Timers programming

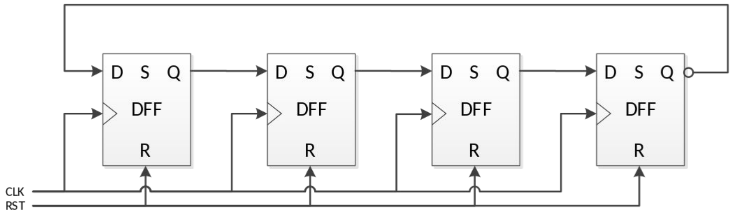
By: Yehia M. Abu Eita

Outlines

- Introduction
- Timer/Counter 0 block diagram
- Normal mode
- CTC mode
- PWM modes
- Counter mode
- Steps to program Timer/Counter 0

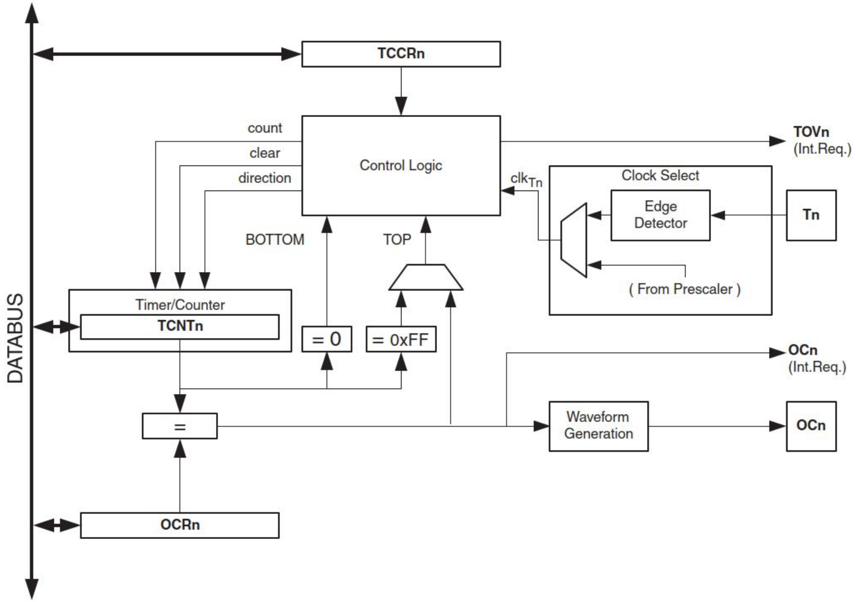
Introduction

- There are two **8-bit** timers, **Timer 0** and **Timer 2**.
- There is one **16-bit** timer, **Timer 1**.
- Timers are a **special type of registers** that is **incremented** automatically according to the feeding **clock signal**.
- Timers can be used to:
  - **Generate delays.**
  - **Generating Pulse Width Modulation signals.**
  - **Counting external events.**
  - **Generate system tick for RTOS.**



Timer/Counter 0 block diagram

- Timer/Counter0 is a **general purpose 8-bit Timer/Counter** module.
- It can be **clocked** by an **internal** or an **external** clock source.
- It has **wave generator** to generate **PWM** signals.



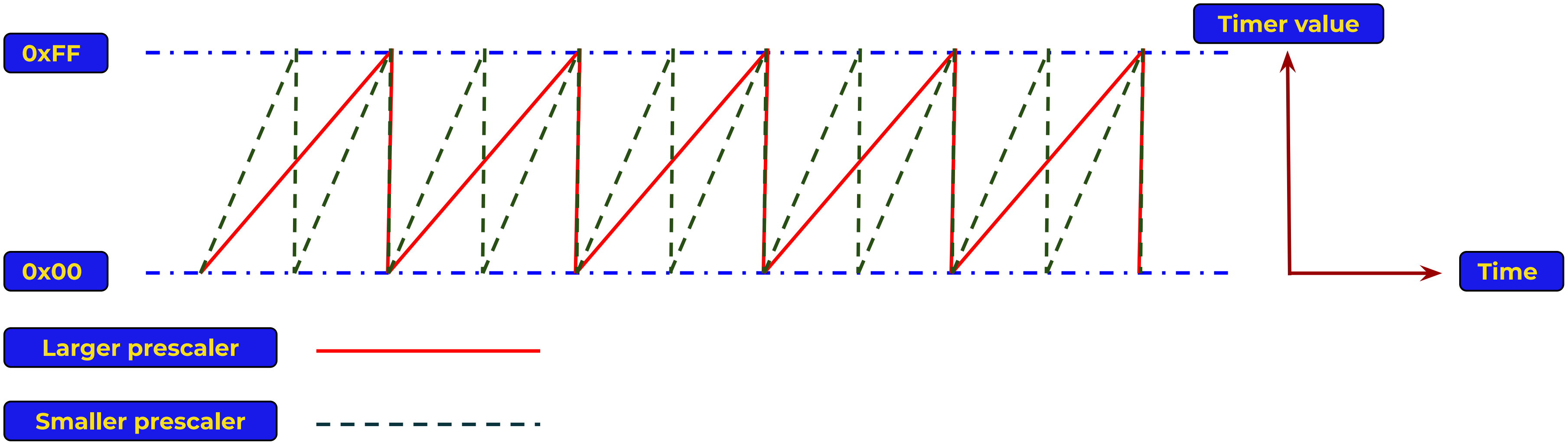
Normal mode

- The timer **counts** to its **maximum/overflow**.
- An **overflow flag** is **set** when overflow is reached.
- Choosing **prescaler** in order to **change tick time**.
- Prescaler is simply a frequency divider.
- **Increasing** the **prescaler** means **reducing frequency** and **increasing tick time**.
- Timer value can be set any time.

7	6	5	4	3	2	1	0
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.

Normal mode



0xFF
0x00
Timer value
Time
Larger prescaler
Smaller prescaler

Normal mode

- **To generate timer delay**:
  - Calculate the tick time with no prescaler.
  - Calculate Maximum delay can be generated.
  - If $T_{delay} < T_{max\ delay}$, set initial timer value and wait for 1 overflow.
  - If $T_{delay} = T_{max\ delay}$, set initial timer value to 0 and wait for 1 overflow.
  - If $T_{delay} > T_{max\ delay}$, calculate the number of needed overflows.
  - Calculate and set the needed timer initial value.

$$T_{tick} = \frac{1}{F_{clock}} = \frac{1}{\frac{F_{CPU}}{Prescaler}} = \frac{Prescaler}{F_{CPU}}$$

$$T_{max\ delay} = T_{tick} \times 2^{n}$$

$$Timer_{initial\ value} = \frac{T_{max\ delay} - T_{delay}}{T_{tick}}$$

$$N_{overflows} = (ceil)\frac{T_{delay}}{T_{max\ delay}}$$

$$Timer_{initial\ value} = 2^{n} - \frac{\frac{T_{delay}}{T_{tick}}}{N_{overflows}}$$

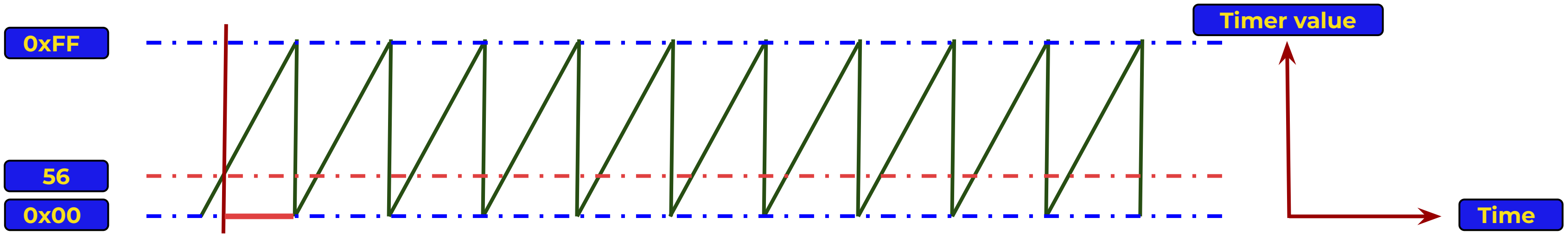
Normal mode

- **Example - No overflow**:
  - For an **8-bit** timer, **n = 8**.
  - For a 1MHz CPU frequency, $\mathbf{F_{CPU} = 10^6}$ **Hz**.
  - We need a delay of 200ms, $\mathbf{T_{delay} = 200\mu s}$.
  - With no prescaling, **prescaler = 1**.

1. $$T_{tick} = \frac{Prescaler}{F_{CPU}} = \frac{1}{10^6} = 1\mu s$$

2. $$T_{max\ delay} = 2^8 \times 1\mu s = 256\mu s$$

3. $$Timer_{initial\ value} = \frac{256\mu s - 200\mu s}{1\mu s} = 56$$



Normal mode

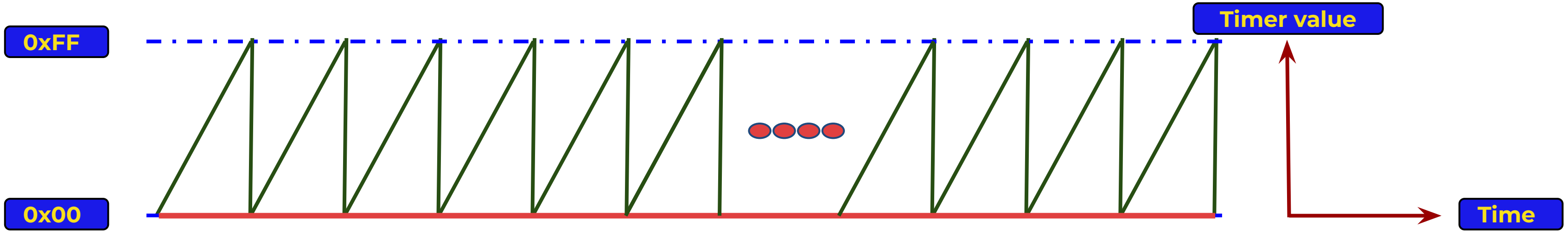
- **Example - overflow and no prescaling**:
  - For an **8-bit** timer, **n = 8**.
  - For a 1MHz CPU frequency, $\mathbf{F_{CPU} = 10^6}$ **Hz**.
  - We need a delay of 512ms, $\mathbf{T_{delay} = 512ms}$.
  - **Prescaler = 1**.

1. $$T_{tick} = \frac{Prescaler}{F_{CPU}} = \frac{1}{10^6} = 1\mu s$$

2. $$T_{max\ delay} = 2^8 \times 1\mu s = 256\mu s$$

3. $$N_{overflows} = (ceil)\frac{512ms}{256\mu s} = 2000$$

4. $$Timer_{initial\ value} = 2^8 - \frac{\frac{512ms}{1\mu s}}{2000} = 0$$



Normal mode

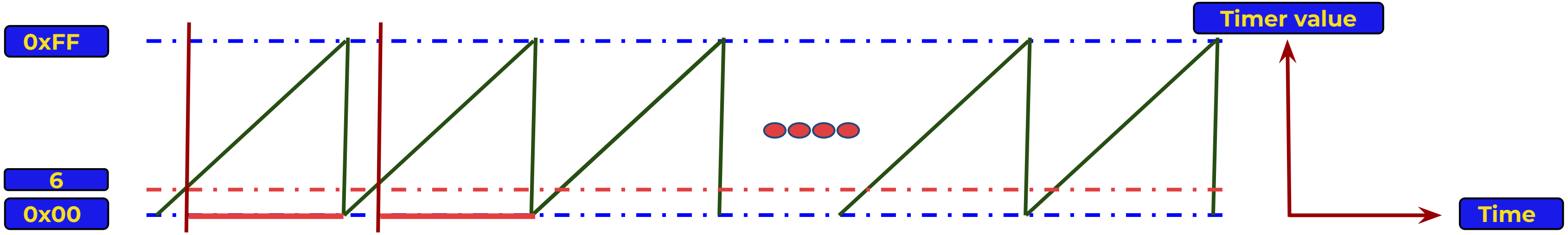
- **Example - overflow with prescaling**:
  - For an **8-bit** timer, **n = 8**.
  - For a 1MHz CPU frequency, $\mathbf{F_{CPU} = 10^6\ Hz}$.
  - We need a delay of 512ms, $\mathbf{T_{delay} = 512ms}$.
  - With **prescaler = 1024**.

1. $$T_{tick} = \frac{Prescaler}{F_{CPU}} = \frac{1024}{10^6} = 1.024ms$$

2. $$T_{max\ delay} = 2^8 \times 1.024ms = 262.144ms$$

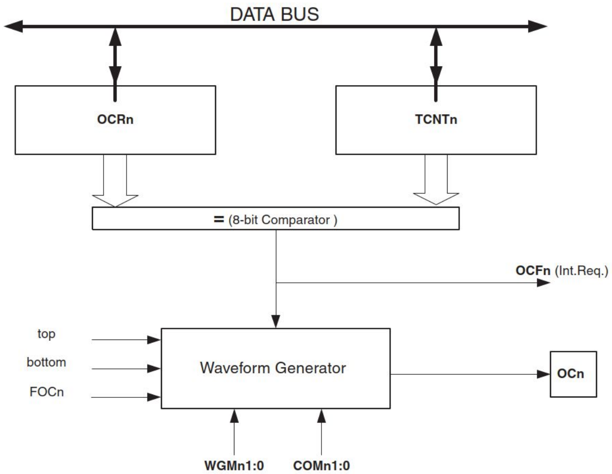
3. $$N_{overflows} = (ceil)\frac{512ms}{262.144ms} = 2$$

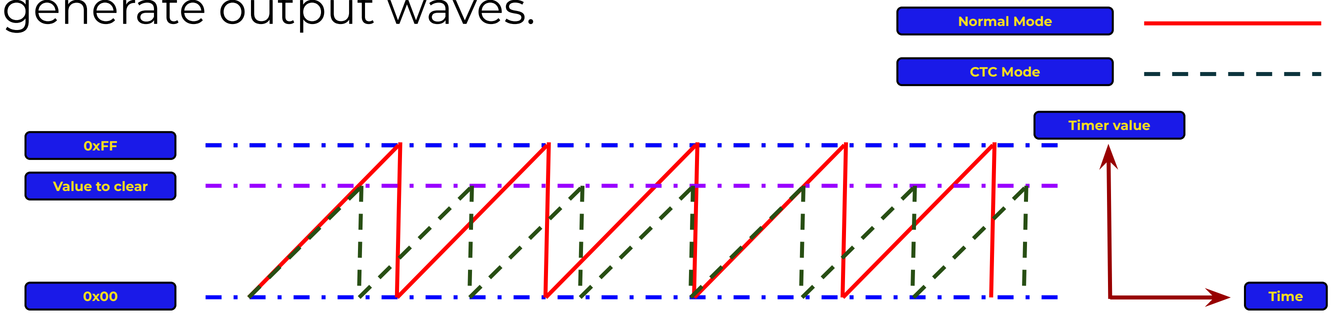
4. $$Timer_{initial\ value} = 2^8 - \frac{\frac{512ms}{1.024ms}}{2} = 6$$



CTC mode

- It is a **C**lear **T**imer on **C**ompare match.
- The timer is **cleared** when it reaches the value in the **OCR0**.
- It may be used to generate output waves.



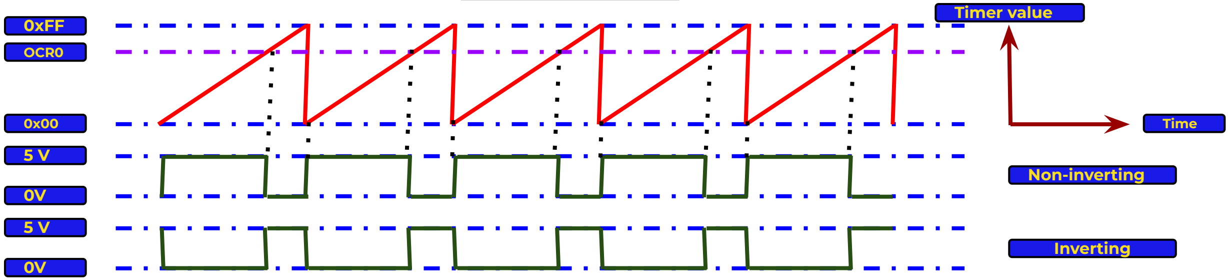


PWM modes

- **Fast PWM mode**.
- PWM means **P**ulse **W**idth **M**odulation.
- Any PWM signal is characterized by:
  - Frequency
  - Duty Cycle
- **Applications**:
  - Controlling motor speed
  - Controlling LED intensity

$$Duty\ Cycle = \frac{T_{on}}{T_{cycle}} \times 100$$

$$T_{cycle} = \frac{1}{F}$$

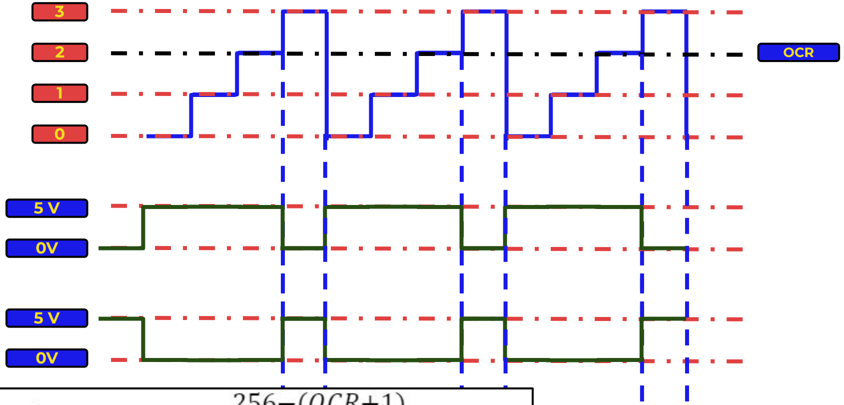


PWM modes

- **Fast PWM mode calculations**:
  - For a **2-bit** timer and **O**utput **C**ompare **R**egister is **2-bit**.
  - $T_{cycle} = 2^2 \times T_{tick}$.
  - For **Non-inverting** configuration:
    - $T_{ON} = (OCR + 1) \times T_{tick}$.
  - For **inverting** configuration:
    - $T_{ON} = T_{cycle} - (OCR + 1) \times T_{tick}$.
  - The same applied for any timer.
  - For **timer 0, 8-bits**:

$$Duty\ Cycle_{non-inverting} = \frac{OCR+1}{2^n} \times 100$$

$$Duty\ Cycle_{inverting} = \frac{2^n-(OCR+1)}{2^n} \times 100$$

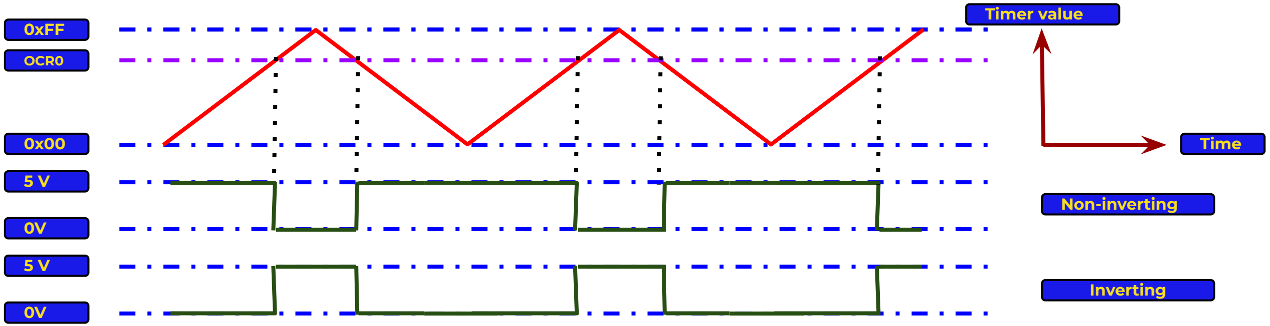


$$Duty\ Cycle_{non-inverting} = \frac{OCR+1}{256} \times 100$$

$$Duty\ Cycle_{inverting} = \frac{256-(OCR+1)}{256} \times 100$$

PWM modes

- **PWM with phase correct mode**.
- It has two advantages over the Fast PWM mode:
  - **Phase correction, i.e pulses center is fixed even if the pulse is changed.**
  - **No OCR lowest value problem.**

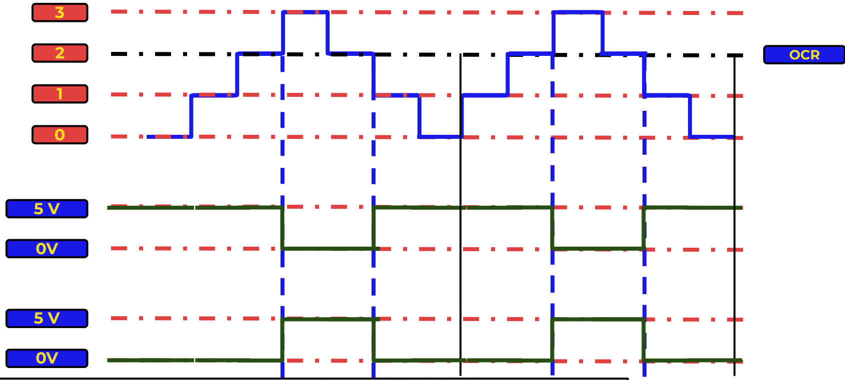


PWM modes

- **PWM with phase correct mode calculations**:
  - For a **2-bit** timer and **O**utput **C**ompare **R**egister is **2-bit**.
  - $T_{cycle} = 2 \times (2^2 - 1) \times T_{tick} = 6 \times T_{tick}$.
  - For **Non-inverting** configuration:
    - $T_{ON} = 2\ OCR \times T_{tick}$.
  - For **inverting** configuration:
    - $T_{ON} = T_{cycle} - 2\ OCR \times T_{tick}$.
  - The same applied for any timer.
  - For **timer 0, 8-bits**:

$$Duty\ Cycle_{non-inverting} = \frac{2 \times OCR}{2 \times (2^n - 1)} \times 100$$

$$Duty\ Cycle_{inverting} = \frac{2 \times (2^n - 1) - (2 \times OCR)}{2 \times (2^n - 1)} \times 100$$

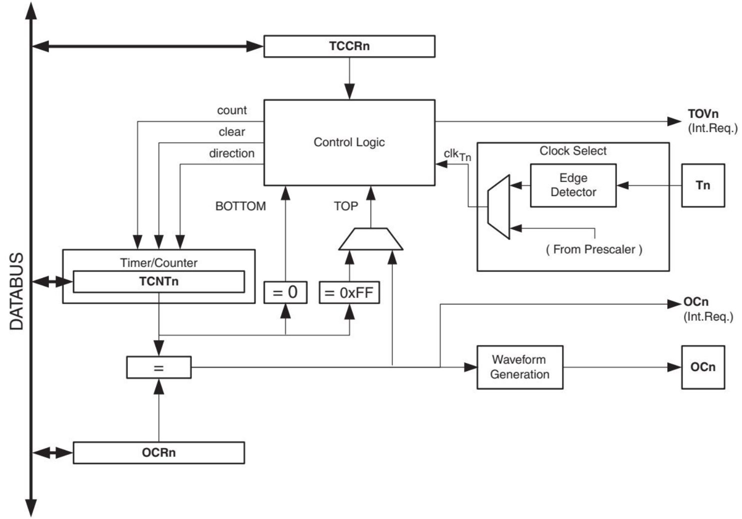


$$Duty\ Cycle_{non-inverting} = \frac{2 \times OCR}{510} \times 100$$

$$Duty\ Cycle_{inverting} = \frac{510 - (2 \times OCR)}{510} \times 100$$

Counter mode

- Counter modes are used to **count external events** or as an **external timer clock source**.
- There are **two modes**:
  - Rising-edge counter
  - Falling-edge counter



Steps to program Timer/Counter 0

- **Set timer configurations**:
  - Choosing **mode of operation**, Normal, CTC, Fast PWM, PWM with phase correct, **TCCR0** register
  - **Set timer** starting **value** according to your calculations, **TCNT0** register
  - **Enable** the required **interrupt if needed**, TIMSK register
- **Start timer**:
  - **Setting the prescaler** according to your calculations, **TCCR0** register, **timer will start count after this step**
- **Get timer state**:
  - **Check** the corresponding **flag**, according to your **mode of choice**, **TIFR** register or
  - **Implement ISR** to handle the interrupt request **if timer interrupt is enabled**

Summary

- You are now familiar with the timer peripheral
- You can program timer 0 and other timers in ATmega32
- Remember that you can generate a PWM signal using normal mode
- Remember that using prescaler helps you to stretch tick time