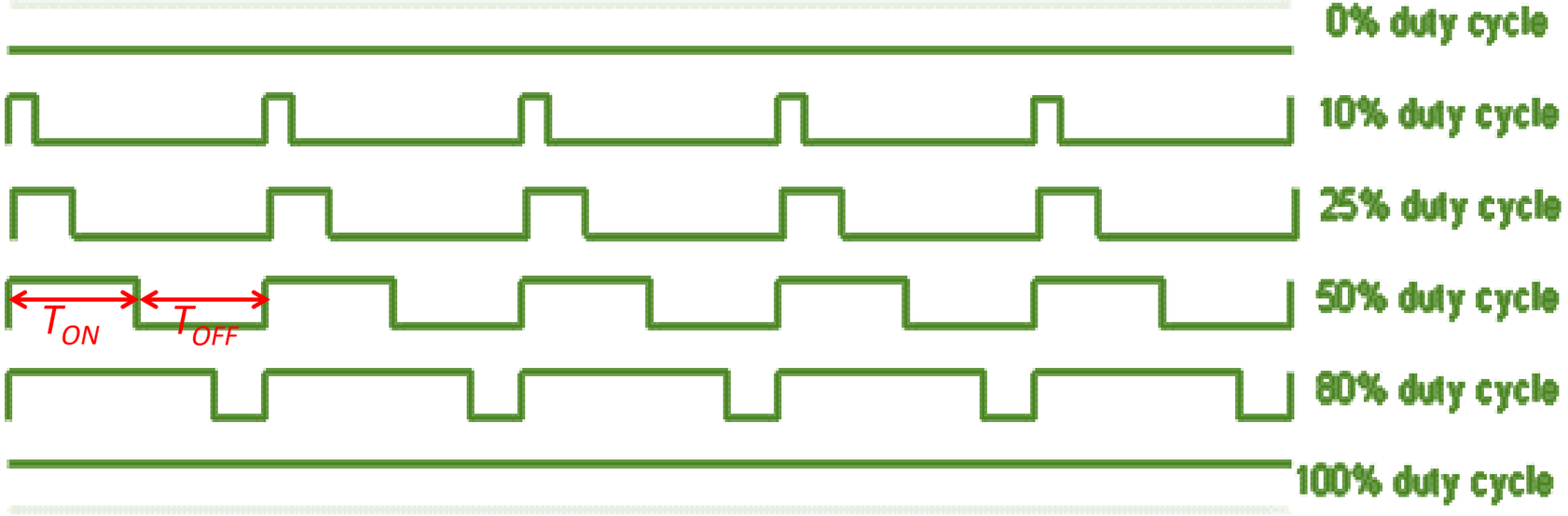
Pulse Width Modulation (PWM)

Department of EEE

PWM

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between the full V_{CC} of the board (e.g., 5 V on UNO) and off (0 Volts). The duration of "on time" is called the **pulse width**. To get varying analog values, the pulse width can be varied. If this on-off pattern is repeated fast enough with an LED, for example, the result is as if the signal is a steady voltage between 0 and V_{CC} controlling the brightness of the LED.



Duty Cycle is defined as the ratio of ON pulse duration to the time period.

$$\boldsymbol{D = \frac{T_{ON}}{T} \times 100\%}$$

Here, T_{ON} = ON pulse duration

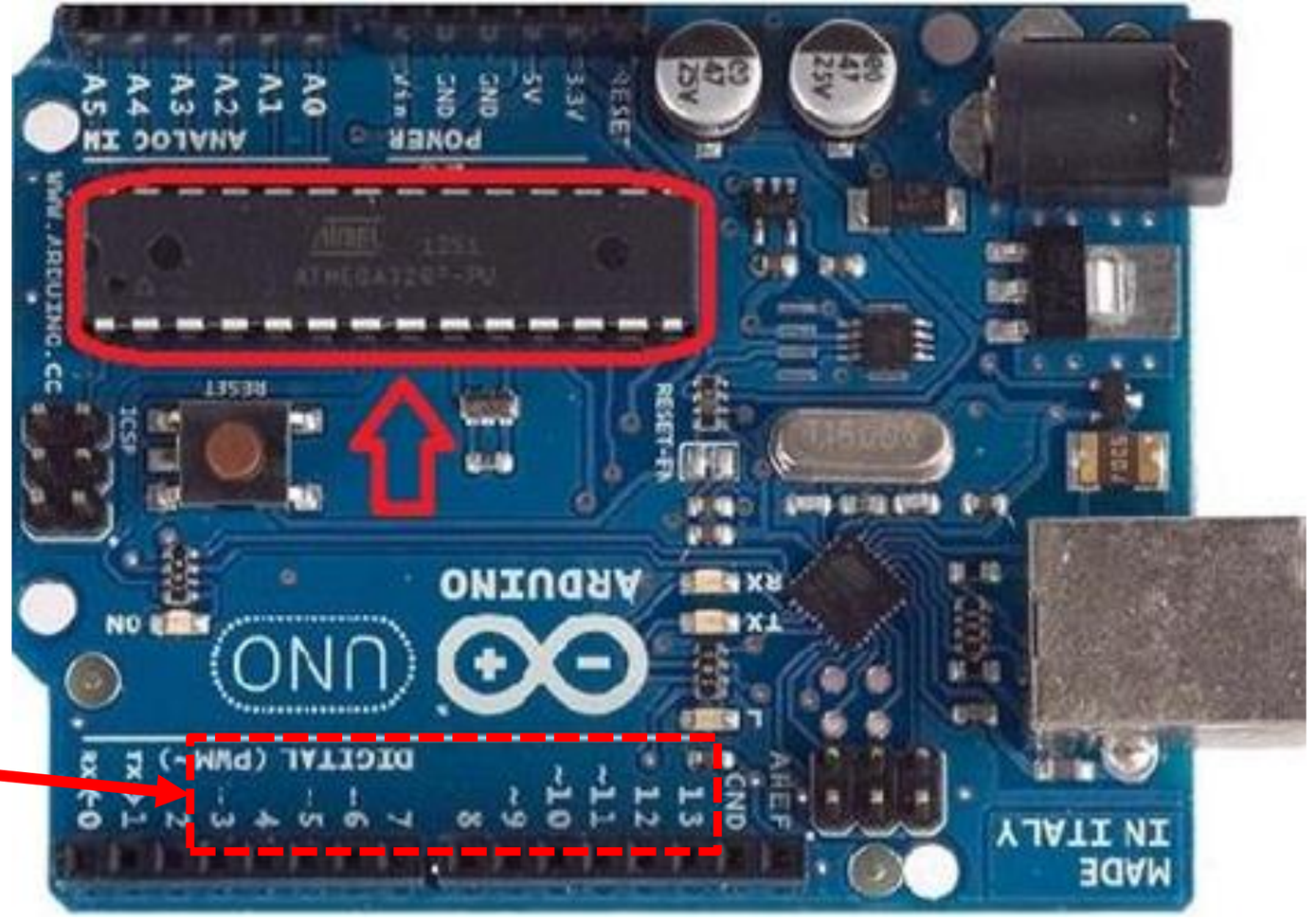
T = Timer period = T_{ON} + T_{OFF} ,where T_{OFF} = OFF pulse duration

PWM

The Arduino's programming language makes PWM easy to use; simply call the built-in function **analogWrite(pin, dutyCycle)**, where **dutyCycle** is a value **from 0 to 255**, and **pin** is one of the **PWM pins (3, 5, 6, 9, 10, or 11).**

The **analogWrite()** function provides a simple interface to the hardware PWM, but doesn't provide any control over frequency.

Note the tilde ~ sign with the pin numbers of the image.

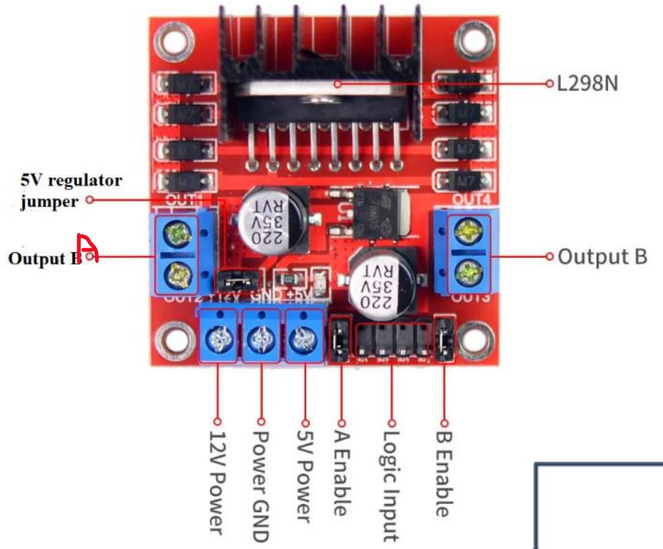


Sample Code for PWM control (ref: PWM lab)

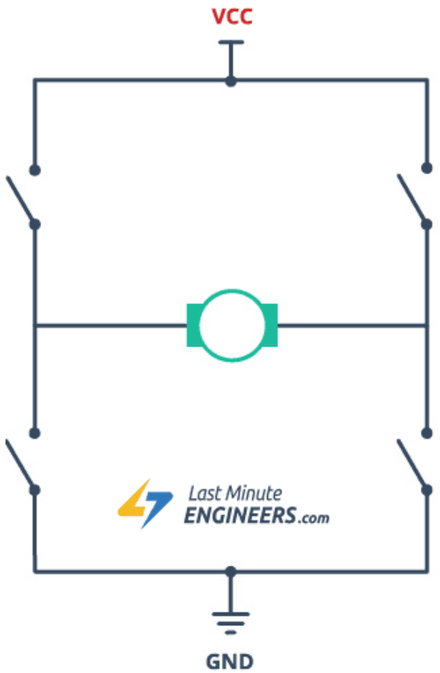
```
int in1 = 9; //Declaring the pins where in1 in2 from the driver are wired
int in2 = 8; //here they are wired with D9 and D8 from Arduino
int enA = 10; //And we add the pin to control the speed after we remove its jumper
              //Make sure it's connected to a pin that can deliver a PWM signal

void setup() {
  pinMode(in1, OUTPUT); //Declaring the pin modes, obviously they're outputs
  pinMode(in2, OUTPUT);
  pinMode(enA, OUTPUT);
}
//Speed range (0-255)

void TurnMotorA(){
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  analogWrite(enA,100);
}

void TurnOFFA(){
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  analogWrite(enA,0);
}
```

```
// in1 and in2 = for logic input pin
// HIGH and LOW = one direction
// LOW and HIGH = another direction
// LOW and LOW = motor stop
// HIGH and HIGH = motor stop
```

```
void TurnMotorA2(){
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  analogWrite(enA,250);
}
void loop() {
  TurnMotorA();
  delay(2000);
   TurnOFFA();
  delay(2000);
   TurnMotorA2();
  delay(4000);
   TurnOFFA();
  delay(2000);
}
```



- **PWM** – to control speed
- **H-Bridge** – to control the spinning direction



```
// Motor A connections
int enA = 9;
int in1 = 8;
int in2 = 7;

void setup() {
// Set all the motor control pins to outputs
pinMode(enA, OUTPUT);
pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);

// Turn off motors - Initial state
digitalWrite(in1, LOW);
digitalWrite(in2, LOW); }

void loop() {
directionControl();
delay(1000);
speedControl();
delay(1000); }

// This function lets you control spinning direction of motor
void directionControl() {
// Set motors to maximum speed
// For PWM maximum possible values are 0 to 255
analogWrite(enA, 255);
// Turn on motor A
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
delay(2000);
// Now change motor directions
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
delay(2000);

// Turn off motors
digitalWrite(in1, LOW);
digitalWrite(in2, LOW); }

// This function lets you control speed of the motors
void speedControl() {
// Turn on motors
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
// Accelerate from zero to maximum speed
for (int i = 0; i < 256; i++) {
analogWrite(enA, i);
delay(20); }

// Deaccelerate from maximum speed to zero
for (int i = 255; i >= 0; --i) {
analogWrite(enA, i);
delay(20); }
// Now turn off motors
digitalWrite(in1, LOW);
digitalWrite(in2, LOW);
}
```

```
// in1 and in2 = for logic input pin
// HIGH and LOW = one direction
// LOW and HIGH = another direction
// LOW and LOW = motor stop
// HIGH and HIGH = motor stop
```

PWM

The **main PWM modes** are "Fast PWM" and "Phase-correct PWM".

There are **two Fast PWM modes** for Timer/Counter 0 unit. These are **modes 3 and 7**, which are selected using the waveform generation mode bits (WGM02, WGM01, and WGM00). The WGM01 and WGM00 bits are located in the TCCR0A register. The WGM02 bit is located in the TCCR0B register.

There are **2 types** of Fast PWM modes: (a) Non-inverting and (b) Inverting Fast PWM

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

PWM

The difference between the mode 3 and mode 7 Fast PWMs is their TOP values.

For mode 3, the TOP value is fixed (0xFF), whereas that for mode 7 is OCRnA value.

So, if mode 7 is used then we have to **load a count value** into the **OCRnA register**.

Once a mode is selected, the timer/counter starts counting from BOTTOM to TOP value and when the TOP value is reached the counting is repeated from BOTTOM.

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

The table in the left shows the **WGM bits combination** for mode 3 and mode 7.

Fast PWM Mode: Setting Modes

In the fast PWM mode, the Compare Output Mode bits COM0A1, COM0A0 (in the TCCR0A register) for the output at OC0A and bits COM0B1, COM0B0 (in the TCCR0A register) for the output at OC0B, are used to configure the output as either non-inverting or inverting mode of operation. Table 12-3 shows how to select COM0A1, COM0A0 bits to generate non-inverting and inverting mode's fast PWM waveform.

Table 12-3. Compare Output Mode, Fast PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on Compare Match, clear OC0A at BOTTOM, (inverting mode).

Fast PWM Mode: Setting Modes

In the fast PWM mode, the compare unit allows the generation of PWM waveforms on the OC0x pins. Table 12.6 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to the fast PWM mode.

Table 12-6. Compare Output Mode, Fast PWM Mode(1)

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on Compare Match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on Compare Match, clear OC0B at BOTTOM, (inverting mode).

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 101 for more details.

Fast PWM Mode: Setting Modes

Setting the COM0x1:0 bits to **10** will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0x1:0 to **11**.

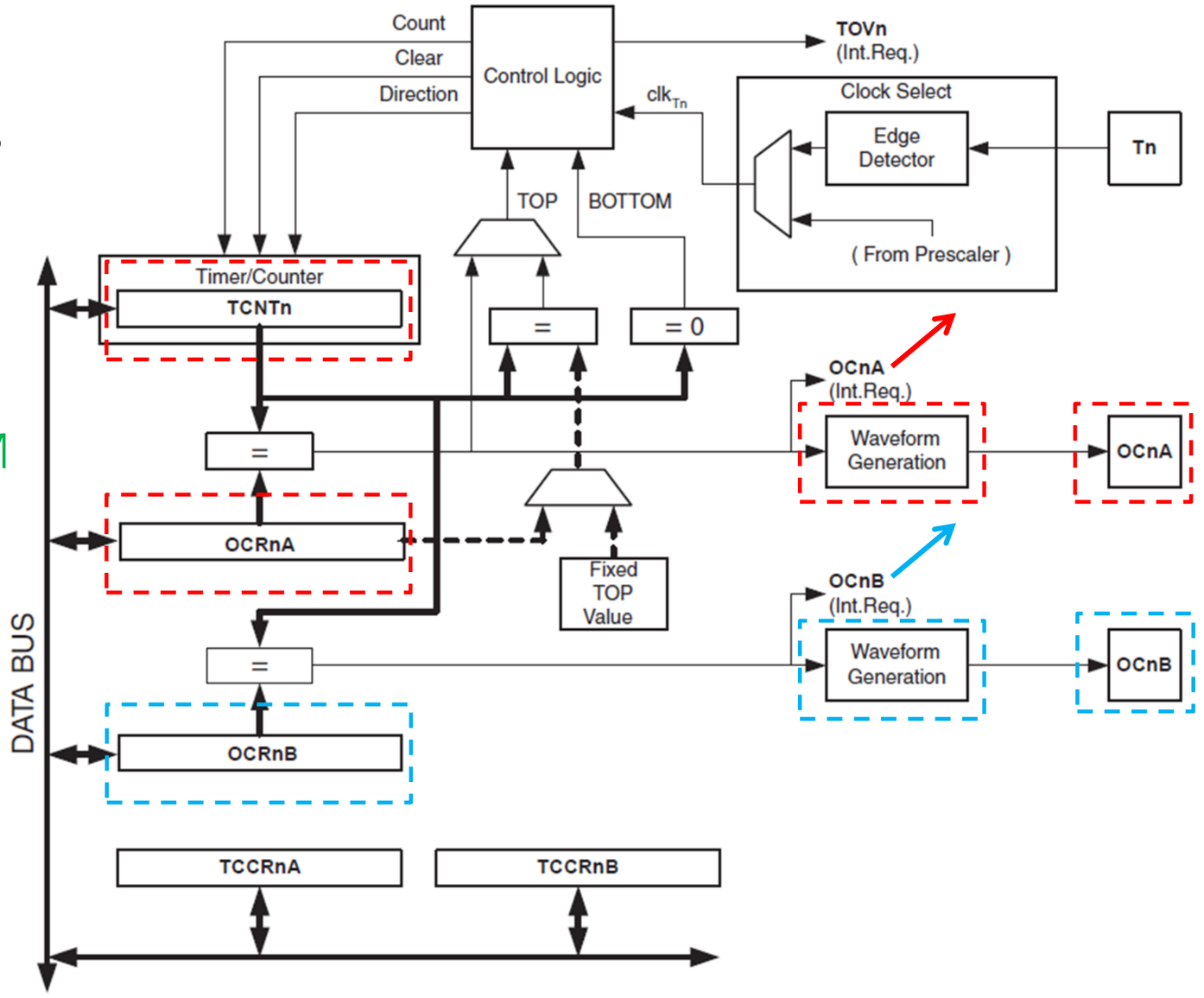
Setting the COM0A1:0 bits to **01** allows the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (see Table 12-6).

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fast PWM Mode

- The **Fast Pulse Width Modulation** or **Fast PWM Mode (WGM02:0 = 3 or 7)** provides a high-frequency PWM waveform generation option. The Fast PWM differs from the other PWM option in its single-slope operation.
- The counter starts counting from BOTTOM to TOP and then restarts from BOTTOM.
- In **Non-inverting Compare Output Mode**, the Output Compare (OC0x) is **cleared** on the compare match between TCNT0 and OCR0x, and **set** at BOTTOM.
- In **Inverting Compare Output Mode**, the output is **set** on compare match and **cleared** at BOTTOM.



Fast PWM Mode

- Due to the single-slope operation, the operating frequency of the Fast PWM mode can be **twice as high as** the phase correct PWM mode that uses dual-slope operation.
- This high frequency makes the fast PWM mode well suited for **power regulation, rectification, and DAC applications**.
- High frequency allows physically small-sized external components (coils, capacitors), and therefore **reduces** the total system cost.
- In fast PWM mode, the counter is incremented until the counter value **matches the TOP** value, the register TCNTn counts from bottom value to maximum value stored in the register OCRn. The counter is then cleared/reset to zero at the following timer clock cycle. If the timer is configured in non-inverting mode, PWM output pin (OCn) goes low when the value of the above two registers matches. The OCn pin becomes high when the TCNTn register reaches at bottom value. In inverting mode, OCn pin behaves opposite to non-inverting mode.

Fast PWM Mode

- The timing diagram for the Fast PWM Mode is shown in Figure 12.6.
- The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation.
- The diagram includes **non-inverted and inverted PWM outputs**.

The **small horizontal line marks** on the TCNT0 slopes represent compare matches between OCR0x and TCNT0.

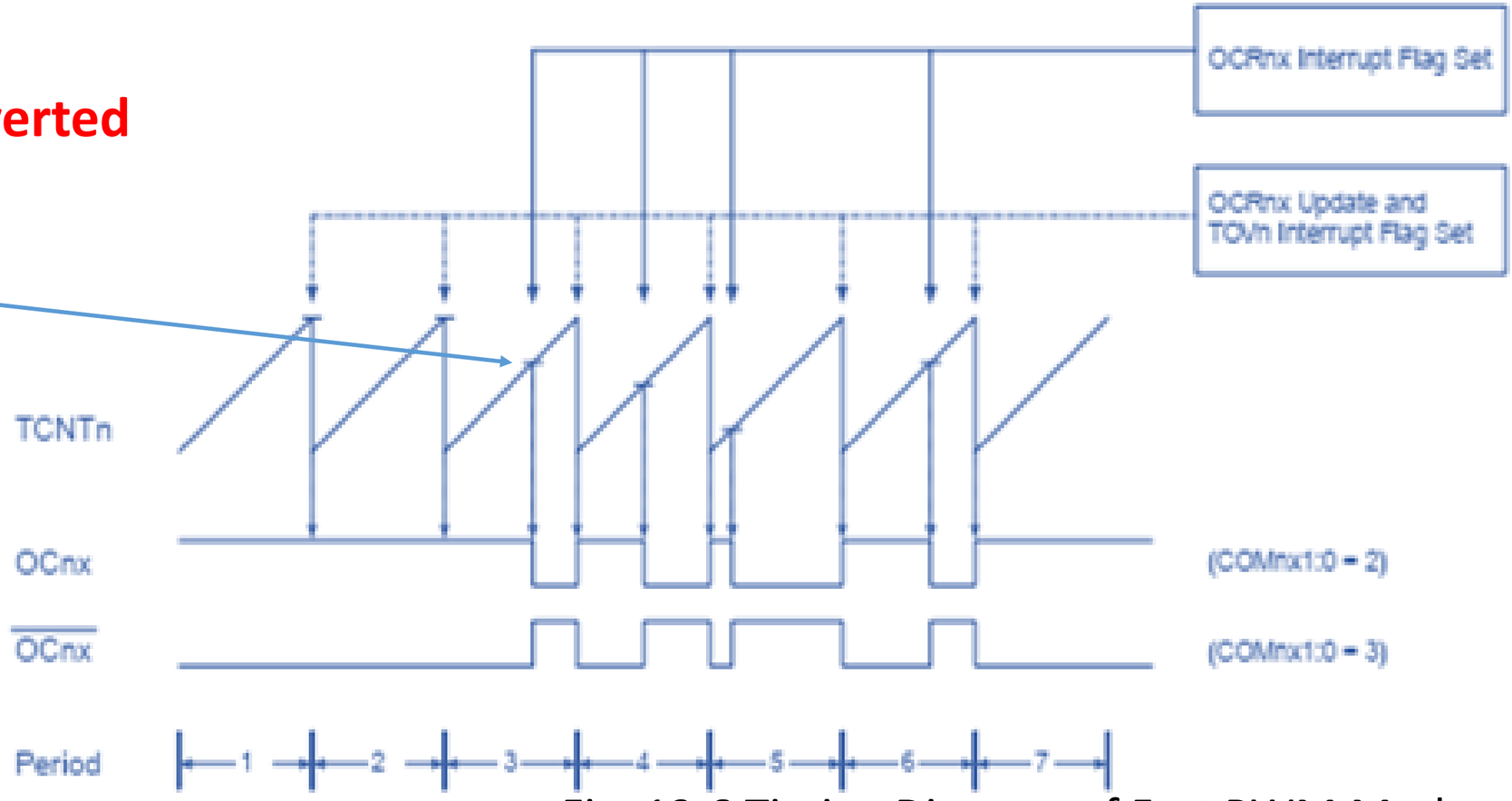


Fig. 12.6 Timing Diagram of Fast PWM Mode

Fast PWM Mode: Output Frequency

- The actual OC0x value will only be visible on the **port pin** if the data direction for the port pin is **set** as **output**.
- The PWM waveform is generated by **setting (or clearing) the OC0x Register** at the **compare match between OCR0x and TCNT0**, and clearing (or setting) the OC0x Register at the timer clock cycle the counter is cleared (i.e., from TOP to BOTTOM).
- The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\_IO}}{N \times 256}$$

- The *N* variable represents the pre-scale factor (1, 8, 64, 256, or 1024).
- The waveform generated will have a maximum frequency of $f_{OC0} = \frac{f_{clk\_IO}}{2}$ when OCR0A is **set to zero**, otherwise the formula would be, $f_{OC0} = \frac{f_{clk\_IO}}{2 \times N \times (1 + OCRnx)}$

Fast PWM Mode: Output Frequency

- **Non-Inverting Fast PWM Duty Cycle**
- The duty cycle of the **Non-Inverting mode** Fast PWM signal is calculated using the following formula.

$$\boldsymbol{OCR0x = \frac{256D}{100} - 1; x = A \; or \; B}$$

- where, D is the Duty cycle that range from 0% to 100%. This duty cycle value is the count value that has to be loaded into the OCR0A (or OCR0B) register.
- **Example:** A PWM signal is to have 75% duty cycle. Compute the value for OCR0A.

$$\boldsymbol{OCR0A = \frac{256 \times 75}{100} - 1 = 192 - 1 = 191}$$

Fast PWM Mode: Output Frequency

- **Inverting Fast PWM Duty Cycle**
- The formula for frequency computation is the same as for non-inverting fast PWM. But, the duty cycle of the **Inverting mode** Fast PWM signal is calculated using the following formula.

$$OCR0x = 255 - \frac{256D}{100}; \quad x = A \text{ or } B$$

- where, D is the Duty cycle that range from 0% to 100%. This duty cycle value is the count value that has to be loaded into the OCR0A (or OCR0B) register.
- **Example:** A PWM signal is to have 75% duty cycle. Compute the value for OCR0A.

$$OCR0A = 255 - \frac{256 \times 75}{100} = 255 - 192 = 63$$

Phase correct PWM Mode

- The phase correct PWM mode (**WGM02:0 = 1 or 5**) provides a high-resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a **dual-slope operation**. The counter counts repeatedly from BOTTOM to TOP and then from TOP to BOTTOM.
- In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the compare match between TCNT0 and OCR0x while up counting, and set on the compare match while down counting.
- In inverting Output Compare mode, the operation is inverted.
- The dual-slope operation has a lower maximum operation frequency than the single-slope operation.
- However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

Phase correct PWM Mode

- In phase correct PWM mode, the counter is incremented until the counter value matches TOP. When the counter reaches TOP, it changes the count direction.
- The TCNT0 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on **Figure 12.7**.
- The TCNT0 value is in the timing diagram shown as a histogram for illustrating the **dual-slope operation**.
- The diagram includes non-inverted and inverted PWM outputs.
- The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0x and TCNT0.

Phase correct PWM Mode

- The diagram includes non-inverted and inverted PWM outputs.
- The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0x and TCNT0.
- The Timer/Counter Overflow Flag (TOV0) 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.

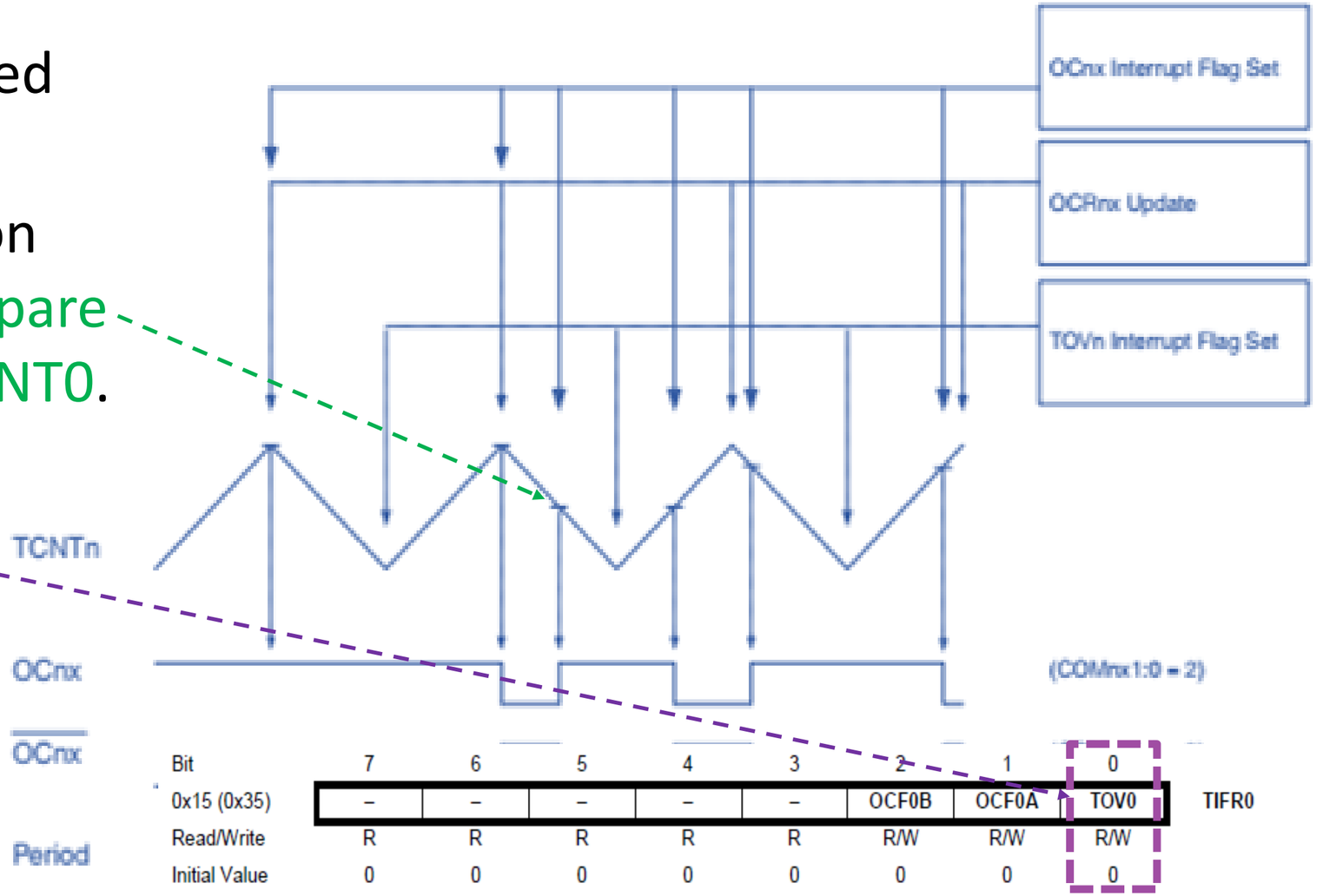


Figure 12.7 Timing Diagram of the Phase Correct PWM Mode

Phase Correct PWM Mode

In Phase Correct PWM mode, the compare unit allows the generation of PWM waveforms on the OC0x pins. Table 12.6 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to Phase Correct PWM mode.

Table 12-7. Compare Output Mode, Phase Correct PWM Mode[1]

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on Compare Match when up-counting. Set OC0B on Compare Match when down-counting.
1	1	Set OC0B on Compare Match when up-counting. Clear OC0B on Compare Match when down-counting.

Phase Correct PWM Mode

- In phase correct PWM mode, the compare unit allows the generation of PWM waveforms on the OC0x pins. Setting the COM0x1:0 bits to **10** will produce a non-inverted PWM.
- An inverted PWM output can be generated by setting the COM0x1:0 to **11**.
- Setting the COM0A0 bits to **01** allows the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (Table 12-7).

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Phase correct PWM Mode

- The PWM waveform is generated by clearing (or setting) the OC0x Register at the compare match between OCR0x and TCNT0 when the counter increments, and setting (or clearing) the OC0x Register at compare match between OCR0x and TCNT0 when the counter decrements.
- The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk\_IO}}{N \times 510}$$

- The *N* variable represents the pre-scale factor (1, 8, 64, 256, or 1024).
- The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk\_IO}$.

Phase correct PWM Mode

- The **extreme values for the OCR0A Register represent special cases** when generating a PWM waveform output in the phase correct PWM mode.
- If the **OCR0A is set equal to BOTTOM**, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode.
- For inverted PWM, the output will have the opposite logic values.

Example to Self practice:

Calculate the PWM frequency for the output when using Fast PWM Mode and Phase Correct PWM Mode when f_{clk_IO} is **10 MHz** and the pre-scale factors are **256** or **1024.** Comment on the results afterward.

Solution:

The PWM frequency for the Fast PWM Mode:

$$f_{OCnxPWM} = \frac{f_{clk\_IO}}{N \times 256} = \frac{10MHz}{1024 \times 256} = 38.15 \text{ Hz}$$

The PWM frequency for the Phase Correct PWM Mode:

$$f_{OCnxPCPWM} = \frac{f_{clk\_IO}}{N \times 510} = \frac{10MHz}{1024 \times 510} = 19.15 \text{ Hz}$$

Now repeat for N=256 and comment afterwards.

Example for Practice

Use ATmega328p Timer 0 Fast PWM mode with TOP at OCR0A (mode 7) and toggle on OC0A pin. Calculate the PWM frequency. With simulation, the frequency changes with OCR0A value loaded, OCR0A = 100 and OCR0A = 200. Calculate the PWM frequency in this mode. The Pre-scaler value, N = 1, and the system clock frequency, f_{clk_IO} = 16 MHz.

The PWM frequency for the Fast PWM Mode:

$$f_{OC0APCPWM} = \frac{f_{clk\_IO}}{2N \times (1 + OCR0A)} = \frac{16MHz}{2 \times 1 \times 101} = 79.2 \text{ kHz}$$

$$f_{OC0APCPWM} = \frac{f_{clk\_IO}}{2N \times (1 + OCR0A)} = \frac{16MHz}{2 \times 1 \times 201} = 39.8 \text{ kHz}$$

Programming Arduino for Fast PWM

```
#ifndef F_CPU
#define F_CPU 8000000UL      // Clock frequency is 8 MHz
#endif
#include <avr/io.h>

int main() {
  // OC0B pin is set as PWM output pin by sending a HIGH to Port D's Data Direction Register, DDRD
  DDRD |= (1<<PD5);
  OCR0B= 191; // Load 191 into OCR0B for setting its duty cycle to 75% (See previous example)
   // Configure TCCR0A and TCCR0B register for (i) non-inverting (10), (ii) Fast PWM mode 3 (011), (iii) No
   // Pre-scalar (001) for frequency control. By default, 0s are there, but you may set them explicitly.
  TCCR0A |= (1 << COM0B1) | (1<<WGM01) | (1<<WGM00);
  TCCR0B |= (1<<CS00);

  while(1);
return 0;
}
```

Programming Arduino for Fast PWM

```
#ifndef F_CPU
#define F_CPU 8000000UL      // Clock frequency is 8 MHz
#endif
#include <avr/io.h>

int main() {
  // OC0B pin is set as PWM output pin by sending a HIGH to Port D's Data Direction Register, DDRD
  DDRD |= (1<<PD5);
  OCR0A = 200; // Top Value of 200 (must be equal or greater than the Duty Cycle)
  OCR0B= 191; // Load 191 into OCR0B for setting its duty cycle to 75% (See previous example)
   // Configure TCCR0A and TCCR0B register for (i) non-inverting (11), (ii) Fast PWM mode 7 (111), (iii) No
   // Pre-scalar (001) for frequency control. By default, 0s are there, but you may set them explicitly.
  TCCR0A |= (1 << COM0B1) | (1 << COM0A0) | (1<<WGM01) | (1<<WGM00);
  TCCR0B |= (1<<WGM02) | (1<<CS00);

  while(1);
return 0;
}
```

Programming Arduino for Fast PWM

```
#ifndef F_CPU
#define F_CPU 8000000UL      // Clock frequency is 8 MHz
#endif
#include <avr/io.h>

int main() {
  // OC0A pin is set as PWM output pin by sending a HIGH to Port D's Data Direction Register, DDRD
  DDRD |= (1<<PD6);
  OCR0A= 63; // Load 63 into OCR0A for setting its duty cycle to 75% (See previous example)
   // Configure TCCR0A and TCCR0B register for (i) inverting (11), (ii) Fast PWM mode 3 (011), (iii) No
   // Pre-scalar (001) for frequency control. By default, 0s are there, but you may set them explicitly.
  TCCR0A |= (1 << COM0A1) | (1 << COM0A0) | (1<<WGM01) | (1<<WGM00);
  TCCR0B |= (1<<CS00);

  while(1);
return 0;
}
```

Thanks for attending….