

LAB 03

TIMERS



UNIVERSITY
OF TRENTO

Prof. Davide Brunelli

Dept. of Industrial Engineering – DII

University of Trento, Italy

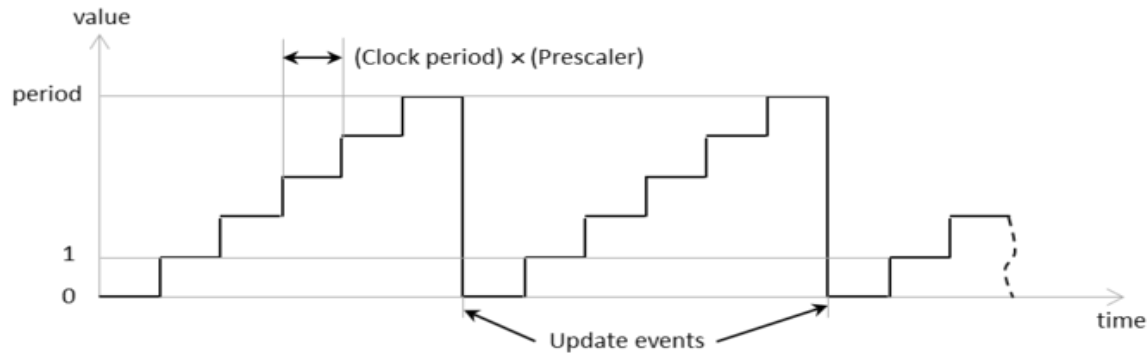
davide.brunelli@unitn.it



TIMERS

TIMERS

A hardware timer is essentially an **independent counter** that counts from zero to its maximum value at a given speed and generates various events. It runs in the background independently from your C/C++ program and its value typically follows the sequence depicted below:



It is basically a **global variable** (timer counter) that **increments** (or **decrements**) on the basis of a **programmable clock source**

TIMERS

The hardware of a timer is composed of **three basic programmable parts**:

- The **clock source**: the circuit that **generates the clock tick** for the timer
- The **time base**: the circuit that **derives the time granularity** from the clock source and contains the timer counter variable
- The **slave circuits**: provide **additional functions** (pulse measure, signal generation, etc.) by exploiting the timer variable

TIMERS

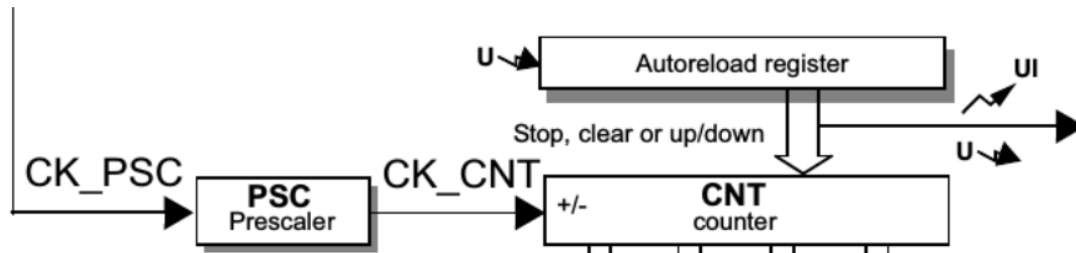
The **STM32 MCU** offers up to 11 different **timer/counters** with the following features:

- **Clock selection** (internal, external, other)
- **16/32-bit counter resolution**
- **Programmable prescaler**
- **Four independent channels** configurable as:
 - Input Capture
 - Output Compare
 - PWM Mode
 - One-pulse Output
- **Interrupt generation** on the basis of the various events that can occur

TIMERS

Counting is handled in the **time-base** by the following registers:

- **TIMx->PSC**: the prescaler register; it directly specifies the **division factor**
- **TIMx->CNT**: the counter register; it holds the counter value and increments (or decrements) it according to the input clock
- **TIMx->ARR**: the auto-reload register; **CNT** counts from 0 to **ARR**, then **CNT** is set to 0 again
- When **CNT** is **reloaded**, an **update event** is generated (the “U” in figure) which can trigger an **interrupt generation**

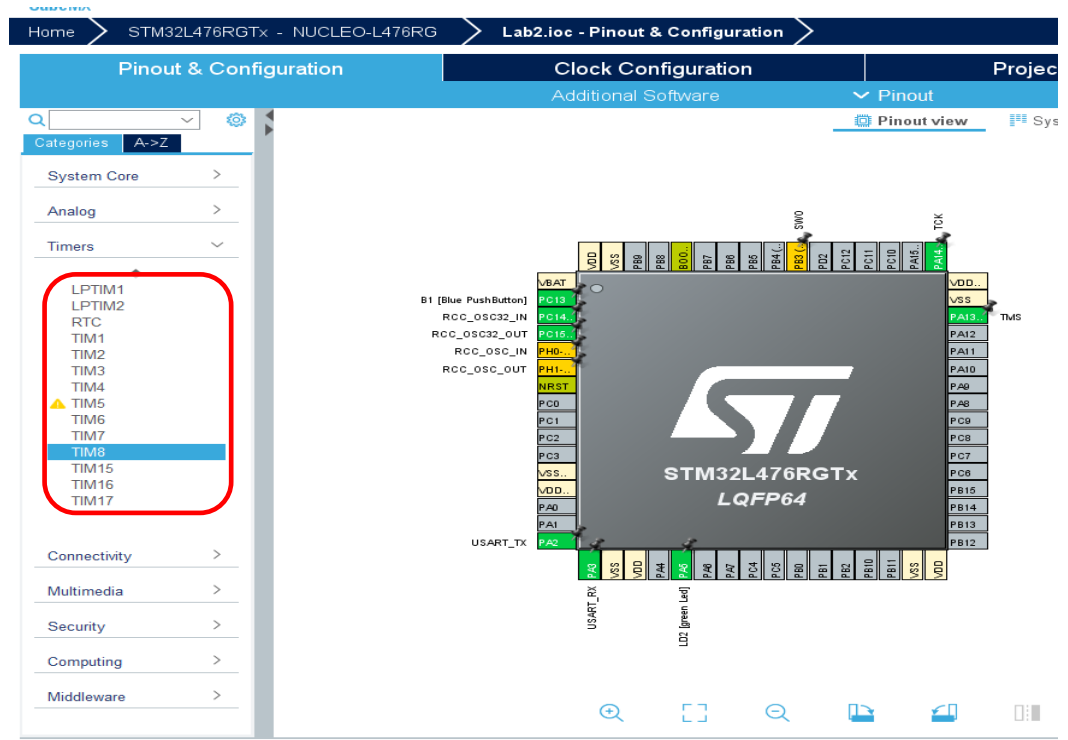


STM32 TIMERS

The STM32 embeds **multiple timers** providing timing resources for software or hardware tasks.

The software tasks mainly consist of providing **time bases, timeout event generation and time-triggers.**

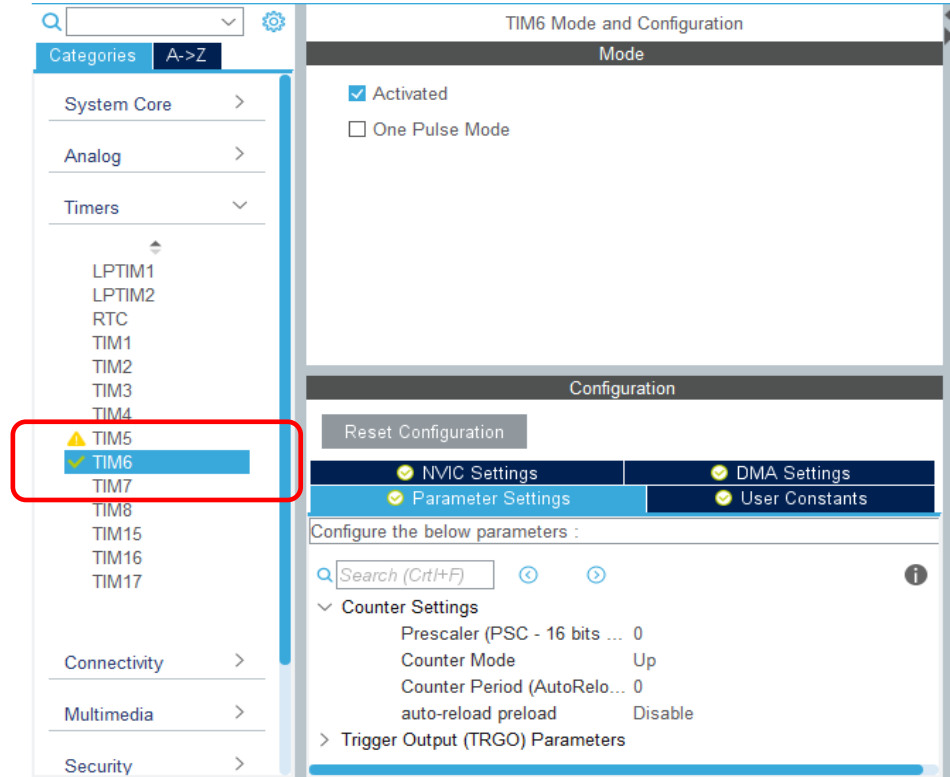
The hardware tasks are related to I/Os: the timers can **generate waveforms** on their outputs, **measure incoming signal** parameters and **react to external events** on their inputs.



STM32 TIMERS

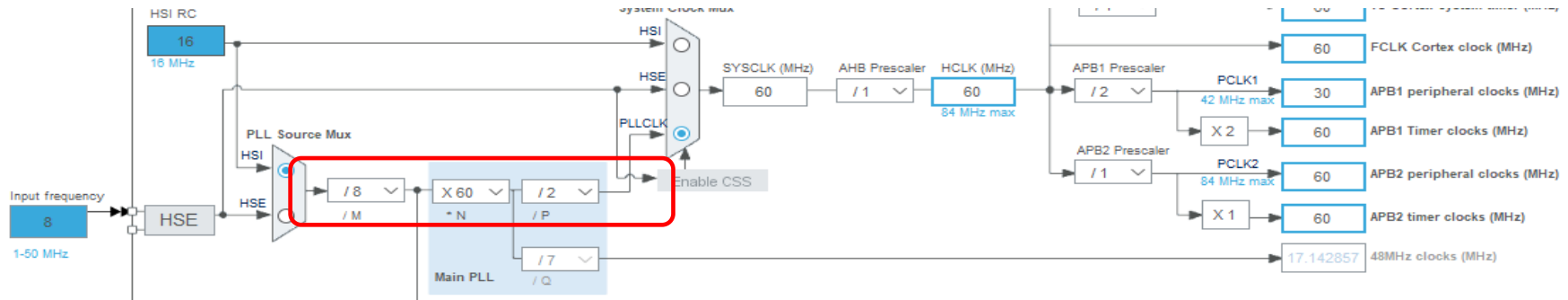
We will use **TIM6** for our board.

TIM6 is a simple Timer acting as a **Counter**. It just counts the clock cycles up to the maximum value set and then reset.



STM32 TIMERS

Timer tick time (or timer resolution) is based on **APBx** clock. It can be scaled using a **16 Bit prescaler**. We can modify the base clock, to simplify the pre-scaler division.



$$t_{TIMx} = \frac{\text{Prescaler} + 1}{APBx_{CLK}}$$

STM32 TIMERS

STM32CubeMX Untitled*: STM32L476RGTx NUCLEO-L476RG

File Window Help

Home > STM32L476RGTx - NUCLEO-L476RG > Untitled - Pinout & Configuration > GENERATE CODE

Pinout & Configuration Clock Configuration Project Manager Tools

Software Packs Pinout

Categories A-Z

System Core

- DMA
- GPIO
- IWDG
- NVIC
- RCC
- SYS
- TSC
- WWDG

Analog

Timers

- LPTIM1
- LPTIM2
- RTC
- TIM1
- TIM2
- TIM3
- TIM4
- TIM5
- TIM6**
- TIM7
- TIM8
- TIM15
- TIM16
- TIM17

Connectivity

TIM6 Mode and Configuration

Mode

☒ Activated
☐ One Pulse Mode

$$T = \frac{(\text{Prescaler} + 1) \cdot (\text{Period} + 1)}{f_{CLK}}$$

Configuration

Reset Configuration

User Constants NVIC Settings DMA Settings

Parameter Settings

Configure the below parameters:

Search (Ctrl+F)

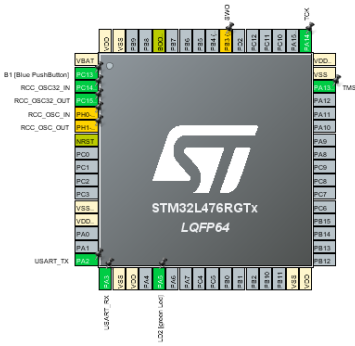
Counter Settings

- Prescaler (PSC - 16 bits value) 1000
- Counter Mode Up
- Counter Period (AutoReload Re... 9000
- auto-reload preload Disable

Trigger Output (TRGO) Parameters

Trigger Event Selection Reset (UG bit from TIMx_EGR)

Pinout view System view



$$\text{Period} = \frac{T \cdot \text{APBx}_{CLK}}{\text{Prescaler} + 1} - 1$$

$$t_{TIMx} = \frac{\text{Prescaler} + 1}{\text{APBx}_{CLK}}$$

STM32 TIMERS – INTERRUPT

We can activate the interrupt using the NVIC configuration tab in CubeMX

The screenshot displays the STM32CubeMX software interface for configuring the NVIC (Nested Vectored Interrupt Controller). The 'NVIC' tab is selected in the left sidebar. The main area shows the 'NVIC Mode and Configuration' settings, including 'Priority Group' and 'Search'. Below this is the 'NVIC Interrupt Table' with columns for the interrupt name, 'Enabled' status, and priority.

Interrupt Name	Enabled	Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0
Memory management fault	<input checked="" type="checkbox"/>	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0
Debug monitor	<input checked="" type="checkbox"/>	0
Pendable request for system service	<input checked="" type="checkbox"/>	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0
PVD/PVM1/PVM2/PVM3/PVM4 interrupts through EXTI lines 16/35/36/37/38	<input type="checkbox"/>	0
Flash global interrupt	<input type="checkbox"/>	0
RCC global interrupt	<input type="checkbox"/>	0
USART2 global interrupt	<input type="checkbox"/>	0
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	0
TIM6 global interrupt, DAC channel1 and channel2 underrun error interrupts	<input checked="" type="checkbox"/>	0
FPD global interrupt	<input type="checkbox"/>	0

STM32 TIMERS – INTERRUPT

Once the timer has reached its counting period, an interrupt will be fired. Using the proper callback function we can create timed tasks (like the blinking of a LED)

```
289 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
290 {
291     // The CUBEMX was set as: Prescaler: 1000, Counter Period (ARR): 9000, HCLK: 9 MHz
292     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
293 }
294
295 /* USER CODE END 4 */
```

After the Peripheral Initializations, remember to start TIM6 In Interrupt Mode.

```
277 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
278 {
279     HAL_TIM_Base_Start_IT(&htim6); // Start the timer
280 }
281
```

STM32 TIMERS – RESOURCES

- General-purpose timer cookbook for STM32 microcontrollers
 - https://www.stmicroelectronics.com.cn/content/ccc/resource/technical/document/application_note/group0/91/01/84/3f/7c/67/41/3f/DM00236305/files/DM00236305.pdf/jcr:content/translations/en.DM00236305.pdf
- STM32L4 Timers
 - https://www.st.com/content/ccc/resource/training/technical/product_training/c4/1b/56/83/3a/a1/47/64/STM32L4_WDG_TIMERS_GPTIM.pdf/files/STM32L4_WDG_TIMERS_GPTIM.pdf/jcr:content/translations/en.STM32L4_WDG_TIMERS_GPTIM.pdf



TOGGLE THE LED USING A TIMER

TOGGLE THE LED USING A TIMER

Use the previously configured timer to toggle the green LED every 1 second.

1. Use CubeMX to **configure a simple timer**
2. Configure the board clock to provide an **APB clock equal to 60 MHz**
3. If needed, pre-scale the clock to a suitable frequency for the timer
4. Set the timer counting period
5. Start the timer **using the blue button**
6. Using the interrupt callback, **toggle the green LED**

CHANGE THE TIMER COUNTING PERIOD

CHANGE THE TIMER COUNTING PERIOD

Using an interrupt generated by an external input, cycle the timer counting period between 0 – 0.25s – 0.5s – 1s – 2s

1. Use CubeMX to **configure a simple timer**
2. Configure the board clock to provide an **APB clock equal to 60 MHz**
3. If needed, pre-scale the clock to a suitable frequency for the timer
4. Set the timer counting period
5. Using the interrupt callback, **cycle the counting period** of the configured timer by updating the relative register.