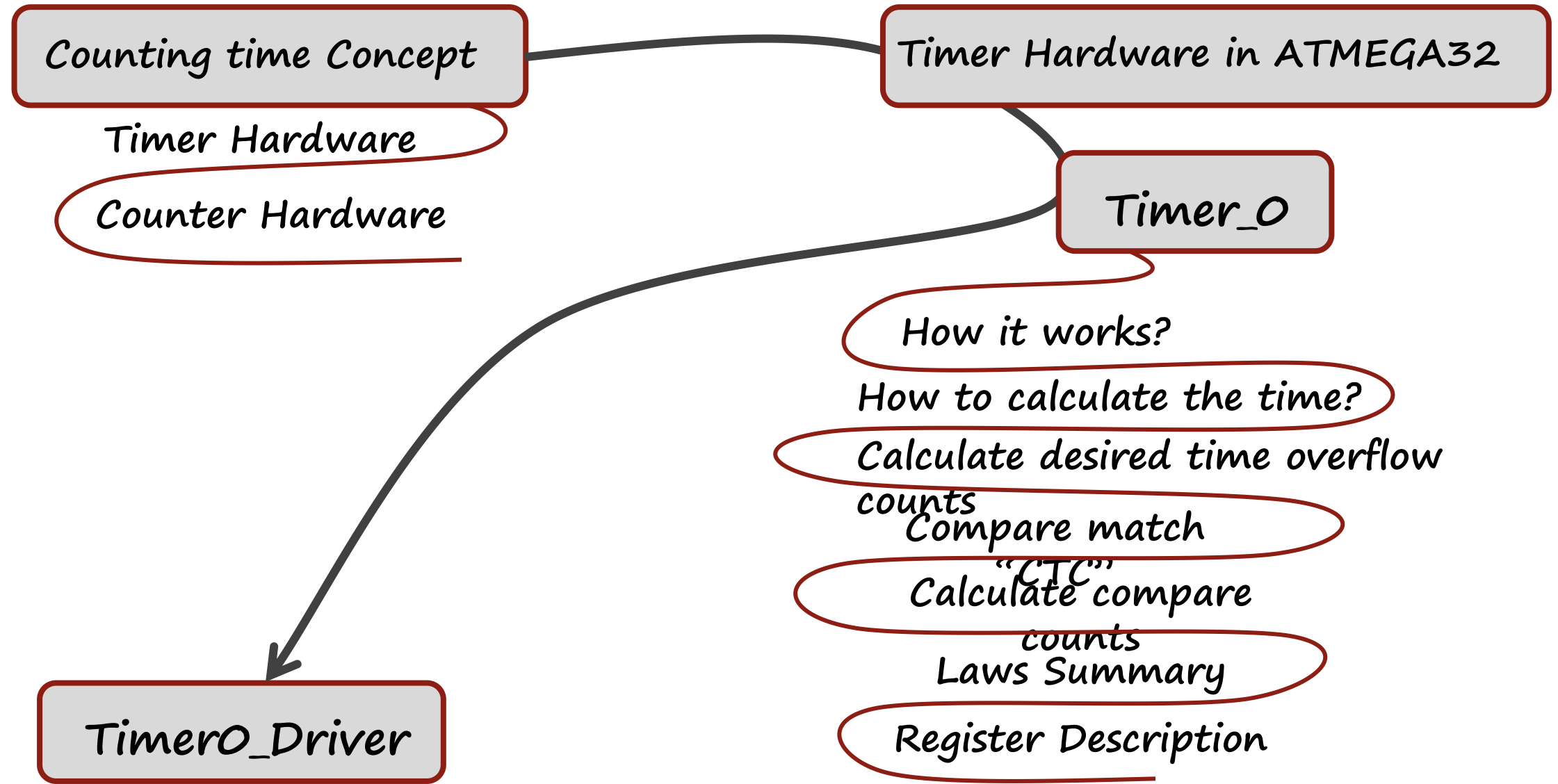


Interfacing Timers “Part one”

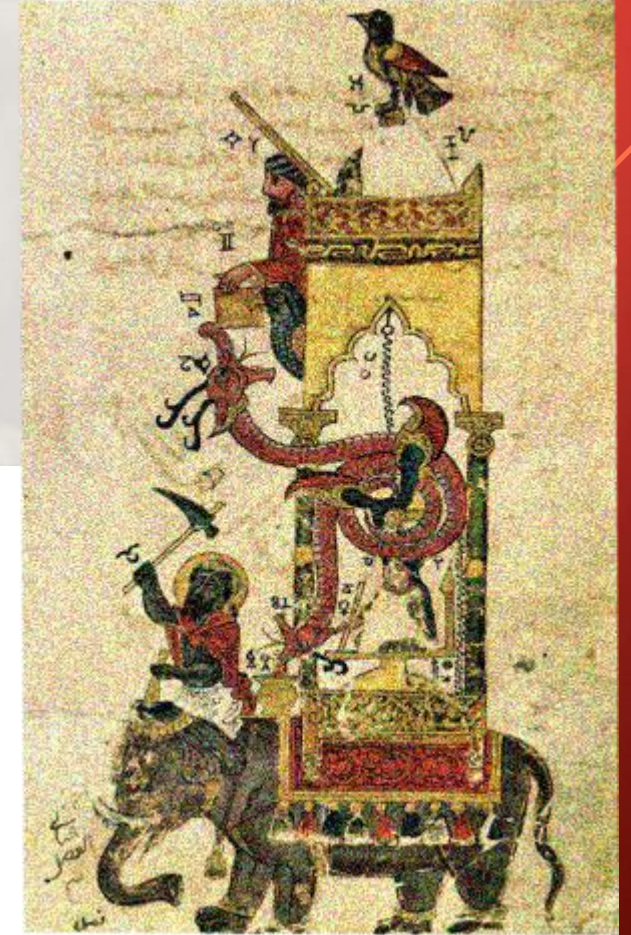
Content



Counting time

Concept

- In middle years, there was a muslim scientist called **Ibn Al-Jazari** who invented a mechanical clock depends on weight and water, it sounds and throw a metal ball into a container every half an hour, so if we counts the balls we can calculate the passed time: like if we find 7 balls, that means about three and half hours have been passed
- Depending the last concept, our **microcontroller** works depending on a Clock cycles, this cycle has a known rate, like our **MC** is **16 MHz**, so if we managed to count these cycles, and knowing the period of every cycle, we can calculate the time
- So, if we have a counter register that is increased by one every clock-cycle, if it is read, its value expresses how many clock-cycles are come, and we can count the time if we has the period of one cycle by multiplying the value by the period of one cycle



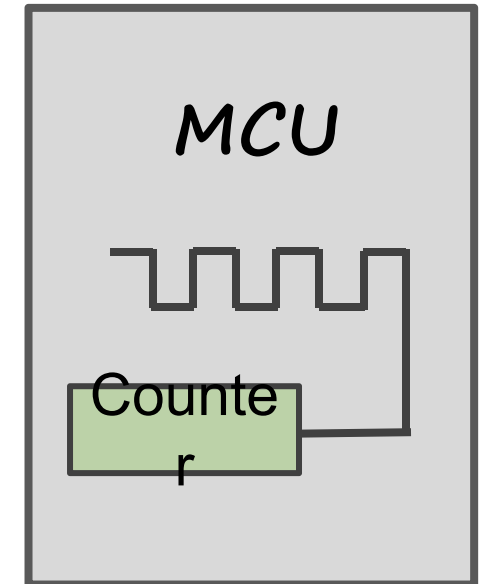
Timer

Hardware

According to the previous theory, any timer hardware depends on a counter register that is increased by one every clock cycle, it depends on the system's clock

It means that any timer is a counter “it will be declared later”, but triggering event of this counter is a clock

Timer can not be transferred to a counter because it internally depends on the system's clock, so no way to be a counter without an external pin



Counter

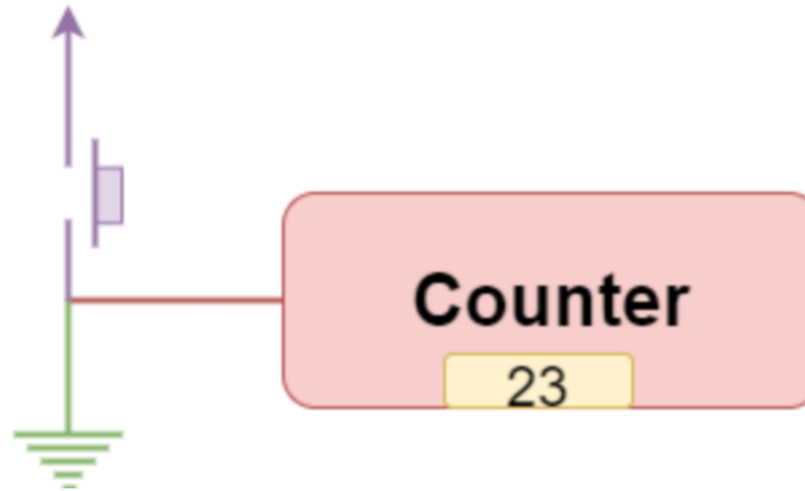
Hardware

It is a hardware circuit depends on increasing one into the counter register every a triggering event occurs

It can be triggered every rising edge of falling edge

It can be used to count any external event like counting how many presses on the switch like the following figure:

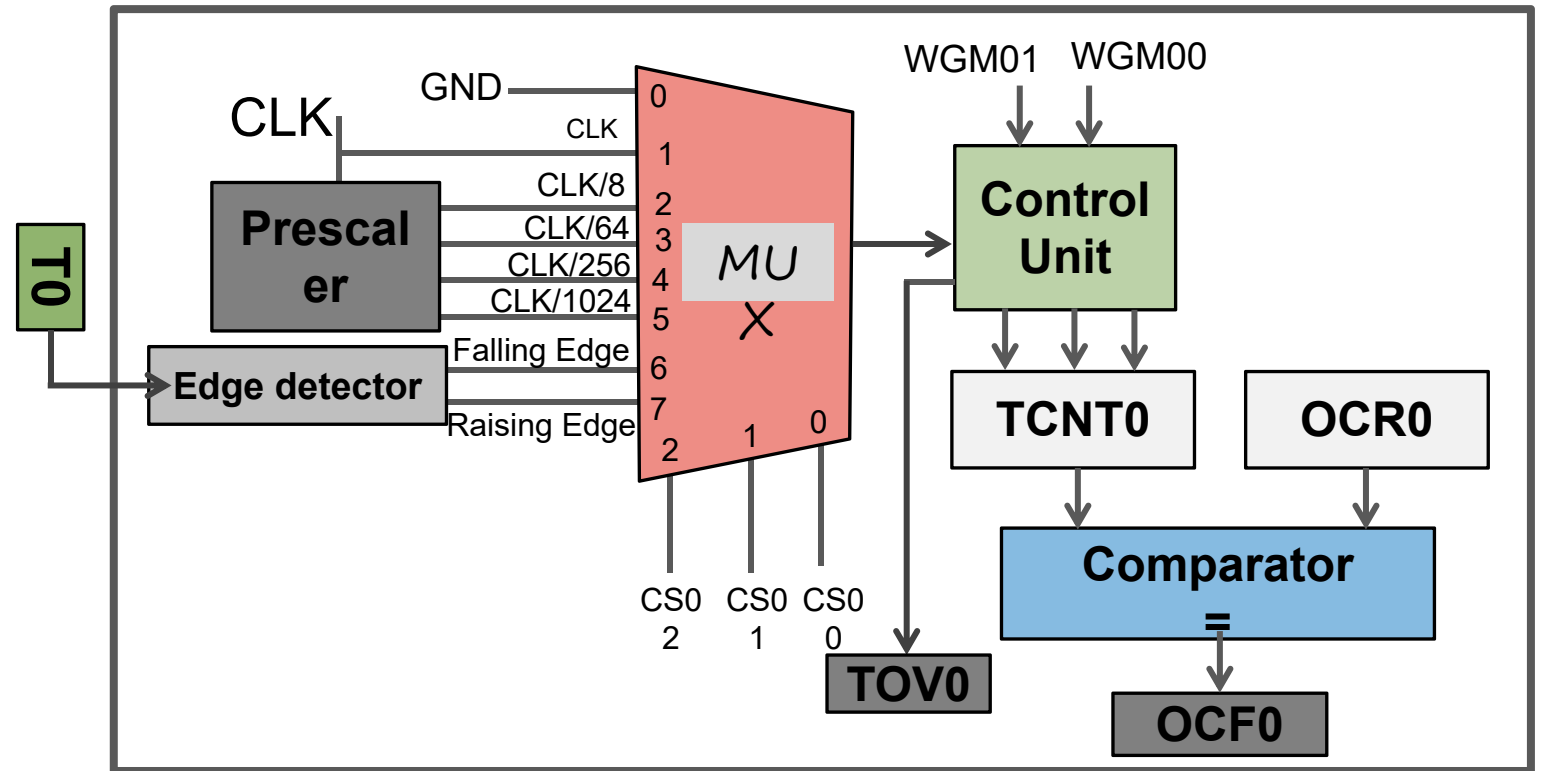
Any counter can be transferred to Timer by triggering it by a clock



Timer Hardware in ATMEGA32

We have three-
peripheral timers into
our microcontroller

TIMER0, 8-bit
resolution

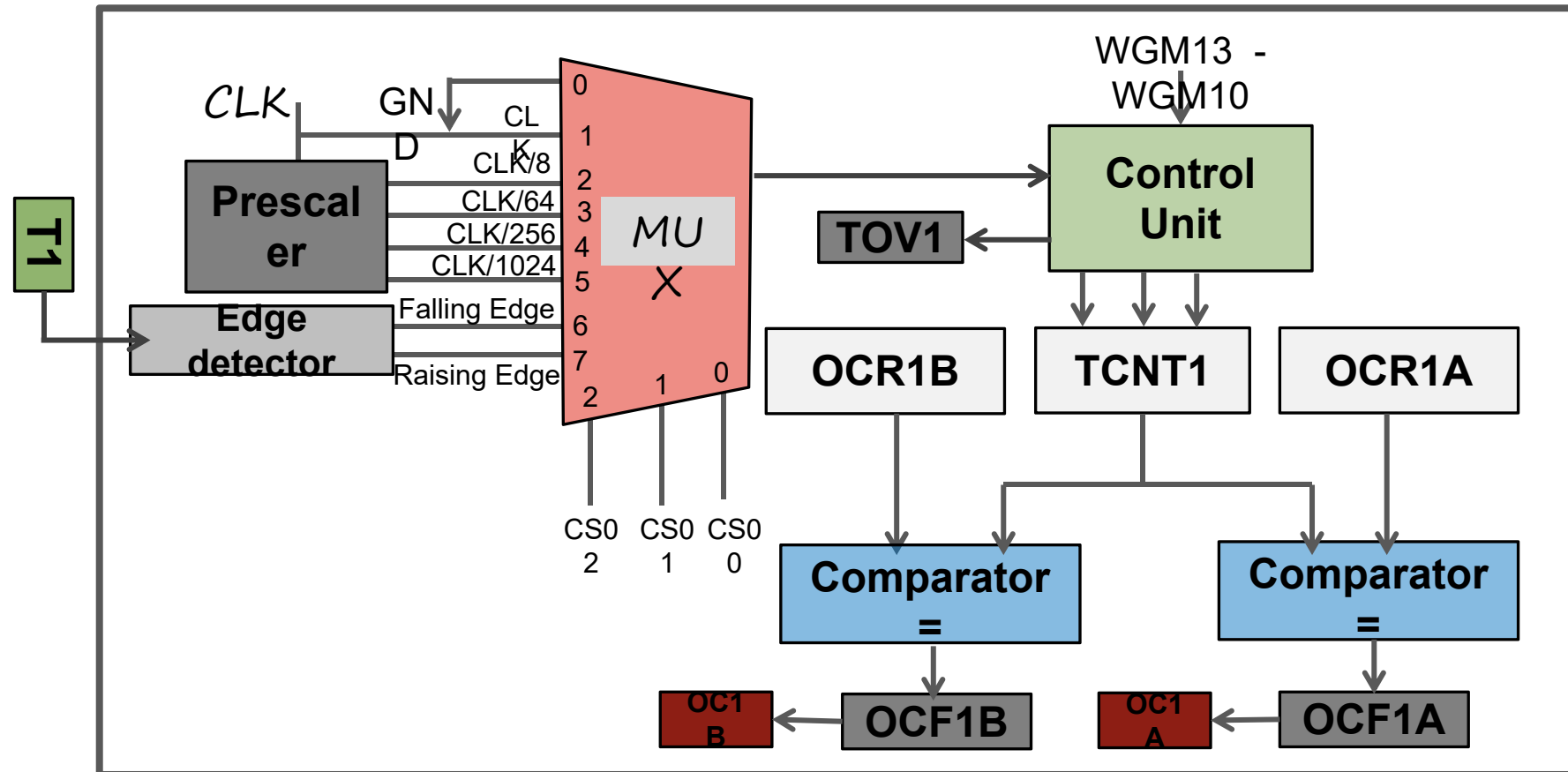


Timer Hardware in ATMEGA32

We have three-
peripheral timers into
our microcontroller

TIMER0, 8-bit
resolution

TIMER1, 16-bit resolution



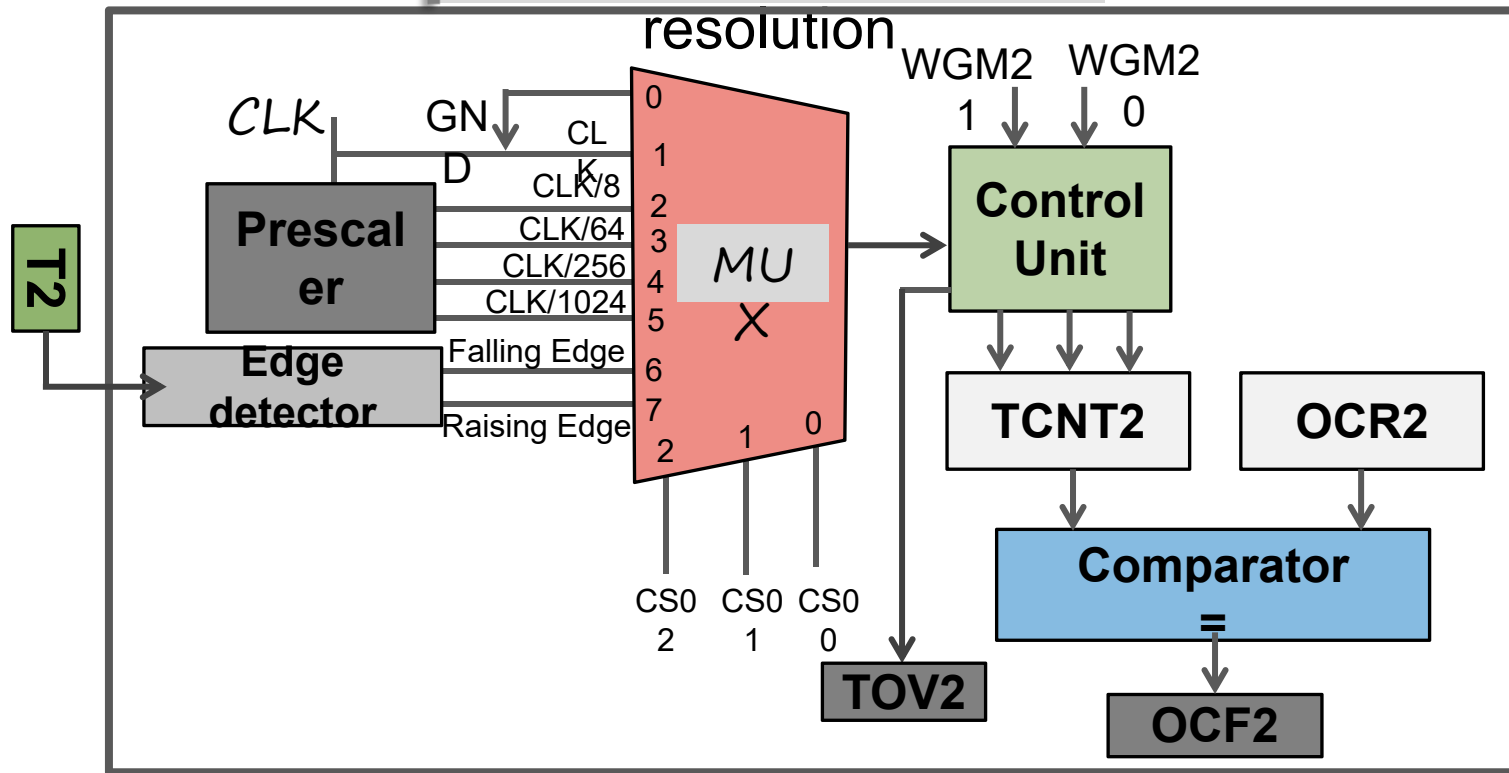
Timer Hardware in ATMEGA32

We have three-
peripheral timers into
our microcontroller

TIMER0, 8-bit
resolution

TIMER1, 16-bit
resolution

TIMER2, 8-bit



Timer_

0

As we mentioned before, timer peripheral depends on **the system clock**, and no way to be a counter without **an external pin**, but our **Timer0** peripheral is a **Timer** and **Counter**, it can be configured to be a **Timer** or **counter**

Timer hardware has a multiplexer that determine clock source will be selected by setting “**CS00, CS01, CS02**”

000 : there is no clock source

001 : system clock is divided by 1

010 : system clock is divided by 8

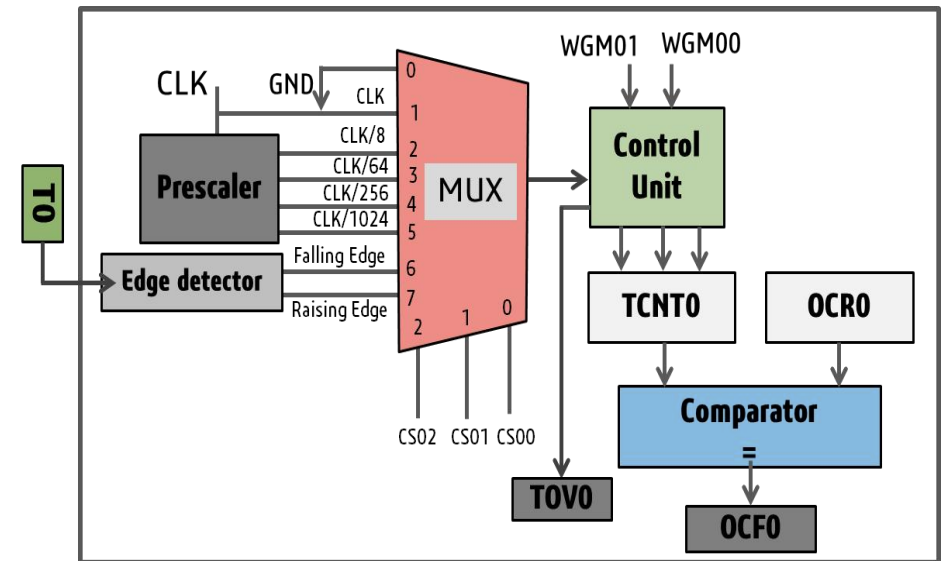
011 : system clock is divided by 64

100 : system clock is divided by 256

101 : system clock is divided by 1024

110 : every falling edge on “**T0**” pin

111 : every rising edge on “**T0**” pin



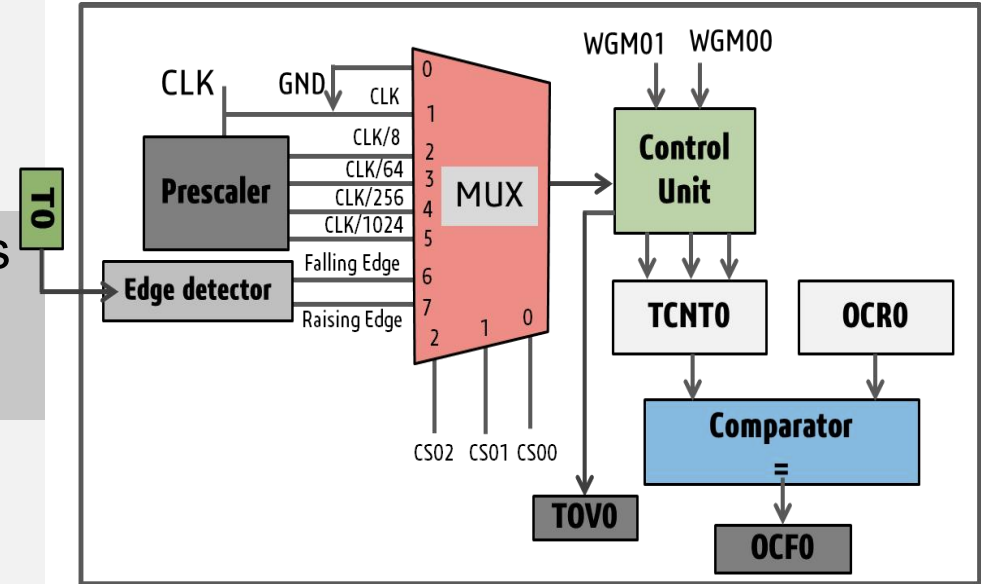
Timer_

If “**CSxx**” bits are selected to “**000**”, this means the timer hardware control will be connected to ground, so the timer will be disabled.

If “**CSxx**” bits are selected to “**001**”, this means the timer hardware control will be directly connected to system clock, so Timer speed will be the same Processor speed

If “**CSxx**” bits are selected from “**010**” to “**101**”, this means the timer hardware control will be connected to system clock, but clock is divided by a specific factor

If “**CSxx**” bits are selected to “**11x**”, this means the timer hardware control will be Counter, which it will be triggered by an external event through “**T0**” pin



Timer_

How it works?

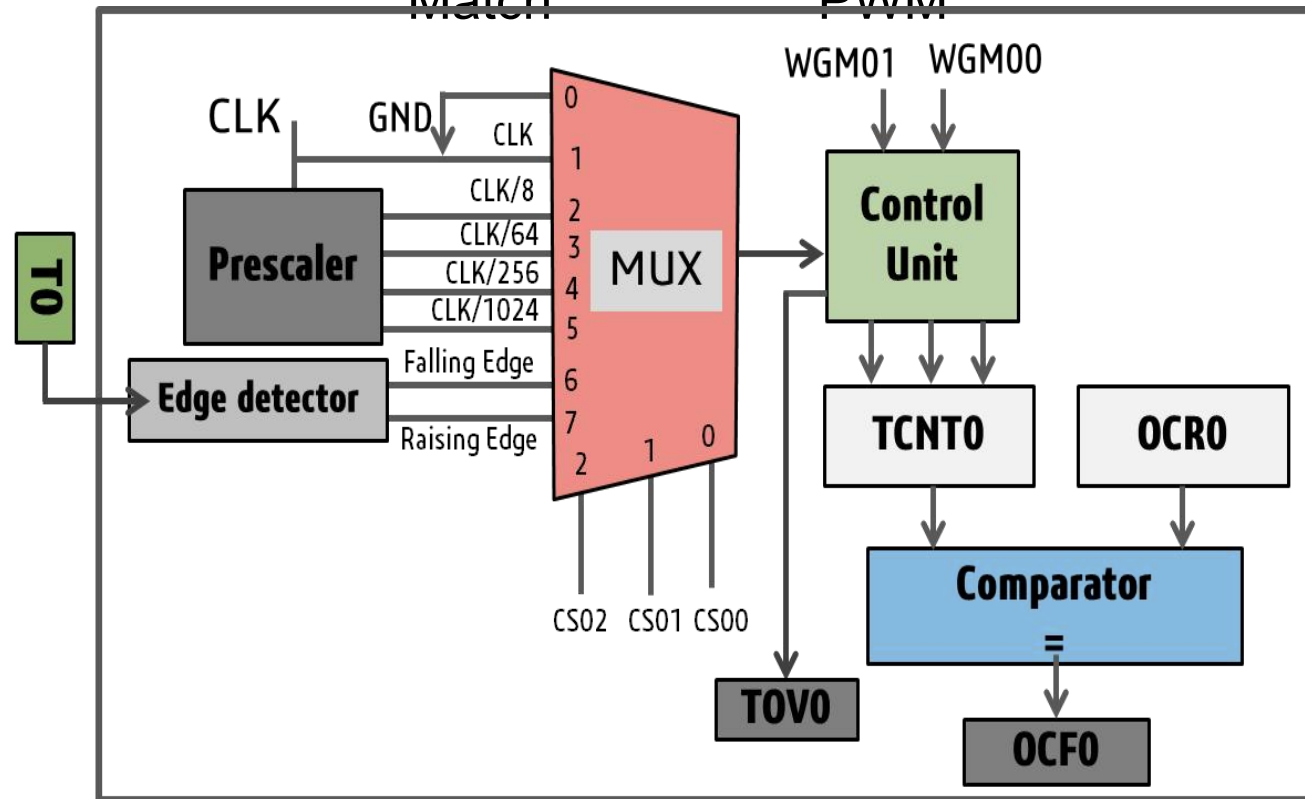
Timer0 has different modes

Normal "Overflow mode"

CTC "Compare Match"

Fast PWM

Phase-correct PWM



Timer_

How it works?

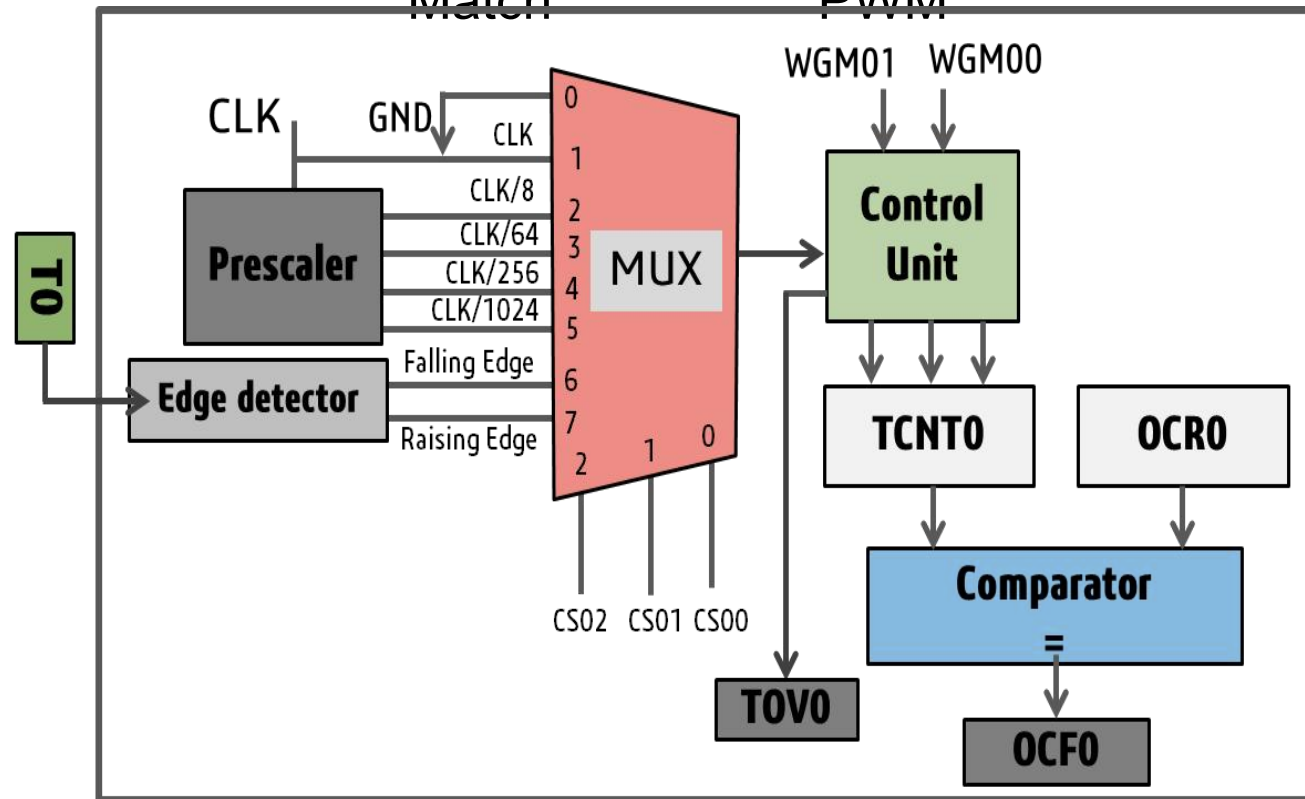
Timer0 has different modes

Normal "Overflow mode"

CTC "Compare Match"

Fast PWM

Phase-correct PWM



Overflow mode in Timer0

- In this mode, the counter register "**TCNT0**" is increased by one every clock cycle or triggered event on "**T0**" until it reaches to its top, then it overflows
- After overflow, "**Tov**" flag will be raised, and it may cause an interrupt service routine if "**Timer PIE**" and "**GIE**"
- After clock-source or edge-type selection, the multiplexer will through out the clock or the event triggered to the control unit of timer
- After that, the control unit will take an action according to its configuration bits "**WGM00, WGM01**"

Overflow mode in Timer0

- “WGM00, WGM01” is configured as an overflow mode, so the control unit will increase “TCNT0” by one
- It will repeated until the register overflow and “Tov” flag is raised
- So, this flag can be used into a blocking system “Polling” or into a foreground-background system “Interrupt”.

Timer_

How to Calculate the Time?

As we mentioned before, "TCNT0" register is increased by one every clock cycle, so by knowing the value of register and the period of the

$$\text{Time of One Cycle} = \frac{1}{\text{Timer frequency}} = \frac{\text{prescaler}}{\text{system frequency}}$$

$$\begin{aligned} \text{Time of Counts} &= \text{All counts of timer} \times \text{Time of one cycle} \\ &= \text{All the counts of timer} \times \frac{\text{prescaler}}{\text{system frequency}} \end{aligned}$$

So, the time of Overflow of Timer0 will be calculated by:

$$\text{Time of Counts} = 256 \times \frac{\text{prescaler}}{\text{system frequency}}$$

Calculation of desired time overflow counts

Assume that the desired time is “**1s**” and the chosen Prescaler is “**256**” and our system frequency is “**16 MHz**”

One second is higher than of overflow time, so we need more than one overflow to get “**1s**”

$$\text{Num of Ovf} = \frac{\text{Desired time}}{\text{time of ovf}} = \frac{1}{256 \times \frac{256}{16 \times 10^6}} = 244.140625$$

The number of Ovfs is an Irrational number, so the number must be approximated to the next integer number “Not an algebraic approximation” even if the number is a rational, so, it will equal “245”

Calculation of desired time overflow counts

The fraction expresses a part of **ovf**, so we can calculate the preload of "**TCNT0**" by:

$$\begin{aligned}\text{Preload} &= \text{max_counts} - \text{Friction counts} \\ &= \text{max_counts} - (\text{max_counts} - \text{fraction} \times \text{max_counts}) \\ &= \text{max_counts} \times (1 - \text{fraction}) = \mathbf{220}\end{aligned}$$

So, TCNT0 must be set by 220 every 245 ovfs

Calculation of desired time overflow

col Sudo code of the previous example

```
u16 counter = 0 ;
int main(void)
{
    set timer0 ;
    TCNT0=220 ;
    sei ;
    while (1) {

        if (counter == 245)
        {
            TCNT0=220;
            counter=0;          // do what you want;
        }

    }

    ISR (TIMER0_OVF){
        counter ++;

    }
```

Calculation of desired time overflow counts

Assume that the desired time is “**1ms**” and the chosen Prescaler is “**256**” and our system frequency is “**16 MHz**”

One second is higher than of overflow time, so we need more than one overflow to get “**1ms**”:

$$\text{Num of Ovf} = \frac{\text{Desired time}}{\text{Timer frequency}} = \frac{0.001}{256 \times \frac{256}{16 \times 10^6}} = 0.244140625$$

The number of Ovfs is an Irrational number, so the number must be approximated to the next integer number “Not an algebraic approximation” even if the number is a rational because fraction means that there is a new needed overflow event but not a complete ovf, part of it , so, it will equal “**1**”

Calculation of desired time overflow counts

The fraction expresses a part of **ovf**, so we can calculate the preload of “**TCNT0**” by:

$$\text{preload} = \text{max_counts} \times (1 - \text{fraction}) = \mathbf{193.5}$$

So, TCNT0 must be set by 193 every Ovf event

Calculation of desired time overflow

col Sudo code of the previous example

```
u8 flag=0;

int main(void){

    set timer0;
    TCNT0=193;
    sei;

    while (1) {
        if (flag == 1) {
            TCNT0 = 193;
            flag =0;          // do what you want;
        }
    }

    ISR (TIMER0_OVF) {
        flag = 1;
    }
}
```

Calculation of desired time overflow counts

- At the previous example, the user only needs about **63** counts into **TCNT0**, so every overflow event it is preloaded by **193**

- This solution is not accurate at all because of setting the **TCNT0** every event

- So, according to the last case, overflow or normal mode is not a perfect chosen mode

- To run the hardware timer to only counts a specific number like the previous example "**63**", the compare match mode is a perfect chosen mode in that case

- But, how can the timer hardware force the "**TCNT0**" to zeroize and start from beginning?

In this case, we need another register to save the value what we need and configure the timer control to compare **TCNT0** register with it every clock cycle, Now, let's recognize on this mode in details

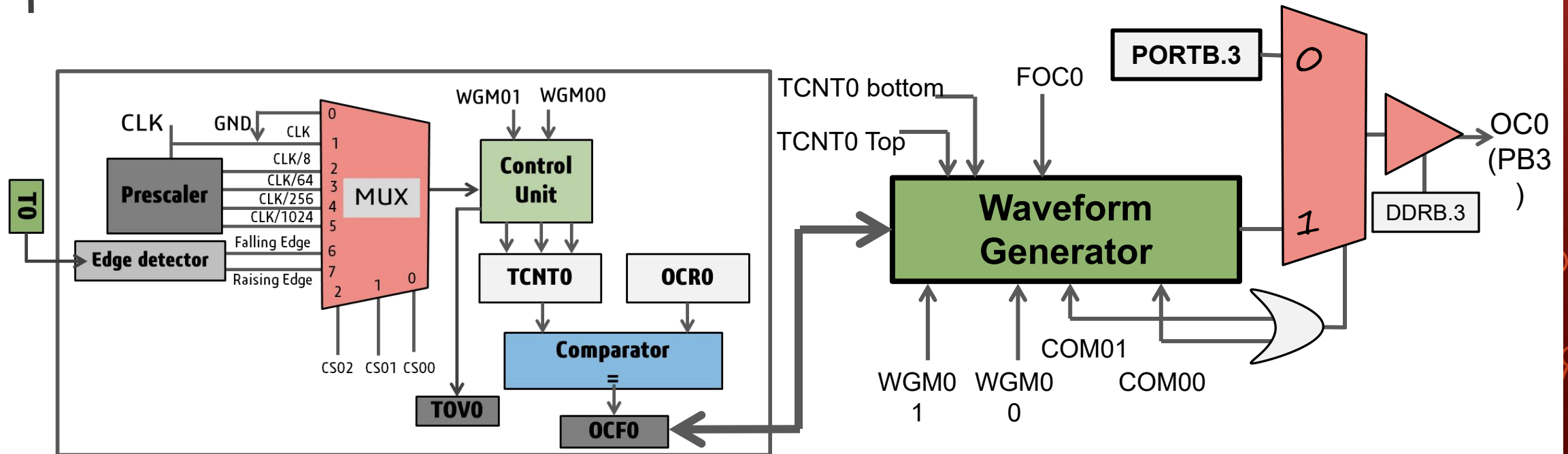
Timer_

Compare Match

“CTC”

At this mode, **Control Unit** of Timer Must be configured to

In this mode, Output Compare Register “**OCR0**” must be used, it is used to save the number what I want, and the comparator compares “**TCNT0**” with “**OCR0**” every clock cycle, if the value of bit order x in “**TCNT0**” equals with the value of same bit order in “**OCR0**”, the output of comparator will raise “**OCF0**” flag and trigger the wave generator

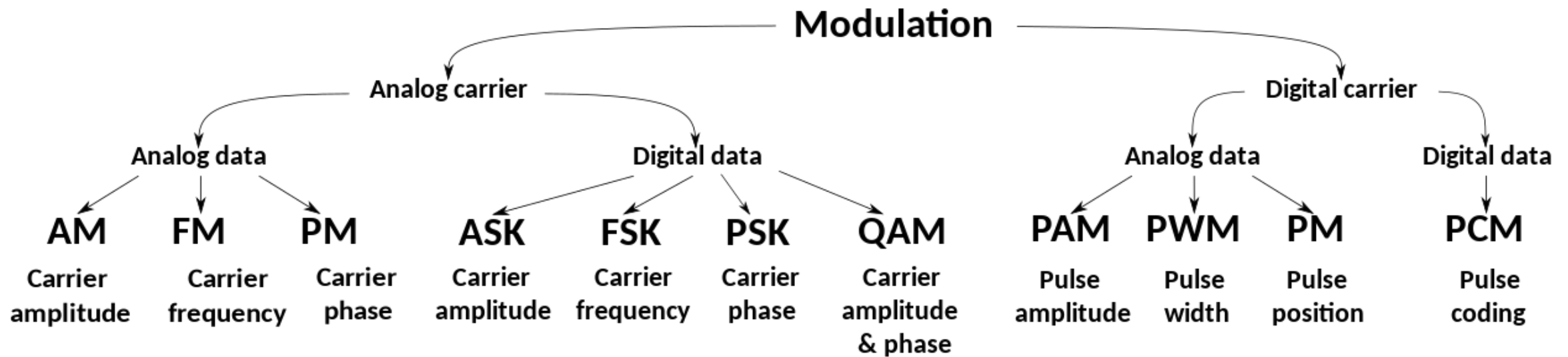


Pulse Width Modulation “PWM”

Modulation : technique used in most communication systems for encoding/decoding and encrypting/decrypting the TX / RX signals that done by generating a pulses that able to carry information

Types of Modulation :

AM(amplitude modulation), PM(phase modulation),FM (frequency modulation)



Pulse Width Modulation “PWM”

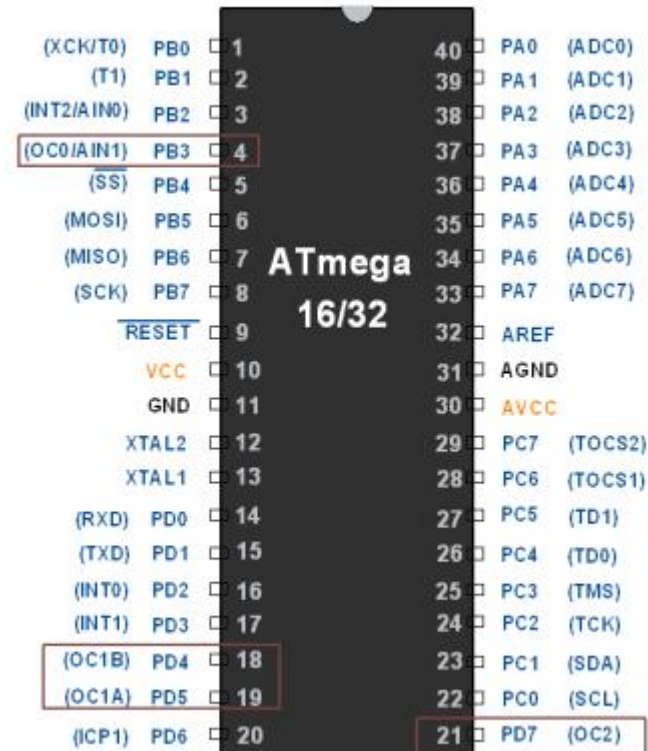
Pulse Width Modulation (PWM) is an elegant current / voltage control technique that enables you to control the speed of motors, and much more in an energy-efficient

And a method of reducing the average power delivered to a device through an electrical signal, by effectively cutting it up into discrete parts

PWM generated by Varying the frequency and duty cycle

SOME APPLICATIONS FOR PWM :

- control speed of fans.
- control output heat of heaters,
- HVAC compressor drives.
- Hybrid and electric vehicle motor .
- brightness and LED Dimmers.



Pulse Width Modulation “**PWM**”

Main Parameters:

Amplitude: The voltage difference between the MAX Voltage and the MIN voltage

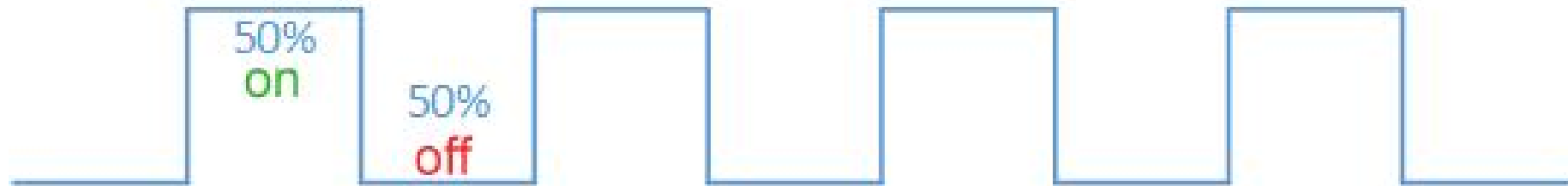
Period : It is the time of one full cycle (High level time + Low level time).

Frequency : It is the total number of cycles “period” per second.

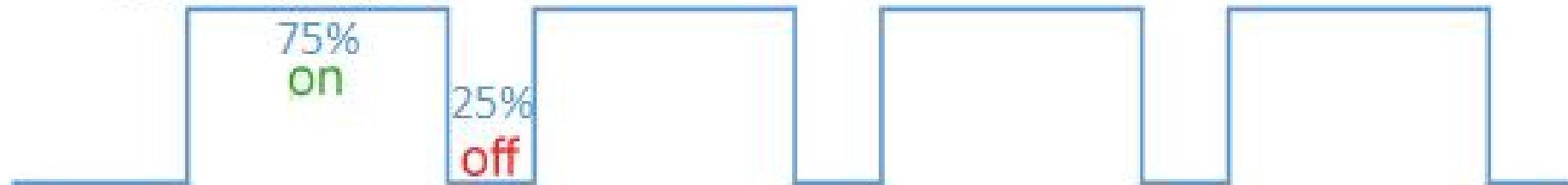
Duty Cycle: It is the ratio between the High-level time “on period” and the total time of the cycle “off period”.

$$\text{Duty Cycle} = \frac{\text{High level period}}{\text{time of full cycle}} = \frac{\text{Pulse Width}}{\text{Period}}$$

50% duty cycle



75% duty cycle



25% duty cycle



Pulse Width Modulation “**PWM**”

the generated wave will be produced on OC pin of the MC

The PWM MODE is combined technique between OVER-FLOW and CTC Mode

Each interrupt change the Level of **PWM** wave form .

The main function of TCNTx is to overflow if it reaches its top.

The main function of OCRx is to Compare with TCNTx

@FAST PWM -> The two interrupts generated through comparing and over flow generating the PWM wave form on OC pin

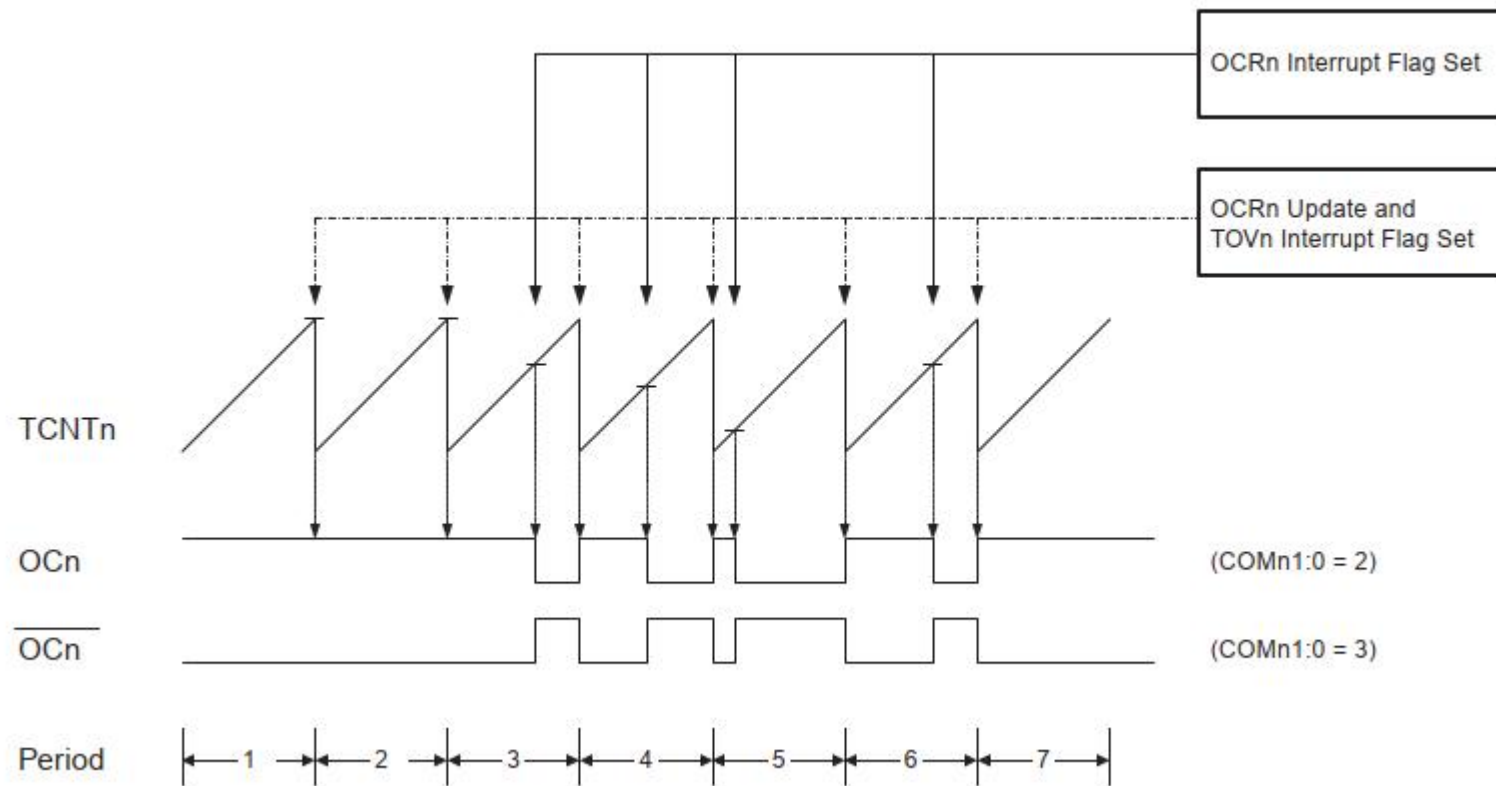
@PHASE CORRECT PWM -> The 1st interrupt generated when compare match occur while TCNT count-up ,The 2nd compare match while TCNT counting-down

Pulse Width Modulation “**FAST PWM**”

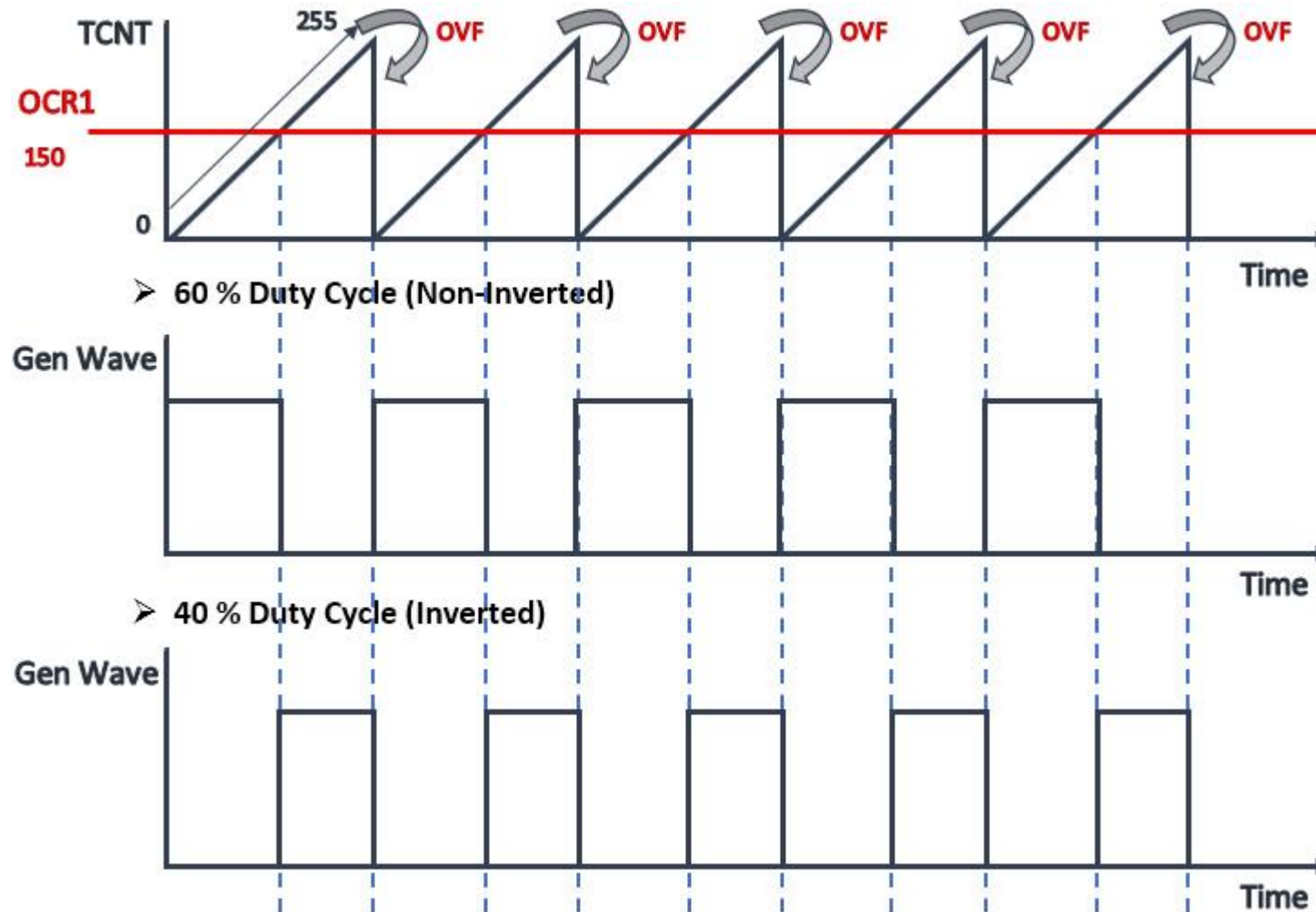
- **There are two types of Pulse Width Modulation (PWM):**
 - ✓ Fast PWM
 - ✓ PWM Phase Correct
- **There are two modes to generate **Fast PWM**:**
 1. Non-Inverted (Clear on compare match, High on overflow)
 2. Inverted (Clear on overflow, High on compare match)

Pulse Width Modulation “FAST PWM”

Figure 32. Fast PWM Mode, Timing Diagram



Pulse Width Modulation “FAST PWM”



Pulse Width Modulation “FAST PWM”

*the frequency of that square wave produced on OC pin of
MC ?*

$$timer_{freq} = \frac{F_{CPU}}{prescaler}$$

$$F_{PWM} = \frac{timer_{freq}}{max\ TCNT\ value}$$

$$F_{PWM} = \frac{16 * 10^6}{pre_{scaler} * 256}$$

Prescaler value	No prescaler	8	64	256	1024
$F_{GW} = \frac{16 * 10^6}{pre_{scaler} * 256}$	62.5 Khz	7.8125 Khz	976.5 Hz	244.14 Hz	61 Hz

To generate more frequencies

we could change the initial start of TCNT Register each overflow

Or By use timer 1 that has 16 bit TCNT with max number of ticks = $2^{16} = 65535$

Pulse Width Modulation “FAST PWM”

.the produced voltage is determined based on the duty cycle :

$$\text{produced voltage} = \text{duty cycle} * 5 \text{ V}$$

. THE DUTY CYCLE VARY ACCORDING TO OCR REGISTER
:

$$\text{Duty cycle} = \frac{\text{HIGH period}}{(\text{HIGH} + \text{LOW}) \text{ period}} = \frac{\text{OCR} + 1}{256} * 100$$

in non-Inverted mode

$$\text{Duty cycle} = \frac{\text{HIGH period}}{(\text{HIGH} + \text{LOW}) \text{ period}} = \frac{256 - (\text{OCR} + 1)}{256} * 100$$

in Inverted mode

Pulse Width Modulation “FAST PWM”

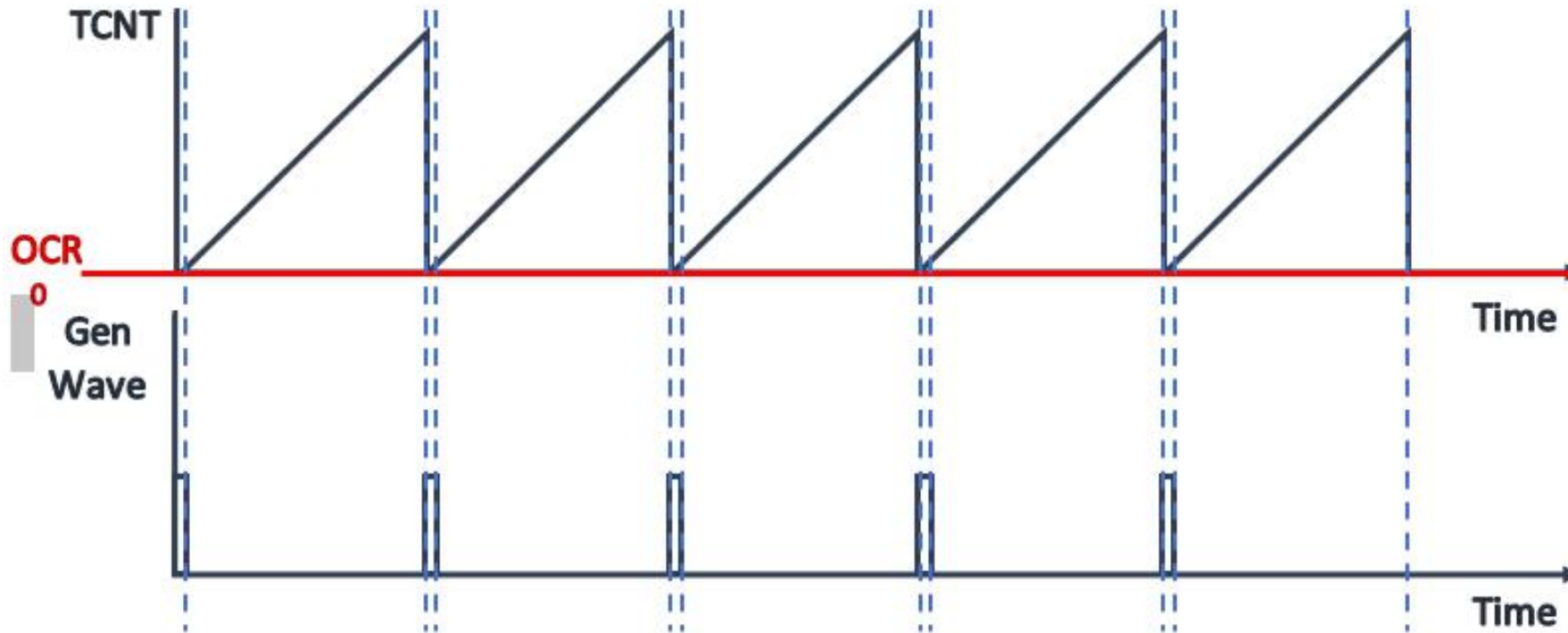
Main mode	Fast PWM			
Sec mode	Non-Inverted mode		Inverted mode	
Duty cycle equation	$\text{Duty cycle} = \frac{OCR + 1}{256} * 100$		$\text{Duty cycle} = \frac{255 - OCR}{256} * 100$	
OCR values	0 % = 0 OCR	100 % = 255 OCR	0 % = 255 OCR	100 % = 0 OCR
Duty cycle	0.39 %	100 %	0 %	99.6 %
voltage	0.0195 V	5 V	0 V	4.98 V

Pulse Width Modulation “FAST PWM”

disadvantage of Fast PWM -> *it's not accurate in producing Duty cycle*

- *in non-Inverted mode if we want to produce 0 % duty cycle it produce 0.39%*

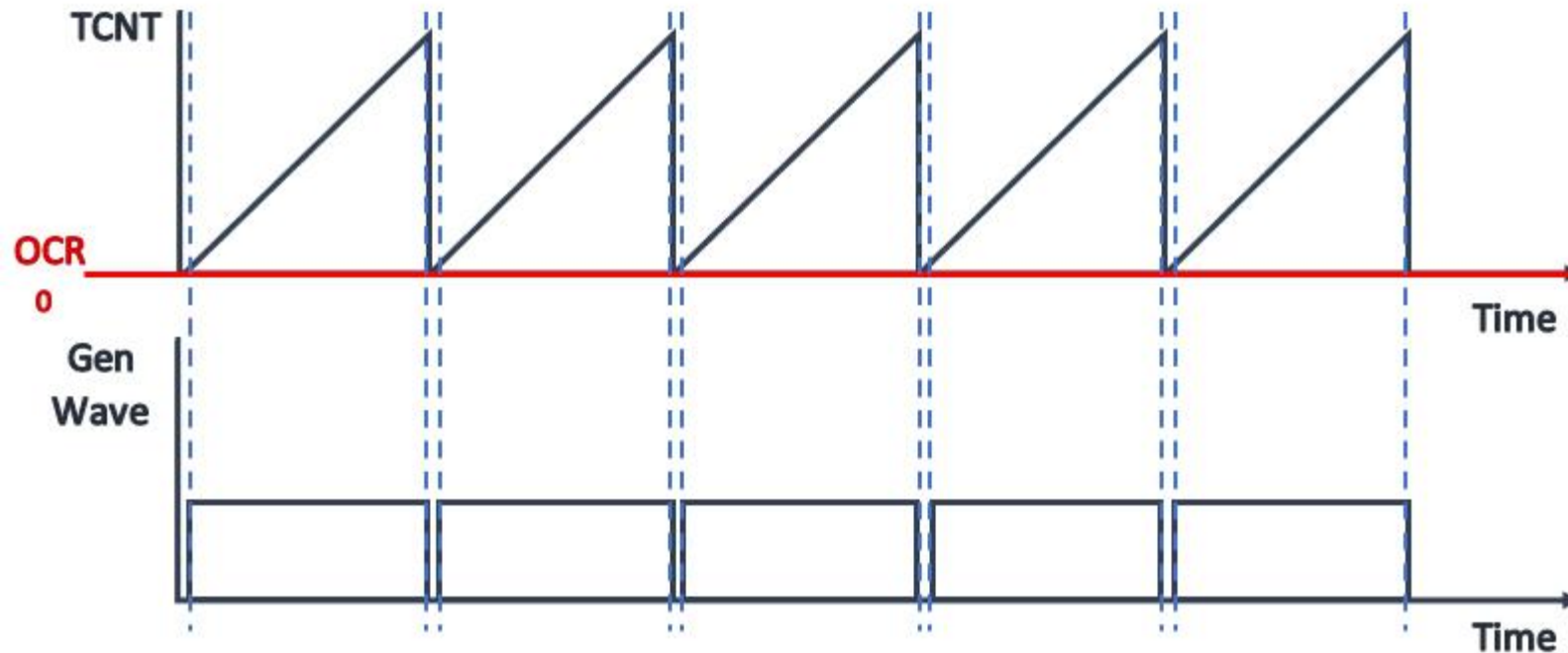
$$\text{Duty cycle} = \frac{0 + 1}{256} = \frac{1}{256} * 100 = 0.39 \%$$



Pulse Width Modulation “FAST PWM”

in Inverted mode if we want to produce 100 % duty cycle it produce 99.6%

$$\text{Duty cycle} = \frac{255 - 0}{256} = \frac{255}{256} * 100 = 99.6 \%$$





***Solution IS **PWM** phase correct
Mode***

Pulse Width Modulation “**PHASE CORRECT PWM**”

In PWM phase correct mode, the TCNT register counts up to 255 then counts down to 0 again to Generate One cycle of PWM .

Note *the number of ticks doubled (ex: TIMER0 Counts = $2 \times 256 = 512$)*

 *That will increase the consumed time to generate the PWM*

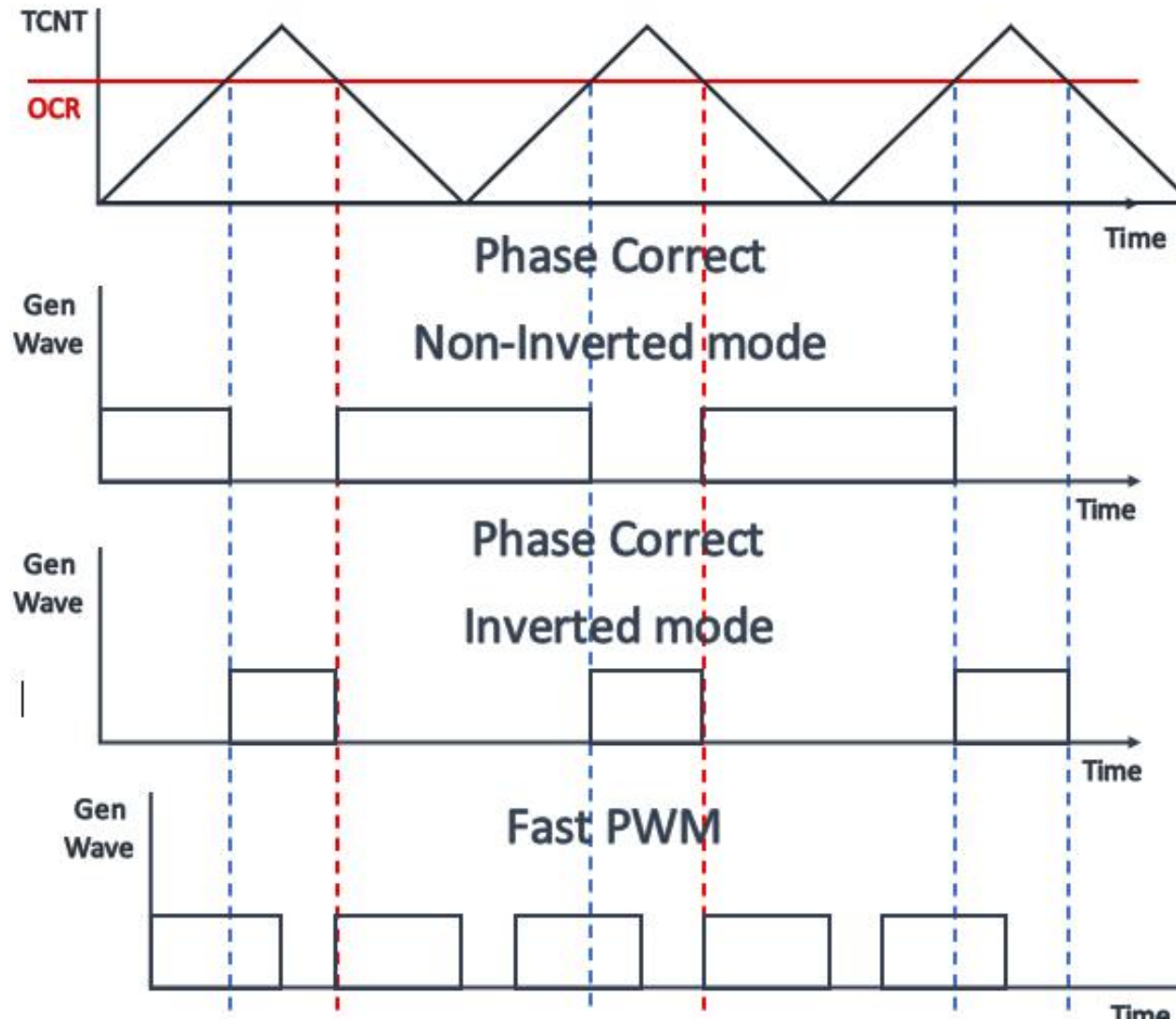
 *which decrease the frequency of the generated wave*

by half

Pulse Width Modulation “**PHASE CORRECT PWM**”

NOTE

*ONE cycle of
PWM phase
correct
is twice the Cycle
of
Fast PWM*



Pulse Width Modulation “PHASE CORRECT PWM”

calculate the Duty cycle of *non-Inverted* modes :

$$\text{Duty cycle} = \frac{2 * OCR}{2 * TOP} * 100$$

$$TOP = 255$$

calculate the Duty cycle of *Inverted* modes :

$$\text{Duty cycle} = \frac{(2 * TOP) - (2 * OCR)}{2 * TOP} * 100$$

Pulse Width Modulation “PHASE CORRECT PWM”

Main mode	PWM Phase Correct			
Sec mode	Non-Inverted mode		Inverted mode	
Duty cycle equation	$Duty\ cycle = \frac{OCR}{TOP} * 100$		$Duty\ cycle = \frac{TOP - OCR}{TOP} * 100$	
OCR values	0 % = 0 OCR	100 % = 255 OCR	0 % = 255 OCR	100 % = 0 OCR
Duty cycle	0 %	100 %	0 %	100 %
voltage	0 V	5 V	0 V	5 V

- ***What is the Advantages of PHASE CORRECT PWM?***

- 1. perfect if we want sharp 0 %, or 100 % duty cycle.***
- 2. due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for stepper and servo motor control applications.***

- ***What is its only disadvantage against Fast PWM?***

is that it is slower than Fast PWM, so can't be used in application that requires a high frequencies like DC motors.

Pulse Width Modulation “PHASE CORRECT PWM”

- *The frequency of the generated wave is as following:*

$$F_{pwm} = \frac{16MHz}{prescaler * 510}$$

Prescaler value	No prescaler	8	64	256	1024
$F_{GW} = \frac{16 * 10^6}{pre_{scaler} * 510}$	31.4 Khz	3.9 Khz	490.2 Hz	122.5 Hz	30.6 Hz

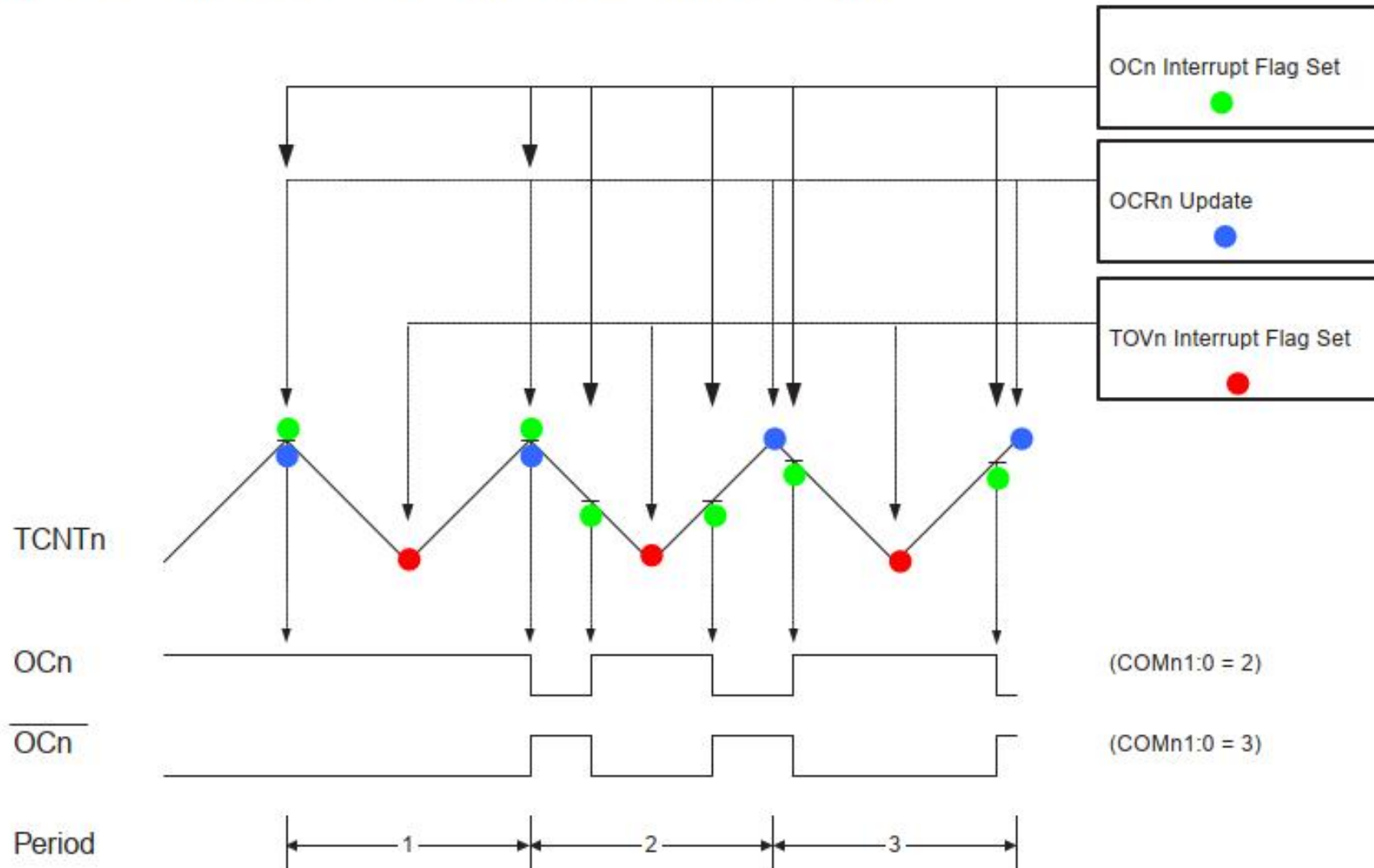
Pulse Width Modulation “**PHASE CORRECT PWM**”

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 use **dual-slope operation**.

Prescaler	Fast PWM at Timer0/2	Phase-Correct PWM at Timer0/2
1	62,500 Hz	31,372.55 Hz
8	7812.5 Hz	3,921.57 Hz
32	1,953.125 Hz (Timer2 only)	980.39 Hz (Timer2 only)
64	976.5625 Hz	490.20 Hz
128	488.28 Hz (Timer2 only)	245.10 Hz (Timer2 only)
256	244.14 Hz	122.55 Hz
1024	61.04 Hz	30.64 Hz

Pulse Width Modulation “PHASE CORRECT PWM”

Figure 59. Phase Correct PWM Mode, Timing Diagram



Pulse Width Modulation “PWM”

There are two cases that give a transition without Compare Match:

- *If OCR0 changes its value from MAX, like in Figure 33. To ensure symmetry around BOTTOM.*
- ***if** The timer starts counting from a value higher than the one in OCR0, and for that reason misses the Compare Match.*

The image features a solid red background with a subtle gradient. In the four corners, there are decorative elements resembling circuit board traces or neural network connections. These elements consist of thin, light-orange lines that branch out and terminate in small circles, creating a symmetrical, abstract pattern.

THANK YOU!