ATmega328P I/O Ports

Narendiran S

July 25, 2020

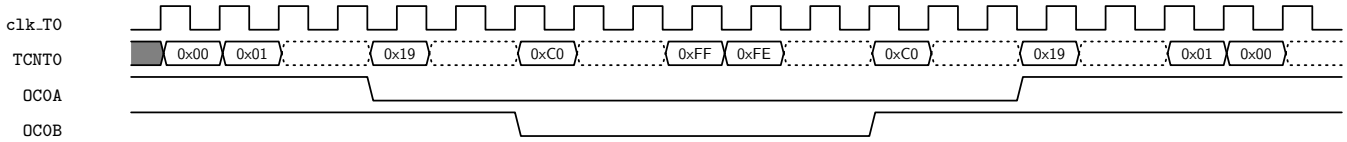
0.1 Phase Corrected PWM Mode

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

0.1.1 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.



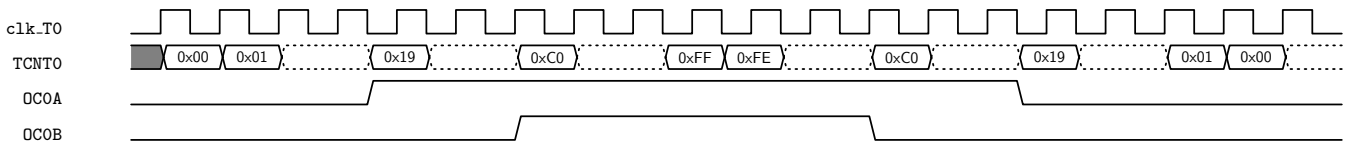

```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 001
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<3);

/* 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();
```

0.1.2 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.



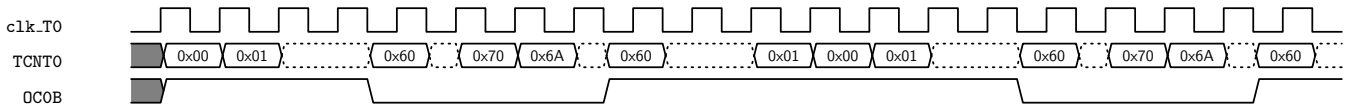
```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 001
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B & ~(1<<3);

/* 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();
```

0.1.3 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.

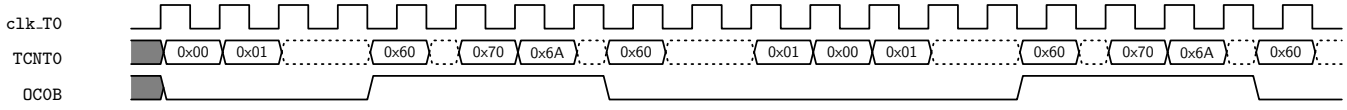



```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 101
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B | (1<<3);

// 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();
```

0.1.4 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.

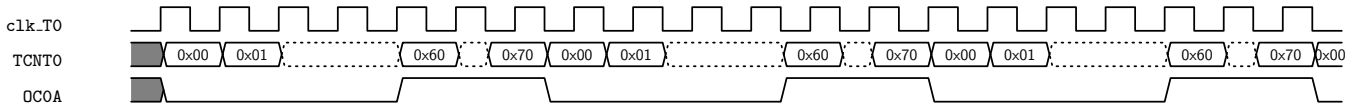


```
// MOde of operation to phase_corrected_pwm_top_max Mode -- WGM0[2:0] === 101
// WGM0[2](bit3) from TCCR0B, WGM0[1](bit1)  from TCCR0A, WGM0[0](bit0)  from TCCR0A
TCCR0A = TCCR0A | (1<<0);
TCCR0A = TCCR0A & ~(1<<1);
TCCR0B = TCCR0B | (1<<3);

// 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();
```

0.1.5 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();
```

0.1.6 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);
}
```