flag (***TOV1***) is set each time the counter reaches TOP.
- The PWM frequency is given by

$$f_{OC1xPWM} = \frac{f_{clkT1}}{N*(1+TOP)}$$

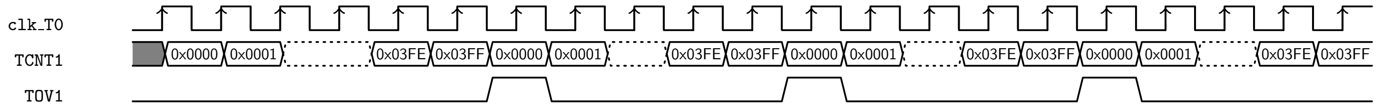
5.3.1 WGM1[3:0] == 0101



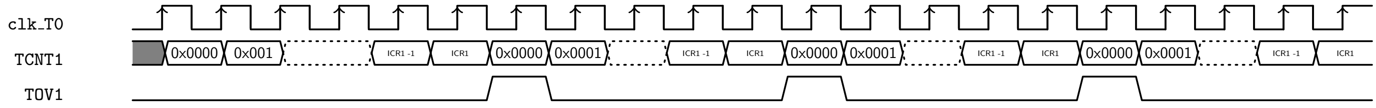
5.3.2 WGM1[3:0] == 0110



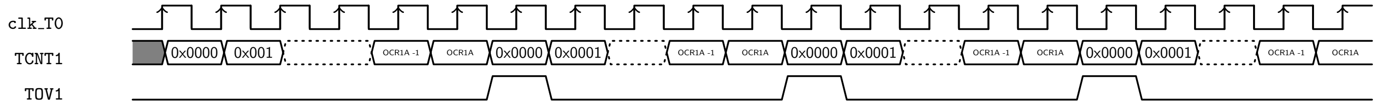
5.3.3 WGM1[3:0] == 0111



5.3.4 WGM1[3:0] == 1110



5.3.5 WGM1[3:0] == 1111

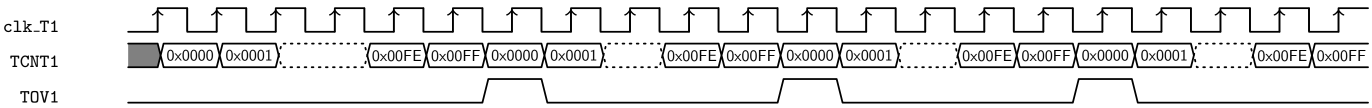


5.4 Phase Correct PWM Mode

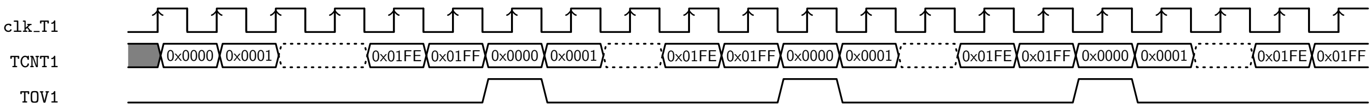
- ***WGM1[3:0]*** −− > 0001 or 0010 or 0011 or 1010 or 1011.
- High resolution phase correct PWM.
- Motor control due to symmetric features
- Dual slope operations causing ower frequency PWM waveform.
- Counter starts from BOTTOM to TOP and then from TOP to BOTTOM.
- TOP is defined by
  - TOP == 0x00FF if ***WGM1[3:0]*** −− > 0001
  - TOP == 0x01FF if ***WGM1[3:0]*** −− > 0010
  - TOP == 0x03FF if ***WGM1[3:0]*** −− > 0011
  - TOP == **ICR1** if ***WGM1[3:0]*** −− > 1010
  - TOP == **OCR1A** if ***WGM1[3:0]*** −− > 1011
- When ***COM1A[1:0]*** == 01, the *OC1A* pin output can be set to toggle its match between **TCNT1** and TOP to generate waveform.
  - The above is possible only when ***WGM12*** bit is set.
  - And only on *OC1A* pin and not on *OC1B* pin.
- In Inverting Compare Mode ***COM1A[1:0]*** == 10 , the *OC1A* or *OC1B* pins is made 1 on compare match between **TCNT1** and TOP and made 0 on reaching BOTTOM.
- In Non-Inverting Compare Mode ***COM1A[1:0]*** == 11 , the *OC1A* or *OC1B* pins is made 0 on compare match between **TCNT1** and TOP and 1 made on reaching BOTTOM.
- The Timer/Counter overflow flag (***TOV1***) is set each time the counter reaches BOTTOM..
- The PWM frequency is given by

$$f_{OC1xPWM} = \frac{f_{clkT1}}{2*N*TOP}$$

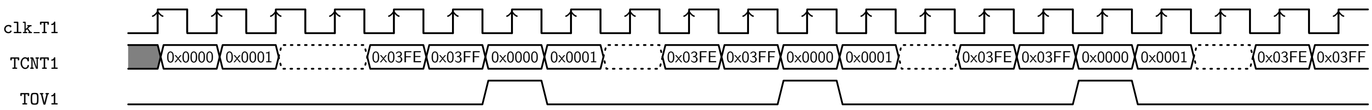
5.4.1 WGM1[3:0] == 0001



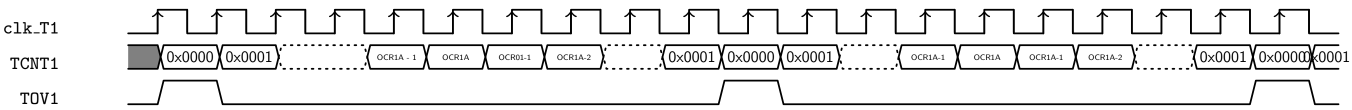
5.4.2 WGM1[3:0] == 0010



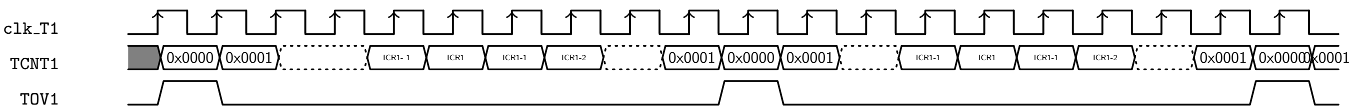
5.4.3 WGM1[3:0] == 0011



5.4.4 WGM[2:0] == 1010



5.4.5 WGM[2:0] == 1011



5.5 Phase and Frequency Corrected PWM Mode

- ***WGM1[3:0]*** $-->$ 1000 or 1001.
- High resolution and Phase correctd PWM.
- Dual-Slope.
- Counter counts from BOTTOM to TOP and then from TOP to BOTTOM.
  - TOP == **OCR1A** if ***WGM1[3:0]*** $-->$ 1001
  - TOP == **ICR1** if ***WGM1[3:0]*** $-->$ 1000
- In Inverting Compare Mode ***COM1x[1:0]*** == 10 the *OC0x* pins is made 1 on compare match between **TCNT1** and TOP when upcounting and made 0 on compare match between **TCNT1** and TOP when downcounting.
- In Non-Inverting Compare Mode ***COM1x[1:0]*** == 11, the *OC0x* pins is made 0 on compare match between **TCNT1** and TOP when upcounting AND made 1 on compare match between **TCNT1** and TOP when downcounting.
- The Timer/Counter overflow flag (***TOV1***) is set each time the counter reaches BOTTOM.
- The interrupt flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.
- The PWM frequency is given by

$$f_{OC1xPWM} = \frac{f_{clkT1}}{2*N*TOP}$$

6 Register Description

TCCR1A – Timer/Counter 1 Control Register A

7	6	5	4	3	2	1	0
COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10

COM1x[1:0]	**Non-PWM modes**	**Fast PWM**	**Phase Corrected PWM & Phase and Frequency Corrected PWM**
00	No output @ *PB1 - OC1A* or *PB2 - OC1B* pin	No output @ *PB1 - OC1A* or *PB2 - OC1B* pin	No output @ *PB1 - OC1A* or *PB2 - OC1B* pin
01	Toggle *PB1 - OC1A* or *PB2 - OC1B* pin on compare Match.	When ***WGM[3:0]*** == 1110 or 1111, Toggle *OC1A* pin on compare match	When ***WGM[3:0]*** == 1110 or 1111, Toggle *OC1A* pin on comapre match.
10	Clear *PB1 - OC1A* or *PB2 - OC1B* pin on compare Match.	Clear *PB1 - OC1A* or *PB2 - OC1B* on compare match and set *PB1 - OC1A* or *PB2 - OC1B* at BOTTOM	Clear *PD5 - OC0B* on compare match when up-counting and set *PB1 - OC1A* or *PB2 - OC1B* on compare match when down-counting.
11	Set *PB1 - OC1A* or *PB2 - OC1B* pin on compare Match.	Set *PB1 - OC1A* or *PB2 - OC1B* on compare match and clear *PB1 - OC1A* or *PB2 - OC1B* at BOTTOM	Set *PD5 - OC0B* on compare match when up-counting and clear *PB1 - OC1A* or *PB2 - OC1B* on compare match when down-counting.

WGM1[3:0]	**Mode of operation**	**TOP**	**TOV1 Flag set on**
0000	Normal	0xFFFF	MAX
0001	PWM Phase corrected – 8bit	0x00FF	BOTTOM
0010	PWM Phase corrected – 9bit	0x01FF	BOTTOM
0011	PWM Phase corrected – 10bit	0x03FF	BOTTOM
0100	CTC	OCR1A	MAX
0101	Fast PWM – 8bit	0x00FF	TOP
0110	Fast PWM – 9bit	0x01FF	TOP
0111	Fast PWM – 10bit	0x03FF	TOP
1000	PWM, phase and frequency corrected	ICR1	BOTTOM
1001	PWM, phase and frequency corrected	OCR1A	BOTTOM
1010	PWM, phase corrected	ICR1	BOTTOM
1011	PWM, phase corrected	OCR1A	BOTTOM
1100	CTC	ICR1	MAX
1110	Fast PWM	ICR1	TOP
1111	Fast PWM	OCR1A	TOP

TCCR1B – Timer/Counter1 Control Register B

7	6	5	4	3	2	1	0
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

- ***ICNC1 - Input Capture Noise Canceler*** - activates the input capture noise canceler.
- ***ICES1 - Input Capture Edge Select*** - selects which edge on the input capture pin (*ICP1*) that is used to trigger a capture event. [1 - Rising edge; 0 - falling edge;]

CS1[2:0]	Description(Prescalar)
000	No clock source(Timer/Counter Stopped)
001	$clk_{I/O}$ – no prescaling
010	$\frac{clk_{I/O}}{8}$
011	$\frac{clk_{I/O}}{64}$
100	$\frac{clk_{I/O}}{256}$
101	$\frac{clk_{I/O}}{1024}$
110	External clock source on *T1* pin. Clock on falling edge.
111	External clock source on *T1* pin. Clock on rising edge.

TCNT1H – Timer/Counter1 Counter Higher Byte

7	6	5	4	3	2	1	0
TCNT1[15:8]							

TCNT1L – Timer/Counter1 Counter Lower Byte

7	6	5	4	3	2	1	0
TCNT1[7:0]							

OCR1AH – Output Compare Register 1 A Higher Byte

7	6	5	4	3	2	1	0
OCR1A[15:8]							

OCR1AL – Output Compare Register 1 A Lower Byte

7	6	5	4	3	2	1	0
OCR1A[7:0]							

OCR1BH – Output Compare Register 1 B Higher Byte

7	6	5	4	3	2	1	0
OCR1B[15:8]							

OCR1BL – Output Compare Register 1 B Lower Byte

7	6	5	4	3	2	1	0
OCR1B[7:0]							

ICR1H – Input Capture Register 1 Higher Byte

7	6	5	4	3	2	1	0
ICR1[15:8]							

ICR1L – Input Capture Register 1 Lower Byte

7	6	5	4	3	2	1	0
ICR1[7:0]							

TIMSK1 – Timer/Counter 1 Interrupt Mask Register

7	6	5	4	3	2	1	0
-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1

Enable interrupts for compare match between **TCNT1** and **OCR1A** or **TCNT1** and **OCR1B** or overflow in **TCNT1** or Input capture interrupt enable.

TIFR1 – Timer/Counter 1 Interrupt Flag Register

7	6	5	4	3	2	1	0
-	-	ICF1	-	-	OCIE1B	OCIE1A	TOIE1

Flag registers for interrupts on compare match between **TCNT0** and **OCR0A** or **TCNT0** and **OCR0B** or overflow in **TCNT1** or capture event occurs on the *ICP1* pin .

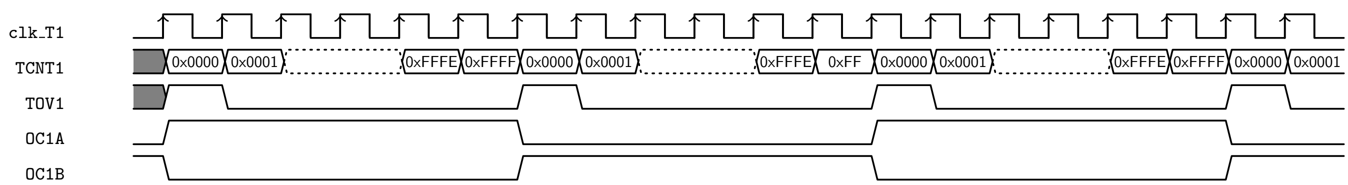
7 Configuring the Timer/Counter

7.1 Normal Mode

7.1.1 As Timer

$$ON\_TIME = \frac{max\_count}{\frac{F\_CPU}{PRESCALAR}}$$

- Depending on PRESCALR value, we get different ON_TIME.
- First, ***WGM1[3:0]*** bits are configured as 0000 for Normal Mode in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1A[1:0]*** and/or ***COM1A[1:0]*** bits are configured to make outputs *OC1A* and/or *OC1B* pins to do nothing, set, clear or toggle in **TCCR1A** register.
- Next, Interrupt is Enabled by ***TOIE1*** (overflow enable) in **TIMSK1** reigster.
- Finally, Timer is started by setting prescalar in ***CS1[2:0]*** bits as needed prescalar of **TCR1B** reigster.
- Global Interrupt is enabled.
- A interrupt Service Routine for Timer1 overflow is Written.
- No need to clear the overflow flag as it is done by hardware.
- The timing when both pins *OC1A* and *OC1B* are made to toggle.



- The code can be seen below,

```
// MOde of operation to Normal Mode -- WGM1[3:0] === 0000
// WGM1[3](bit4) from TCCR1B, WGM1[2](bit3) from TCCR1B, WGM1[1](bit1)  from TCC1RA, WGM1[0](bit0)
↪  from TCCR1A
TCCR1A = TCCR1A & ~(1<<WGM10);
TCCR1A = TCCR1A & ~(1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
```

```c
TCCR1B = TCCR1B & ~(1<<WGM13);

/* What to do when timer reaches the MAX(0xFFFF) value */
// toggle OC1A on each time when reaches the MAX(0xFFFF)
// which is reflected in PB1
// Output OC1A to toglle when reaches MAX -- COM1A[1:0] === 01
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A & ~(1<<COM1A1);
TCCR1A = TCCR1A | (1<<COM1A0);

// toggle OC1B on each time when reaches the MAX(0xFFFF)
// which is reflected in PB2
// Output OC1B to toglle when reaches MAX -- COM1B[:0] === 01
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A & ~(1<<COM1B1);
TCCR1A = TCCR1A | (1<<COM1B0);


//Enable Interrupt of OVERFLOW flag so that interrupt can be generated
TIMSK1 = TIMSK1 | (1<<TOV1);

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

// enabling global interrupt

sei();
// SO ON TIME = max_count / (F_CPU / PRESCALAR)
// ON TIME = 0xFFFF / (16000000/1) = 4.096ms
// since symmetric as toggling OFF TIME = 4.096ms
// hence, we get a square wave of fequency 1 / 8.192ms = 122.07Hz
```

7.1.2 As Counter

- Every rising/falling edge the count increases.
- So to reach 0xFFFF count, it would take a time of $\frac{0xFFFF}{frequency@T1pin}$.
- First, ***WGM1[3:0]*** bits are configured as 0000 for Normal Mode in **TCCR1A** and **TCCR1B** registers.
- Finally, Counter is started by configuring ***CS1[2:0]*** bits to 110 or 111 for external falling or rising edge on *T1 - PD5*.
- The code when *T1* pin is used as counter @ falling edge.

```c
// MOde of operation to Normal Mode -- WGM1[3:0] === 0000
// WGM1[3](bit4) from TCCR1B, WGM1[2](bit3) from TCCR1B, WGM1[1](bit1)  from TCC1RA, WGM1[0](bit0)
↪  from TCCR1A
TCCR1A = TCCR1A & ~(1<<WGM10);
TCCR1A = TCCR1A & ~(1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

/* to count external event -we must connect source to T1 (PD5) */
// THE CLK IS CLOCKED FROM external source
// Falling edge of T1(PD5) -- CS1[2:0] === 110
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B & ~(1<<CS10);
```

```
TCCR1B = TCCR1B | (1<<CS11);
TCCR1B = TCCR1B | (1<<CS12);
```

7.1.3 As Input Capture

- Capture the value of **TCNT1** into **ICR1** register when there is rising or falling edge.
- First, ***WGM1[3:0]*** bits are configured as 0000 for Normal Mode in **TCCR1A** and **TCCR1B** registers.
- Next, the falling or rising edge for the *ICP1* pin is selected by ***ICES1*** bit in **TCCR1B**.
- The interrupts for input capture is enabled by setting the ***ICIE1*** bit in **TIMSK1**.
- A interrupt service routing is written.
- Finally, Timer is started by setting prescalar in *CS1[2:0]* bits as needed prescalar of **TCR1B** reigster.
- The code when *ICP1 - PB0* pin is used as capture @ rising edge.

```
// MOde of operation to Normal Mode -- WGM1[3:0] === 0000
// WGM1[3](bit4) from TCCR1B, WGM1[2](bit3) from TCCR1B, WGM1[1](bit1)  from TCC1RA, WGM1[0](bit0)
↪  from TCCR1A
TCCR1A = TCCR1A & ~(1<<WGM10);
TCCR1A = TCCR1A & ~(1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

// Select the edge for Input Capture
// ICES1(bit6) from TCCR1B
// Capture on Rising edge, ICES1 === 1
TCCR1B |= (1<<ICES1);

//Enable Interrupt of Input Capture Interrupt Enable so that interrupt can be generated
TIMSK1 = TIMSK1 | (1<<ICIE1);

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

// enabling global interrupt

sei();

ISR(TIMER1_CAPT_vect)
{
        if((TIFR1 & (1<<ICF1)) != 0)
        {
                capVal = ICR1L;
                capVal = (ICR1H<<8) | (capVal & 0xFF);
                // see datamemory
        }
}
```

7.1.4 Application I - Delay

```
/* TCNT1 starts from 0X0000 goes upto 0XFFFF and restarts */
/* No possible use case as it just goes upto 0xFFFF and restarts */
// MOde of operation to Normal Mode -- WGM1[3:0] === 0000
```

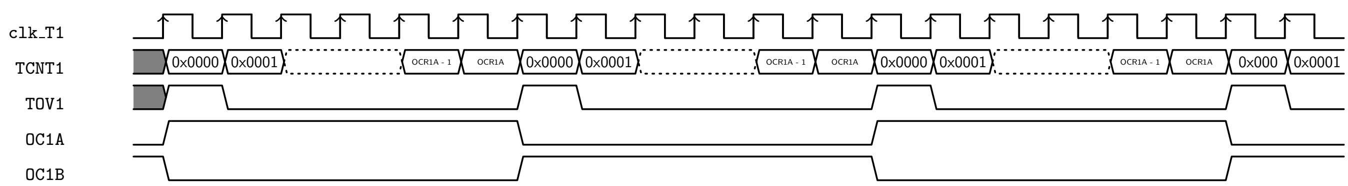
```c
// WGM1[3](bit4) from TCCR1B, WGM1[2](bit3) from TCCR1B, WGM1[1](bit1)  from TCC1RA, WGM1[0](bit0)
↪  from TCCR1A
TCCR1A = TCCR1A & ~(1<<WGM10);
TCCR1A = TCCR1A & ~(1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

/* What to do when timer reaches the MAX(0xFFFF) value */
// nothing should be done on OC1A for delay
// nothing  -- COM1A[1:0] === 00
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A & ~(1<<COM1A1);
TCCR1A = TCCR1A & ~(1<<COM1A0);

/* The delay possible = 0xffff / (F_CPU/prescalar) */
// lowest delay = 0xffff / (16000000 / 1) = 4.096ms
// when prescalar == 8 --> delay = 0xffff / (16000000 / 8) = 32.768ms
// when prescalar == 64 --> delay = 0xffff / (16000000 / 64) = 262.144ms
// when prescalar == 256 --> delay = 0xffff / (16000000 / 256) = 1.048576s
// highest delay possible = 0xffff / (16000000 / 1024) = 4.194304s

// start timer by setting the clock prescalar
// divede by 64 from I/O clock
// divede by 64-- CS1[2:0] === 101
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B | (1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);


// actual delaying - wait until delay happens
while((TIFR1 & 0x01) == 0x00); // checking overflow flag when overflow happns
// clearing the overflag so that we can further utilize
TIFR1 = TIFR1 | 0x01;
```

7.2 CTC Mode

7.2.1 As Timer

$$ON\_TIME = \frac{1+OCR1A}{\frac{F\_CPU}{PRESCALAR}}$$

- Depending on **OCR1A** register and/or **ICR1** register and PRESCALR value, we get different ON_TIME.
- First, ***WGM1[3:0]*** bits are configured as 0100 or 1100 for CTC Mode in **TCCR2A** and **TCCR1B** registers.
- Next, ***COM1A[1:0]*** and/or ***COM1B[1:0]*** bits are configured to make outputs *OC1A* and/or *OC1B* pins to do nothing, set, clear or toggle in **TCCR0A** register.
- Next, Interrupt is Enabled by ***OCIE1A*** (output compare on match on **OCR1A** register enable) in **TIMSK1** reigster.
- Finally, Timer is started by setting prescalar in ***CS1[2:0]*** bits as needed prescalar of **TCR1B** reigster.
- Global Interrupt is enabled.
- A interrupt Service Routine for Timer1 compare is Written.
- No need to clear the overflow flag as it is done by hardware.
- The timing when both pins *OC1n* are made to toggle.



- The code can be seen below,

```
// MOde of operation to Normal Mode -- WGM1[3:0] === 0100(TOP = OCR1A) or 1100(TOP = ICR1)
// WGM1[3](bit4) from TCCR1B, WGM1[2](bit3) from TCCR1B, WGM1[1](bit1)  from TCC1RA, WGM1[0](bit0)
↪  from TCCR1A
// we take TOP to be OCR1A for custom frequency
TCCR1A = TCCR1A & ~(1<<WGM10);
TCCR1A = TCCR1A & ~(1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

/* What to do when timer reaches the OCR1A value */
// toggle OC1A on each time when reaches the OCR1A
// which is reflected in PB1
// Output OC1A to toglle when reaches OCR1A -- COM1A[1:0] === 01
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1A0);
TCCR1A = TCCR1A & ~(1<<COM1A1);

// toggle OC1B on each time when reaches the OCR1A
// which is reflected in PB2
// Output OC1B to toglle when reaches OCR1A -- COM1B[1:0] === 01
// COM1B[1](bi57) from TCCR1A, COM1B[0](bit64) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1B0);
TCCR1A = TCCR1A & ~(1<<COM1B1);

// Enable Interrupt when counter matches OCR1A Rgister
//  OCIE1A  bit is enabled
TIMSK1 = TIMSK1 | (1<<OCIE1A);

// setting the value till the counter should reach in OCR1A
// for toggling of OC1A pin
OCR1A = 0x4861;

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

// enabling global interrupt

sei();
// SO ON TIME = (1 + OCR1A) / (F_CPU / PRESCALAR)
// ON TIME = 0x4861 / (16000000/1) = 1.15ms
// since symmetric as toggling OFF TIME = 1.15ms
// hence, we get a square wave of fequency 1 / 2.31ms = 431Hz
```

```
ISR(TIMER1_COMPA_vect)
{
    // do the thing when overflows.
}
```

7.2.2 As Counter

- Every rising/falling edge the count increases.
- So to reach required count, it would take a time of $\frac{OCR1A}{frequency@T1_{pin}}$.
- First, ***WGM1[3:0]*** bits are configured as 0100 or 1100 for CTC Mode in **TCCR2A** and **TCCR1B** registers.

- Finally, Counter is started by configuring *CS1[2:0]* bits to 110 or 111 for external falling or rising edge on *T1 - PD5* pin.
- The code when *T1* pin is used as counter @ falling edge.

```
// MOde of operation to Normal Mode -- WGM1[3:0] === 0100(TOP = OCR1A) or 1100(TOP = ICR1)
// WGM1[3](bit4) from TCCR1B, WGM1[2](bit3) from TCCR1B, WGM1[1](bit1)  from TCC1RA, WGM1[0](bit0)
↪  from TCCR1A
TCCR1A = TCCR1A & ~(1<<WGM10);
TCCR1A = TCCR1A & ~(1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

/* What to do when timer reaches the OCR1A value */
// toggle OC1A on each time when reaches the OCR1A
// which is reflected in PB1
// Output OC1A to toglle when reaches OCR1A -- COM1A[1:0] === 01
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1A0);
TCCR1A = TCCR1A & ~(1<<COM1A1);


//we count till OCR1A register value and toggle
// lets' count 10 pulses
OCR1A = 0x000a;

/* to count external event -we must connect source to T1 (PD5) */
// THE CLK IS CLOCKED FROM external source
// Falling edge of T1(PD5) -- CS1[2:0] === 110
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B & ~(1<<CS10);
TCCR1B = TCCR1B | (1<<CS11);
TCCR1B = TCCR1B | (1<<CS12);


// since for every rising edge the count increase
// so to reach 10 count, it would take 0xa / (frequency of input at T1 pin or PD5)
// we wave used 5kHz so it would take ==> 2ms to toggle as we have made OC1A toggle when overflows
↪  (by setting COMA[1:0])
// also we canuse TCNT1 as edge counter
```

7.3 Application I - Delay

```
// minimum delay being 4us -- choose like that - because, of the the delay for execution, - we get
↪  us if we use toggling of pins OC1A or OC1B
// use PRESCALAR OF 1 -- 4us - 4.096ms -- usage 4us - 4ms -- factor=0 -- CS1[2:0]=1
// use PRESCALAR OF 8 -- 4us - 32.768ms -- usage 5ms - 32ms -- factor=3 -- CS1[2:0]=2
// use PRESCALAR OF 64 -- 4us - 262.144ms -- usage 33ms - 260ms -- factor=6 -- CS0[2:0]=3
// use PRESCALAR OF 256 -- 16us - 1.048s -- usage 261ms - 1.048s -- factor=8 -- CS0[2:0]=4


/* TCNT1 starts from 0X0000 goes upto OCR1A or ICR1 and restarts */
// MOde of operation to Normal Mode -- WGM1[3:0] === 0100(TOP = OCR1A) or 1100(TOP = ICR1)
// WGM1[3](bit4) from TCCR1B, WGM1[2](bit3) from TCCR1B, WGM1[1](bit1)  from TCC1RA, WGM1[0](bit0)
↪  from TCCR1A
// we take TOP to be OCR1A for custom frequency
TCCR1A = TCCR1A & ~(1<<WGM10);
TCCR1A = TCCR1A & ~(1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

/* What to do when timer reaches the MAX(0xFFFF) value */
```

```
// nothing should be done on OC1A for delay
// nothing  -- COM1A[1:0] === 00
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A & ~(1<<COM1A1);
TCCR1A = TCCR1A & ~(1<<COM1A0);



if(delay_in_us <=3)
{
    // if delay_in_us <= 3us -- so we stop clock

    OCR1A = 0;
    // stop clcok
    // stop clcok-- CS1[2:0] === 000
    // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
    TCCR1B = TCCR1B & ~(1<<CS10);
    TCCR1B = TCCR1B & ~(1<<CS11);
    TCCR1B = TCCR1B & ~(1<<CS12);
}
else if((3 < delay_in_us)  && (delay_in_us <= 4000))
{
    OCR1A = ((delay_in_us * 16) >> 0) - 1;
    // start timer by setting the clock prescalar
    // SAME AS from I/O clock
    // same-- CS1[2:0] === 001
    // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
    TCCR1B = TCCR1B | (1<<CS10);
    TCCR1B = TCCR1B & ~(1<<CS11);
    TCCR1B = TCCR1B & ~(1<<CS12);
}
else if((4000 < delay_in_us)  && (delay_in_us <= 32000))
{
    OCR1A = ((delay_in_us * 16) >> 3) - 1;
    // start timer by setting the clock prescalar
    // divide by 8 from I/O clock
    // divide by 8 CS1[2:0] === 010
    // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
    TCCR1B = TCCR1B & ~(1<<CS10);
    TCCR1B = TCCR1B | (1<<CS11);
    TCCR1B = TCCR1B & ~(1<<CS12);
}
else if((32000 < delay_in_us)  && (delay_in_us <= 260000))
{
    OCR1A = ((delay_in_us * 16) >> 6) - 1;
    // start timer by setting the clock prescalar
    // divide by 64 from I/O clock
    // divide by 64 CS1[2:0] === 011
    // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
    TCCR1B = TCCR1B | (1<<CS10);
    TCCR1B = TCCR1B | (1<<CS11);
    TCCR1B = TCCR1B & ~(1<<CS12);
}
else if((260000 < delay_in_us)  && (delay_in_us <= 1000000))
{
    OCR1A = ((delay_in_us * 16) >> 8) - 1;
    // start timer by setting the clock prescalar
    // divide by 256 from I/O clock
    // divide by 256 CS1[2:0] === 100
    // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
    TCCR1B = TCCR1B & ~(1<<CS10);
    TCCR1B = TCCR1B & ~(1<<CS11);
    TCCR1B = TCCR1B | (1<<CS12);
```

```
}
else if(delay_in_us > 1000000)
{
    Timer1_asDelayIn_us(delay_in_us - 1000000);
    OCR1A = ((1000000 * 16) >> 8) - 1;
    // start timer by setting the clock prescalar
    // divide by 256 from I/O clock
    //divide by 256 CS1[2:0] === 100
    // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
    TCCR1B = TCCR1B & ~(1<<CS10);
    TCCR1B = TCCR1B & ~(1<<CS11);
    TCCR1B = TCCR1B | (1<<CS12);
}

// actual delaying - wait until delay happens
while((TIFR1 & 0x02) == 0x00); // checking OCF1A (compare match flag A) flag when match happns
// clearing the compare match flag so that we can further utilize
TIFR1 = TIFR1 | 0x02;
```

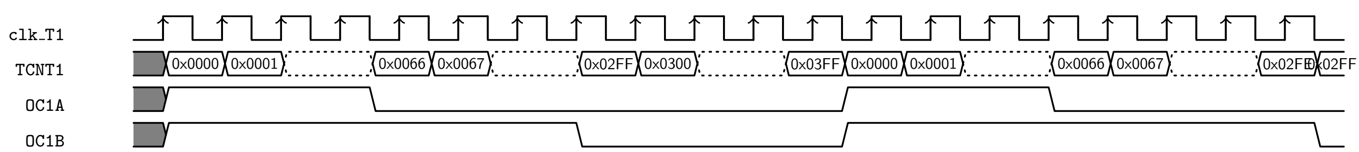
7.4 Fast PWM Mode

```
ISR(TIMER1_OVF_vect)
{
}
ISR(TIMER1_COMPA_vect)
{
}
ISR(TIMER1_COMPB_vect)
{
}
```

7.4.1 Non-Inverting PWM with TOP at MAX(0x00FF or 0x01FF or 0x03FF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR1A** and/or **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 0101 or 0110 or 0111 for Fast PWM Mode with TOP at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1A[1:0]*** and/or ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make outputs *OC1A* and/or *OC01* pins to generate PWM by comparing between **OCR1A** and/or **OCR1B** respectively. That is for Non-Inverting, ***COM1x[1:0]*** is written 10.
- Next, the duty cycle value is loaded into **OCR1A** and/or **OCR1B** register for *OC1A* and/or *OC1B* pins.
- Also, the ***OCIE1A*** and/or ***OCIE1B*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match and/or overflow.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 10% duty cycle *OC1A* and 75% duty cycle*OC1B* pins are shown assuming .
  - WGM1[3:0] === 0111 – TOP equals 0x03FF
  - 0x66 for OCR1A.
  - 0x2FF for OCR1B.



```
/* TCNT1 starts from 0X0000 goes upto TOP and restarts from 0X00*/
/* Mode of operation:
    WGM1[3:0] --> 0101 --        TOP--> 0X00FF
    WGM1[3:0] --> 0110 --        TOP--> 0x01FF
    WGM1[3:0] --> 0111 --        TOP--> 0x03FF
    WGM1[3:0] --> 1110 --        TOP--> ICR1
    WGM1[3:0] --> 1111 --        TOP--> OCR1A
*/
// we take 0x03FF for fixed frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 0111 for OCR1A as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

// here we set COM0A[1:0] as 10 for non-inverting
// here we set COM0B[1:0] as 10 for non-inverting

// which is reflected in PD6
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1A1);
TCCR1A = TCCR1A & ~(1<<COM1A0);

// which is reflected in PD65
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1B1);
TCCR1A = TCCR1A & ~(1<<COM1B0);

// Enable Interrupt when TOV1 overflows TOP - here 0x03FF
//  TOIE1 bit is enabled
TIMSK1 = TIMSK1 | (1<<TOIE1);

/* we use OCF1A flag - which is set at every time TCN0 reaches OCR1A */
TIMSK1 = TIMSK1 | (1<<OCIE1A);
/* we use OCF1B flag - which is set at every time TCN0 reaches OCR1B */
TIMSK1 = TIMSK1 | (1<<OCIE1B);


// Next we set values for OCR1A and OCR2B
// Since, TCNT1 goes till max(0x3FF), we can choose OCR1A and OCR1B to any value below max(0x03FF)
OCR1A = 102; // for 10% duty clcle
OCR1B = 767; // for 75% duty clcle


// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

//enabled global interrupt
sei();
```

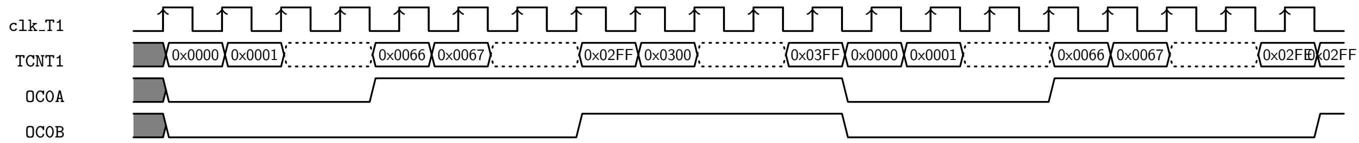
7.4.2 Inverting PWM with TOP at MAX(0x00FF or 0x01FF or 0x03FF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR1A** and/or **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 0101 or 0110 or 0111 for Fast PWM Mode with TOP at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1A[1:0]*** and/or ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make outputs *OC1A* and/or *OC01* pins to generate PWM by comparing between **OCR1A** and/or **OCR1B** respectively. That is for

Inverting, ***COM1x[1:0]*** is written 11.

- Next, the duty cycle value is loaded into **OCR1A** and/or **OCR1B** register for *OC1A* and/or *OC1B* pins.
- Also, the ***OCIE0A*** and/or ***OCIE0B*** bits of **TIMSK0** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match and/or overflow.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 10% duty cycle *OC1A* and 75% duty cycle*OC1B* pins are shown assuming .
  - WGM1[3:0] === 0111 – TOP equals 0x03FF
  - 0x66 for OCR1A.
  - 0x2FF for OCR1B.



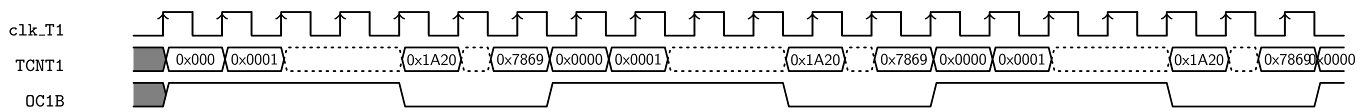

```
/* TCNT1 starts from 0X0000 goes upto TOP and restarts from 0X00*/
/* Mode of operation:
    WGM1[3:0] --> 0101 --        TOP--> 0X00FF
    WGM1[3:0] --> 0110 --        TOP--> 0x01FF
    WGM1[3:0] --> 0111 --        TOP--> 0x03FF
    WGM1[3:0] --> 1110 --        TOP--> ICR1
    WGM1[3:0] --> 1111 --        TOP--> OCR1A
*/
// we take 0x03FF for fixed frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 0111 for OCR1A as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

// here we set COMOA[1:0] as 11 for inverting
// here we set COMOB[1:0] as 11 for inverting

// which is reflected in PD6
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1A1);
TCCR1A = TCCR1A | (1<<COM1A0);

// which is reflected in PD65
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1B1);
TCCR1A = TCCR1A | (1<<COM1B0);

// Enable Interrupt when TOV1 overflows TOP - here 0x03FF
//  TOIE1 bit is enabled
TIMSK1 = TIMSK1 | (1<<TOIE1);

/* we use OCF1A flag - which is set at every time TCN0 reaches OCR1A */
TIMSK1 = TIMSK1 | (1<<OCIE1A);
/* we use OCF1B flag - which is set at every time TCN0 reaches OCR1B */
TIMSK1 = TIMSK1 | (1<<OCIE1B);


// Next we set values for OCR1A and OCR2B
// Since, TCNT1 goes till max(0x3FF), we can choose OCR1A and OCR1B to any value below max(0x03FF)
OCR1A = 102; // for 10% duty clcle
OCR1B = 767; // for 75% duty clcle
```

```
// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

//enabled global interrupt
sei();
```

7.4.3 Non-Inverting PWM with TOP at OCR1A

Frequency is chosen by **OCR1A** and Duty cycle by **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 1110 or 1111 for Fast PWM Mode with **ICR1** or **OCR1A** at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make output *OC1B* pins to generate PWM by comparing between **TCNT1** and **OCR1B**. That is for Non-Inverting, ***COM1B[1:0]*** is written 10.
- The frequency of duty cycle is loaded into **OCR01A** register.
- Next, the duty cycle value is loaded into **OCR1B** register for *OC1B* bits.
- Also, the ***OCIE01B*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 37% duty cycle *OC1B* pins are shown assuming .
  - 0x7869 for OCR0A.
  - 0x1A20 for OCR0B.



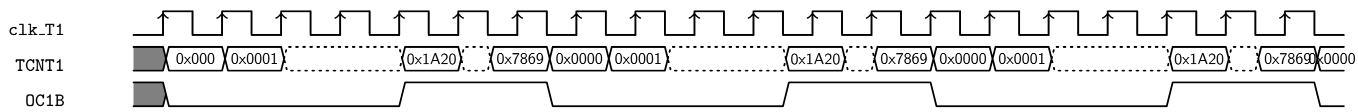

```
/* TCNT1 starts from 0X0000 goes upto TOP and restarts from 0X00*/
/* Mode of operation:
    WGM1[3:0] --> 0101 --        TOP--> 0X00FF
    WGM1[3:0] --> 0110 --        TOP--> 0x01FF
    WGM1[3:0] --> 0111 --        TOP--> 0x03FF
    WGM1[3:0] --> 1110 --        TOP--> ICR1
    WGM1[3:0] --> 1111 --        TOP--> OCR1A
*/
// we take OCR1A for custom frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 1111 for OCR1A as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
TCCR1B = TCCR1B | (1<<WGM13);


// for non-inverting on  OC1B we use 10 for and COM1B[1:0]
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A & ~(1<<COM1B0);
TCCR1A = TCCR1A | (1<<COM1B1);

// Next we set values for OCR1A and OCR1B
// Since, TCNT1 goes till OCR1A, we can choose OCR1B to any value below OCR1A
OCR1A = 0x7869; // for freqeuncy
```

```
OCR1B = 0x1A20; // for pwm duty cylc

// Enable interrupt when count reaches the overflow value
TIMSK1 |= (1<<TOV1);

// Enable interrupt when count reaches the OCR1B
TIMSK1 |= (1<<OCF1B);

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

//e enabel globalinterrupt
sei();
```

7.4.4 Inverting PWM with TOP at OCR1A

Frequency is chosen by **OCR1A** and Duty cycle by **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 1110 or 1111 for Fast PWM Mode with **ICR1** or **OCR1A** at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make output *OC1B* pins to generate PWM by comparing between **TCNT1** and **OCR1B**. That is for Inverting, ***COM1B[1:0]*** is written 11.
- The frequency of duty cycle is loaded into **OCR01A** register.
- Next, the duty cycle value is loaded into **OCR1B** register for *OC1B* bits.
- Also, the ***OCIE01B*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 37% duty cycle(0x60) *OC1B* pins are shown assuming .
  - 0x7869 for OCR0A.
  - 0x1A20 for OCR0B.



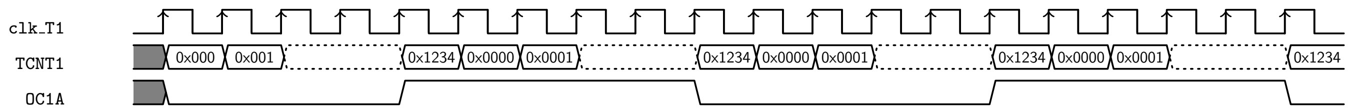

```
/* TCNT1 starts from 0X0000 goes upto TOP and restarts from 0X00*/
/* Mode of operation:
    WGM1[3:0] --> 0101 --         TOP--> 0X00FF
    WGM1[3:0] --> 0110 --         TOP--> 0x01FF
    WGM1[3:0] --> 0111 --         TOP--> 0x03FF
    WGM1[3:0] --> 1110 --         TOP--> ICR1
    WGM1[3:0] --> 1111 --         TOP--> OCR1A
*/
// we take OCR1A for custom frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 1111 for OCR1A as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
TCCR1B = TCCR1B | (1<<WGM13);


// for ninverting on  OC1B we use 11 for and COM1B[1:0]
```

```
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1B0);
TCCR1A = TCCR1A | (1<<COM1B1);

// Next we set values for OCR1A and OCR1B
// Since, TCNT1 goes till OCR1A, we can choose OCR1B to any value below OCR1A
OCR1A = 0x7869; // for freqeuncy
OCR1B = 0x1A20; // for pwm duty cylc

// Enable interrupt when count reaches the overflow value
TIMSK1 |= (1<<TOV1);

// Enable interrupt when count reaches the OCR1B
TIMSK1 |= (1<<OCF1B);

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

//e enabel globalinterrupt
sei();
```

7.4.5 Toggling mode square Wave

Frequency is chosen by **OCR1A** register.

- First, ***WGM1[3:0]*** bits are configured as 1111 for Fast PWM Mode with **OCR1A** at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1A[1:0]*** bits of **TCCR1A** register are configured to make output *OC1A* pins to generate PWM by comparing between **OCR1A** and **TCNT1**. That is for Toggling square wave ***COM1A[1:0]*** is written 01.
- The frequency of duty cycle is loaded into **OCR1A** register.
- Also, the ***OCIE1A*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for squared wave on *OC1A* pins are shown assuming.
  - 0x1234 for OCR1A.




```
/* TCNT1 starts from 0X0000 goes upto TOP and restarts from 0X00*/
/* Mode of operation:
WGM1[3:0] --> 0101 --        TOP--> 0X00FF
WGM1[3:0] --> 0110 --        TOP--> 0x01FF
WGM1[3:0] --> 0111 --        TOP--> 0x03FF
WGM1[3:0] --> 1110 --        TOP--> ICR1
WGM1[3:0] --> 1111 --        TOP--> OCR1A
*/
// we take OCR1A for custom frequency
// choose WGM1[3:0] --> 1111 for OCR1A as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B | (1<<WGM12);
```

```
TCCR1B = TCCR1B | (1<<WGM13);

// here we set COM1B[1:0] as 01 for toggling of OC1A
// which is reflected in PB1
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A & ~(1<<5);
TCCR1A = TCCR1A | (1<<4);

OCR1A = 0x1234; // for freqeuncy


// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

//enabled global interrupt
sei();
}
```

7.4.6 Application I - PWM generation

```
void Timer1_FastPWMGeneration(uint32_t on_time_us, uint32_t off_time_us)
{
        uint32_t total_time = on_time_us + off_time_us;

        /* TCNT1 starts from 0X0000 goes upto TOP and restarts from 0X00*/
        /* Mode of operation:
                WGM1[3:0] --> 0101 --        TOP--> 0X00FF
                WGM1[3:0] --> 0110 --        TOP--> 0x01FF
                WGM1[3:0] --> 0111 --        TOP--> 0x03FF
                WGM1[3:0] --> 1110 --        TOP--> ICR1
                WGM1[3:0] --> 1111 --        TOP--> OCR1A
        */

        // we take OCR1A for custom frequency and OCR1B for PWM on time(duty cycle)

        // choose WGM1[3:0] --> 1111 for OCR1A as TOP for custom frequency
        TCCR1A = TCCR1A | (1<<WGM10);
        TCCR1A = TCCR1A | (1<<WGM11);
        TCCR1B = TCCR1B | (1<<WGM12);
        TCCR1B = TCCR1B | (1<<WGM13);


        // COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
        TCCR1A = TCCR1A | (1<<COM1B0);
        TCCR1A = TCCR1A | (1<<COM1B1);

        if(total_time <4)
        {
                // if total_time <= 3us -- so we stop clock

                OCR1A = 0;
                OCR1B = 0;
                // start timer by setting the clock prescalar
                //  use the same clock from I/O clock
                //  CS1[2:0] === 001
                // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
                TCCR1B = TCCR1B & ~(1<<0);
```

```
            TCCR1B = TCCR1B & ~(1<<1);
            TCCR1B = TCCR1B & ~(1<<2);
    }
    else if((3 < total_time)  && (total_time <= 4000))
    {
            OCR1A = ((total_time * 16) >> 0) - 1;
            OCR1B = ((on_time_us * 16) >> 0) - 1;
            // start timer by setting the clock prescalar
            //  use the same clock from I/O clock
            //  CS1[2:0] === 001
            // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
            TCCR1B = TCCR1B | (1<<0);
            TCCR1B = TCCR1B & ~(1<<1);
            TCCR1B = TCCR1B & ~(1<<2);
    }
    else if((4000 < total_time)  && (total_time <= 32000))
    {
            OCR1A = ((total_time * 16) >> 3) - 1;
            OCR1B = ((on_time_us * 16) >> 3) - 1;
            // start timer by setting the clock prescalar
            //  dived by 8 from I/O clock
            //  CS1[2:0] === 010
            // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
            TCCR1B = TCCR1B & ~(1<<0);
            TCCR1B = TCCR1B | (1<<1);
            TCCR1B = TCCR1B & ~(1<<2);
    }
    else if((32000 < total_time)  && (total_time <= 260000))
    {
            OCR1A = ((total_time * 16) >> 6) - 1;
            OCR1B = ((on_time_us * 16) >> 6) - 1;
            // start timer by setting the clock prescalar
            //  dived by 64 from I/O clock
            //  CS1[2:0] === 011
            // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
            TCCR1B = TCCR1B | (1<<0);
            TCCR1B = TCCR1B | (1<<1);
            TCCR1B = TCCR1B & ~(1<<2);

    }
    else if((260000 < total_time)  && (total_time <= 1000000))
    {
            OCR1A = ((total_time * 16) >> 8) - 1;
            OCR1B = ((on_time_us * 16) >> 8) - 1;
            // start timer by setting the clock prescalar
            //  divide by256 from I/O clock
            //  CS1[2:0] === 100
            // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
            TCCR1B = TCCR1B & ~(1<<0);
            TCCR1B = TCCR1B & ~(1<<1);
            TCCR1B = TCCR1B | (1<<2);

    }
    else if(total_time > 1000000)
    {
            // dont' cross more than 1s
    }
}
void PWMGeneration(double duty_cycle_percent,uint32_t freqeuncy)
{
    double total_time_us = (1000000.0/freqeuncy);
    double on_time_us = (duty_cycle_percent/100.0) * total_time_us;
    if (on_time_us<1.0)
```

```
        {
                on_time_us = 1;
        }

        // max time = 1S -- min freqency = 1 Hz
        //  min time = 4us -- max frequency = 250000 = 250khz
        Timer1_FastPWMGeneration(on_time_us, total_time_us - on_time_us);
}
```

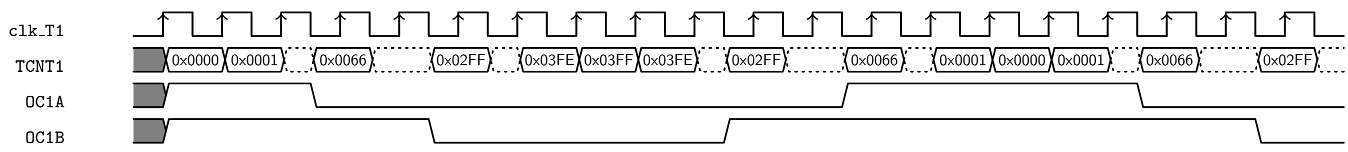
7.5 Phase Corrected PWM Mode

```
ISR(TIMER1_OVF_vect)
{
}
ISR(TIMER1_COMPA_vect)
{
}
ISR(TIMER1_COMPB_vect)
{
}
```

7.5.1 Non-Inverting PWM with TOP at MAX(0x00FF or 0x01FF or 0x03FF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR1A** and/or **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 0001 or 0010 or 0011 for Phase Corrected PWM Mode with TOP at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1A[1:0]*** and/or ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make outputs *OC1A* and/or *OC01* pins to generate PWM by comparing between **OCR1A** and/or **OCR1B** respectively. That is for Non-Inverting, ***COM1x[1:0]*** is written 10.
- Next, the duty cycle value is loaded into **OCR1A** and/or **OCR1B** register for *OC1A* and/or *OC1B* pins.
- Also, the ***OCIE1A*** and/or ***OCIE1B*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 10% duty cycle *OC1A* and 75% duty cycle*OC1B* pins are shown assuming .
  - WGM1[3:0] === 0011 – TOP equals 0x03FF
  - 0x66 for OCR1A.
  - 0x2FF for OCR1B.




```
/* TCNT1 starts from 0X0000 goes upto TOP and from TOP to BOTTOM*/
/* Mode of operation:
    WGM1[3:0] --> 0001 --         TOP--> 0X00FF
    WGM1[3:0] --> 0010 --         TOP--> 0x01FF
    WGM1[3:0] --> 0011 --         TOP--> 0x03FF
    WGM1[3:0] --> 1010 --         TOP--> ICR1
    WGM1[3:0] --> 1011 --         TOP--> OCR1A
*/
// we take 0x03FF for fixed frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 0011 for 0x03FF as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
```

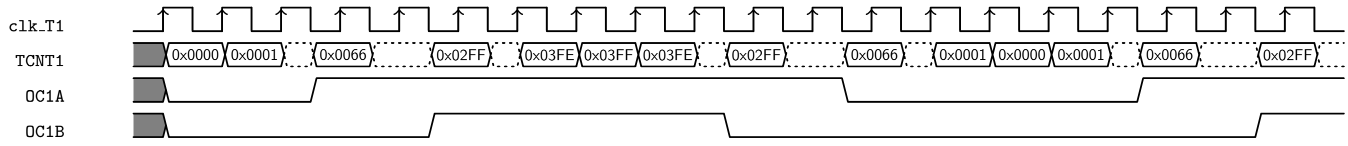
```
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

/* in timer0_phase_pwm_top_max, only two possiblites are there for COM0B[1:0] and COM0A[1:0] i.e)
↪  10(Inverting) and 11(Non- inverting) */

// here we set COM0A[1:0] as 10 for non-inverting
// here we set COM0B[1:0] as 10 for non-inverting

// which is reflected in PD6
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1A1);
TCCR1A = TCCR1A & ~(1<<COM1A0);

// which is reflected in PD65
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1B1);
TCCR1A = TCCR1A & ~(1<<COM1B0);

// Enable Interrupt when TOV1 overflows TOP - here 0x03FF
//  TOIE1 bit is enabled
TIMSK1 = TIMSK1 | (1<<TOIE1);

/* we use OCF1A flag - which is set at every time TCN0 reaches OCR1A */
TIMSK1 = TIMSK1 | (1<<OCIE1A);
/* we use OCF1B flag - which is set at every time TCN0 reaches OCR1B */
TIMSK1 = TIMSK1 | (1<<OCIE1B);

// Next we set values for OCR1A and OCR2B
// Since, TCNT1 goes till max(0x3FF), we can choose OCR1A and OCR1B to any value below max(0x03FF)
OCR1A = 102; // for 10% duty clcle
OCR1B = 767; // for 75% duty clcle

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

//enabled global interrupt
sei();
```

7.5.2 Inverting PWM with TOP at MAX(0x00FF or 0x01FF or 0x03FF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR1A** and/or **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 0001 or 0010 or 0011 for Phase Corrected PWM Mode with TOP at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1A[1:0]*** and/or ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make outputs *OC1A* and/or *OC01* pins to generate PWM by comparing between **OCR1A** and/or **OCR1B** respectively. That is for Inverting, ***COM1x[1:0]*** is written 11.
- Next, the duty cycle value is loaded into **OCR1A** and/or **OCR1B** register for *OC1A* and/or *OC1B* pins.
- Also, the ***OCIE1A*** and/or ***OCIE1B*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 10% duty cycle *OC1A* and 75% duty cycle*OC1B* pins are shown assuming .

- WGM1[3:0] === 0011 – TOP equals 0x03FF
- 0x66 for OCR1A.
- 0x2FF for OCR1B.



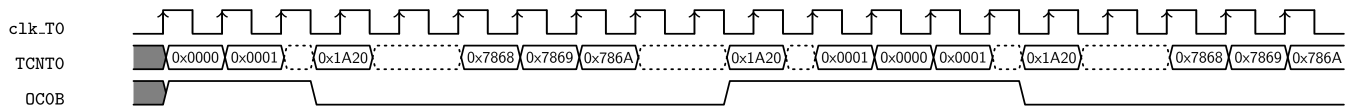

```
/* TCNT1 starts from 0X0000 goes upto TOP and from TOP to BOTTOM*/
/* Mode of operation:
    WGM1[3:0] --> 0001 --        TOP--> 0X00FF
    WGM1[3:0] --> 0010 --        TOP--> 0x01FF
    WGM1[3:0] --> 0011 --        TOP--> 0x03FF
    WGM1[3:0] --> 1010 --        TOP--> ICR1
    WGM1[3:0] --> 1011 --        TOP--> OCR1A
*/
// we take 0x03FF for fixed frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 0011 for 0x03FF as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
TCCR1B = TCCR1B & ~(1<<WGM13);

/* in timer0_phase_pwm_top_max, only two possiblites are there for COM0B[1:0] and COM0A[1:0] i.e)
↪  10(Inverting) and 11(Non- inverting) */

// here we set COM0A[1:0] as 11 for inverting
// here we set COM0B[1:0] as 11 for inverting

// which is reflected in PD6
// COM1A[1](bit7) from TCCR1A, COM1A[0](bit6) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1A1);
TCCR1A = TCCR1A | (1<<COM1A0);

// which is reflected in PD65
// COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
TCCR1A = TCCR1A | (1<<COM1B1);
TCCR1A = TCCR1A | (1<<COM1B0);

// Enable Interrupt when TOV1 overflows TOP - here 0x03FF
//  TOIE1 bit is enabled
TIMSK1 = TIMSK1 | (1<<TOIE1);

/* we use OCF1A flag - which is set at every time TCN0 reaches OCR1A */
TIMSK1 = TIMSK1 | (1<<OCIE1A);
/* we use OCF1B flag - which is set at every time TCN0 reaches OCR1B */
TIMSK1 = TIMSK1 | (1<<OCIE1B);

// Next we set values for OCR1A and OCR2B
// Since, TCNT1 goes till max(0x3FF), we can choose OCR1A and OCR1B to any value below max(0x03FF)
OCR1A = 102; // for 10% duty clcle
OCR1B = 767; // for 75% duty clcle

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);

//enabled global interrupt
```

```
sei();
```

7.5.3 Non-Inverting PWM with TOP at OCR1A

Frequency is chosen by **OCR1A** and Duty cycle by **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 1011 for Phase Corrected PWM Mode with OCR1A at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make output *OC1B* pins to generate PWM by comparing between **OCR1B** respectively. That is for Non-Inverting, ***COM1B[1:0]*** is written 10.
- The frequency of duty cycle is loaded into **OCR1A** register.
- Next, the duty cycle value is loaded into **OCR1B** register for *OC1B* bits.
- Also, the ***OCIE1B*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match.
- Finally, Timer is started by setting ***CS1[2:0]*** bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 37% duty cycle *OC1B* pins are shown assuming .
  - 0x7869 for OCR1A.
  - 0x1A20 for OCR1B.



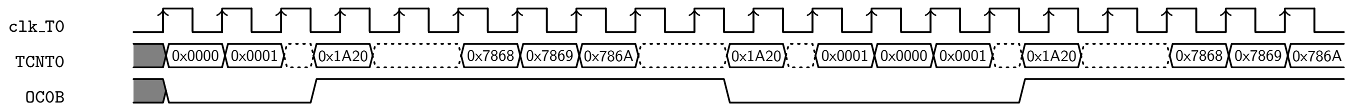

```
/* TCNT1 starts from 0X0000 goes upto TOP and from TOP to BOTTOM*/
/* Mode of operation:
    WGM1[3:0] --> 0001 --
    TOP--> 0X00FF
    WGM1[3:0] --> 0010 --
    TOP--> 0x01FF
    WGM1[3:0] --> 0011 --
    TOP--> 0x03FF
    WGM1[3:0] --> 1010 --
    TOP--> ICR1
    WGM1[3:0] --> 1011 --
    TOP--> OCR1A
*/
// we take 0x03FF for fixed frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 1011 for OCR1A as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
TCCR1B = TCCR1B | (1<<WGM13);

// here we set COM1A[1:0] as 10 for non-inverting
// which is reflected in PD5
// COM1B[1](bit5) from TCCR1A, COMOB[0](bit4) from TCCR1A
TCCR1A = TCCR1A | (1<<5);
TCCR1A = TCCR1A & ~(1<<4);

// Next we set values for OCR1A and OCR1B
// Since, TCNT1 goes till OCR1A, we can choose OCR1B to any value below OCR1A
OCR1A = 0x7869; // for freqeuncy
OCR1B = 0x1A20; // for pwm duty cylc

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
```

```
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);
//enabled global interrupt
sei();
```

7.5.4 Inverting PWM with TOP at OCR1A

Frequency is chosen by **OCR1A** and Duty cycle by **OCR1B** register.

- First, ***WGM1[3:0]*** bits are configured as 1011 for Phase Corrected PWM Mode with OCR1A at MAX in **TCCR1A** and **TCCR1B** registers.
- Next, ***COM1B[1:0]*** bits of **TCCR1A** register are configured to make output *OC1B* pins to generate PWM by comparing between **OCR1B** respectively. That is for Inverting, ***COM1B[1:0]*** is written 11.
- The frequency of duty cycle is loaded into **OCR1A** register.
- Next, the duty cycle value is loaded into **OCR1B** register for *OC1B* bits.
- Also, the ***OCIE1B*** bits of **TIMSK1** register are enabled for Output Compare Interupts if needed.
- The interrupt Service routine is written if needed for compare match.
- Finally, Timer is started by setting *CS1[2:0]* bit as needed prescalar in **TCR1B** register.
- The timing for PWM on 37% duty cycle *OC1B* pins are shown assuming .
  - 0x7869 for OCR1A.
  - 0x1A20 for OCR1B.




```
/* TCNT1 starts from 0X0000 goes upto TOP and from TOP to BOTTOM*/
/* Mode of operation:
    WGM1[3:0] --> 0001 --
    TOP--> 0X00FF
    WGM1[3:0] --> 0010 --
    TOP--> 0x01FF
    WGM1[3:0] --> 0011 --
    TOP--> 0x03FF
    WGM1[3:0] --> 1010 --
    TOP--> ICR1
    WGM1[3:0] --> 1011 --
    TOP--> OCR1A
*/
// we take 0x03FF for fixed frequency and OCR1B for PWM on time(duty cycle)
// choose WGM1[3:0] --> 1011 for OCR1A as TOP for custom frequency
TCCR1A = TCCR1A | (1<<WGM10);
TCCR1A = TCCR1A | (1<<WGM11);
TCCR1B = TCCR1B & ~(1<<WGM12);
TCCR1B = TCCR1B | (1<<WGM13);

// here we set COM1A[1:0] as 11 for inverting
// which is reflected in PD5
// COM1B[1](bit5) from TCCR1A, COMOB[0](bit4) from TCCR1A
TCCR1A = TCCR1A | (1<<5);
TCCR1A = TCCR1A | (1<<4);

// Next we set values for OCR1A and OCR1B
// Since, TCNT1 goes till OCR1A, we can choose OCR1B to any value below OCR1A
OCR1A = 0x7869; // for freqeuncy
```

```
OCR1B = 0x1A20; // for pwm duty cylc

// start timer by setting the clock prescalar
// SAME AS from I/O clock
// same-- CS1[2:0] === 001
// CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
TCCR1B = TCCR1B | (1<<CS10);
TCCR1B = TCCR1B & ~(1<<CS11);
TCCR1B = TCCR1B & ~(1<<CS12);
//enabled global interrupt
sei();
```

7.5.5 Application I - PWM generation

```
void Timer1_PhaseCorrectedPWMGeneration(uint32_t On_time_us, uint32_t Off_time_us)
{
    // Since, it is dual slope, the time would be doubled for one cylce, so we divide by 2
    uint32_t total_time = (On_time_us>>1) + (Off_time_us>>1);
    uint32_t on_time_us = On_time_us >> 1;

    /* TCNT1 starts from 0X0000 goes upto TOP and from TOP to BOTTOM*/
    /* Mode of operation:
        WGM1[3:0] --> 0001 --
        TOP--> 0X00FF
        WGM1[3:0] --> 0010 --
        TOP--> 0x01FF
        WGM1[3:0] --> 0011 --
        TOP--> 0x03FF
        WGM1[3:0] --> 1010 --
        TOP--> ICR1
        WGM1[3:0] --> 1011 --
        TOP--> OCR1A
    */
    // we take 0x03FF for fixed frequency and OCR1B for PWM on time(duty cycle)
    // choose WGM1[3:0] --> 1011 for OCR1A as TOP for custom frequency
    TCCR1A = TCCR1A | (1<<WGM10);
    TCCR1A = TCCR1A | (1<<WGM11);
    TCCR1B = TCCR1B & ~(1<<WGM12);
    TCCR1B = TCCR1B | (1<<WGM13);

    // COM1B[1](bit5) from TCCR1A, COM1B[0](bit4) from TCCR1A
    TCCR1A = TCCR1A | (1<<COM1B0);
    TCCR1A = TCCR1A | (1<<COM1B1);

    if(total_time <4)
    {
        // if total_time <= 3us -- so we stop clock
        OCR1A = 0;
        OCR1B = 0;
        // start timer by setting the clock prescalar
        // use the same clock from I/O clock
        // CS1[2:0] === 001
        // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
        TCCR1B = TCCR1B & ~(1<<0);
        TCCR1B = TCCR1B & ~(1<<1);
        TCCR1B = TCCR1B & ~(1<<2);
    }
    else if((3 < total_time) && (total_time <= 4000))
    {
        OCR1A = ((total_time * 16) >> 0) - 1;
        OCR1B = ((on_time_us * 16) >> 0) - 1;
        // start timer by setting the clock prescalar
```

```c
        // use the same clock from I/O clock
        // CS1[2:0] === 001
        // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
        TCCR1B = TCCR1B | (1<<0);
        TCCR1B = TCCR1B & ~(1<<1);
        TCCR1B = TCCR1B & ~(1<<2);
    }
    else if((4000 < total_time) && (total_time <= 32000))
    {
        OCR1A = ((total_time * 16) >> 3) - 1;
        OCR1B = ((on_time_us * 16) >> 3) - 1;
        // start timer by setting the clock prescalar
        // dived by 8 from I/O clock
        // CS1[2:0] === 010
        // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
        TCCR1B = TCCR1B & ~(1<<0);
        TCCR1B = TCCR1B | (1<<1);
        TCCR1B = TCCR1B & ~(1<<2);
    }
    else if((32000 < total_time) && (total_time <= 260000))
    {
        OCR1A = ((total_time * 16) >> 6) - 1;
        OCR1B = ((on_time_us * 16) >> 6) - 1;
        // start timer by setting the clock prescalar
        // dived by 64 from I/O clock
        // CS1[2:0] === 011
        // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
        TCCR1B = TCCR1B | (1<<0);
        TCCR1B = TCCR1B | (1<<1);
        TCCR1B = TCCR1B & ~(1<<2);
    }
    else if((260000 < total_time) && (total_time <= 1000000))
    {
        OCR1A = ((total_time * 16) >> 8) - 1;
        OCR1B = ((on_time_us * 16) >> 8) - 1;
        // start timer by setting the clock prescalar
        // divide by256 from I/O clock
        // CS1[2:0] === 100
        // CS1[2](bit2) from TCCR1B,CS1[1](bit1) from TCCR1B,CS1[0](bit0) from TCCR1B
        TCCR1B = TCCR1B & ~(1<<0);
        TCCR1B = TCCR1B & ~(1<<1);
        TCCR1B = TCCR1B | (1<<2);
    }
    else if(total_time > 1000000)
    {
        // dont' cross more than 1s
    }
}
void PWMGeneration(double duty_cycle_percent,uint32_t freqeuncy)
{
    double total_time_us = (1000000.0/freqeuncy);
    double on_time_us = (duty_cycle_percent/100.0) * total_time_us;
    if (on_time_us<1.0)
    {
        on_time_us = 1;
    }

    // max time = 8ms -- min freqency = 125 Hz
    //  min time = 8us -- max frequency = 250000 = 125khz
    Timer1_PhaseCorrectedPWMGeneration(on_time_us, total_time_us - on_time_us);
}
```