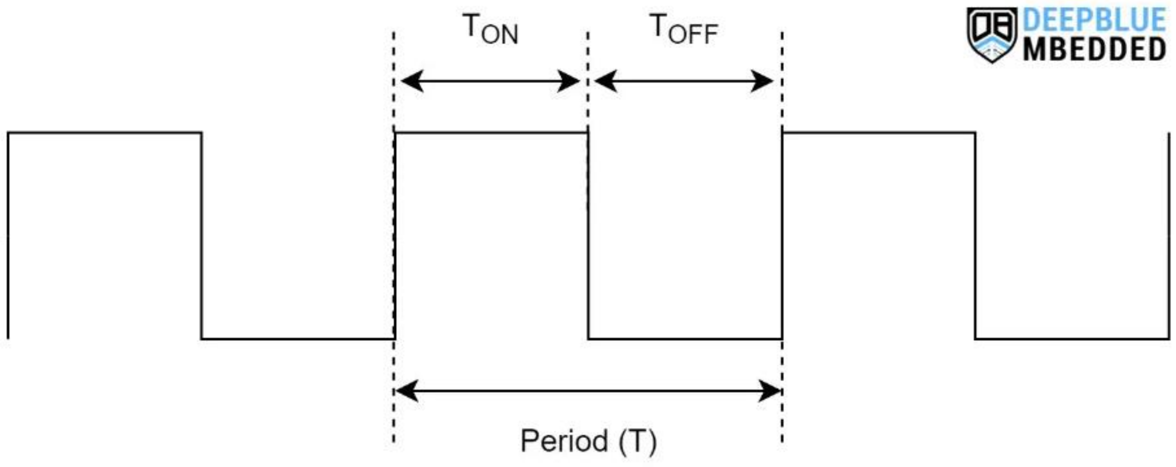
PWM Tutorial

STM32 PWM Introduction

Pulse **W**idth **M**odulation (**PWM**) is a technique for generating a continuous HIGH/LOW alternating digital signal and programmatically controlling its pulse width and frequency. Certain loads like (LEDs, Motors, etc) will respond to the **average voltage** of the signal which gets higher as the PWM signal's pulse width is increased. This technique is widely used in embedded systems to control LEDs brightness, motor speed, and other applications.

PWM Frequency

The PWM signal captures a few features. The first of which is the frequency, which is basically a measure of how fast the PWM signal keeps alternating between HIGH and LOW. The frequency is measured in Hz and it's the inverse of the full period time interval. Here is how it looks graphically and its mathematical formula.



$$Frequency[PWM] = \frac{1}{Period(T)} Hz$$

PWM Duty Cycle

The PWM’s duty cycle is the most important feature that we’re always interested in. It’s a measure of how long the PWM signal stays ON relative to the full PWM’s cycle period. The PWM’s duty cycle equation is as follows:

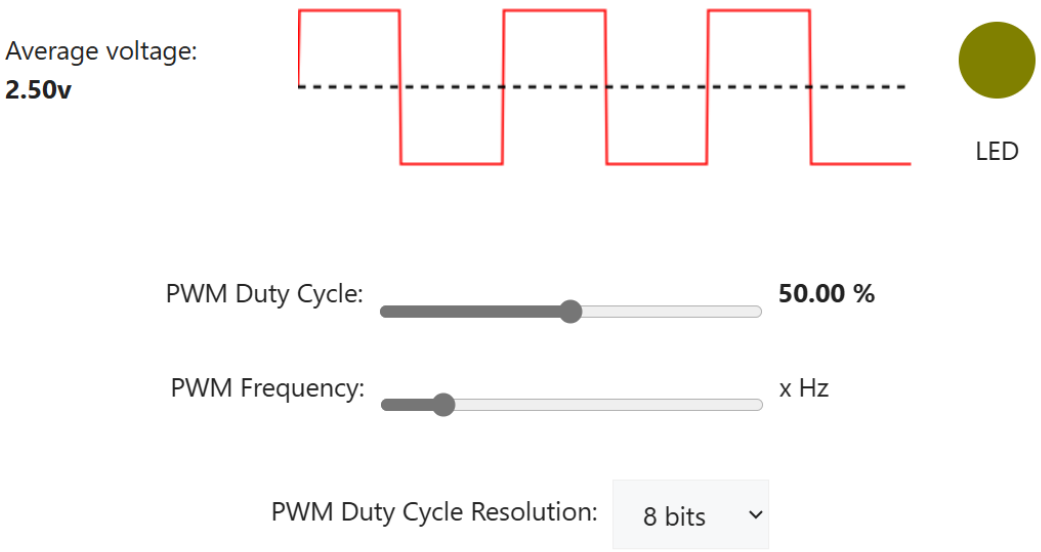
$$DutyCycle[PWM] = \frac{T_{ON}}{T_{ON}+T_{OFF}} \times 100 = \frac{T_{ON}}{Period(T)} \times 100[\%]$$

The duty cycle is usually expressed as a percentage (**%**) value because it’s a ratio between two-time quantities. And it directly affects the PWM’s total (average) voltage that most devices respond to. That’s why we typically change the duty cycle to control things like LED brightness, DC motor speed, etc.

PWM Resolution

The PWM resolution is expressed in (bits). It’s the number of bits that are used to represent the duty cycle value. It can be 8bits, 10, 12, or even 16bits. The PWM resolution can be as the number of discrete duty cycle levels between 0% and 100%. The higher the PWM resolution, the higher number of discrete levels over the entire range of the PWM’s duty cycle.

A PWM resolution of only **3 bits** means there are only **8 discrete levels** for the duty cycle over the entire range (from 0% up to 100%). On the other hand, a PWM with a resolution of **8 bits** will have **256 discrete levels** for the duty cycle over the entire range (from 0% up to 100%). You can use the interactive tool below to test this yourself.



Average voltage: **2.50v**

PWM Duty Cycle: **50.00 %**

PWM Frequency: x Hz

PWM Duty Cycle Resolution: 8 bits

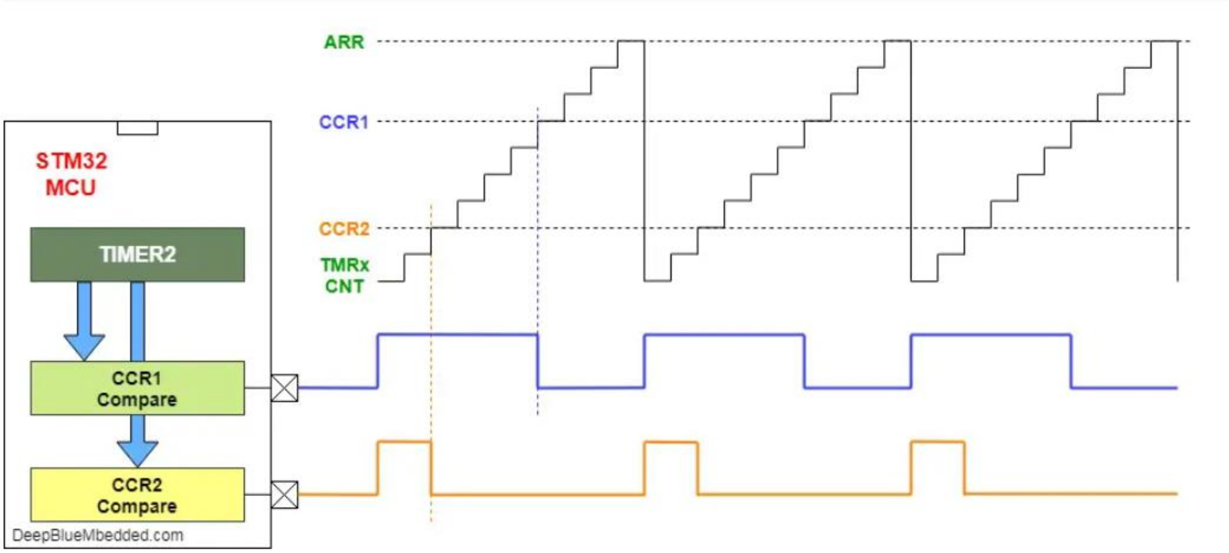
Set the PWM resolution to 3-bit and sweep across the entire range of PWM's duty cycle. And change the resolution up to 8 bits or even 16 bits to see the huge difference in the degree of control you'll have over the duty cycle value. It becomes incredibly smooth as the resolution increases and we become more able to fine-tune the duty cycle. You can use the keyboard arrow keys for fine adjustment of the duty cycle while testing high resolutions.

STM32 PWM Mode in Timer Explained

As we've discussed in an earlier tutorial, the STM32 timer modules can operate a variety of modes one of which is the PWM mode. Where the timer gets clocked from an internal source and counts up to the auto-reload register value, then the output channel pin is driven HIGH. And it remains until the timer counts reach the CCRx register value, the match event causes the output channel pin to be driven LOW. And it remains until the timer counts up to the auto-reload register value, and so on.

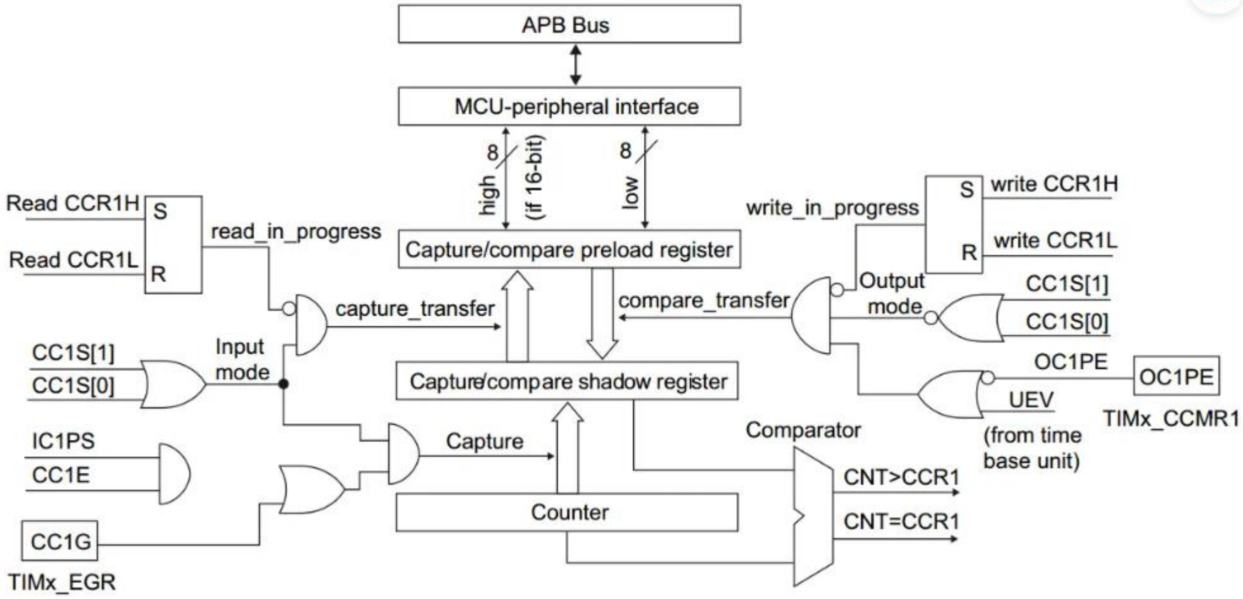
The resulting waveform is called PWM (pulse-width modulated) signal. Whose frequency is determined by the internal clock, the Prescaler, and the ARRx register. And its duty cycle is defined by the channel CCRx register value. The PWM doesn't always have to be following this exact same procedure for PWM generation, however, it's the very basic one and the easier to understand the concept. It's called the up-counting PWM mode. We'll discuss further advanced PWM generation techniques as we go on in this series of tutorials.

The following diagram shows you how the ARR value affects the period (frequency) of the PWM signal. And how the CCRx value affects the corresponding PWM signal's duty cycle. And illustrates the whole process of PWM signal generation in the up-counting normal mode.



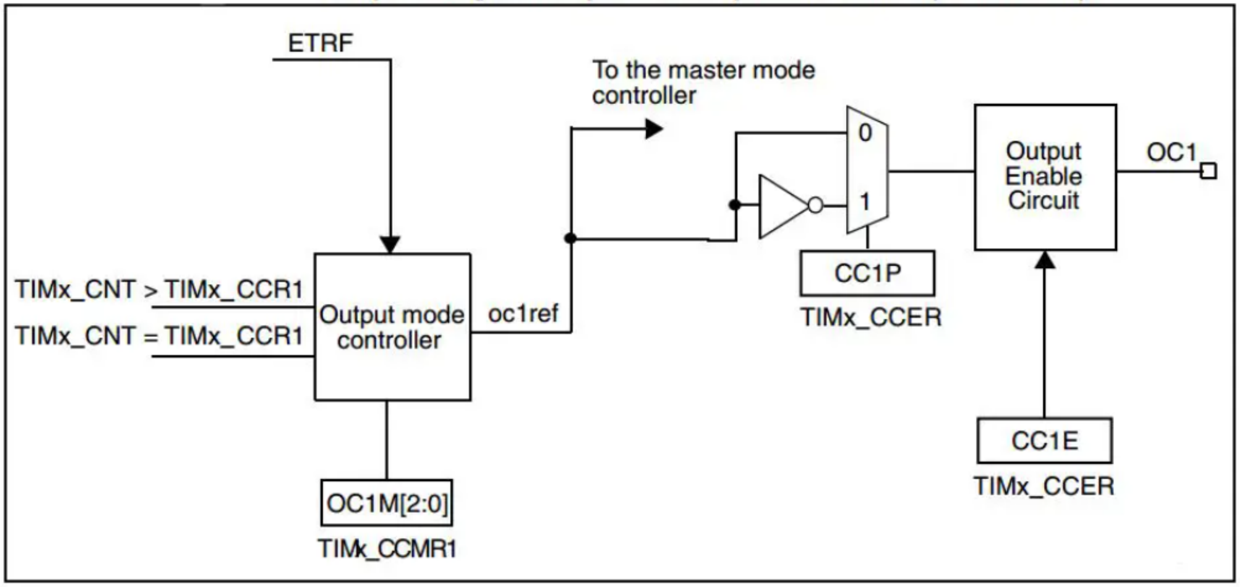
STM32 Timers - PWM Output Channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), an input stage for capture (with a digital filter, multiplexing, and Prescaler) and an output stage (with comparator and output control). The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.



And here is a diagram for the output stage that driver the OCx pins

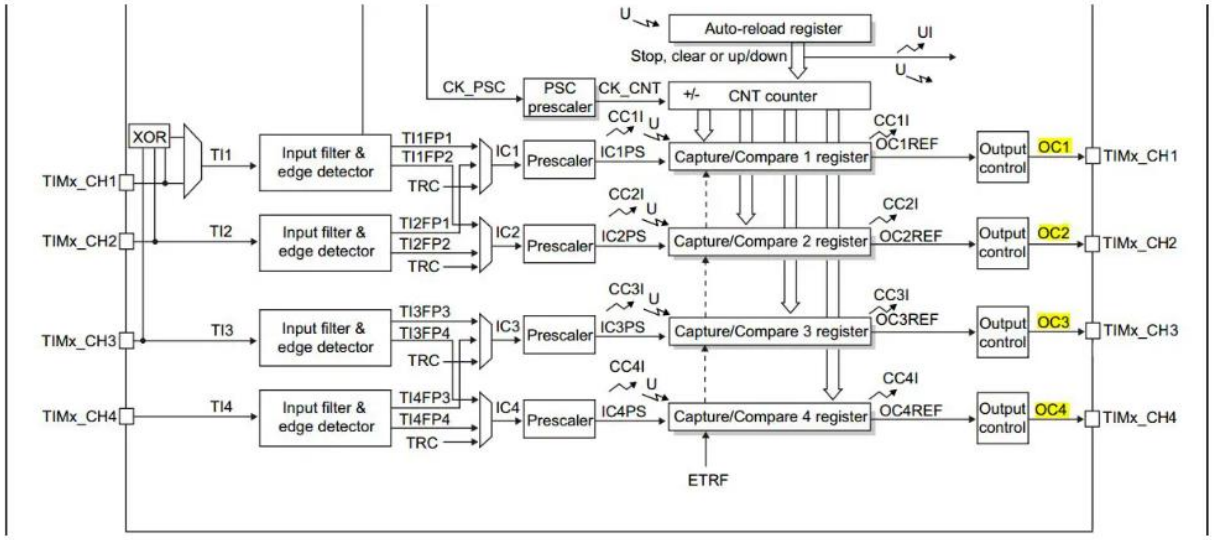
Output stage of capture/compare channel (channel 1)



A single STM32 timer usually has multiple channels (4, 6, or whatever is found in the datasheet). Therefore, using a single timer you can independently generate multiple PWM signals with different duty cycles of course, but they'll share the same timing (same frequency), and all of them will be in sync. We'll do this in the

2nd LAB in this tutorial after we set up a single PWM channel and get everything up and running.

Here is a snapshot of the general-purpose Timer2 diagram, which highlights the presence of multiple output compare channels and output drivers.



STM32 Timers In PWM Mode

Pulse width modulation mode allows for generating a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register. The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or ‘111 (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register.

The user must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register by setting the ARPE bit in the TIMx_CR1 register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. For applications where you need to generate complementary PWM signals, this option will be suitable for you.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx ≤ TIMx_CNT or TIMx_CNT ≤ TIMx_CCRx (depending on the direction of the counter). The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

Note

Note That PWM signals have a lot of properties that we need to control in various applications. The first of which is the frequency of the signal. And secondly, and probably the most important one, is the duty cycle. Third, is the PWM resolution. And much more to be discussed in later tutorials, we'll get into those 3 properties in the next sections below

STM32 PWM Frequency

In various applications, you'll be in need to generate a PWM signal with a specific frequency. In servo motor control, LED drivers, motor drivers, and many more situations where you'll be in need to set your desired frequency for the output PWM signal.

The PWM period ($1/F_{PWM}$) is defined by the following parameters: ARR value, the Prescaler value, and the internal clock itself which drives the timer module F_{CLK}. The formula down below is to be used for calculating the F_{PWM} for the output. You can set the clock you're using, the Prescaler, and solve for the ARR value in order to control the F_{PWM} and get what you want.

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1)}$$

STM32 PWM Duty Cycle

In normal settings, assuming you're using the timer module in PWM mode and generating a PWM signal in edge-aligned mode with up-counting configuration. The duty cycle percentage is controlled by changing the value of the CCRx register. And the duty cycle equals (CCRx/ARR) [%].

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARRx}[\%]$$

STM32 PWM Resolution

One of the most important properties of a PWM signal is the resolution. It's the number of discrete duty cycle levels that you can set it to. This number determines how many steps the duty cycle can take until it reaches the maximum value. So, the step size or the number of duty cycle steps can tell how fine can you change the duty cycle in order to achieve a certain percentage. This can be extremely important in some audio applications, motor control, or even light control systems.

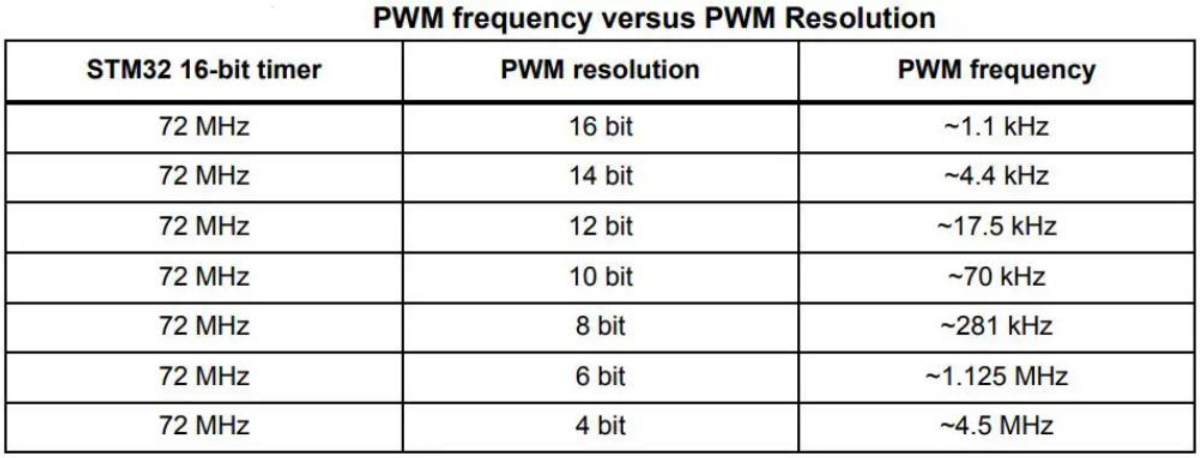
This is the STM32 PWM resolution formula that can be used to calculate the resolution of the PWM signal at a specific frequency or even the opposite. If you're willing to get a 10-Bit resolution PWM signal, what should the frequency be in order to achieve this? And so on..

$$Resolution_{PWM} = \frac{log(\frac{F_{CLK}}{F_{PWM}})}{log(2)}[Bits]$$

In other situations, you'll need to adjust the ARR value. Therefore, you'll need to know the relationship between it and the PWM resolution. This is not a new formula, it's derived from the first one and the F_{PWM} equation that you've seen earlier in this tutorial. We'll need it in later tutorials to design our Motor driver library and some other applications.

$$Resolution_{PWM}[Bits] = \frac{log(ARR + 1))}{log(2)}$$

Check this table which shows you some example frequencies and the PWM resolution at each F_{PWM} frequency.



PWM frequency versus PWM Resolution

STM32 16-bit timer	PWM resolution	PWM frequency
72 MHz	16 bit	~1.1 kHz
72 MHz	14 bit	~4.4 kHz
72 MHz	12 bit	~17.5 kHz
72 MHz	10 bit	~70 kHz
72 MHz	8 bit	~281 kHz
72 MHz	6 bit	~1.125 MHz
72 MHz	4 bit	~4.5 MHz

STM32 PWM Different Modes

The PWM signal generation can be done in different modes, I'll be discussing two of them in this section. The edge-aligned and the center-aligned modes.

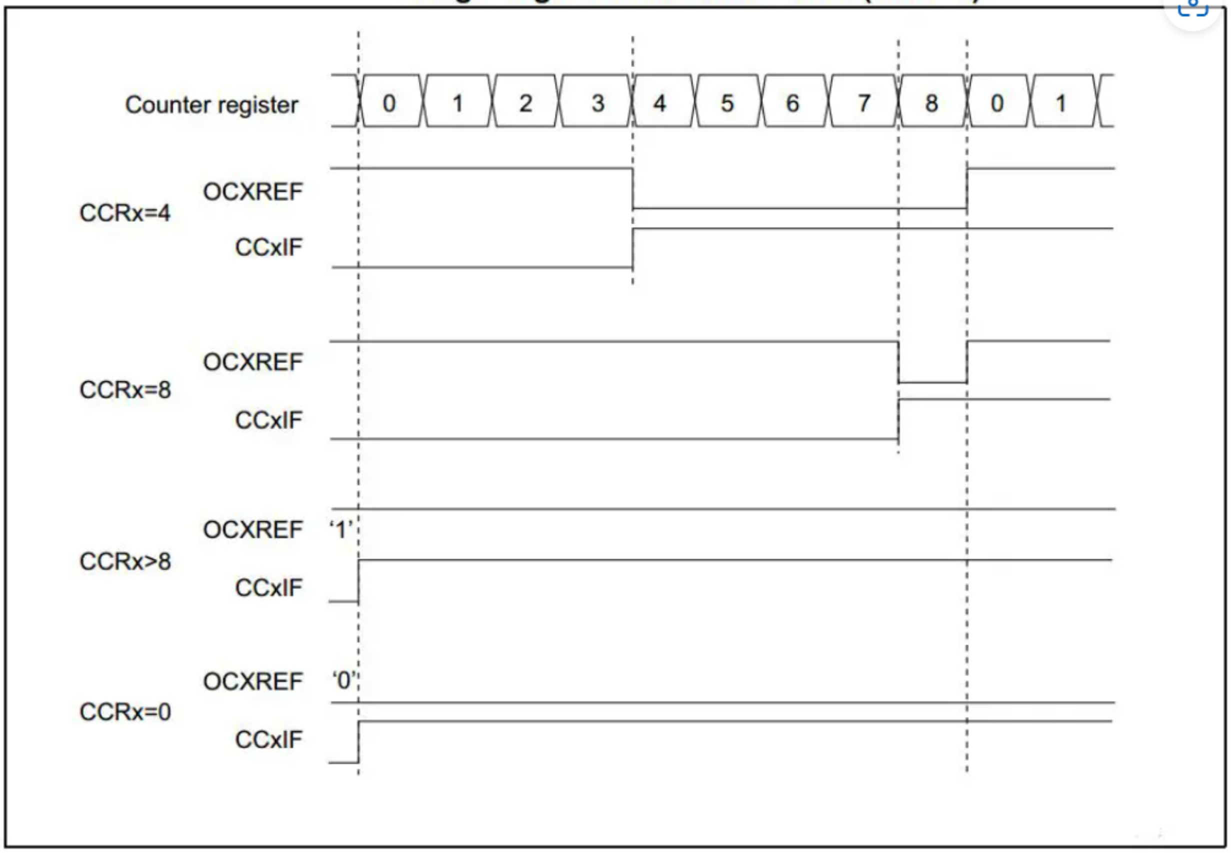
1- Edge-Aligned Mode

In the edge-aligned PWM mode there exist a couple of possible configurations:

- Up-Counting Configuration
- Down-Counting Configuration

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as TIMx_CNT <TIMx_CCRx else it becomes low. If the compare value in TIMx_CCRx is greater than the auto-reload value (in TIMx_ARR) then OCxREF is held at ‘1. If the compare value is 0 then OCxREF is held at ‘0.

Edge-aligned PWM waveforms (ARR=8)

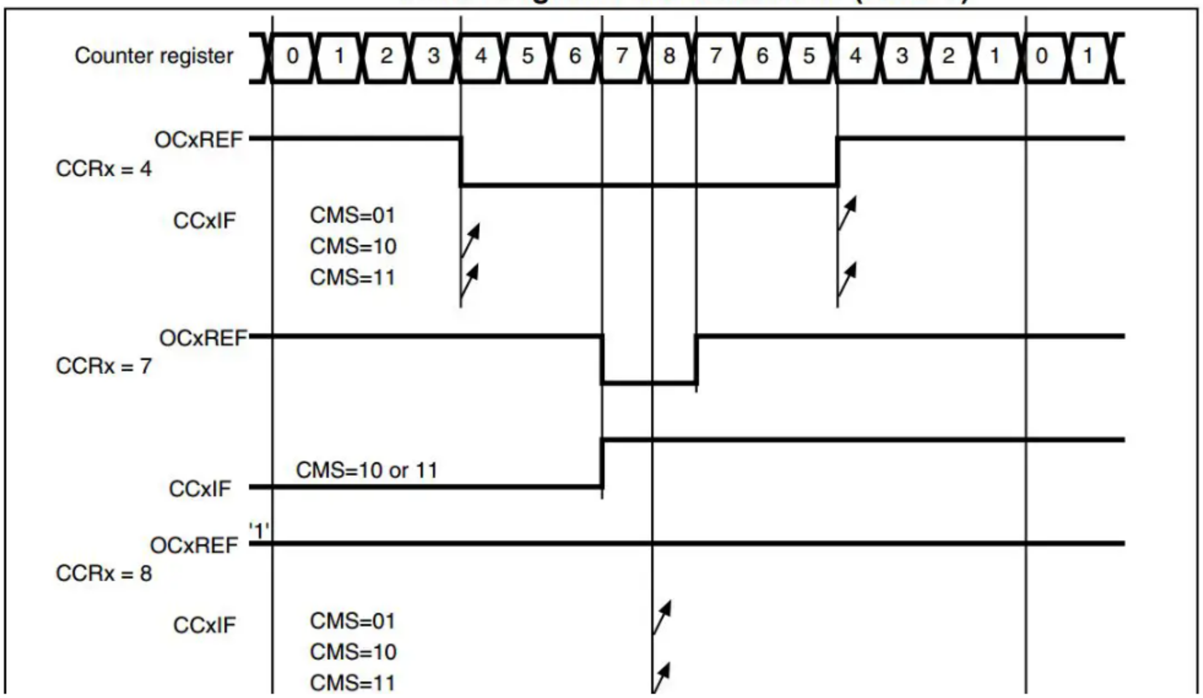


2- Center-Aligned Mode

The compare flag is set when the counter counts up when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx_CR1 register is updated by hardware and must not be changed by software.

The diagram below shows some center-aligned PWM waveforms in an example where: TIMx_ARR=8, PWM mode is the PWM mode 1.

Center-aligned PWM waveforms (ARR=8)



Steps to program for PWM

1. Enable the clocks for port A and Timer 2.

2. Configure PA5 for alternate function thru GPIOA->MODER.

3. Select the specific alternate functionality for PA5.(refer data sheet page 57, for PA5 -> AF1)

4. As a part of configuring the timers for PWM, first load a ARR value on TIM2 timer.

5. In the TIM2->CCMR register CC1S bits load a value which is relevant to PWM mode 1 upcounting.

6. Configure - OC1 signal as output on TIM2->CCER register.

7. Start the timer2 on CEN bit.

8. In an infinite loop keep varying the value you load onTIM2->CCR1 register with proper intermittent delays.

9. Observe the varying intensity of light on LED pin i.e PA5.