ATmega328P Timer/Counter 1

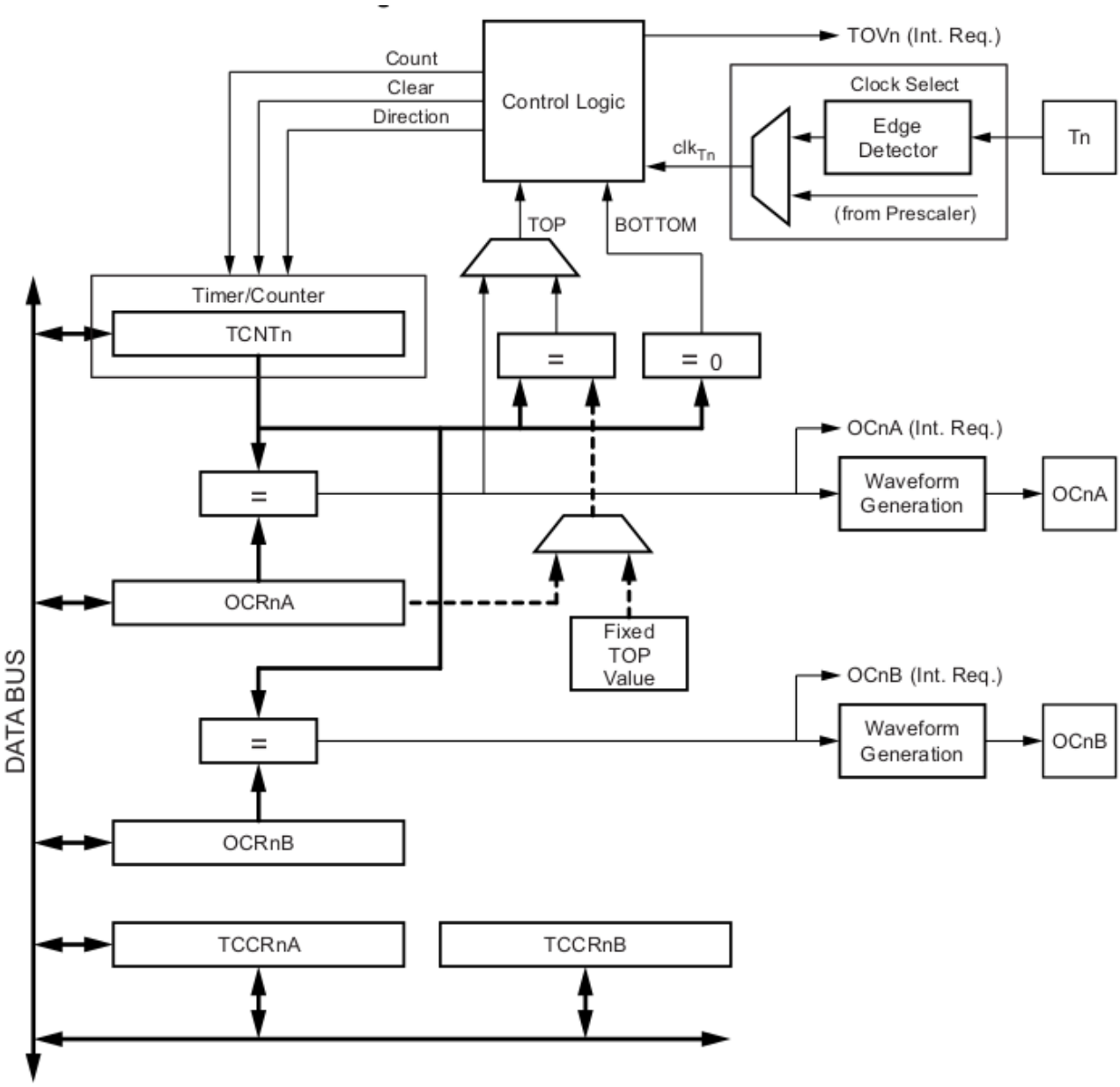
Narendiran S

July 26, 2020

1 Features

- General purpose 8-bit Timer/Counter module.
- Two independent output compare units.
- Variable PWM.
- Three independent interrupt sources (TOV0, OCF0A, and OCF0B).
- Clear timer on compare match (auto reload)

2 Block Diagram



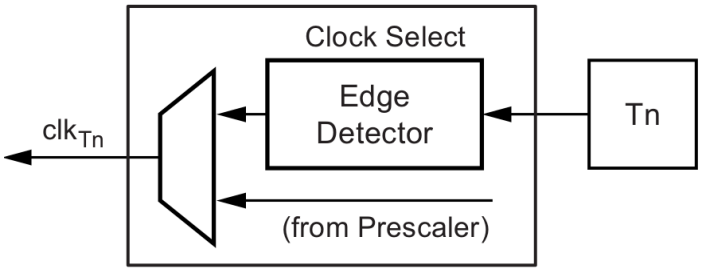
3 Terminologies and Registers

Parameter	Description
BOTTOM	counter reaches 0x00
MAX	ounter reaches 0xFF
TOP	counter reaches highest value (depends on mode of operation can be 0xFF, OCR0A).

Register - 8 bit	Name
TCNT0	Timer/Counter0 count value
TCCR0A	Timer/Coutner0 Control Register A
TCCR0B	Timer/Coutner0 Control Register B
OCBR0A	Output compare register A
OCBR0B	Output compare register B
TIFR0	Timer Interrupt Flag Register
TIMSK0	Timer interrupt Mask Register

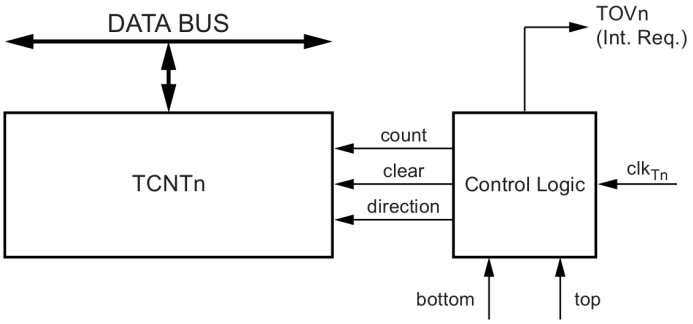
4 Timer/Counter0 Units

4.1 Clock Source/Select Unit



- The source for the Timer/Counter0 can be external or internal.
- External clock source is from *T0* pin.
- While Internal Clock source can be clocked via a prescalar.
- The output of this unit is the timer clock (clk_{T0}).
- It uses ***CS0[2:0]*** bits in **TCCR0B** register to select the source.

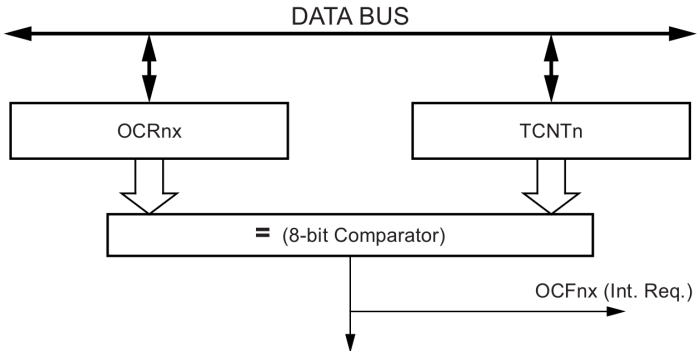
4.2 Counter Unit



Signal	Description
count	Increment or decrement **TCNT0** by 1
direction	Select between increment or decrement
clear	Clears **TCNT0** to 0x00
clk_{T0}	Timer/Coutner0 clock
top	Signalize that **TCNT0** has maximum value
bottom	Signalize that **TCNT0** has minimum value(0x00)

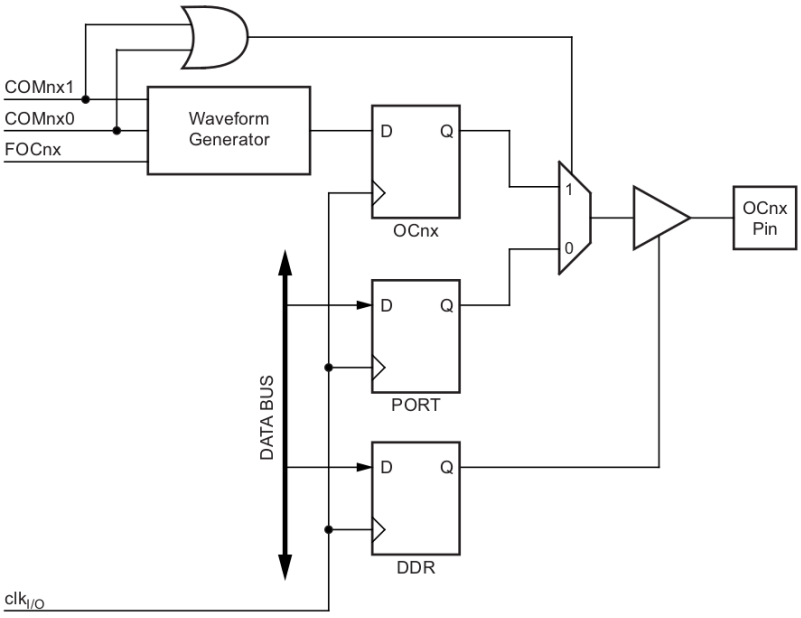
- The main part of the 8-bit Timer/Counter is the programmable bi-directional counter.
- Depending the mode of operation the counter is cleared, incremented, or decremented at each timer clock (clk_{T0}).
- Counting sequence is determined by ***WGM0[1:0]*** bits of **TCCR0A** -Timer/Counter0 Control register A and ***WGM02*** bit of **TCCR0B** - Timer/Counter0 Control register B.
- The Timer/Counter0 Overflow flag ***TOV0*** is set and can generate interrupt according to the mode.

4.3 Output Compare Unit



- 8-bit comparator continuously compares **TCNT0** with both **OCR0A** and **OCR0B**.
- When **TCNT0** equals **OCR0A** or **OCR0B**, 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 ***WGM0[2:0]*** bits and compare output mode ***COM0x[1:0]*** bits.

4.4 Compare Match Output Unit



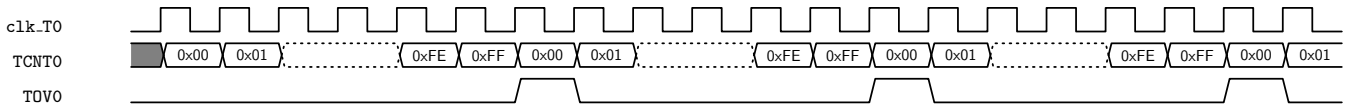
- This unit is used for changing the state of *OC0A* and *OC0B* pins by configuring the ***COM0x[1:0]*** bits.
- But, general I/O port function is overriiden by DDR reigster.

5 Modes of Operation

- The mode of operation can be defined by combination of waveform generation mode (***WGM0[2:0]***) and compare output mode(***COM0[1:0]***) bits.
- The waveform generation mode (***WGM0[2:0]***) bits affect the counting sequence.
- For non-PWM mode, ***COM0[1:0]*** bits control if the output should be set, cleared or toggled at a compare match.
- For PWM mode, ***COM0[1:0]*** bits control if the PWM generated should be inverted or non-inverted.

5.1 Normal Mode - Non-PWM Mode

- ***WGM0[2:0]*** $--> 000$.
- Counter counts up and no counter clear.
- Overruns TOP(0XFF) and restarts from BOTTOM(0X00).
- ***TOV0*** Flag is only set when overrun.
- We have to clear ***TOV0*** flag inorder to have next running.
- But, if we use interrupt we don't need to clear it as interrupt automatically clear the ***TOV0*** flag.
- The timing can be seen below.

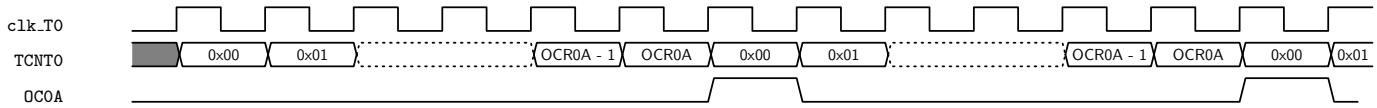


5.2 Clear Timer on Compare Match(CTC) Mode - Non-PWM Mode

- ***WGM0[2:0]*** $--> 010$.
- Counter value clears when **TCNT0** reaches **OCR0A**.
- Interrupt can be generated each time **TCNT0** reaches **OCR0A** register value by ***OCF0A*** flag.
- When ***COM0A[1:0]*** $== 01$, the *OC0A* pin output can be set to toggle its match between **TCNT0** and **OCR0A** to generate waveform.
- The frequency of the waveform its

$$f_{OC0A} = \frac{f_{clkT0}}{2*N*(1+OCR0A)}$$

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

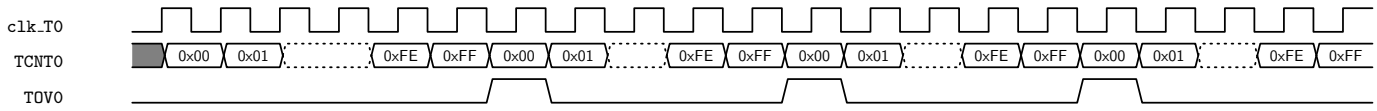


5.3 Fast PWM Mode

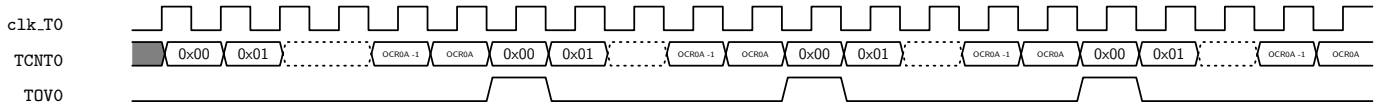
- ***WGM0[2:0]*** $--> 011$ or 111.
- Power Regulation, Rectification, DAC applications.
- Single slope operations causing high frequency PWM waveform.
- Counter starts from BOTTOM to TOP and then restarts from BOTTOM.
- TOP is defined by
  - TOP $== 0$xFF if ***WGM0[2:0]*** $--> 011$
  - TOP $==$ **OCR0A** if ***WGM0[2:0]*** $--> 111$
- When ***COM0A[1:0]*** $== 01$, the *OC0A* pin output can be set to toggle its match between **TCNT0** and TOP to generate waveform.
  - The above is possible only when ***WGM02*** bit is set.
  - And only on *OC0A* pin and not on *OC0B* pin.
- In Inverting Compare Mode ***COM0A[1:0]*** $== 10$, the *OC0A* or *OC0B* pins is made 1 on compare match between **TCNT0** and TOP and made 0 on reaching BOTTOM.
- In Non-Inverting Compare Mode ***COM0A[1:0]*** $== 11$, the *OC0A* or *OC0B* pins is made 0 on compare match between **TCNT0** and TOP and 1 made on reaching BOTTOM.
- The Timer/Counter overflow flag (***TOV0***) is set each time the counter reaches TOP.
- The PWM frequency is given by

$$f_{OC0xPWM} = \frac{f_{clkT0}}{N*256}$$

5.3.1 WGM[2:0] == 011



5.3.2 WGM[2:0] == 011

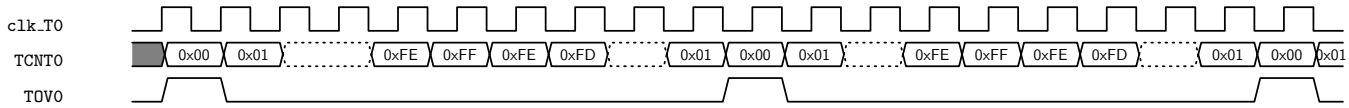


5.4 Phase Correct PWM Mode

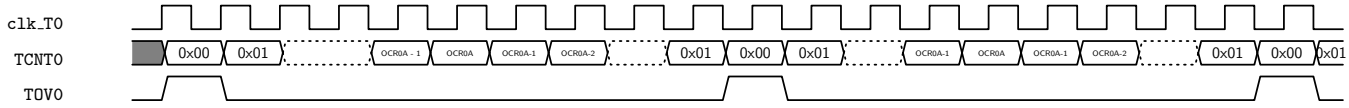
- ***WGM0[2:0]*** $--> 001$ or 101.
- High resolution phase correct PWM.
- Motor control due to symmetric features
- Dual slope operations causing ower frequency PWM waveform.
- Counter starts from BOTTOM to TOP and then from TOP to BOTTOM.
- TOP is defined by
  - TOP == 0xFF if ***WGM0[2:0]*** $--> 001$
  - TOP == **OCR0A** if ***WGM0[2:0]*** $--> 101$
- When ***COM0A[1:0]*** == 01, the *OC0A* pin output can be set to toggle its match between **TCNT0** and TOP to generate waveform.
  - The above is possible only when ***WGM02*** bit is set.
  - And only on *OC0A* pin and not on *OC0B* pin.
- In Inverting Compare Mode ***COM0A[1:0]*** == 10 , the *OC0A* or *OC0B* pins is made 1 on compare match between **TCNT0** and TOP and made 0 on reaching BOTTOM.
- In Non-Inverting Compare Mode ***COM0A[1:0]*** == 11 , the *OC0A* or *OC0B* pins is made 0 on compare match between **TCNT0** and TOP and 1 made on reaching BOTTOM.
- The Timer/Counter overflow flag (***TOV0***) is set each time the counter reaches BOTTOM..
- The PWM frequency is given by

$$f_{OC0xPWM} = \frac{f_{clkT0}}{N*510}$$

5.4.1 WGM[2:0] == 001



5.4.2 WGM[2:0] == 101



6 Register Description

TCCR0A – Timer/Counter Control Register A

7	6	5	4	3	2	1	0
COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00

COM0B[1:0]	**Non-PWM modes**	**Fast PWM**	**Phase Corrected PWM**
00	No output @ *PD5 - OC0B* pin	No output @ *PD5 - OC0B*	No output @ *PD5 - OC0B*
01	Toggle *PD5 - OC0B* pin on compare Match.	Reserved	Reserved
10	Clear *PD5 - OC0B* pin on compare Match.	Clear *PD5 - OC0B* on compare match and set *PD5 - OC0B* at BOTTOM	Clear *PD5 - OC0B* on compare match when up-counting and set *PD5 - OC0B* on compare match when down-counting.
11	Set *PD5 - OC0B* pin on compare Match.	Set *PD5 - OC0B* on compare match and clear *PD5 - OC0B* at BOTTOM	Set *PD5 - OC0B* on compare match when up-counting and clear *PD5 - OC0B* on compare match when down-counting

COM0A[1:0]	**Non-PWM modes**	**Fast PWM**	**Phase Corrected PWM**
00	No output @ *PD6 - OC0A* pin	No output @ *PD6 - OC0A*	No output @ *PD6 - OC0A*
01	Toggle *PD6 - OC0A* pin on compare Match.	When WGM0[2] == 1, Toggle *PD6 - OC0A* pin on Compare match	Toggle *PD6 - OC0A* pin on Compare match
10	Clear *PD6 - OC0A* pin on compare Match.	Clear *PD6 - OC0A* on compare match and set *PD6 - OC0A* at BOTTOM	Clear *PD6 - OC0A* on compare match when up-counting and set *PD6 - OC0A* on compare match when down-counting.
11	Set *PD6 - OC0A* pin on compare Match.	Set *PD6 - OC0A* on compare match and clear *PD6 - OC0A* at BOTTOM	Set *PD6 - OC0A* on compare match when up-counting and clear *PD6 - OC0A* on compare match when down-counting

WGM0[2:0]	**Mode of operation**	**TOP**	**TOV0 Flag set on**
000	Normal	0xFF	MAX
001	PWM Phase Corrected	0xFF	BOTTOM
010	CTC	OCRA	MAX
011	Fast PWM	0xFF	MAX
101	PWM Phase Corrected	OCR0A	BOTTOM
111	Fast PWM	OCR0A	TOP

TCCR0B – Timer/Counter Control Register B

7	6	5	4	3	2	1	0
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

CS0[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 *T0* pin. Clock on falling edge.
111	External clock source on *T0* pin. Clock on rising edge.

TIMSK0 – Timer/Counter Interrupt Mask Register

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
-	-	-	-	-	OCIE0B	OCIE0A	TOIE0

Enable interrupts for compare match between **TCNT0** and **OCR0A** or **TCNT0** and **OCR0B** or overflow in **TCNT**0.

TIFR0 – Timer/Counter 0 Interrupt Flag Register

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
-	-	-	-	-	OCIE0B	OCIE0A	TOIE0

FLag registers for interrupts on compare match between **TCNT0** and **OCR0A** or **TCNT0** and **OCR0B** or overflow in **TCNT0**.

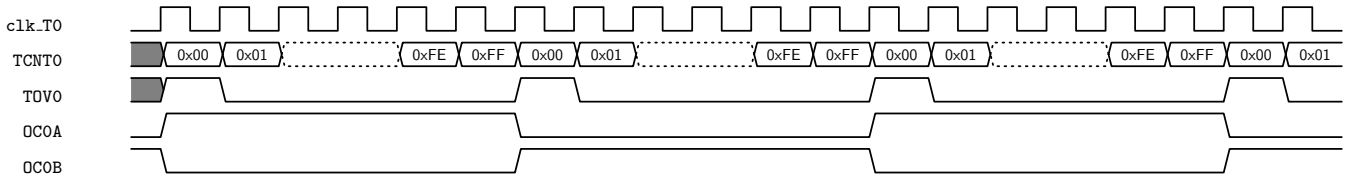
7 Configuring the Timer/Counter

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, ***WGM0[2:0]*** bits are configured as 000 for Normal Mode in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0A[1:0]*** and/or ***COM0A[1:0]*** bits are configured to make outputs *OC0A* and/or *OC0B* pins to do nothing, set, clear or toggle in **TCCR0A** register.
- Next, Interrupt is Enabled by ***TOIE0*** (overflow enable) in **TIMSK0** reigster.
- Finally, Timer is started by setting prescalar in ***CS0[2:0]*** bits as needed prescalar of **TCR0B** reigster.
- Global Interrupt is enabled.
- A interrupt Service Routine for Timer0 overflow is Written.
- No need to clear the overflow flag as it is done by hardware.
- The timing when both pins *OC0A* and *OC0B* are made to toggle.



- The code can be seen below,

```
// MOde of operation to Normal Mode -- WGM0[2:0] === 000
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A & (~(1<<0) & ~(1<<1));
TCCR0B = TCCR0B & ~(1<<3);

/* What to do when timer reaches the MAX(0xFF) value */
// toggle OC0A and OC0B on each time when reaches the MAX(0xFF)
// which is reflected in PD6 and PD5

// Output OC0A to toglle when reaches MAX -- COM0A[1:0] === 01
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A & ~(1<<7);
TCCR0A = TCCR0A | (1<<6);

// Output OC0B to toglle when reaches MAX -- COM0B1:0] === 01
// COM0B[1](bit7) from TCCR0A, COM0B[0](bit6) from TCCR0A
TCCR0A = TCCR0A & ~(1<<5);
TCCR0A = TCCR0A | (1<<4);

//Enable Interrupt of OVERFLOW flag so that interrupt can be generated
TIMSK0 = TIMSK0 | (1<<0);

// start timer by setting the clock prescalar
// DIVIDE BY 8 from I/O clock
// DIVIDE BY 8-- CS0[2:0] === 010
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<1);
TCCR0B = TCCR0B & (~(1<<0) & ~(1<<2));

// enabling global interrupt
sei();
// SO ON TIME = max_count / (F_CPU / PRESCALAR)
```

```
// ON TIME = 0xFF / (16000000/8) = 128us
// since symmetric as toggling OFF TIME = 128us
// hence, we get a square wave of fequency 1 / 256us = 3.906kHz
```

```
ISR(TIMER0_OVF_vect)
{
    // do the thing when overflows.
}
```

7.1.2 As Counter

- Every rising/falling edge the count increases.
- So to reach 256 count, it would take a time of $\frac{0xFF}{frequency@T0_{pin}}$.
- First, ***WGM0[2:0]*** bits are configured as 000 for Normal Mode in **TCCR0A** and **TCCR0B** registers.
- Finally, Counter is started by configuring ***CS0[2:0]*** bits to 110 or 111 for external falling or rising edge on *T0 - PD4*.
- The code when *T0* pin is used as counter @ falling edge.

```
// MOde of operation to Normal Mode -- WGM0[2:0] === 000
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A & (~(1<<0) & ~(1<<1));
TCCR0B = TCCR0B & ~(1<<3);

/* to count external event -we must connect source to T0 (PD4) */
// THE CLK IS CLOCKED FROM external source
// Falling edge of T0(PD4) -- CS0[2:0] === 110
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<2);
TCCR0B = TCCR0B | (1<<1);
TCCR0B = TCCR0B & ~(1<<0);
```

7.1.3 Application I - Delay

```
/* TCNT0 starts from 0X00 goes upto 0XFF and restarts */
/* No possible use case as it just goes upto 0xFF and restarts */
// MOde of operation to Normal Mode -- WGM0[2:0] === 000
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A & (~(1<<0) & ~(1<<1));
TCCR0B = TCCR0B & ~(1<<3);

/* What to do when timer reaches the MAX(0xFF) value */
// nothing should be done on OC0A for delay
// nothing  -- COM0A[1:0] === 00
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A & ~(1<<7);
TCCR0A = TCCR0A & ~(1<<6);

/* The delay possible = 0xff / (F_CPU/prescalar) */
// lowest delay = 0xff / (16000000 / 1) = 16us
// when prescalar == 8 --> delay = 0xff / (16000000 / 8) = 128us
// when prescalar == 64 --> delay = 0xff / (16000000 / 64) = 1.024ms
// when prescalar == 256 --> delay = 0xff / (16000000 / 256) = 4.096ms
// highest delay possible = 0xff / (16000000 / 1024) = 16.38ms

// start timer by setting the clock prescalar
// DIVIDE BY 8 use the same clock from I/O clock
// DIVIDE BY 8-- CS0[2:0] === 010
```

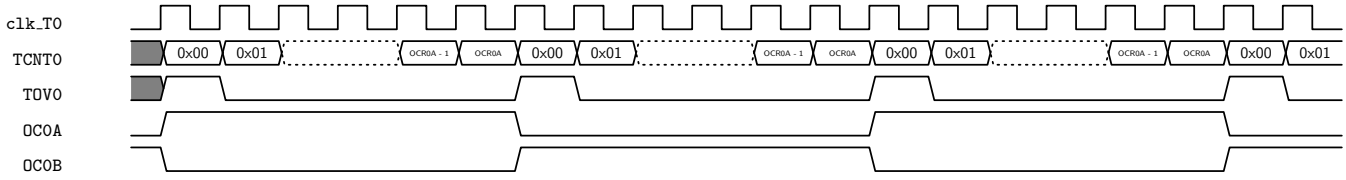
```
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B & ~(1<<0);
TCCR0B = TCCR0B | (1<<1);
TCCR0B = TCCR0B & ~(1<<2);


// actual delaying - wait until delay happens
while((TIFR0 & 0x01) == 0x00); // checking overflow flag when overflow happns
// clearing the overflag so that we can further utilize
TIFR0 = TIFR0 | 0x01;
```

7.2 CTC Mode

7.2.1 As Timer

$$ON\_TIME = \frac{1+OCR0A}{\frac{F\_CPU}{PRESCALAR}}$$

- Depending on **OCR0A** register and PRESCALR value, we get different ON_TIME.
- First, ***WGM0[2:0]*** bits are configured as 010 for CTC Mode in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0A[1:0]*** and/or ***COM0B[1:0]*** bits are configured to make outputs *OC0A* and/or *OC0B* pins to do nothing, set, clear or toggle in **TCCR0A** register.
- Next, Interrupt is Enabled by ***OCIE01A*** (utput compare on match on **OCR0A** register enable) in **TIMSK0** reigster.
- Finally, Timer is started by setting prescalar in ***CS0[2:0]*** bits as needed prescalar of **TCR0B** reigster.
- Global Interrupt is enabled.
- A interrupt Service Routine for Timer0 Compare is Written.
- No need to clear the overflow flag as it is done by hardware.
- The timing when both pins *OC0n* are made to toggle.



- The code can be seen below,

```
// MOde of operation to CTC Mode -- WGM0[2:0] === 010
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A & ~(1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B & ~(1<<3);

/* What to do when timer reaches the OCR0A */
// toggle OC0A on each time when reaches the OCR0A
// which is reflected in PD6
// Output OC0A to toglle when reaches MAX -- COM0A[1:0] === 01
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A & ~(1<<7);
TCCR0A = TCCR0A | (1<<6);

// Output OC0B to toglle when reaches MAX -- COM0B1:0] === 01
// COM0B[1](bit7) from TCCR0A, COM0B[0](bit6) from TCCR0A
TCCR0A = TCCR0A & ~(1<<5);
TCCR0A = TCCR0A | (1<<4);
```

```
// Enable Interrupt when counter matches OCR0A Rgister
//  OCIE0A bit is enabled
TIMSK0 = TIMSK0 | (1<<1);


// setting the value till the counter should reach in OCR0A
// for toggling of OC0A pin
OCR0A = 0x32;

// start timer by setting the clock prescalar
// DIVIDE BY 8 from I/O clock
// DIVIDE BY 8-- CS0[2:0] === 010
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<1);
TCCR0B = TCCR0B & (~(1<<0) & ~(1<<2));

// enabling global interrupt
sei();
// SO ON TIME = (1 + OCR0A) / (F_CPU / PRESCALAR)
// ON TIME = 0X32 / (16000000/8) = 25.5us
// since symmetric as toggling OFF TIME = 25.5us
// hence, we get a square wave of fequency 1 / 50us = 20kHz
```

```
ISR(TIMER0_COMPA_vect)
{
    // do the thing when compare match between TCNT0 matches OCR0A.
}
```

7.2.2 As Counter

- Every rising/falling edge the count increases.
- So to reach required count, it would take a time of $\frac{OCR0A}{frequency@T0_{pin}}$.
- First, ***WGM0[2:0]*** bits are configured as 010 for CTC Mode in **TCCR0A** and **TCCR0B** registers.
- Finally, Counter is started by configuring ***CS0[2:0]*** bits to 110 or 111 for external falling or rising edge on *T0 - PD4* pin.
- The code when *T0* pin is used as counter @ falling edge.

```
// MOde of operation to CTC Mode -- WGM0[2:0] === 010
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A & ~(1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B & ~(1<<3);

// Disbale Interrupt when counter matches OCR0A Rgister
//  OCIE0A bit is disabled
TIMSK0 = TIMSK0 & ~(1<<1);

//we count till OCR0A register value and reset and continue
OCR0A = 0xA;

/* to count external event -we must connect source to T0 (PD4) */
// THE CLK IS CLOCKED FROM external source
// Falling edge of T0(PD4) -- CS0[2:0] === 110
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<2);
TCCR0B = TCCR0B | (1<<1);
TCCR0B = TCCR0B & ~(1<<0);
```

7.2.3 Application I - Delay in ms

```
// minimum delay being 4us -- choose like that
// use PRESCALAR OF 1 -- 3us - 16us -- usage 3us - 16us -- factor=0 -- CS0[2:0]=1
// use PRESCALAR OF 8 -- 3us - 128us -- usage 17us - 128us -- factor=3 -- CS0[2:0]=2
// use PRESCALAR OF 64 -- 4us - 1.024ms -- usage 129us - 1024us -- factor=6 -- CS0[2:0]=3
// use PRESCALAR OF 256 -- 16us - 4.096ms -- usage 1025us - 4096us -- factor=8 -- CS0[2:0]=4

// MOde of operation to ctc Mode -- WGM0[2:0] === 010
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A & ~(1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B & ~(1<<3);

while(delayInMs--)
{
    // for 1ms delay
    OCR0A = 249;
    // start timer by setting the clock prescalar
    //  dived by 64 from I/O clock
    //  CS0[2:0] === 011
    // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
    TCCR0B = TCCR0B | (1<<0);
    TCCR0B = TCCR0B | (1<<1);
    TCCR0B = TCCR0B & ~(1<<2);

    // actual delaying - wait until delay happens
    while((TIFR0 & 0x02) == 0x00); // checking OCF0A (compare match flag A) flag when match happns
    // clearing the compare match flag so that we can further utilize
    TIFR0 = TIFR0 | 0x02;
}
```

7.3 Fast PWM Mode

```
ISR(TIMER0_OVF_vect)
{
}
ISR(TIMER0_COMPA_vect)
{
}
ISR(TIMER0_COMPB_vect)
{
}
```

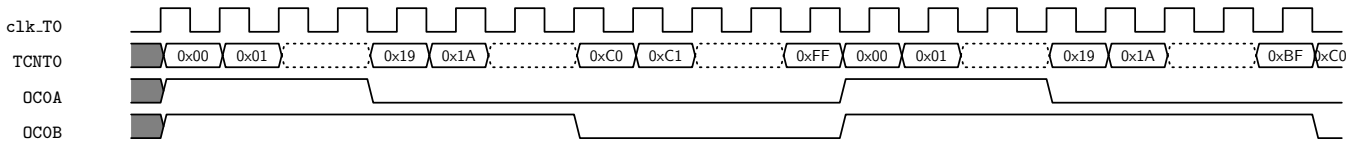
7.3.1 Non-Inverting PWM with TOP at MAX(0xFF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR0A** and/or **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 011 for Fast PWM Mode with TOP at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0A[1:0]*** and/or ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make outputs *OC0A* and/or *OC0B* pins to generate PWM by comparing between **OCR0A** and/or **OCR0B** respectively. That is for Non-Inverting, ***COM0x[1:0]*** is written 10.
- Next, the duty cycle value is loaded into **OCR0A** and/or **OCR0B** register for *OC0A* and/or *OC0B* 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.
- Finally, Timer is started by setting ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 10% duty cycle *OC0A* and 75% duty cycle*OC0B* pins are shown assuming .

– 0x19 for OCR0A.

– 0xC0 for OCR0B.



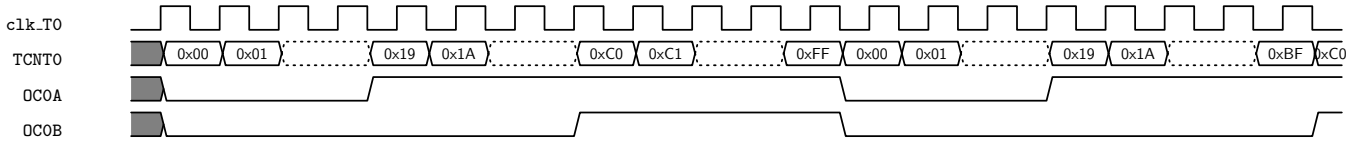

```
// MOde of operation to fast_pwm_top_max Mode -- WGM0[2:0] === 011
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B & ~(1<<3);

// 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
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A | (1<<7);
TCCR0A = TCCR0A & ~(1<<6);

// which is reflected in PD65
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A & ~(1<<4);

// Enable Interrupt when TCN0 overflows TOP - here 0xFF
//  TOV0 bit is enabled
TIMSK0 = TIMSK0 | (1<<0);

/* we use OCF0A flag - which is set at every time TCN0 reaches OCR0A
here we clear led(PC1),  so that we obtain the PWM when TCN0 reaches OCR0A*/
TIMSK0 = TIMSK0 | (1<<1);
/* we use OCF0B flag - which is set at every time TCN0 reaches OCR0B
here we clear led(PC2),  so that we obtain the PWM when TCN0 reaches OCR0B*/
TIMSK0 = TIMSK0 | (1<<2);


// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till max(0xFF), we can choose OCR0A and OCR0B to any value below max(0xFFF)
OCR0A = 0x19; // for 10% duty clcle
OCR0B = 0xC0; // for 75% duty clcle


// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.3.2 Inverting PWM with TOP at MAX(0xFF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR0A** and/or **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 011 for Fast PWM Mode with TOP at MAX in **TCCR0A** and **TCCR0B** registers.

- Next, ***COM0A[1:0]*** and/or ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make outputs *OC0A* and/or *OC0B* pins to generate PWM by comparing between **OCR0A** and/or **OCR0B** respectively. That is for Inverting, ***COM0x[1:0]*** is written 11.
- Next, the duty cycle value is loaded into **OCR0A** and/or **OCR0B** register for *OC0A* and/or *OC0B* bits.
- 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.
- Finally, Timer is started by setting ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 10% duty cycle *OC0A* and 75% duty cycle *OC0B* pins are shown assuming .
  - 0x19 for OCR0A.
  - 0xC0 for OCR0B.



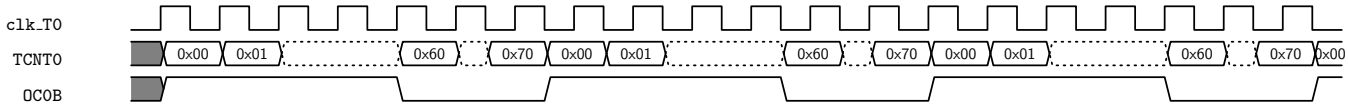

```
// MOde of operation to fast_pwm_top_max Mode -- WGM0[2:0] === 011
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B & ~(1<<3);

// 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
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A | (1<<7);
TCCR0A = TCCR0A | (1<<6);

// which is reflected in PD65
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A | (1<<4);

// Enable Interrupt when TCN0 overflows TOP - here 0xFF
//  TOV0 bit is enabled
TIMSK0 = TIMSK0 | (1<<0);

/* we use OCF0A flag - which is set at every time TCN0 reaches OCR0A
    here we clear led(PC1),  so that we obtain the PWM when TCN0 reaches OCR0A*/
TIMSK0 = TIMSK0 | (1<<1);
/* we use OCF0B flag - which is set at every time TCN0 reaches OCR0B
    here we clear led(PC2),  so that we obtain the PWM when TCN0 reaches OCR0B*/
TIMSK0 = TIMSK0 | (1<<2);


// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till max(0xFF), we can choose OCR0A and OCR0B to any value below max(0xFFF)
OCR0A = 0x19; // for 10% duty clcle
OCR0B = 0xC0; // for 75% duty clcle


// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
```

```
TCCR0B = TCCR0B & ~(1<<2);

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

7.3.3 Non-Inverting PWM with TOP at OCR0A

Frequency is chosen by **OCR0A** and Duty cycle by **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 111 for Fast PWM Mode with **OCR0A** at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make output *OC0B* pins to generate PWM by comparing between **TCNT0** and **OCR0B**. That is for Non-Inverting, ***COM0B[1:0]*** is written 10.
- The frequency of duty cycle is loaded into **OCR0A** register.
- Next, the duty cycle value is loaded into **OCR0B** register for *OC0B* bits.
- Also, the ***OCIE0B*** bits of **TIMSK0** 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 ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 85% duty cycle(0x60) *OC0B* pins are shown assuming .
  - 0x70 for OCR0A.
  - 0x60 for OCR0B.

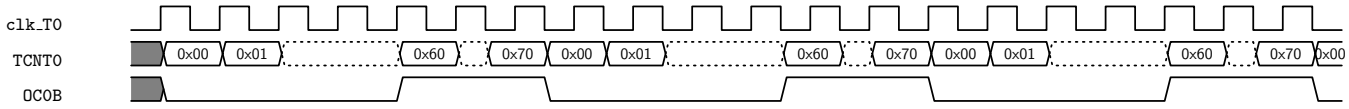



```
// MOde of operation to fast_pwm_top_max Mode -- WGM0[2:0] === 111
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B | (1<<3);

// here we set COM0B[1:0] as 10 for non-inverting
// which is reflected in PD5
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A & ~(1<<4);

// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till OCR0A, we can choose OCR0B to any value below OCR0A
OCR0A = 0x70; // for freqeuncy
OCR0B = 0x60; // for pwm duty cylc

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.3.4 Inverting PWM with TOP at OCR0A

Frequency is chosen by **OCR0A** and Duty cycle by **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 111 for Fast PWM Mode with **OCR0A** at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make output *OC0B* pins to generate PWM by comparing between **TCNT0** and **OCR0B**. That is for Inverting, ***COM0B[1:0]*** is written 11.
- The frequency of duty cycle is loaded into **OCR0A** register.
- Next, the duty cycle value is loaded into **OCR0B** register for *OC0B* bits.
- Also, the ***OCIE0B*** bits of **TIMSK0** 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 ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 85% duty cycle *OC0B* pins are shown assuming .
  - 0x70 for OCR0A.
  - 0x60 for OCR0B.



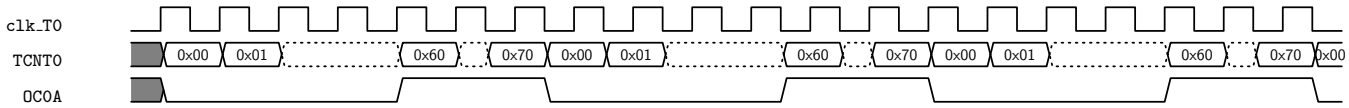

```
// MOde of operation to fast_pwm_top_max Mode -- WGM0[2:0] === 111
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B | (1<<3);

// here we set COM0B[1:0] as 11 for inverting
// which is reflected in PD5
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A | (1<<4);

// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till OCR0A, we can choose OCR0B to any value below OCR0A
OCR0A = 0x70; // for freqeuncy
OCR0B = 0x60; // for pwm duty cylc

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.3.5 Toggling mode square Wave

Frequency is chosen by **OCR0A** register.

- First, ***WGM0[2:0]*** bits are configured as 111 for Fast PWM Mode with OCR0A at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0A[1:0]*** bits of **TCCR0A** register are configured to make output *OC0A* pins to generate PWM by comparing between **OCR0A**. That is for Toggling square wave ***COM0A[1:0]*** is written 01.
- The frequency of duty cycle is loaded into **OCR0A** register.

- Also, the *OCIE0A* bits of **TIMSK0** 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 *CS0[2:0]* bit as needed prescalar in **TCR0B** register.
- The timing for squared wave on *OC0A* pins are shown assuming.
  - 0x70 for OCR0A.




```
// MOde of operation to fast_pwm_top_max Mode -- WGM0[2:0] === 111
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A | (1<<1);
TCCR0B = TCCR0B | (1<<3);

// here we set COM0B[1:0] as 01 for toggling of OC0A
// which is reflected in PD6
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A & ~(1<<7);
TCCR0A = TCCR0A | (1<<6);

// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till OCR0A, we can choose OCR0B to any value below OCR0A
OCR0A = 0x70; // for freqeuncy

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.3.6 Application I - PWM generation

```
void Timer0_FastPWMGeneration(uint32_t on_time_us, uint32_t off_time_us)
{
        uint32_t total_time = on_time_us + off_time_us;

        // MOde of operation to fast_pwm_top_max Mode -- WGM0[2:0] === 111
        // WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
        TCCR0A = TCCR0A | (1<<0);
        TCCR0A = TCCR0A | (1<<1);
        TCCR0B = TCCR0B | (1<<3);

        // which is reflected in PD5
        // COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
        TCCR0A = TCCR0A | (1<<5);
        TCCR0A = TCCR0A & ~(1<<4);

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

                OCR0A = 0;
                // start timer by setting the clock prescalar
```

```
                //  use the same clock from I/O clock
                //  CS0[2:0] === 001
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B & ~(1<<0);
                TCCR0B = TCCR0B & ~(1<<1);
                TCCR0B = TCCR0B & ~(1<<2);
        }
        else if((3 < total_time)  && (total_time <= 16))
        {
                OCR0A = ((total_time * 16) >> 0) - 1;
                OCR0B = ((on_time_us * 16) >> 0) - 1;
                // start timer by setting the clock prescalar
                //  use the same clock from I/O clock
                //  CS0[2:0] === 001
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B | (1<<0);
                TCCR0B = TCCR0B & ~(1<<1);
                TCCR0B = TCCR0B & ~(1<<2);
        }
        else if((16 < total_time)  && (total_time <= 128))
        {
                OCR0A = ((total_time * 16) >> 3) - 1;
                OCR0B = ((on_time_us * 16) >> 3) - 1;
                // start timer by setting the clock prescalar
                //  dived by 8 from I/O clock
                //  CS0[2:0] === 010
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B & ~(1<<0);
                TCCR0B = TCCR0B | (1<<1);
                TCCR0B = TCCR0B & ~(1<<2);
        }
        else if((128 < total_time)  && (total_time <= 1024))
        {
                OCR0A = ((total_time * 16) >> 6) - 1;
                OCR0B = ((on_time_us * 16) >> 6) - 1;
                // start timer by setting the clock prescalar
                //  dived by 64 from I/O clock
                //  CS0[2:0] === 011
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B | (1<<0);
                TCCR0B = TCCR0B | (1<<1);
                TCCR0B = TCCR0B & ~(1<<2);

        }
        else if((1024 < total_time)  && (total_time <= 4096))
        {
                OCR0A = ((total_time * 16) >> 8) - 1;
                OCR0B = ((on_time_us * 16) >> 8) - 1;
                // start timer by setting the clock prescalar
                //  divide by256 from I/O clock
                //  CS0[2:0] === 100
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B & ~(1<<0);
                TCCR0B = TCCR0B & ~(1<<1);
                TCCR0B = TCCR0B | (1<<2);

        }
        else if(total_time > 4096)
        {
                // dont' cross more than 4.096ms
        }
}
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 = 4ms -- min freqency = 250 Hz
        //  min time = 4us -- max frequency = 250000 = 250khz
        Timer0_FastPWMGeneration(on_time_us, total_time_us - on_time_us);
}
```

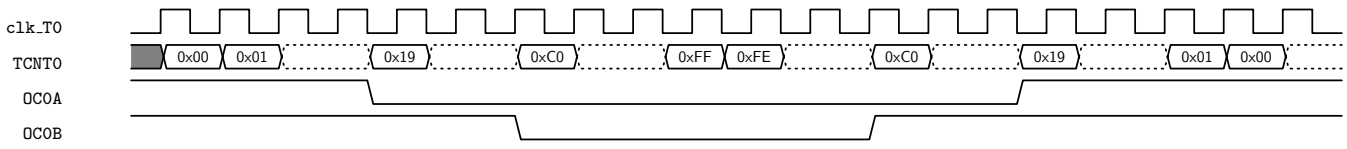
7.4 Phase Corrected PWM Mode

```
ISR(TIMER0_OVF_vect)
{
}
ISR(TIMER0_COMPA_vect)
{
}
ISR(TIMER0_COMPB_vect)
{
}
```

7.4.1 Non-Inverting PWM with TOP at MAX(0xFF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR0A** and/or **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 001 for Phase Corrected PWM Mode with TOP at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0A[1:0]*** and/or ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make outputs *OC0A* and/or *OC0B* pins to generate PWM by comparing between **OCR0A** and/or **OCR0B** respectively. That is for Non-Inverting, ***COM0x[1:0]*** is written 10.
- Next, the duty cycle value is loaded into **OCR0A** and/or **OCR0B** register for *OC0A* and/or *OC0B* bits.
- 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.
- Finally, Timer is started by setting ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 10% duty cycle *OC0A* and 75% duty cycle*OC0B* pins are shown assuming .
  - 0x19 for OCR0A.
  - 0xC0 for OCR0B.



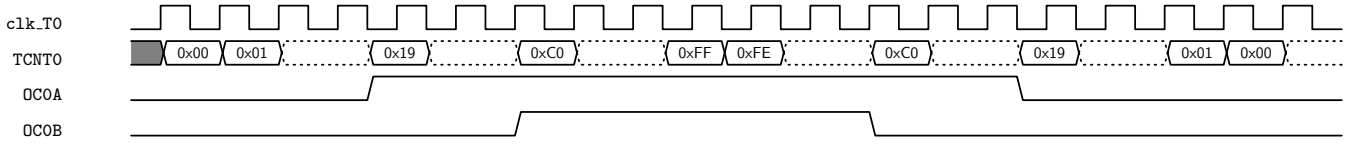

```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 001
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<3);

/* in timer0_phase_pwm_top_max, only two possiblites are there for COM0B[1:0] and COM0A[1:0] i.e) 10(Inver

// 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
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A | (1<<7);
TCCR0A = TCCR0A & ~(1<<6);

// which is reflected in PD65
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A & ~(1<<4);

/* we use overflow flag -- which is set at every time TCN0 reaches TOP here 0xFF
here, we toggle an led(PC0) at every overflow interrupt - this led(PC0) would give the frequency of PWM be
Also, we set the other leds(PC1 and PC2) so that they are make one when TCN0 reaches 0x00 */
// Enable Interrupt when TCN0 overflows TOP - here 0xFF
//  TOV0 bit is enabled
TIMSK0 = TIMSK0 | (1<<0);


// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till max(0xFF), we can choose OCR0A and OCR0B to any value below max(0xFFF)
OCR0A = 0x19; // for 10% duty clcle
OCR0B = 0xC0; // for 75% duty clcle

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.4.2 Inverting PWM with TOP at MAX(0xFF)

Frequency is chosen by PRESCALAR and Duty cycle by **OCR0A** and/or **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 001 for Phase Corrected PWM Mode with TOP at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0A[1:0]*** and/or ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make outputs *OC0A* and/or *OC0B* pins to generate PWM by comparing between **OCR0A** and/or **OCR0B** respectively. That is for Inverting, ***COM0x[1:0]*** is written 11.
- Next, the duty cycle value is loaded into **OCR0A** and/or **OCR0B** register for *OC0A* and/or *OC0B* bits.
- 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.
- Finally, Timer is started by setting ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 10% duty cycle *OC0A* and 75% duty cycle*OC0B* pins are shown assuming .
  - 0x19 for OCR0A.
  - 0xC0 for OCR0B.



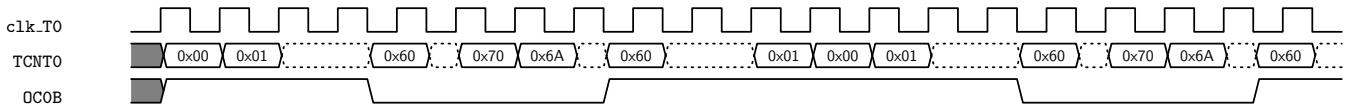
```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 001
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<3);

/* in timer0_phase_pwm_top_max, only two possiblites are there for COM0B[1:0] and COM0A[1:0] i.e) 10(Inver

// 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
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A | (1<<7);
TCCR0A = TCCR0A & ~(1<<6);

// which is reflected in PD65
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A & ~(1<<4);

/* we use overflow flag -- which is set at every time TCN0 reaches TOP here 0xFF
here, we toggle an led(PC0) at every overflow interrupt - this led(PC0) would give the frequency of PWM be
Also, we set the other leds(PC1 and PC2) so that they are make one when TCN0 reaches 0x00 */
// Enable Interrupt when TCN0 overflows TOP - here 0xFF
//  TOV0 bit is enabled
TIMSK0 = TIMSK0 | (1<<0);


// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till max(0xFF), we can choose OCR0A and OCR0B to any value below max(0xFFF)
OCR0A = 0x19; // for 10% duty clcle
OCR0B = 0xC0; // for 75% duty clcle

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.4.3 Non-Inverting PWM with TOP at OCR0A

Frequency is chosen by **OCR0A** and Duty cycle by **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 101 for Phase Corrected PWM Mode with OCR0A at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make output *OC0B* pins to generate PWM by comparing between **OCR0B** respectively. That is for Non-Inverting, ***COM0B[1:0]*** is written 10.
- The frequency of duty cycle is loaded into **OCR0A** register.
- Next, the duty cycle value is loaded into **OCR0B** register for *OC0B* bits.
- Also, the ***OCIE0B*** bits of **TIMSK0** 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 ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 85% duty cycle(0x60) *OC0B* pins are shown assuming .

- 0x70 for OCR0A.
- 0x60 for OCR0B.

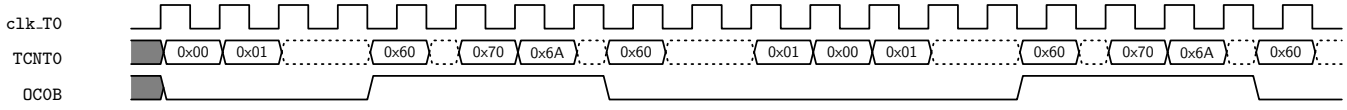



```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 101
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B | (1<<3);

// here we set COM0A[1:0] as 10 for non-inverting
// which is reflected in PD5
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A & ~(1<<4);

// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till OCR0A, we can choose OCR0B to any value below OCR0A
OCR0A = 0x70; // for freqeuncy
OCR0B = 0x60; // for pwm duty cylc

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.4.4 Inverting PWM with TOP at OCR0A

Frequency is chosen by **OCR0A** and Duty cycle by **OCR0B** register.

- First, ***WGM0[2:0]*** bits are configured as 101 for Phase Corrected PWM Mode with OCR0A at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0B[1:0]*** bits of **TCCR0A** register are configured to make output *OC0B* pins to generate PWM by comparing between **OCR0B** respectively. That is for Inverting, ***COM0B[1:0]*** is written 11.
- The frequency of duty cycle is loaded into **OCR0A** register.
- Next, the duty cycle value is loaded into **OCR0B** register for *OC0B* bits.
- Also, the ***OCIE0B*** bits of **TIMSK0** 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 ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for PWM on 85% duty cycle(0x60) *OC0B* pins are shown assuming .
  - 0x70 for OCR0A.
  - 0x60 for OCR0B.

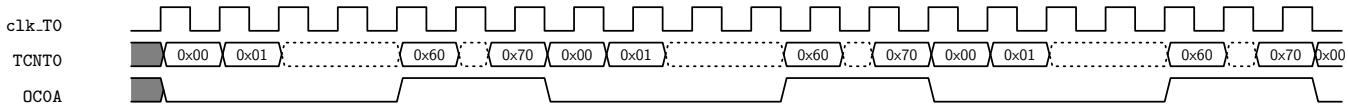


```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 101
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B | (1<<3);

// here we set COM0A[1:0] as 11 for inverting
// which is reflected in PD5
// COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
TCCR0A = TCCR0A | (1<<5);
TCCR0A = TCCR0A | (1<<4);

// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till OCR0A, we can choose OCR0B to any value below OCR0A
OCR0A = 0x70; // for freqeuncy
OCR0B = 0x60; // for pwm duty cylc

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.4.5 Toggling mode square Wave

Frequency is chosen by **OCR0A** register.

- First, ***WGM0[2:0]*** bits are configured as 101 for Phase Corrected PWM Mode with OCR0A at MAX in **TCCR0A** and **TCCR0B** registers.
- Next, ***COM0A[1:0]*** bits of **TCCR0A** register are configured to make output *OC0A* pins to generate PWM by comparing between **OCR0A**. That is for Toggling square wave ***COM0A[1:0]*** is written 01.
- The frequency of duty cycle is loaded into **OCR0A** register.
- Also, the ***OCIE0A*** bits of **TIMSK0** 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 ***CS0[2:0]*** bit as needed prescalar in **TCR0B** register.
- The timing for squared wave on *OC0A* pins are shown assuming.
  - 0x70 for OCR0A.




```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 101
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B | (1<<3);

// here we set COM0B[1:0] as 01 for toggling of OC0A
// which is reflected in PD6
// COM0A[1](bit7) from TCCR0A, COM0A[0](bit6) from TCCR0A
TCCR0A = TCCR0A & ~(1<<7);
TCCR0A = TCCR0A | (1<<6);
```

```
// Next we set values for OCR0A and OCR0B
// Since, TCNT0 goes till OCR0A, we can choose OCR0B to any value below OCR0A
OCR0A = 0x70; // for freqeuncy

// start the timer by selecting the prescalr
//  use the same clock from I/O clock
//  CS0[2:0] === 001
// CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
TCCR0B = TCCR0B | (1<<0);
TCCR0B = TCCR0B & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<2);

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

7.4.6 Application I - PWM generation

```
void Timer0_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;

        // MOde of operation to phase_corrected_phase_top_max Mode -- WGM0[2:0] === 101
        // WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
        TCCR0A = TCCR0A | (1<<0);
        TCCR0A = TCCR0A & ~(1<<1);
        TCCR0B = TCCR0B | (1<<3);

        // which is reflected in PD5
        // COM0B[1](bit5) from TCCR0A, COM0B[0](bit4) from TCCR0A
        TCCR0A = TCCR0A | (1<<5);
        TCCR0A = TCCR0A & ~(1<<4);

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

                OCR0A = 0;
                // start timer by setting the clock prescalar
                //  use the same clock from I/O clock
                //  CS0[2:0] === 001
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B & ~(1<<0);
                TCCR0B = TCCR0B & ~(1<<1);
                TCCR0B = TCCR0B & ~(1<<2);
        }
        else if((3 < total_time)  && (total_time <= 16))
        {
                OCR0A = ((total_time * 16) >> 0) - 1;
                OCR0B = ((on_time_us * 16) >> 0) - 1;
                // start timer by setting the clock prescalar
                //  use the same clock from I/O clock
                //  CS0[2:0] === 001
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B | (1<<0);
                TCCR0B = TCCR0B & ~(1<<1);
                TCCR0B = TCCR0B & ~(1<<2);
        }
        else if((16 < total_time)  && (total_time <= 128))
        {
                OCR0A = ((total_time * 16) >> 3) - 1;
```

```
                OCR0B = ((on_time_us * 16) >> 3) - 1;
                // start timer by setting the clock prescalar
                //  dived by 8 from I/O clock
                //  CS0[2:0] === 010
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B & ~(1<<0);
                TCCR0B = TCCR0B | (1<<1);
                TCCR0B = TCCR0B & ~(1<<2);
        }
        else if((128 < total_time)  && (total_time <= 1024))
        {
                OCR0A = ((total_time * 16) >> 6) - 1;
                OCR0B = ((on_time_us * 16) >> 6) - 1;
                // start timer by setting the clock prescalar
                //  dived by 64 from I/O clock
                //  CS0[2:0] === 011
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B | (1<<0);
                TCCR0B = TCCR0B | (1<<1);
                TCCR0B = TCCR0B & ~(1<<2);

        }
        else if((1024 < total_time)  && (total_time <= 4096))
        {
                OCR0A = ((total_time * 16) >> 8) - 1;
                OCR0B = ((on_time_us * 16) >> 8) - 1;
                // start timer by setting the clock prescalar
                //  divide by256 from I/O clock
                //  CS0[2:0] === 100
                // CS0[2](bit2) from TCCR0B,CS0[1](bit1) from TCCR0B,CS0[0](bit0) from TCCR0B
                TCCR0B = TCCR0B & ~(1<<0);
                TCCR0B = TCCR0B & ~(1<<1);
                TCCR0B = TCCR0B | (1<<2);

        }
        else if(total_time > 4096)
        {
                // dont' cross more than 4.096ms
        }
}
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
        Timer0_PhaseCorrectedPWMGeneration(on_time_us, total_time_us - on_time_us);
}
```