LABORATORIO DI ARCHITETTURE E PROGRAMMAZIONE DEI SISTEMI ELETTRONICI INDUSTRIALI

Laboratory Lesson 4:

-Timers

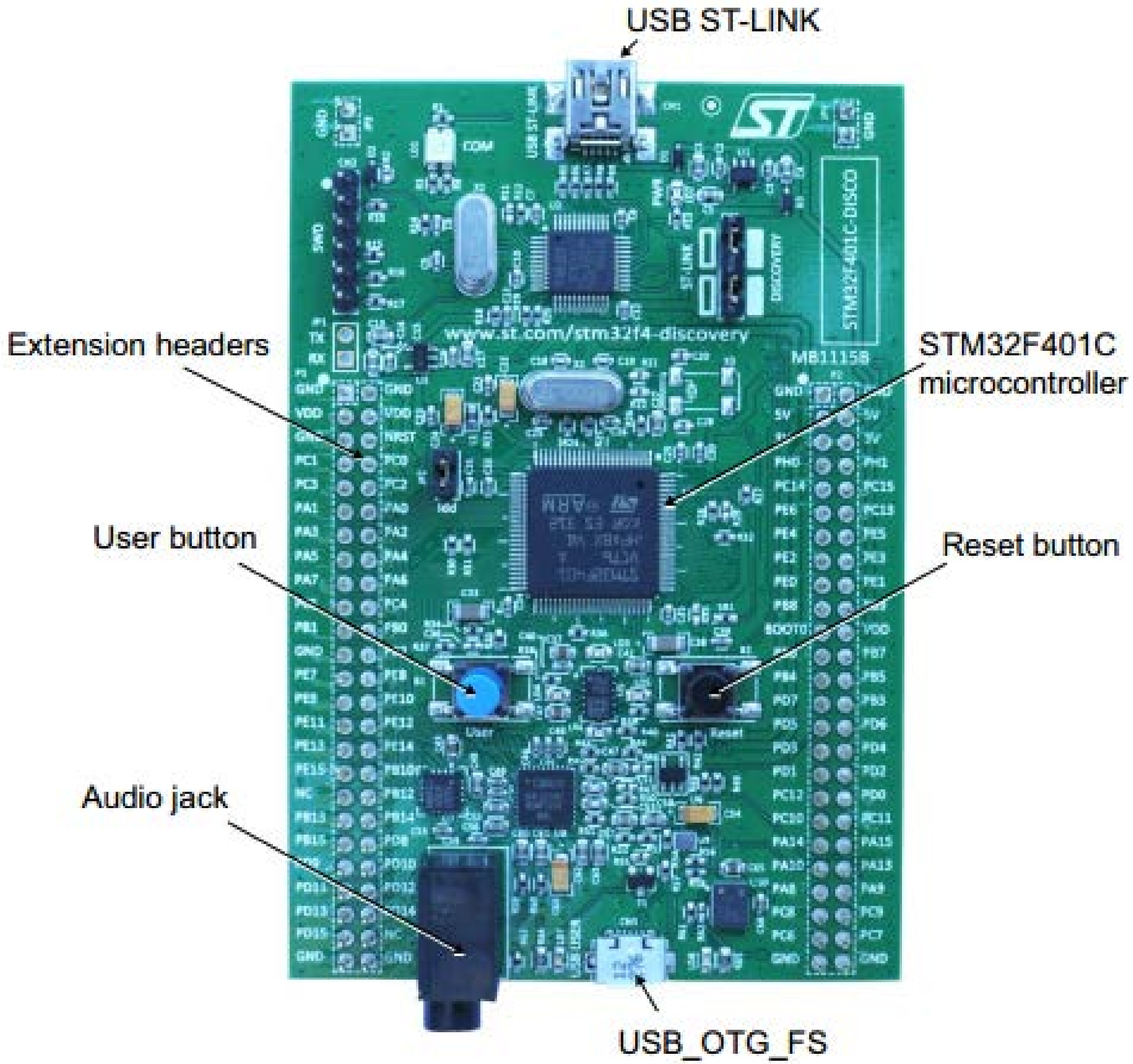
Prof. Luca Benini <luca.benini@unibo.it>

Domenico Balsamo <domenico.balsamo@unibo.it>
Filippo Casamassima <filippo.casamassima@unibo.it>

Course Organization

- Hands-on session LAB1 **Thursday 15.00 – 19.00**
- Prof Benini Friday **9.00 – 11.00 room 5.5**
- Lab is available **Friday 11.00 – 13.00**
- Check website for **announcements, course material:** http://www-micrel.deis.unibo.it/LABARCH
- Final Exam:
  - Homeworks (**to be checked weekly**)
  - Final project
  - Final discussion (homeworks + final project)

STM32F401 Discovery Kit



References:

- STM32F401xB STM32F401xC datasheet

- STM32F40xxx advanced ARM®-based 32-bit MCUs reference manual (RM0344)

- Discovery kit for STM32F401 line (UM1669)

- Getting started with STM32F401 Discovery software development tools (UM1671)

#6 Timers

Timers

- The general-purpose timers consist of a **16-bit auto-reload counter** driven by a programmable prescaler.

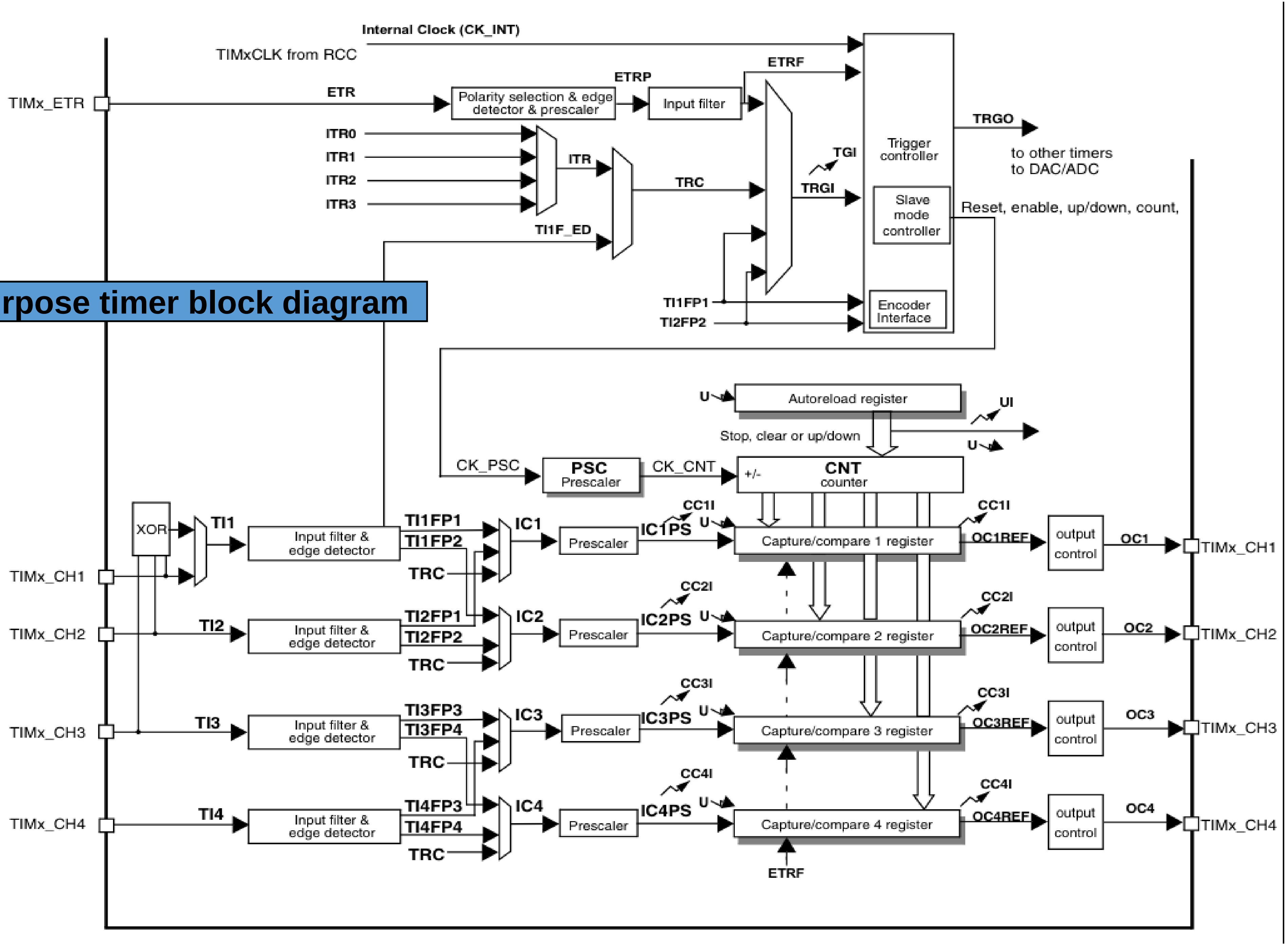
- They may be used for a variety of purposes, including **measuring the pulse lengths of input signals** (input capture) or **generating output waveforms** (PWM).

- Pulse lengths and waveform periods can be modulated **from a few microseconds to several milliseconds** using the timer prescaler and the RCC clock controller prescalers.

Timers

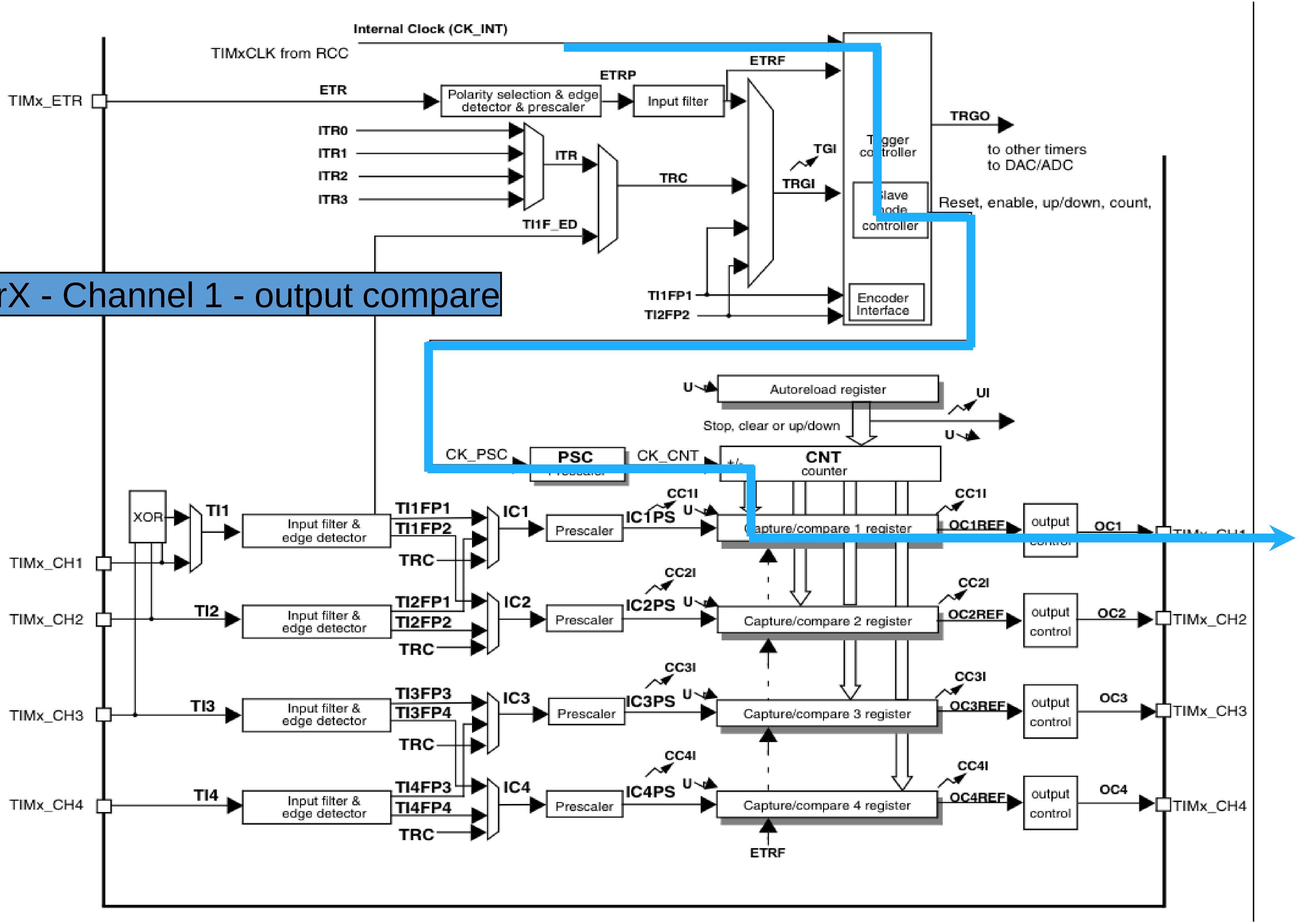
- General-purpose TIMx timer features include:
  - 16-bit up, down, up/down auto-reload counter.
  - 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535.
  - **Up to 4 independent channels** for:
    - Input capture
    - Output compare
    - PWM generation (Edge- and Center-aligned modes)
    - One-pulse mode output

Timers



General-purpose timer block diagram

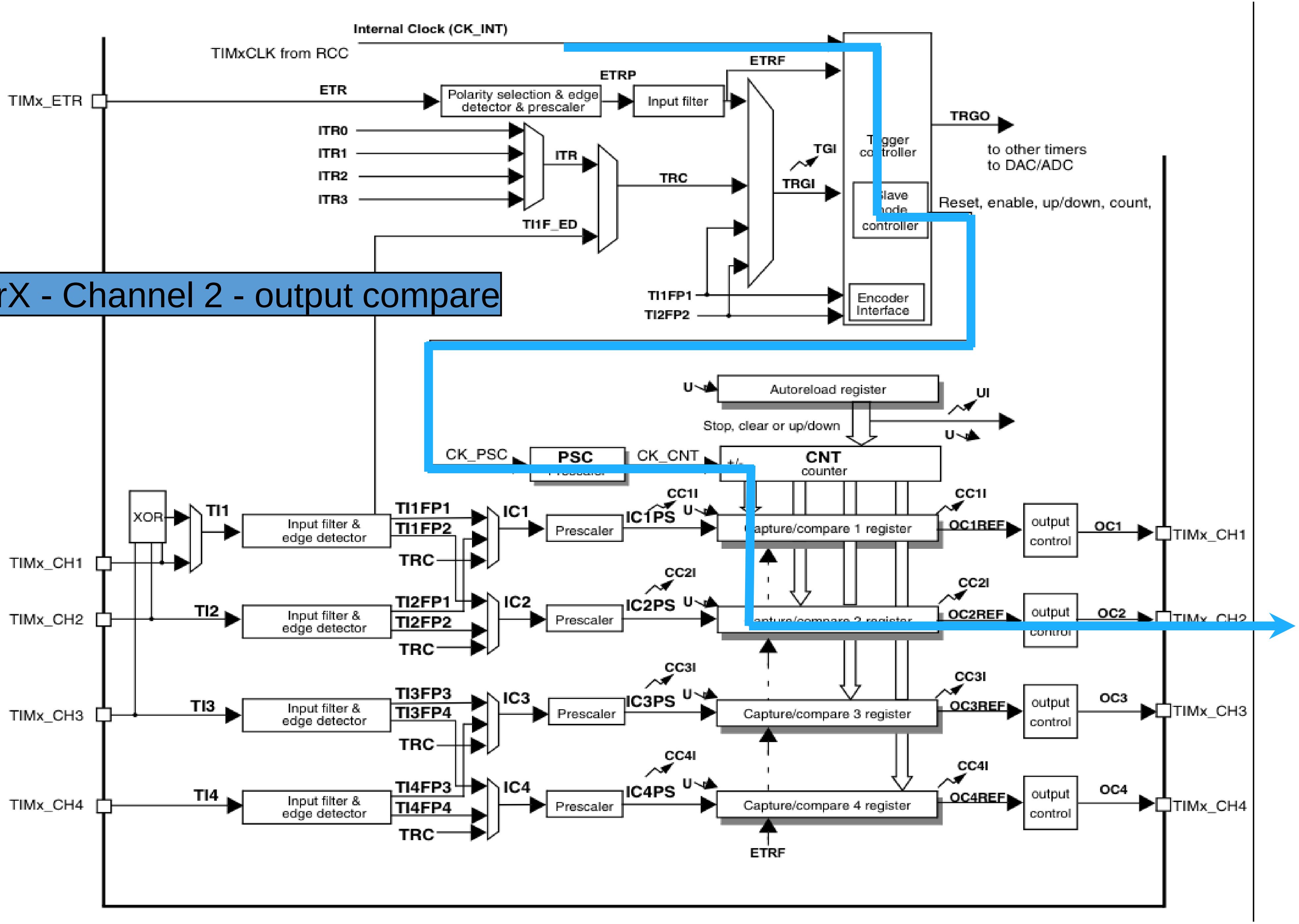
Timers



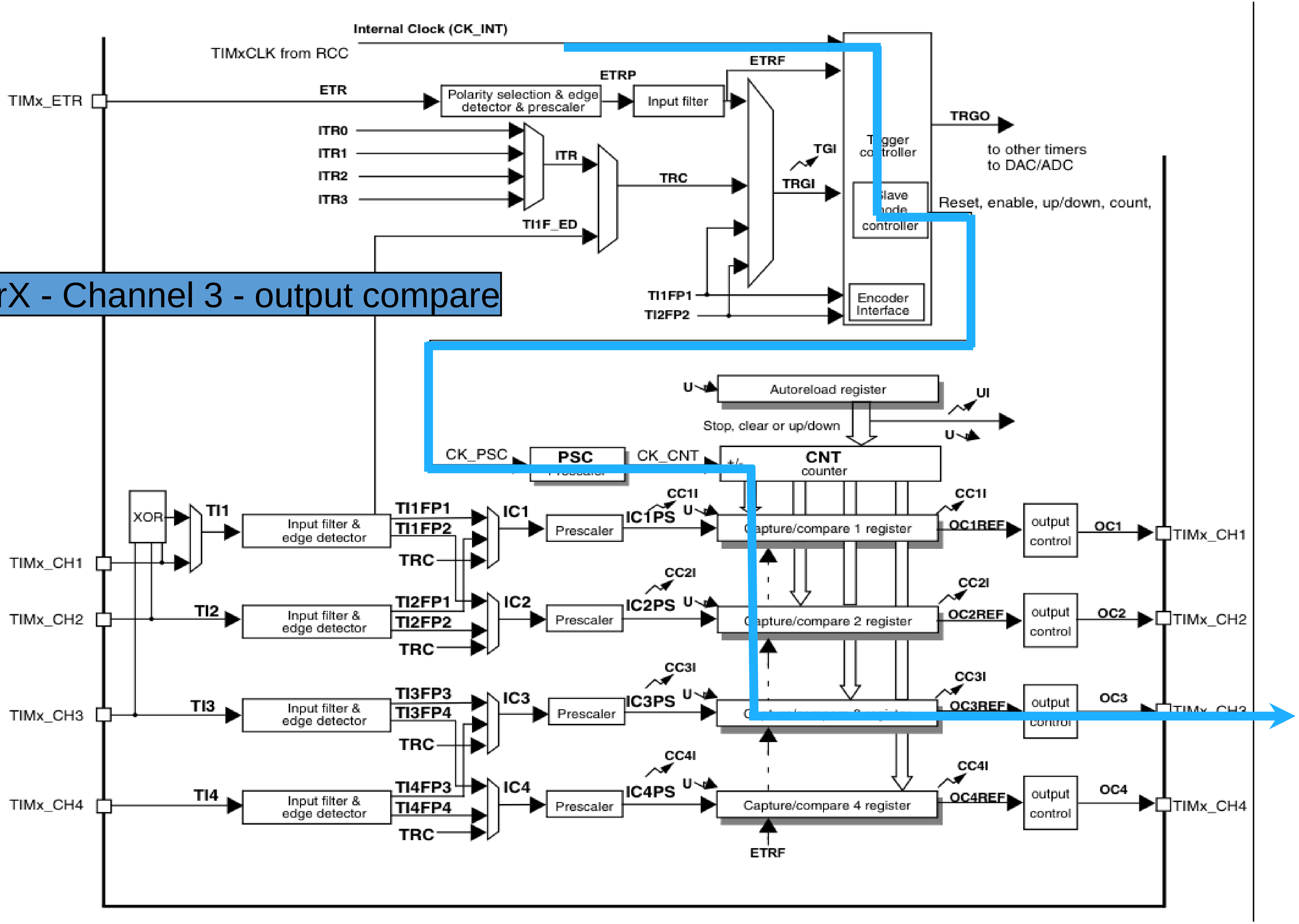
PATH: TimerX - Channel 1 - output compare

Timers

PATH: TimerX - Channel 2 - output compare

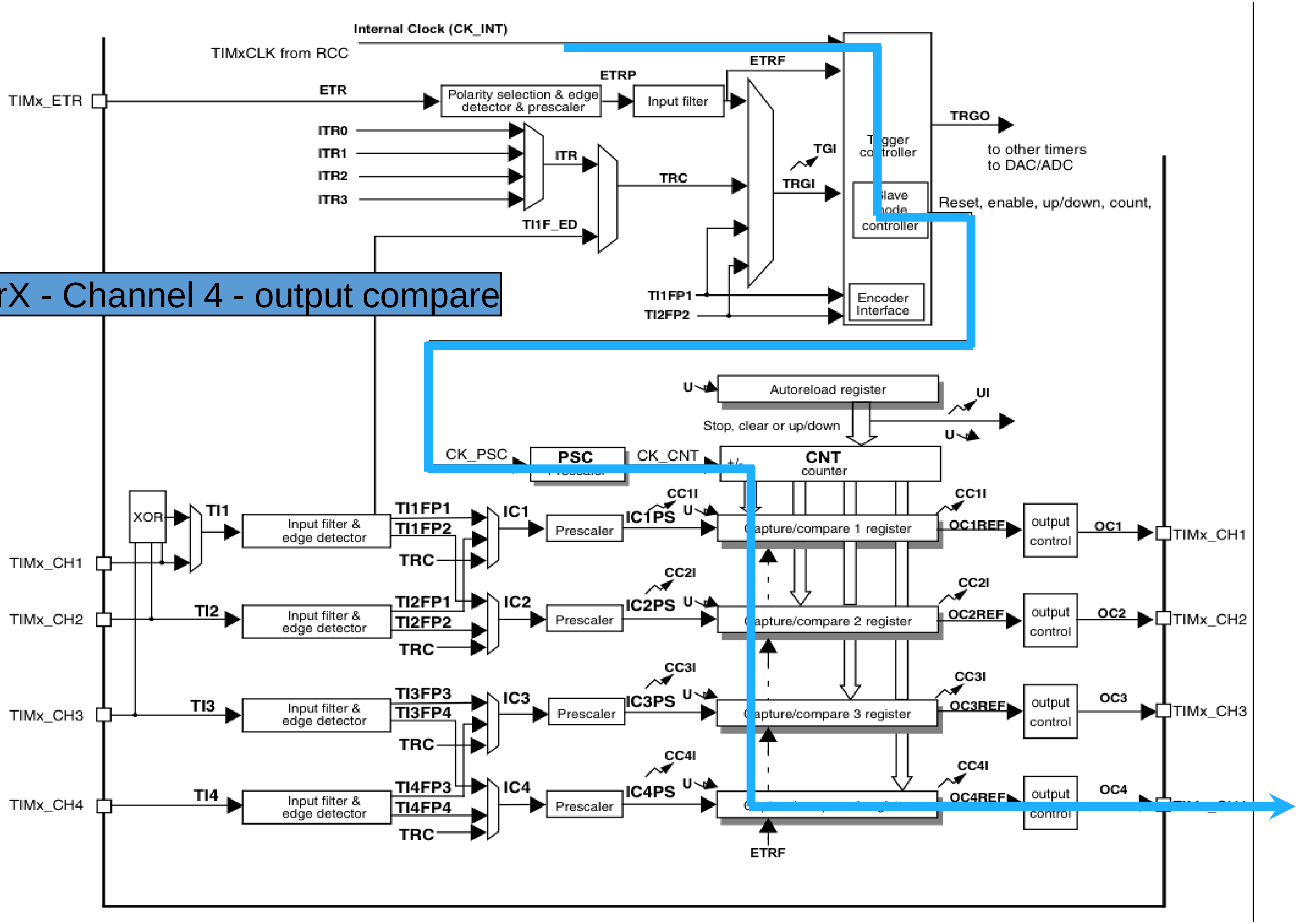


Timers



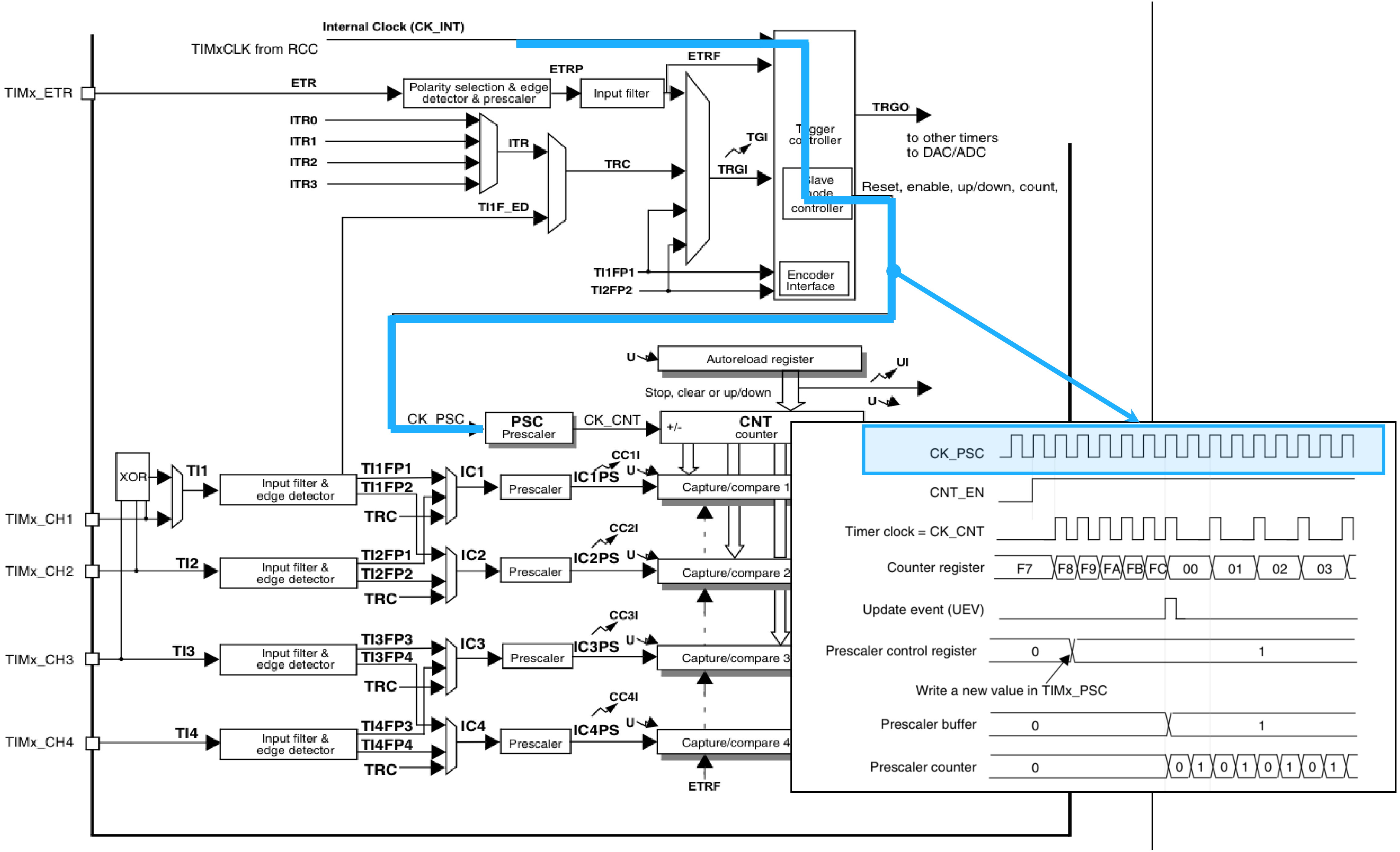
PATH: TimerX - Channel 3 - output compare

Timers



PATH: TimerX - Channel 4 - output compare

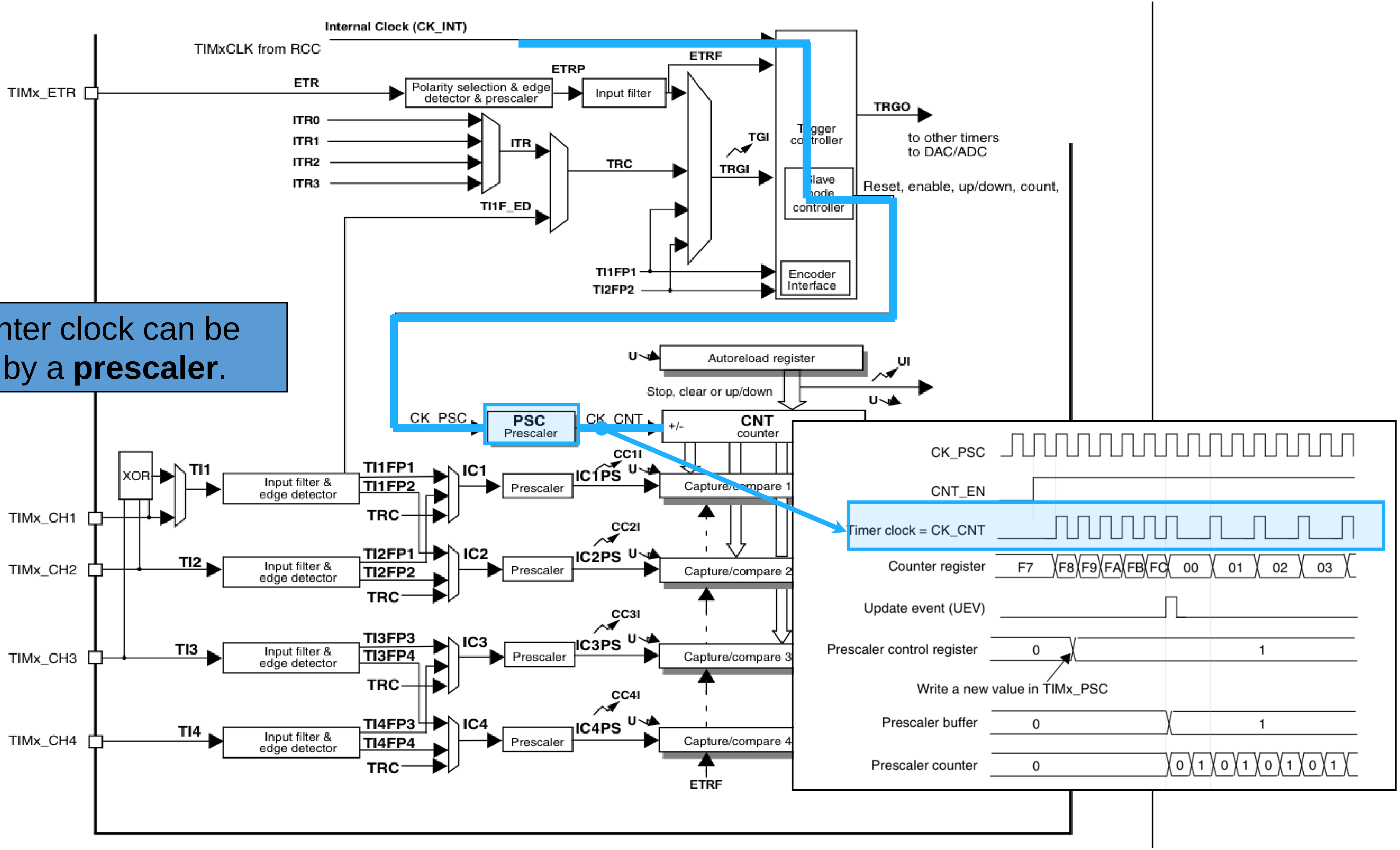
Timers



Internal Clock (CK_INT)
TIMxCLK from RCC
ETRF
ETRP
TIMx_ETR
ETR
Polarity selection & edge detector & prescaler
Input filter
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TI1F_ED
TGI
TRGI
Trigger controller
Slave mode controller
Encoder Interface
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
TI1FP1
TI2FP2
U
Autoreload register
UI
Stop, clear or up/down
U
CK_PSC
PSC
Prescaler
CK_CNT
+/-
CNT
counter
XOR
TI1
TIMx_CH1
Input filter & edge detector
TI1FP1
TI1FP2
TRC
IC1
Prescaler
CC1I
U
IC1PS
Capture/compare 1
TIMx_CH2
TI2
Input filter & edge detector
TI2FP1
TI2FP2
TRC
IC2
Prescaler
CC2I
U
IC2PS
Capture/compare 2
TIMx_CH3
TI3
Input filter & edge detector
TI3FP3
TI3FP4
TRC
IC3
Prescaler
CC3I
U
IC3PS
Capture/compare 3
TIMx_CH4
TI4
Input filter & edge detector
TI4FP3
TI4FP4
TRC
IC4
Prescaler
CC4I
U
IC4PS
Capture/compare 4
ETRF
CK_PSC
CNT_EN
Timer clock = CK_CNT
Counter register
F7
F8
F9
FA
FB
FC
00
01
02
03
Update event (UEV)
Prescaler control register
0
1
Write a new value in TIMx_PSC
Prescaler buffer
0
1
Prescaler counter
0
0
1
0
1
0
1
0
1

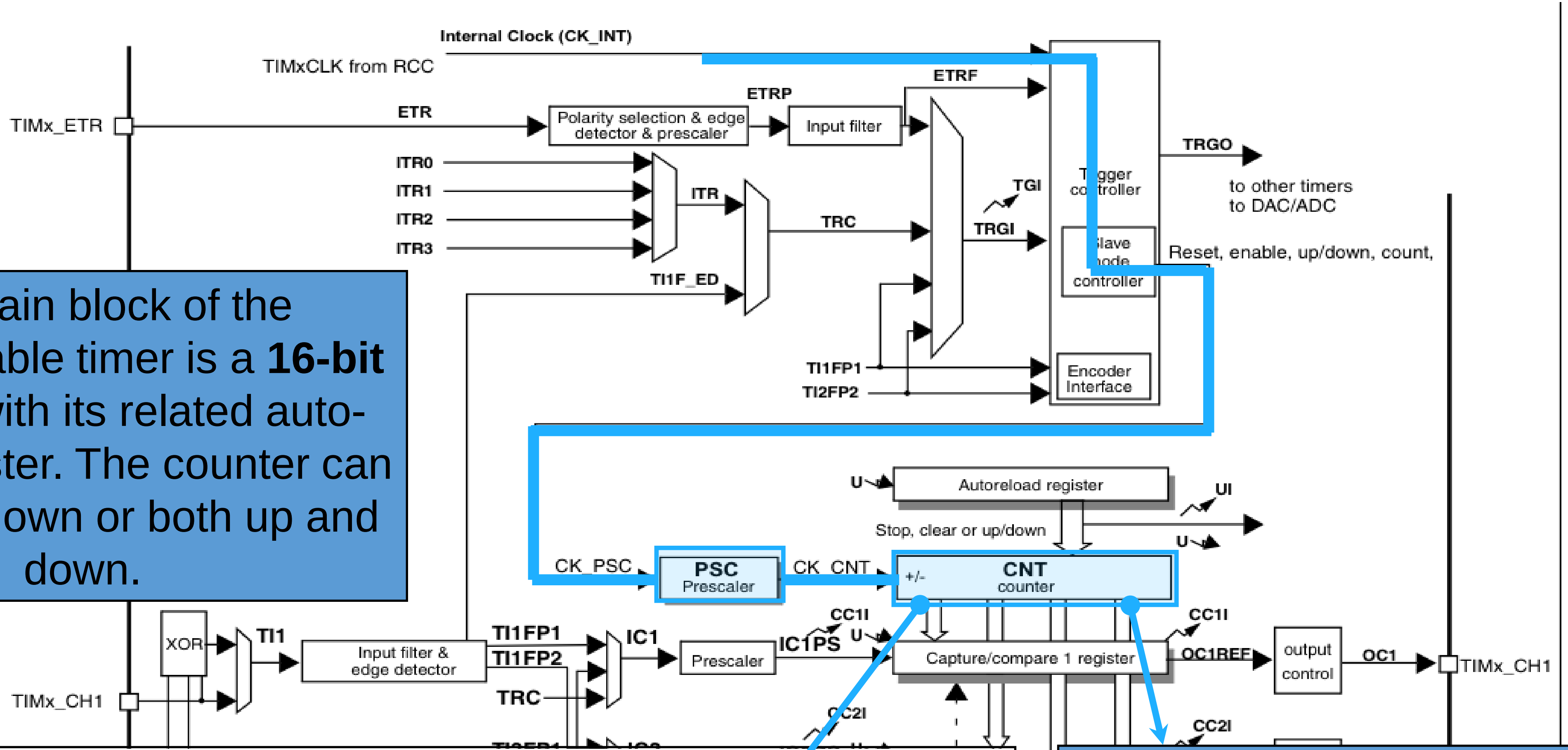
Timers

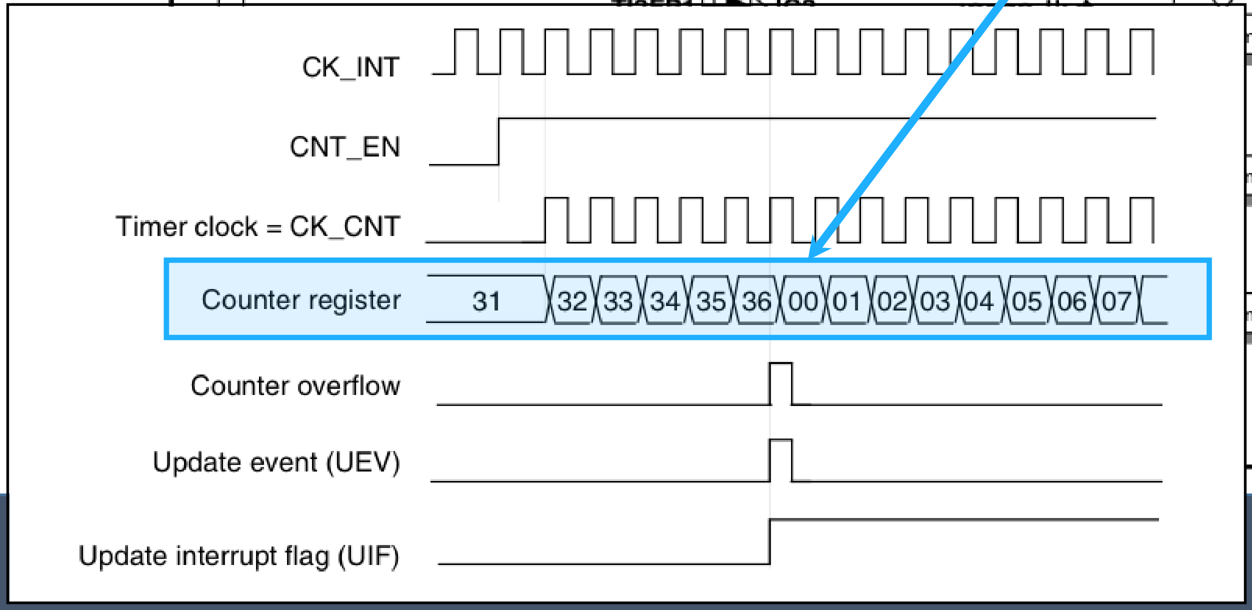
The counter clock can be divided by a **prescaler**.

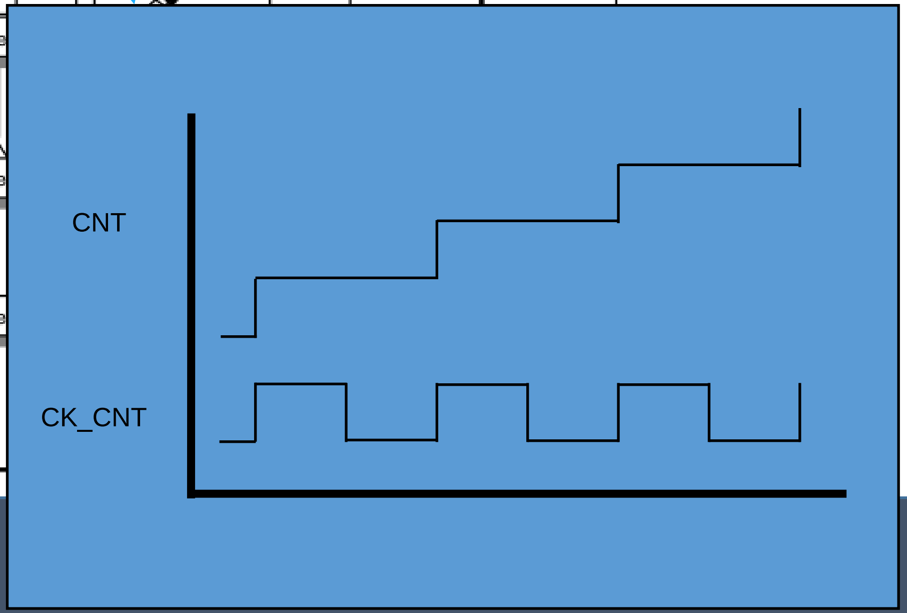


Timers

The main block of the programmable timer is a **16-bit counter** with its related auto-reload register. The counter can count up, down or both up and down.

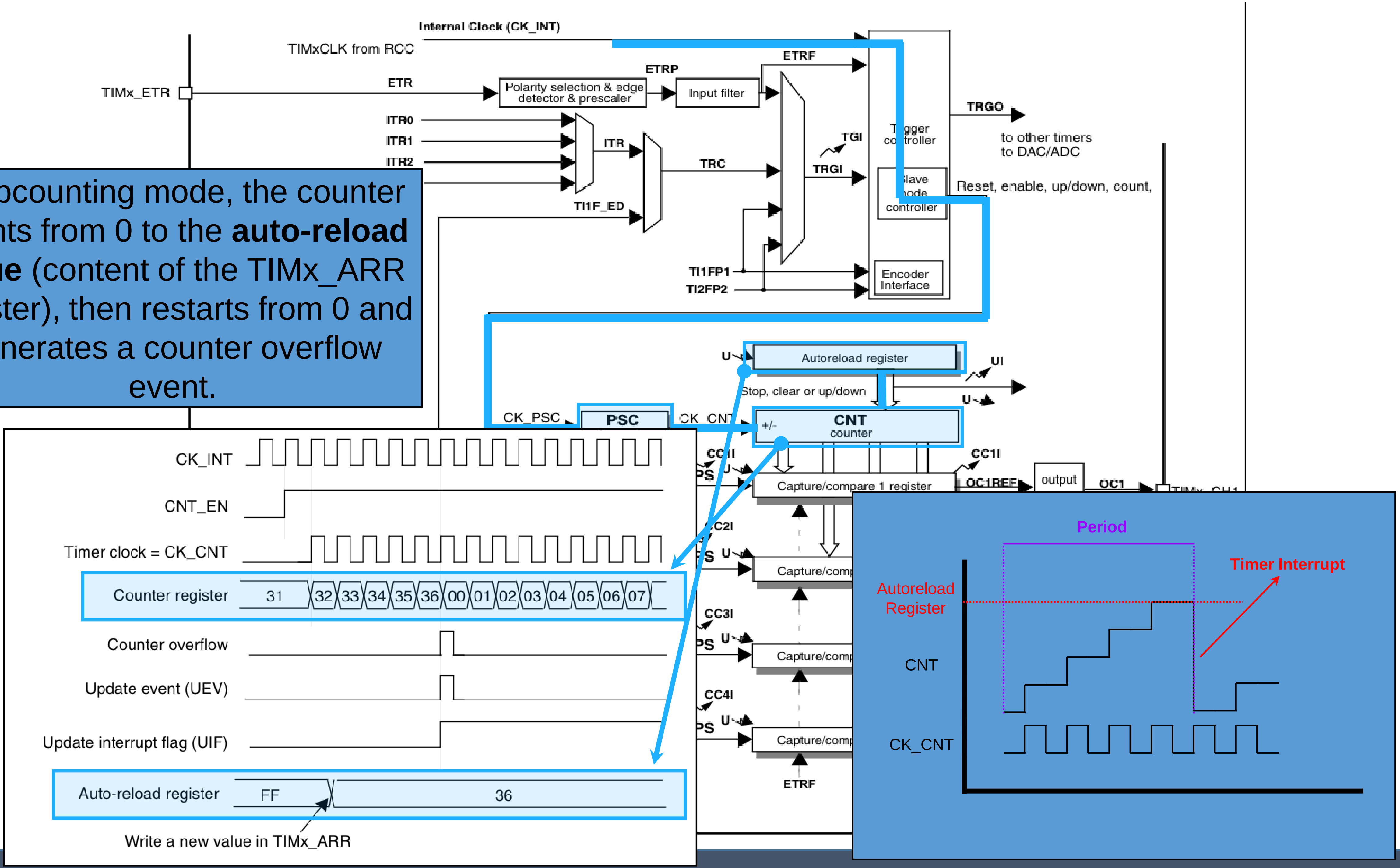






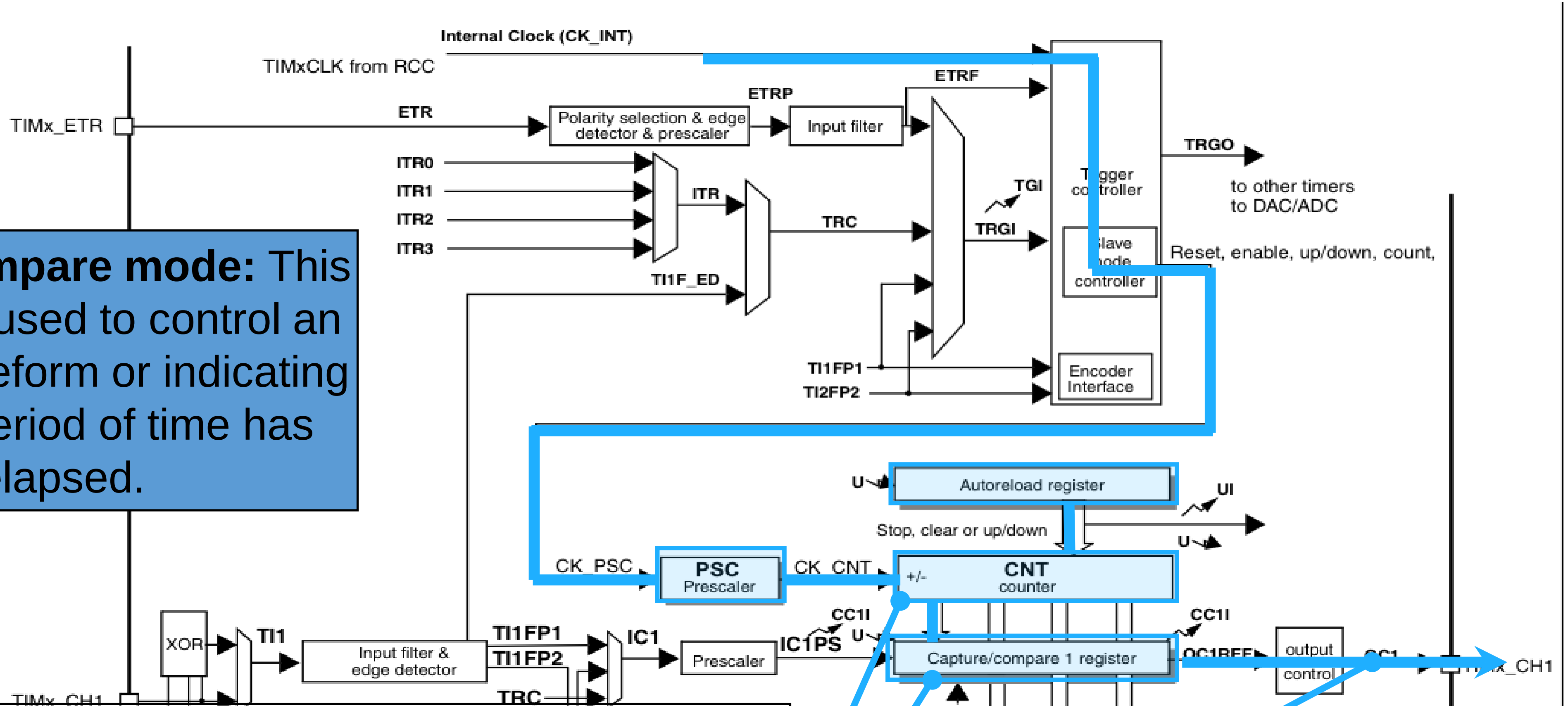
Timers

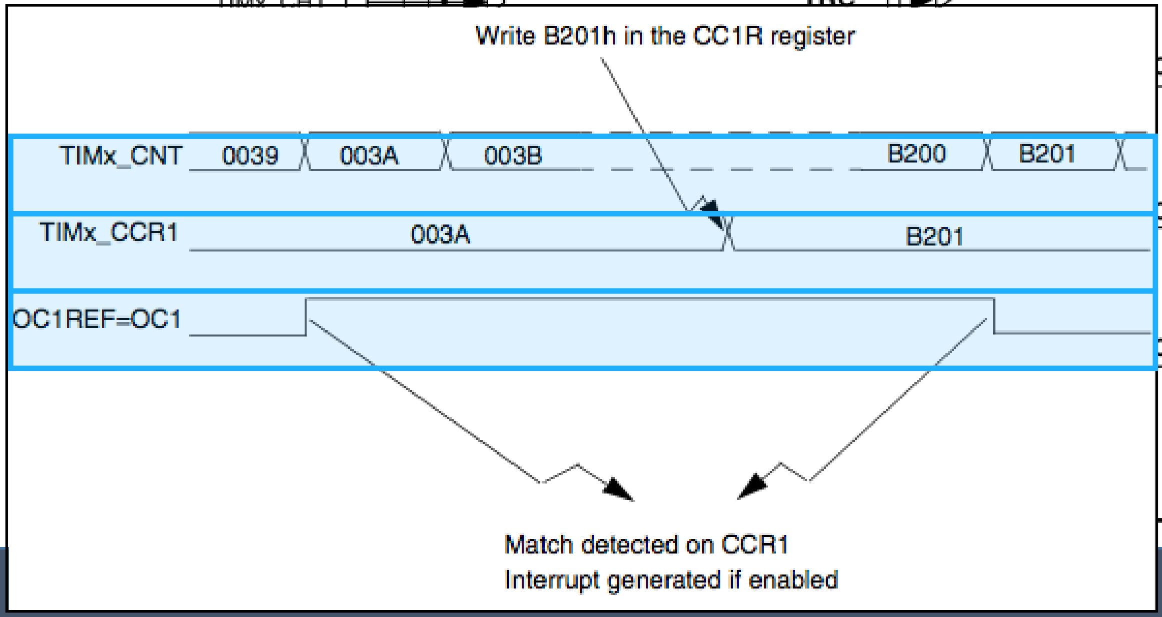
In upcounting mode, the counter counts from 0 to the **auto-reload value** (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

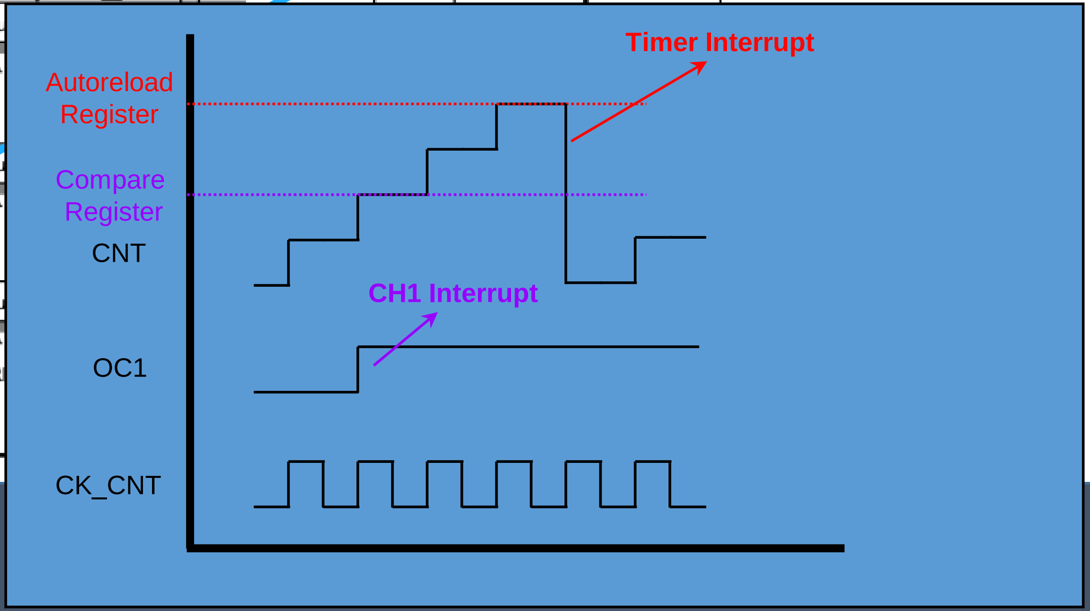


Timers

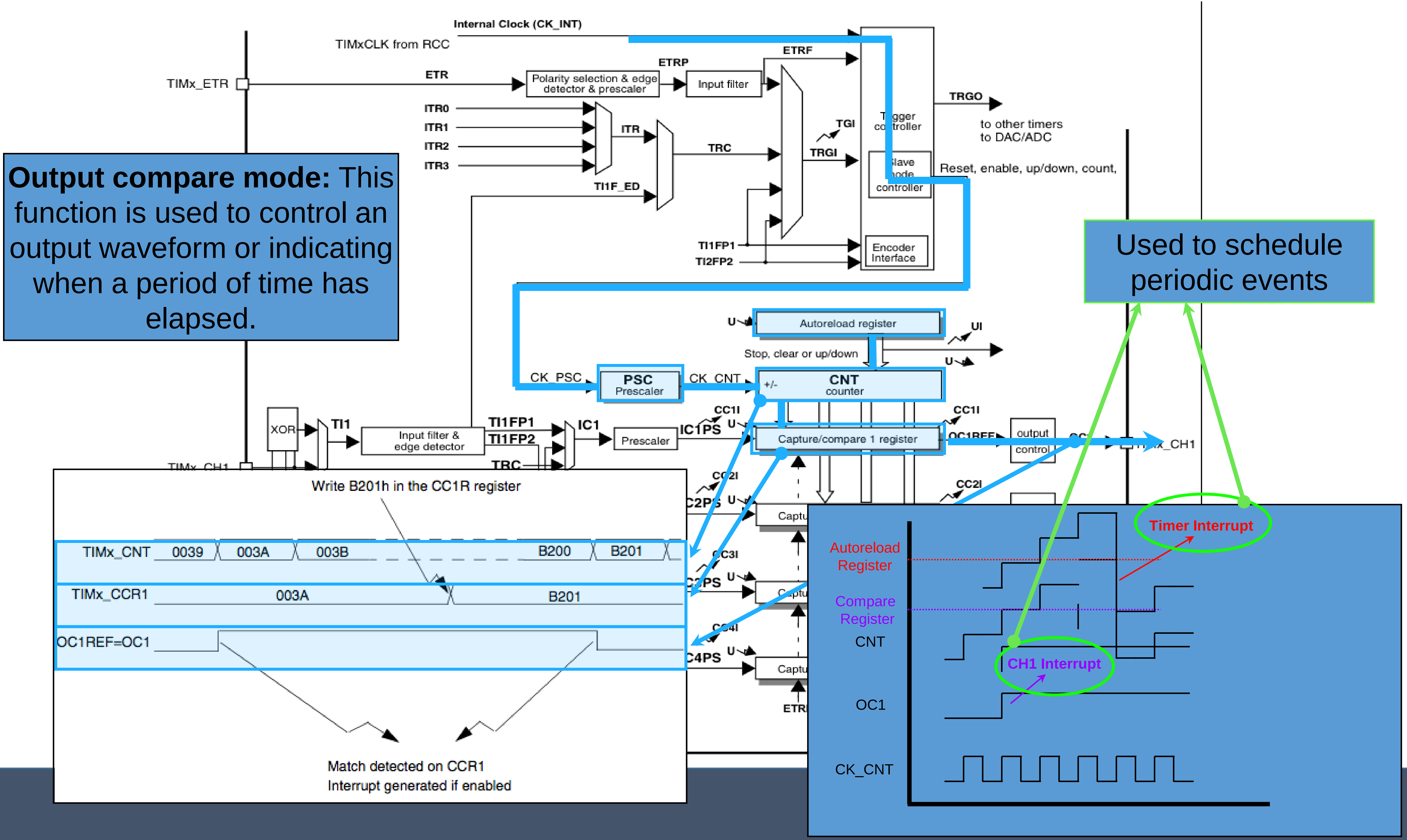
Output compare mode: This function is used to control an output waveform or indicating when a period of time has elapsed.







Timers



Output compare mode: This function is used to control an output waveform or indicating when a period of time has elapsed.

Used to schedule periodic events

Timers (What)

- I want a LED blinking at a given frequency using a timer.

  - **We need to setup the GPIO port and pin the LED is connected to**

    ➡We already know how to do that

  - **Since we are going to use interrupts generated by timers we need to setup NVIC**

    ➡The IRQChannel for TIMER3 is TIM3_IRQn

    ➡The ISR is void TIM2_IRQHandler(void)

  - **We need a generic timer because we want the LED blinking at a fixed frequency**

  - **We use the TIM3_CH1 (Timer 3 channel 1) in output compare mode**

Timers (How)

We need to setup the **GPIO** port and pin the LED is connected to:

```
void LEDs_Configuration(void)
{

GPIO_InitTypeDef  GPIO_InitStructure;

    /* Enable the GPIO_LED Clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

    /* Configure the GPIO_LED pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

    GPIO_Init(GPIOD, &GPIO_InitStructure);
}
```

Timers (How)

Since we are going to use interrupts generated by timers we need to setup **NVIC**

```
void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure; /* TIM3 clock enable */


    /* Enable the TIM3 gloabal Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

     NVIC_Init(&NVIC_InitStructure);

}
```

Timers (How)

We need a **generic timer** because we want the LED blinking at a fixed frequency:

```
uint16_t PrescalerValue = 0;
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
```

As usual a struct is used for the configuration of the peripheral

Timers (How)

We need a **generic timer** because we want the LED blinking at a fixed frequency:

```
uint16_t PrescalerValue = 0;
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

/* Enable the TIM3 gloabal Interrupt */

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
```

Clock enable for the TIMER3

Timers (How)

We need a **generic timer** because we want the LED blinking at a fixed frequency:

```
uint16_t PrescalerValue = 0;
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

/* Enable the TIM3 gloabal Interrupt */

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

TIM_TimeBaseStructure.TIM_Period = 65535;  // 2^16
```

upcounting mode

Timers (How)

We need a **generic timer** because we want the LED blinking at a fixed frequency:

```
uint16_t PrescalerValue = 0;
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

/* Enable the TIM3 gloabal Interrupt */

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

TIM_TimeBaseStructure.TIM_Period = 65535;  // 2^16
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
```

upcounting mode

Timers (How)

We need a **generic timer** because we want the LED blinking at a fixed frequency:

```
uint16_t PrescalerValue = 0;
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

/* Enable the TIM3 gloabal Interrupt */

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

TIM_TimeBaseStructure.TIM_Period = 65535;
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); /* Prescaler configuration */
```

As usual the init routine

Timers (How)

We need a **generic timer** because we want the LED blinking at a fixed frequency:

```
uint16_t PrescalerValue = 0;
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

/* Enable the TIM3 gloabal Interrupt */

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

TIM_TimeBaseStructure.TIM_Period = 65535;
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); /* Prescaler configuration */

TIM_PrescalerConfig(TIM3, PrescalerValue, TIM_PSCReloadMode_Immediate);
```

As usual the init routine

Timers (How)

Prescaler Value:

To set the prescaler we use the formula:

Prescaler = ((SystemCoreClock / 2) Fx) - 1

where Fx is the counter clock of the TIMER (CK_CNT) we want.

```
/* Compute the prescaler value */

PrescalerValue = (uint16_t) ((SystemCoreClock / 2) / 500000) - 1; // 500 KHz
```

This value has to be set before calling TIM_PrescalerConfig function

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;
```

**As usual a struct is used for the configuration
of the peripheral**

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
```

The comparison between the output compare register and the counter has no effect on the outputs.
(this mode is used to generate a timing base).
We are interested in interrupt not in output waveform.

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;

 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
```

OC1 signal is active high on the corresponding output pin

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;

 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInitStructure.TIM_Pulse = 0;
```

The compare register is set to 0.

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;

 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInitStructure.TIM_Pulse = 0;

TIM_OC1Init(TIM3, &TIM_OCInitStructure);
```

Init as usual

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;

 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInitStructure.TIM_Pulse = 0;

TIM_OC1Init(TIM3, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);
```

The compare register can be written at anytime,
the new value is taken in account immediately

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;

 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInitStructure.TIM_Pulse = 0;

TIM_OC1Init(TIM3, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);
TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE);
```

We are interested in the **interrupt** of the CHANNEL 1 of the TIMER3

Timers (How)

- We use the TIM3_CH1 (Timer 3 channel 1) in **output compare mode**

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;

 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInitStructure.TIM_Pulse = 0;

TIM_OC1Init(TIM3, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);
TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE);
TIM_Cmd(TIM3, ENABLE);
```

TIMER2 **enabled**

Timers (How)

- Handle the interrupt

```
void TIM3_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET)
    {
          TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);

          // Turn on the LED

    }
}
```

1. Check the flags to see what channel the interrupt is related to
2. Clear the flag
3. Turn on/off the LED

Timers (Code:Main)

```
#include "stm32f4xx.h "
#include "stm32f401_discovery.h "

void NVIC_Configuration(void)
{
        NVIC_InitTypeDef NVIC_InitStructure; /* TIM3 clock enable */

        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

        NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

        NVIC_Init(&NVIC_InitStructure);
}

void LEDs_Configuration(void)
{
      GPIO_InitTypeDef  GPIO_InitStructure;

      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
      GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
      GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

      GPIO_Init(GPIOD, &GPIO_InitStructure);
}
```

Timers (Code:Main)

```
int main(void)
{
        TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
        TIM_OCInitTypeDef  TIM_OCInitStructure;

        uint16_t PrescalerValue = 0;

        LEDs_Configuration();
        NVIC_Configuration();
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

        /* Compute the prescaler value */
        PrescalerValue = (uint16_t) ((SystemCoreClock /2)/ 500000) - 1;

        /* Time base configuration */
        TIM_TimeBaseStructure.TIM_Period = 65535;
        TIM_TimeBaseStructure.TIM_Prescaler = 0;
        TIM_TimeBaseStructure.TIM_ClockDivision = 0;
        TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

         TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
         TIM_PrescalerConfig(TIM3, PrescalerValue, TIM_PSCReloadMode_Immediate);

        /* Output Compare Timing Mode configuration: Channel1 */
        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
        TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_Pulse = 0;
        TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

         TIM_OC1Init(TIM2, &TIM_OCInitStructure);
        TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);
        TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE);

        /* TIM2 enable counter */
        TIM_Cmd(TIM3, ENABLE);

        while(1);
        return(0);
}
```

Timers (Code:IT)

stm32f4xx_it.c

```
...
void TIM3_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET)
    {
                    TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
                     GPIO_WriteBit(GPIOD, GPIO_Pin_12, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOD, GPIO_Pin_12)));
    }
}
...
```

Timers (Esercizi)

1. Fai lampeggiare un LED a 2Hz, ed un’altro a 3Hz

➡Tip: usa TIM_GetCapture1() e TIM_SetCompare1()

➡Tip: devi utilizzare due differenti canali

2. Utilizza un bottone per modificare la frequenza di lampeggio di un LED (usando ovviamente i timers).

3. Genera un onda quadra a 500Hz in output da un TIM2_CH1 pin (controlla con l’oscilloscopio) (opzionale)

➡Tips: Dovreste usare il Toggle del Output Compare mode

➡STM32F4xx Standard Peripherals Library,

Timers (Domande)

1. Quali sono gli altri possibili modi per configurare il TIM_OCMode, nel campo di TIM_OCInitTypeDef **structure** (guardare anche I manuali messi a disposizione)

2. Come funziona la modalità TIM_CounterMode_CenterAligned in comparazione alla modalità TIM_CounterMode_Up) **?**

4. Cosa accade se omettiamo TIM_OC1PreloadConfig(...) **?**

5. Che cos'è ClockDivision?