POLITECNICO
DI MILANO

www.polimi.it

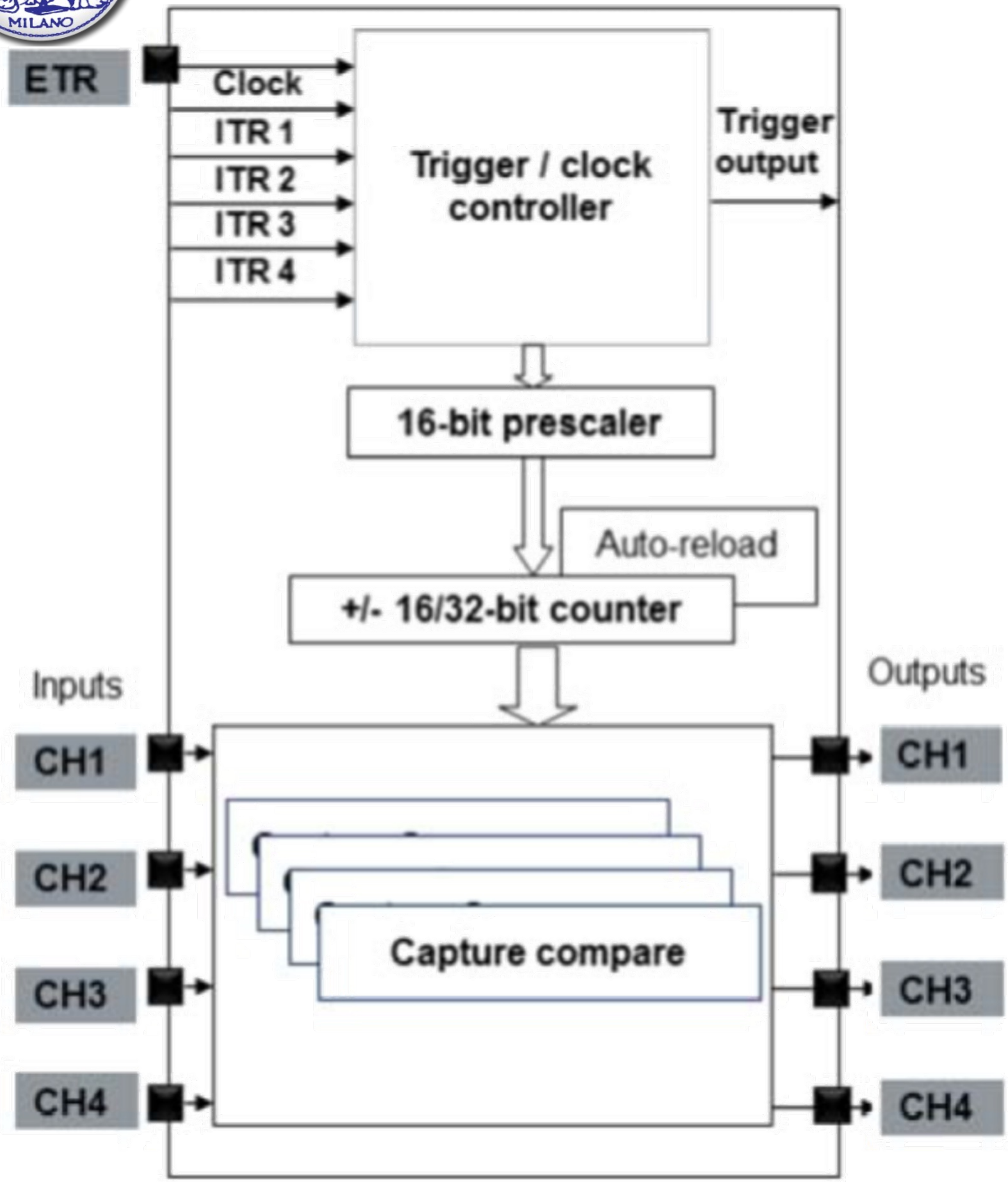




Timers

Dr. Federica Villa

Timer structure



Trigger / clock controller

- selects input clock
- synchronization with other clocks
  (Master → Output Trigger
  Slave → Input Trigger)

Prescaler

Divides the frequency the counter increases, skipping a portion of the incoming events.

Counter

Up, Down, Up/Down counter

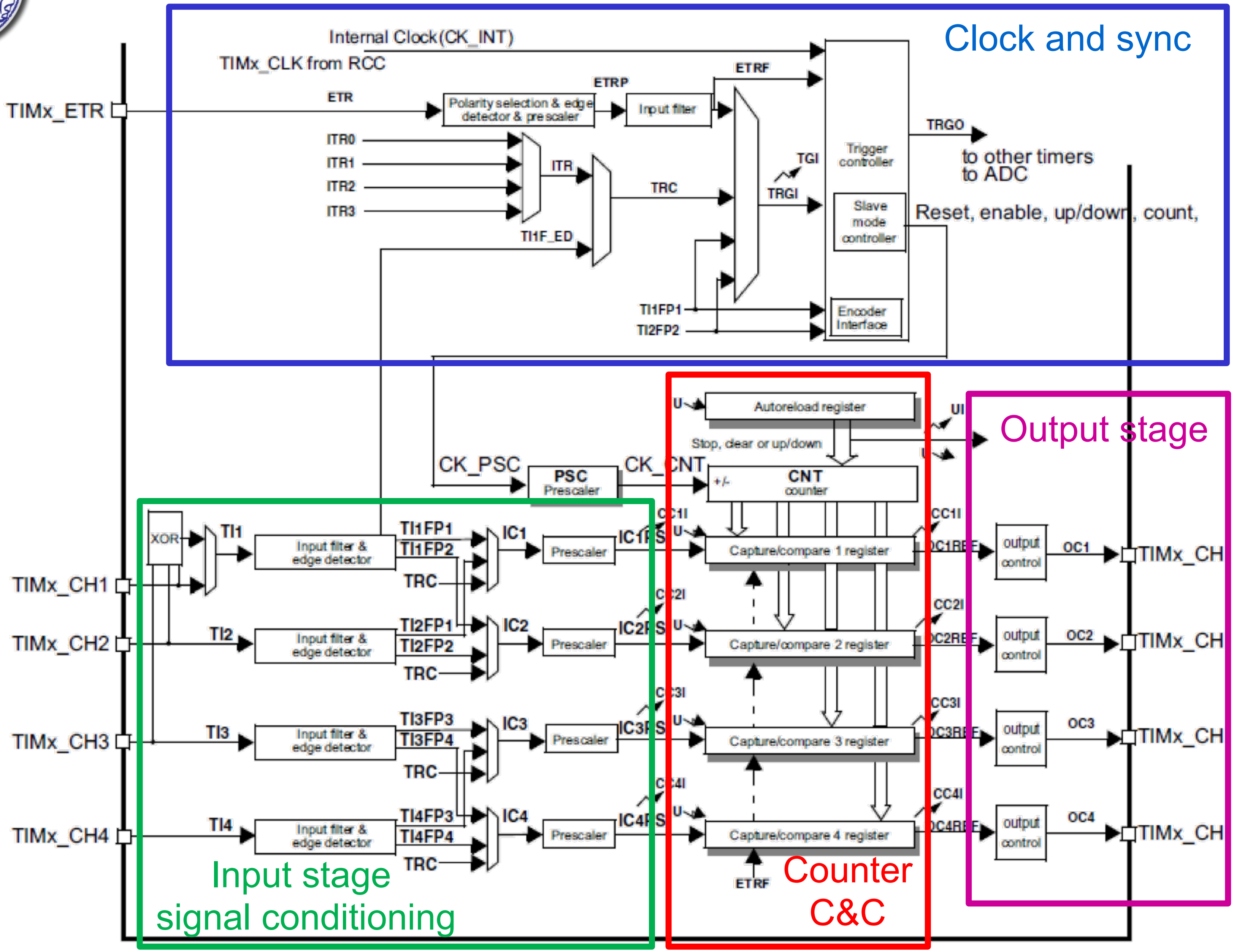
Capture and Compare registers

One for each channel

Auto-reload register

Sets the timer FSR

Timer block diagram



Timers functionalities

- **Time base**

  Generates triggers and interrupts at the overflow (or at the compare value) (e.g., trigger ADC conversion, software management, generic interrupts).

- **Input capture**

  Capture the counter value in the capture register:
  - one input can be mapped to 2 capture channels
  - programmable edge sensitivity
  - event prescaler
  - digital filter

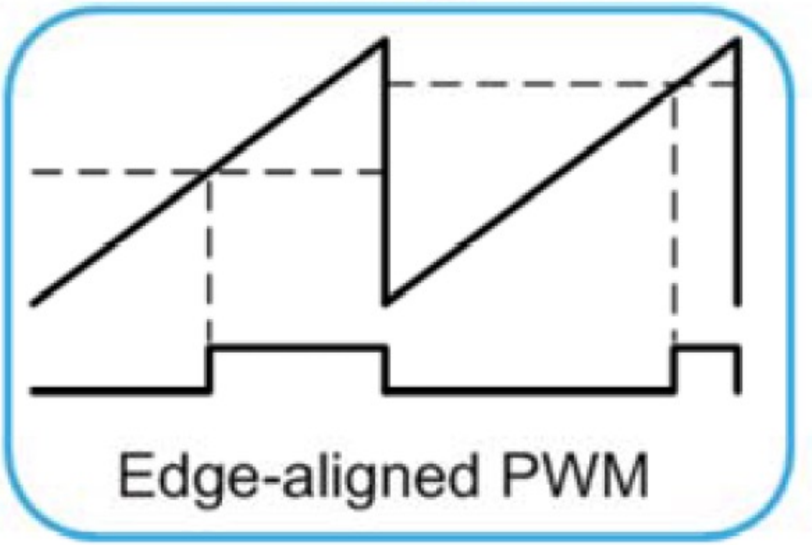
- **Output compare**

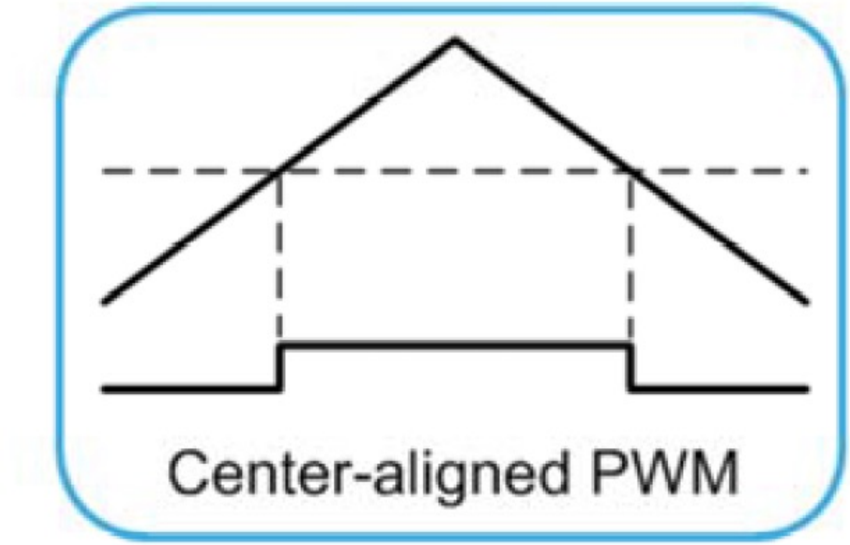
  Every time that the compare value is reached:
  - corresponding output pin is SET/RESET/TOGGLE/UNCHANGED
  - generate an interrupt / trigger

- **PWM generation**

  Edge aligned: counter Up or Down mode
  Center aligned: counter Up/Down mode

Pulse Width Modulator (PWM)





PWM frequency: $$f_{PWM} = \frac{f_{TIM}}{(ARR+1)\cdot(PSC+1)}$$

PWM Duty Cycle: $$DC = \frac{CCRx+1}{ARR+1}$$

Registers

- **Prescaler (PSC)**

  Note that in order to divide by N the clock frequency you need to write N-1 in the Prescaler register

- **Auto Reload Register (ARR)**

  FSR of the counter (counter overflow)

- **Capture / Compare Register** (one for each channel) **(CCRx)**

  Capture = counter value when trigger is received
  Compare = counter value which generate an event (trigger or PWM commutation).

A complete list of the timer registers can be found in the Reference Manual.

Timer HAL functions

There are many HAL functions for timers.

In the next projects we will use the following functions:

HAL_StatusTypeDef **HAL_TIM_PWM_Start**(TIM_HandleTypeDef *htim, uint32_t Channel)

HAL_StatusTypeDef **HAL_TIM_PWM_Stop**(TIM_HandleTypeDef *htim, uint32_t Channel)

Starts(stops) the PWM signal generation

HAL_StatusTypeDef **HAL_TIM_Base_Start**(TIM_HandleTypeDef * htim)

HAL_StatusTypeDef **HAL_TIM_Base_Start_IT**(TIM_HandleTypeDef * htim)

HAL_StatusTypeDef **HAL_TIM_Base_Stop**(TIM_HandleTypeDef * htim)

HAL_StatusTypeDef **HAL_TIM_Base_Stop_IT**(TIM_HandleTypeDef * htim)

Starts(stops) the TIM Base generation (*x*_IT with interrupt generation).

Project 1c: Blinking LED - PWM

Objective of this project is
to **blink the NUCLEO board green LED**
at 1 Hz with 50% DC,
using a PWM.

Project hints

1. Use the NUCLEO board Manual to find the GPIO connected to the green LED.
2. Set in CUBE the GPIO of the green LED as the output of a PWM
3. Check the TIMER clock frequency and configure the corresponding timer in order to obtain the required frequency and duty cycle.
4. Generate the c code.
5. In the main, start the correct channel of the timer connected to the LED.
6. Debug and verify the blinking frequency.

Project 2: Play note through speaker

Objective of this project is
to **play a tone using the speaker**,
using a PWM.

Project hints

1. Use the NUCLEO board Manual to find the GPIO connected to the speaker.

2. Set in CUBE the GPIO of the speaker as the output of a PWM

3. Check the TIMER clock frequency and configure the corresponding timer in order to obtain a 440 Hz tone (LA4 / A4 note) with 50 % duty cycle.

4. Generate the c code.

5. In the main, start the correct channel of the timer connected to the speaker. Stop it a few seconds later.

6. Debug and verify the note frequency. (Spectrum analyzer apps are easily available for smartphones)

Project 2B: Play a song

Objective of this project is
to **play a song using the speaker**

Suggested song score

Timer configuration for the various notes can ba computed using the file "Musical notes.xslx" available on WeBeep

Project hints

1. Start from the same configuration as in the previous project.

2. Identify the setup functions that initialize the timer frequency and pulse duration. Use the required code from those functions to create a new function to setup a new frequency and pulse duration of the timer.

3. Define a struct to store the note frequency and duration

4. Create an array of structs to store the song

5. Play the song by stepping through the array, every time setting the correct frequency and for the correct duration of the note.

Project 2C: Play a song (interrupt)

Objective of this project is
to **play a song using the speaker**
when the microphone detects
a loud sound

Project hints

1. Start from the same configuration as in the previous project.

2. Setup the correct interrupt and using the correct callback, play the song

3. Is the song being played correctly? If not, try to debug your code and propose a possible solution to the problem.

4. If the song is playing correctly, does it stop at the end? If not, debug your code and propose a possible solution to the problem.