Laboratorio di Architetture e Programmazione dei Sistemi Elettronici Industriali

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

Simone Benatti<simone.benatti@unibo.it>

Filippo Casamassima<filippo.casamassima@unibo.it>

#6 Timers

Timers

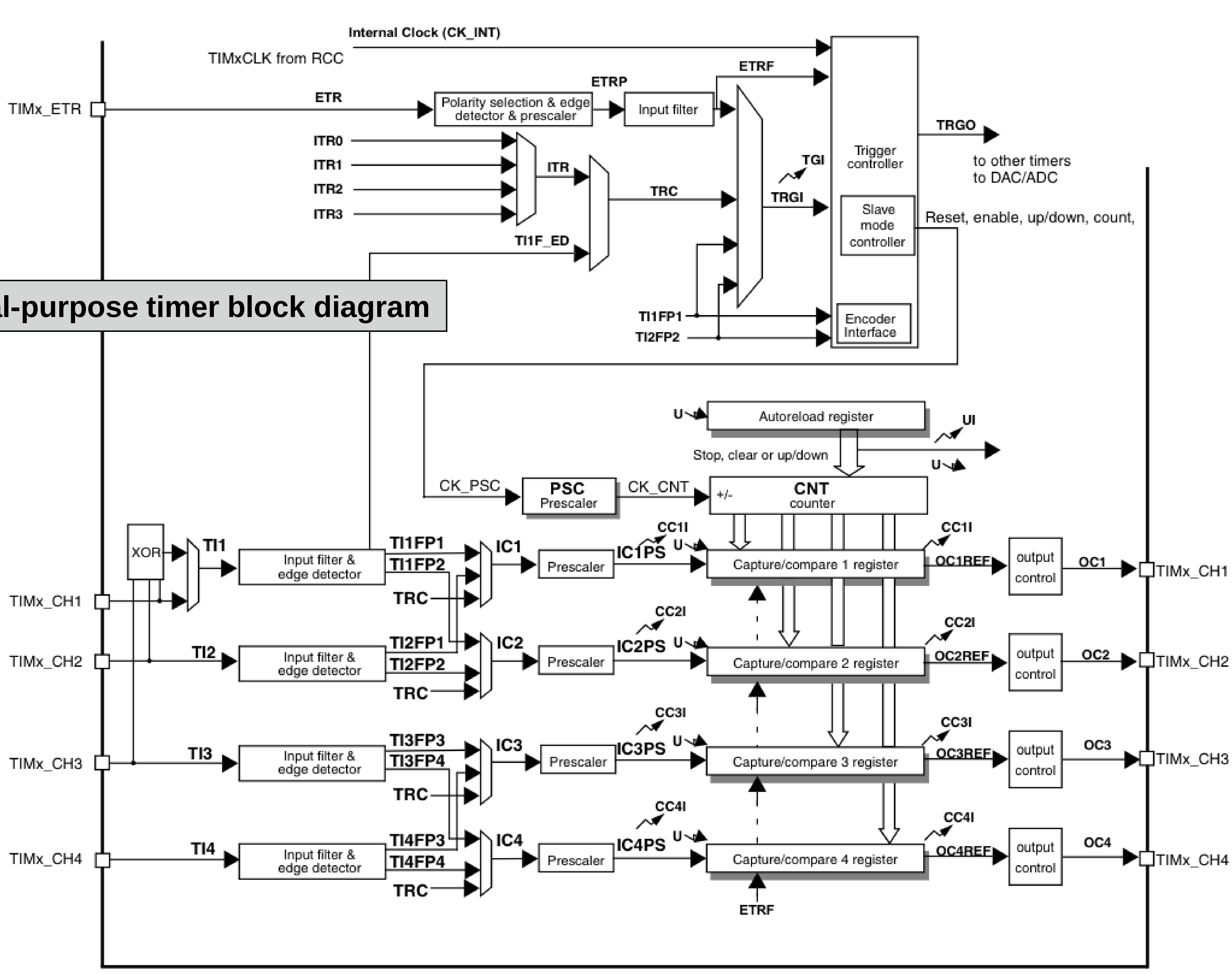
- The general-purpose timers consist of a **16-bit auto-reload counter** driven by a programmable prescaler.

- 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.

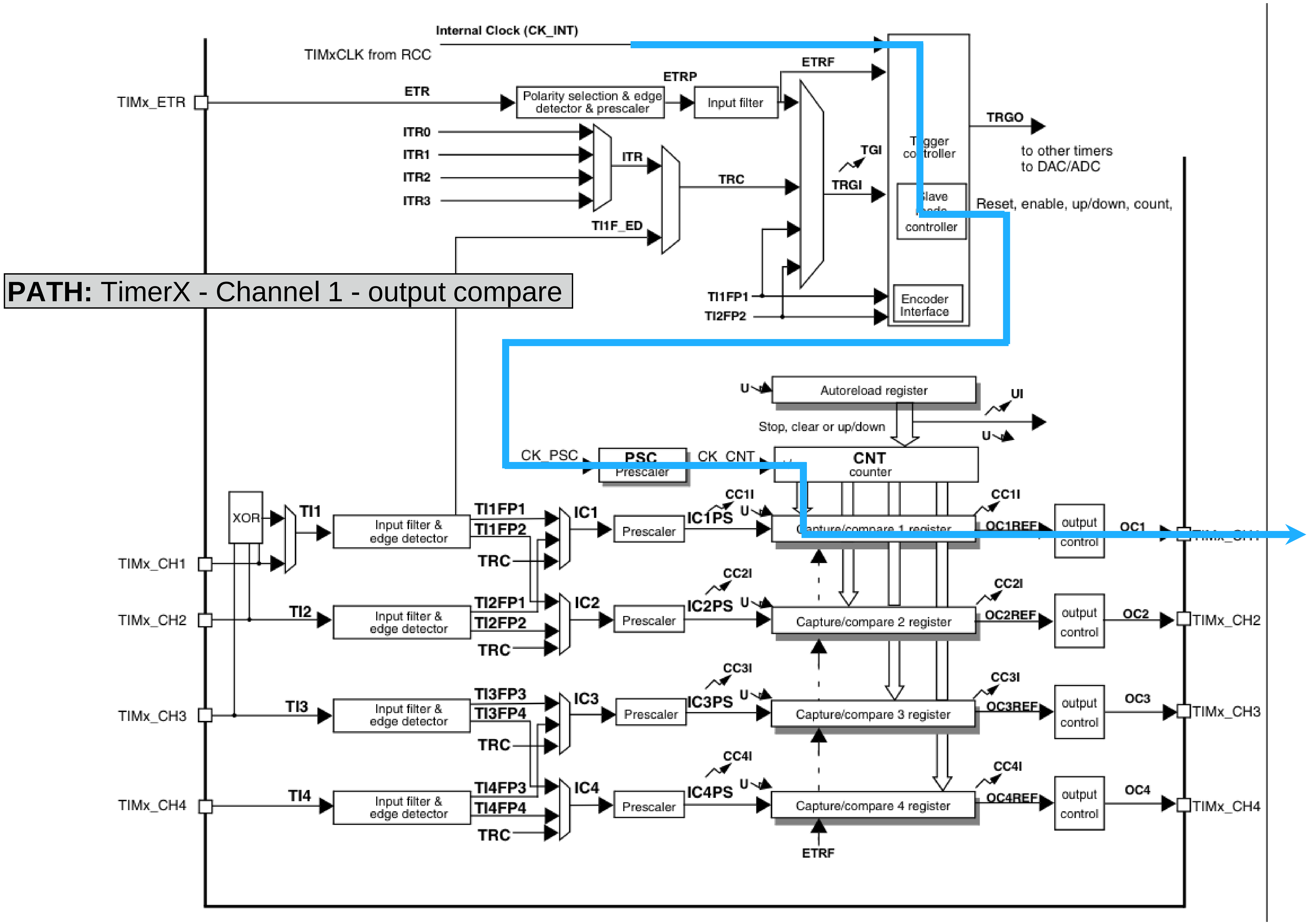
- General-purpose TIMx timer features include:
  - 16-bit up, down, up/down auto-reload counter.
  - 16-bit programmable prescaler used to divide (also "on the fly") the counter clock frequency by any factor between 1 and 65535.
  - **Up to 4 independent channels** for:
    - Input capture
    - Output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

Timers



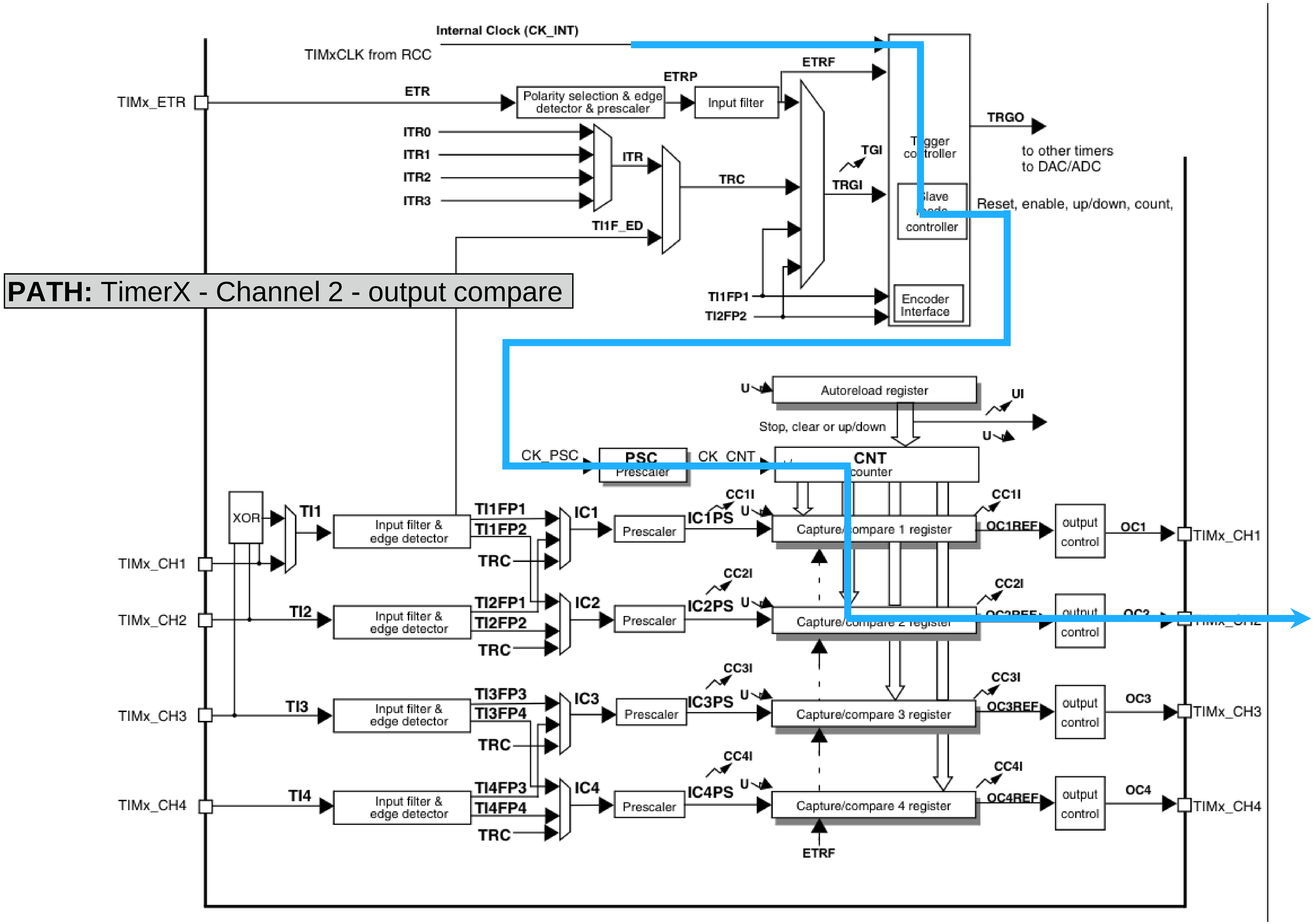
General-purpose timer block diagram

Timers



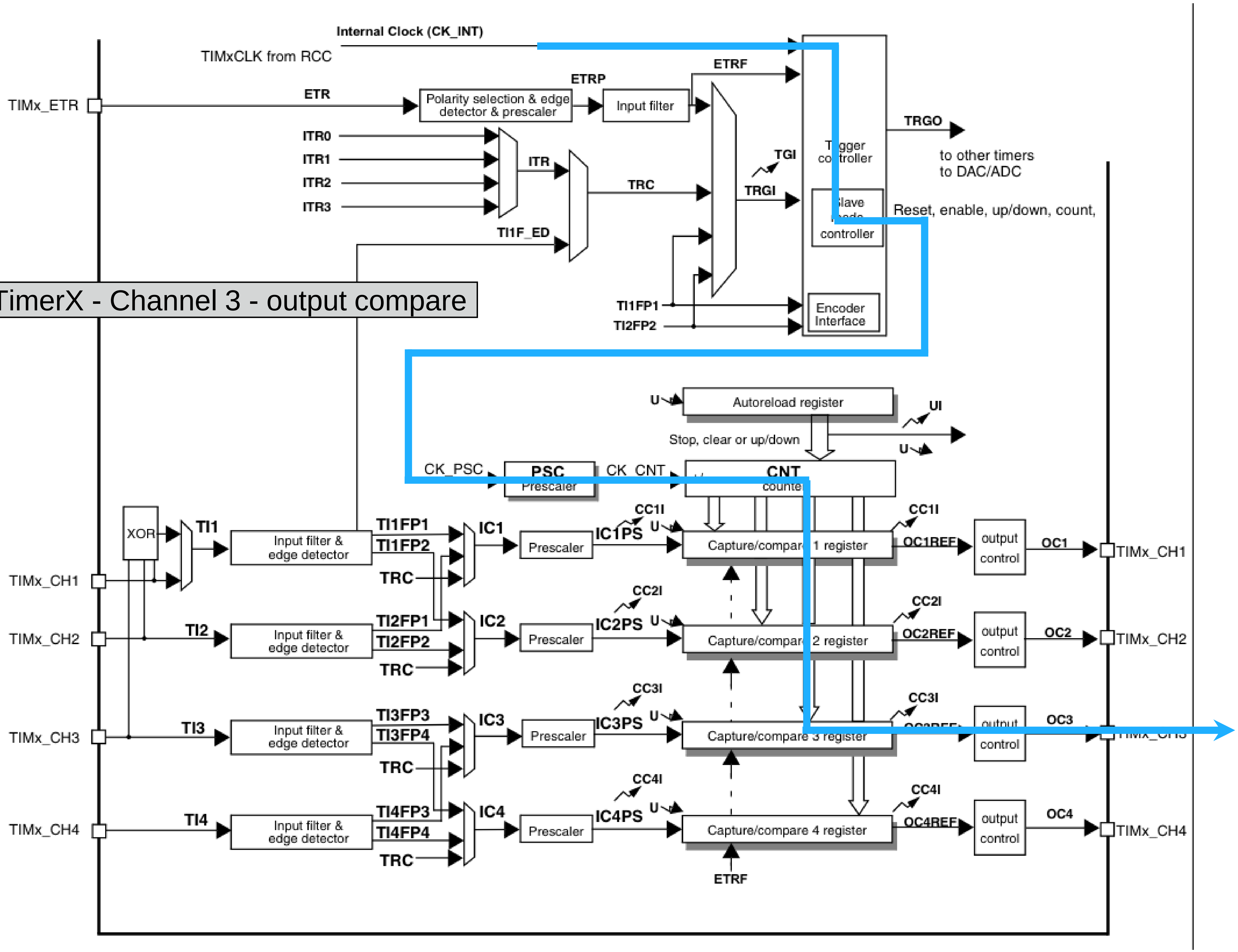
Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
Polarity selection & edge detector & prescaler
ETRP
Input filter
ETRF
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TI1F_ED
TGI
TRGI
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
controller
Encoder Interface
TI1FP1
TI2FP2
PATH: TimerX - Channel 1 - output compare
Autoreload register
UI
Stop, clear or up/down
CK_PSC
PSC
Prescaler
CK_CNT
CNT
counter
XOR
TI1
Input filter & edge detector
TI1FP1
TI1FP2
TRC
IC1
Prescaler
IC1PS
CC1I
OC1REF
output control
OC1
TIMx_CH1
TIMx_CH2
TI2
TI2FP1
TI2FP2
IC2
IC2PS
CC2I
Capture/compare 2 register
OC2REF
OC2
TIMx_CH3
TI3
TI3FP3
TI3FP4
IC3
IC3PS
CC3I
Capture/compare 3 register
OC3REF
OC3
TIMx_CH4
TI4
TI4FP3
TI4FP4
IC4
IC4PS
CC4I
Capture/compare 4 register
OC4REF
OC4
ETRF

Timers

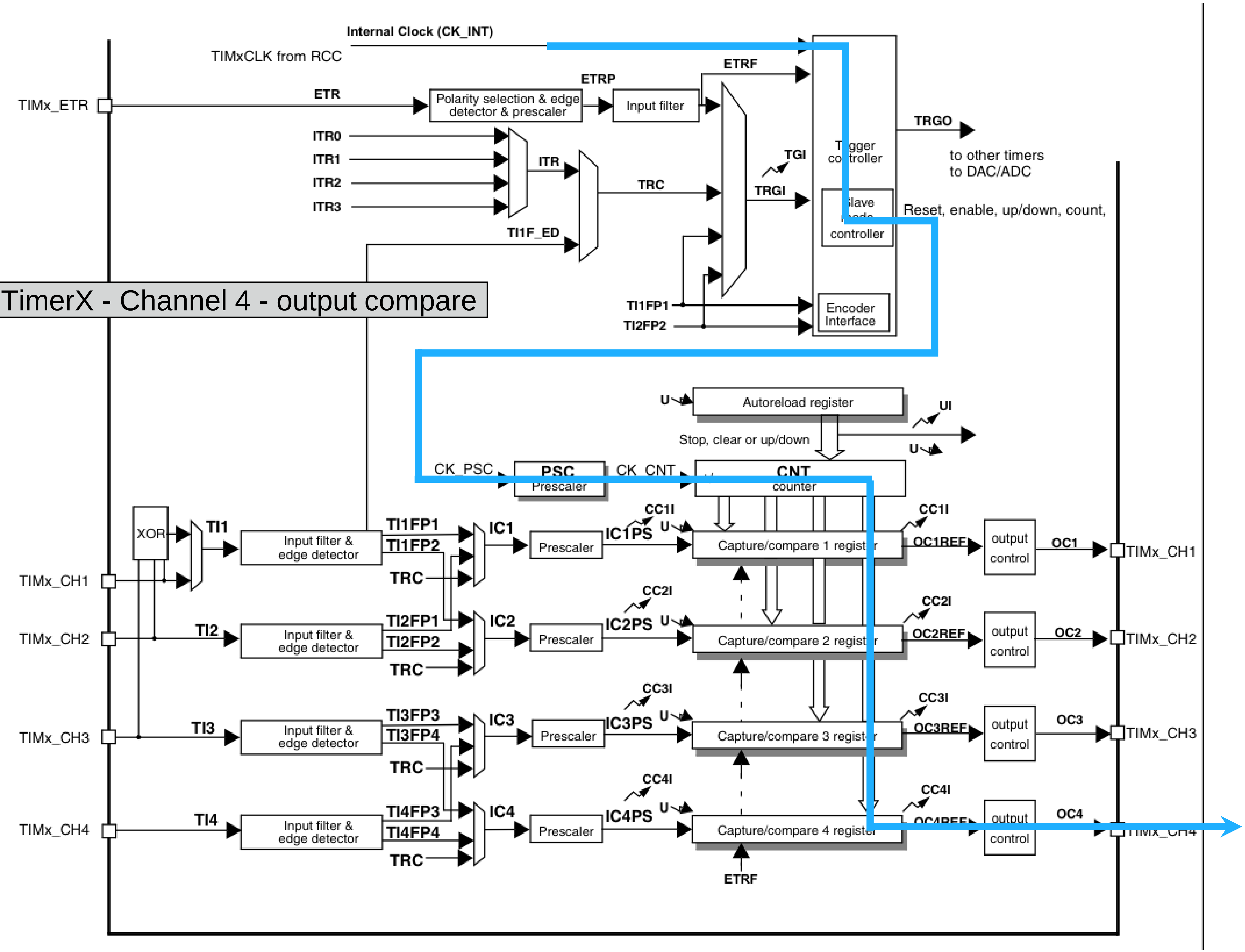

PATH: TimerX - Channel 2 - output compare
Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
Polarity selection & edge detector & prescaler
ETRP
Input filter
ETRF
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TI1F_ED
TGI
TRGI
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
controller
Encoder Interface
TI1FP1
TI2FP2
Autoreload register
UI
Stop, clear or up/down
CK_PSC
PSC
CK_CNT
CNT
counter
XOR
TI1
TI2
TI3
TI4
TIMx_CH1
TIMx_CH2
TIMx_CH3
TIMx_CH4
Input filter & edge detector
TI1FP1
TI1FP2
TI2FP1
TI2FP2
TI3FP3
TI3FP4
TI4FP3
TI4FP4
TRC
IC1
IC2
IC3
IC4
Prescaler
IC1PS
IC2PS
IC3PS
IC4PS
CC1I
CC2I
CC3I
CC4I
Capture/compare 3 register
Capture/compare 4 register
OC1REF
OC3REF
OC4REF
output control
OC1
OC3
OC4
ETRF

Timers

PATH: TimerX - Channel 3 - output compare

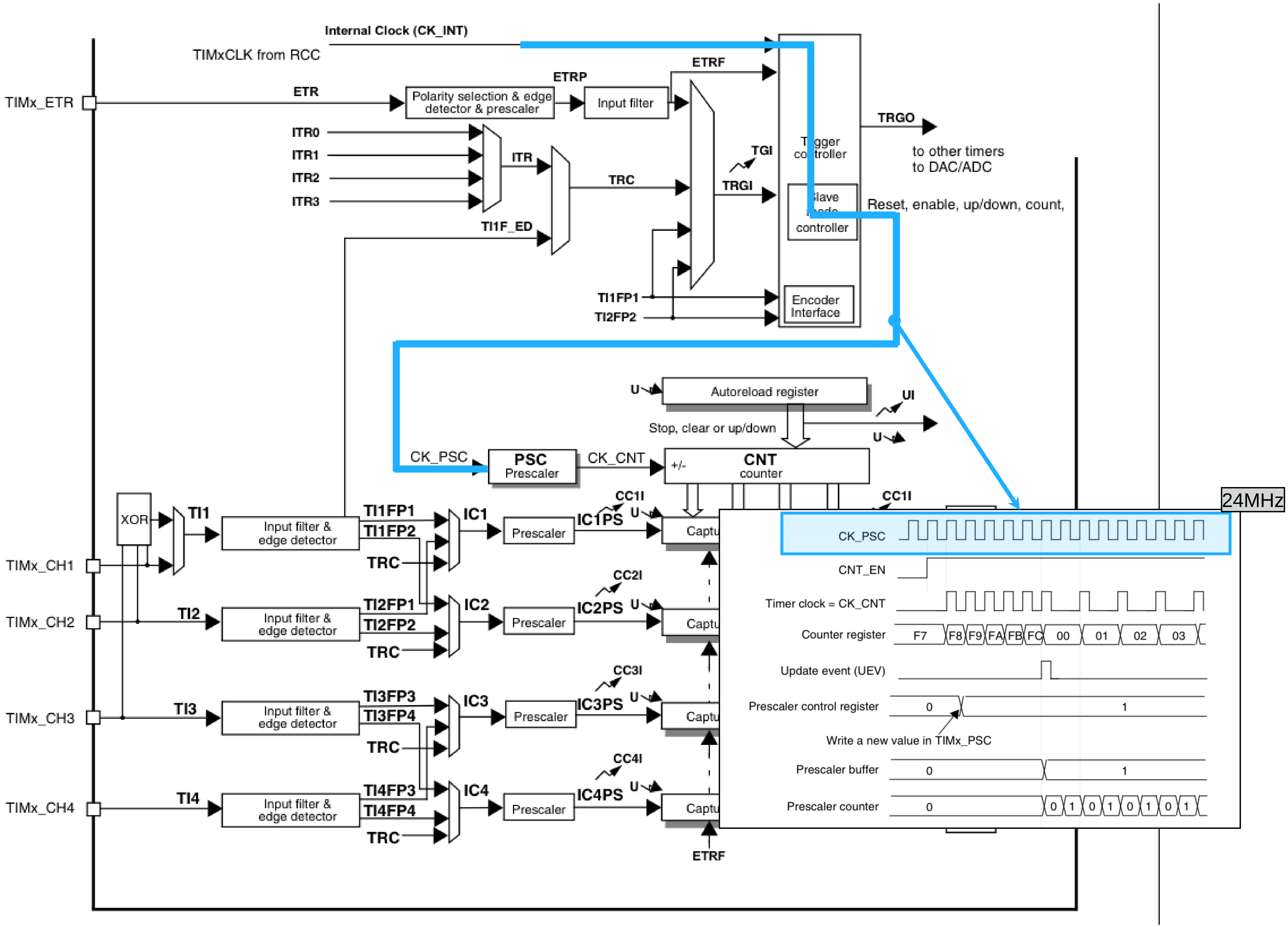


Timers



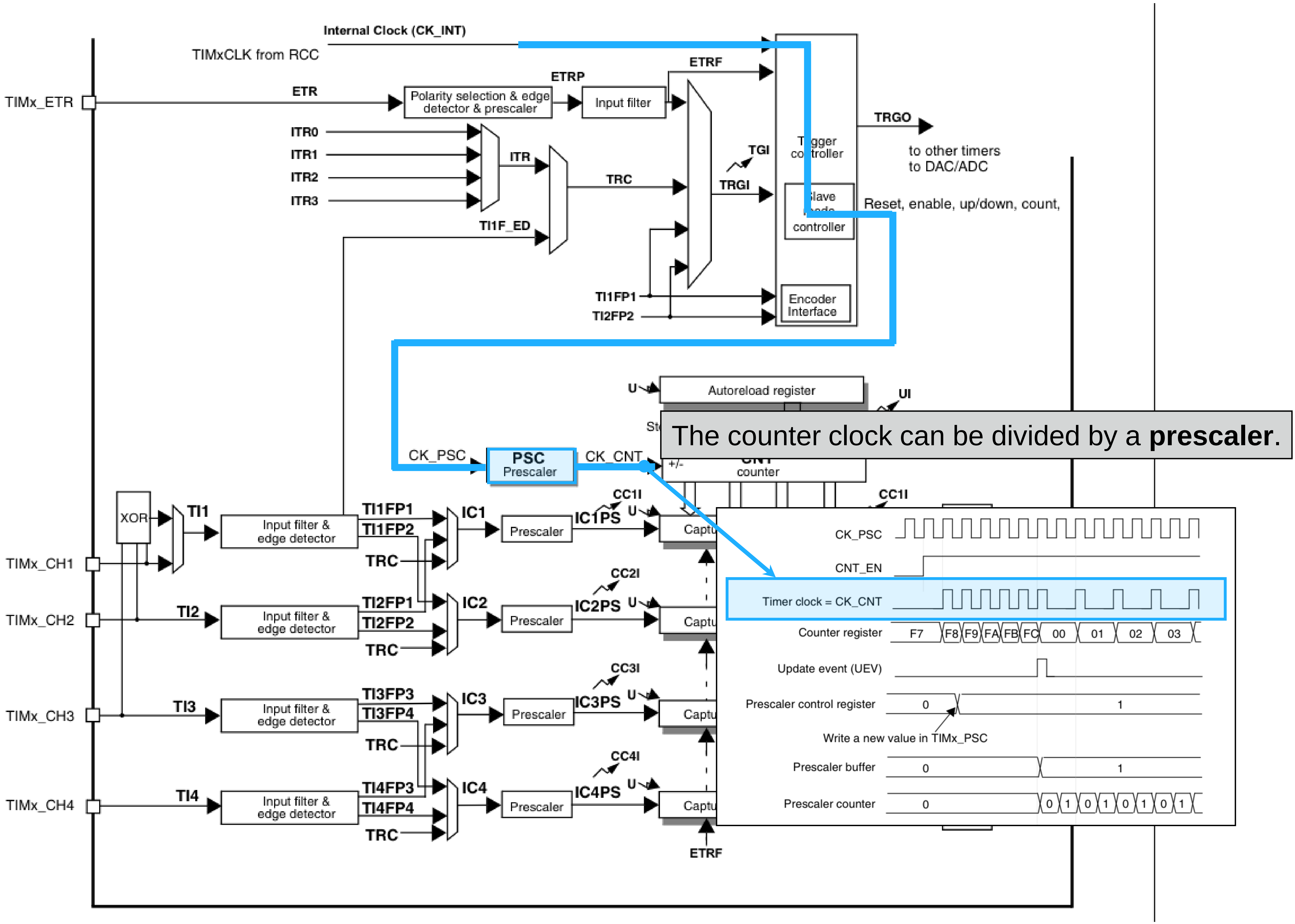
PATH: TimerX - Channel 4 - output compare

Timers



Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
Polarity selection & edge detector & prescaler
ETRP
Input filter
ETRF
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TI1F_ED
TGI
TRGI
TI1FP1
TI2FP2
Trigger controller
Slave mode controller
Encoder Interface
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
Autoreload register
UI
Stop, clear or up/down
CK_PSC
PSC Prescaler
CK_CNT
+/-
CNT counter
XOR
TI1
TI2
TI3
TI4
TIMx_CH1
TIMx_CH2
TIMx_CH3
TIMx_CH4
Input filter & edge detector
TI1FP1
TI1FP2
TI2FP1
TI2FP2
TI3FP3
TI3FP4
TI4FP3
TI4FP4
IC1
IC2
IC3
IC4
Prescaler
IC1PS
IC2PS
IC3PS
IC4PS
CC1I
CC2I
CC3I
CC4I
ETRF
24MHz
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
Write a new value in TIMx_PSC
Prescaler buffer
Prescaler counter

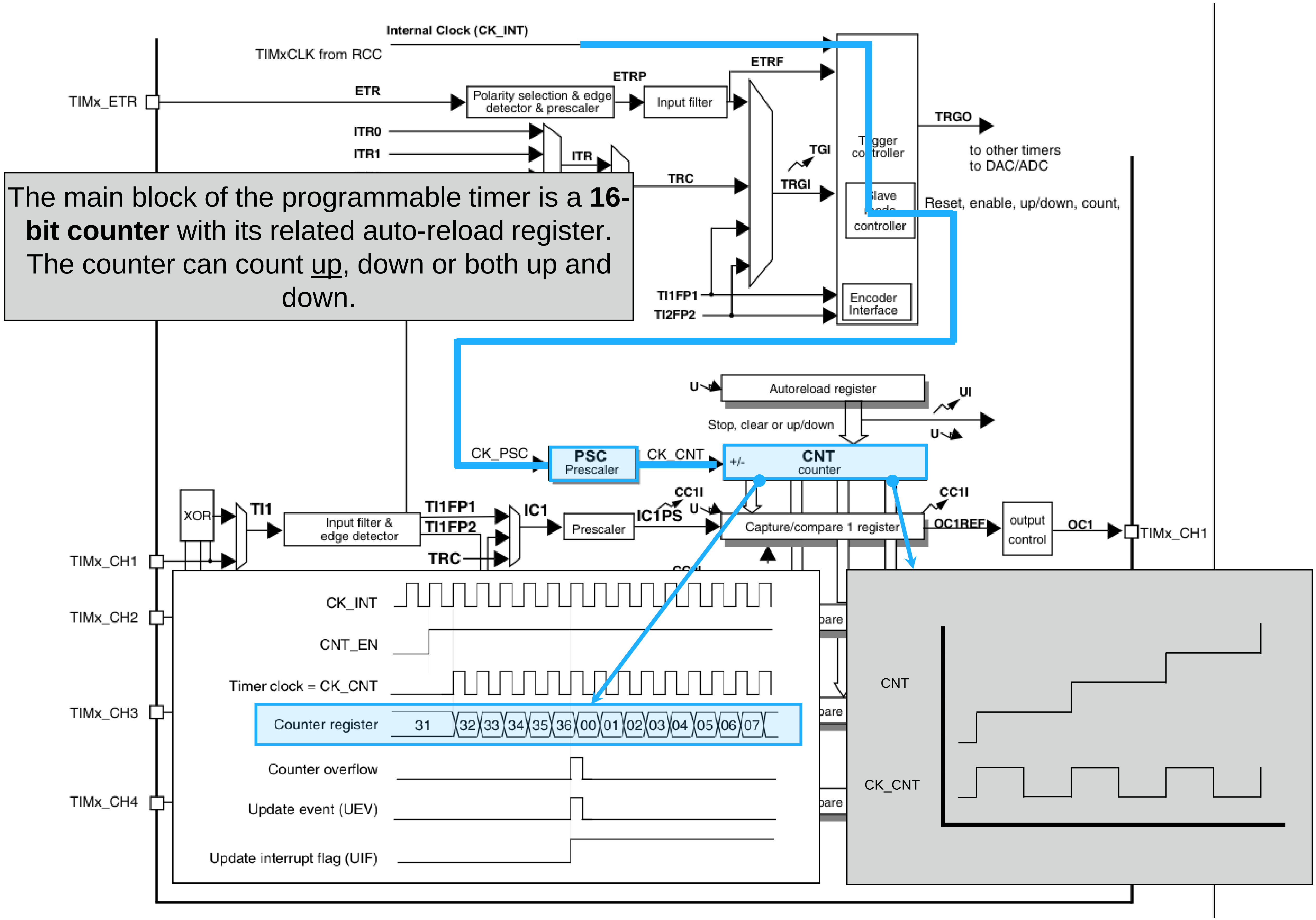
Timers



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

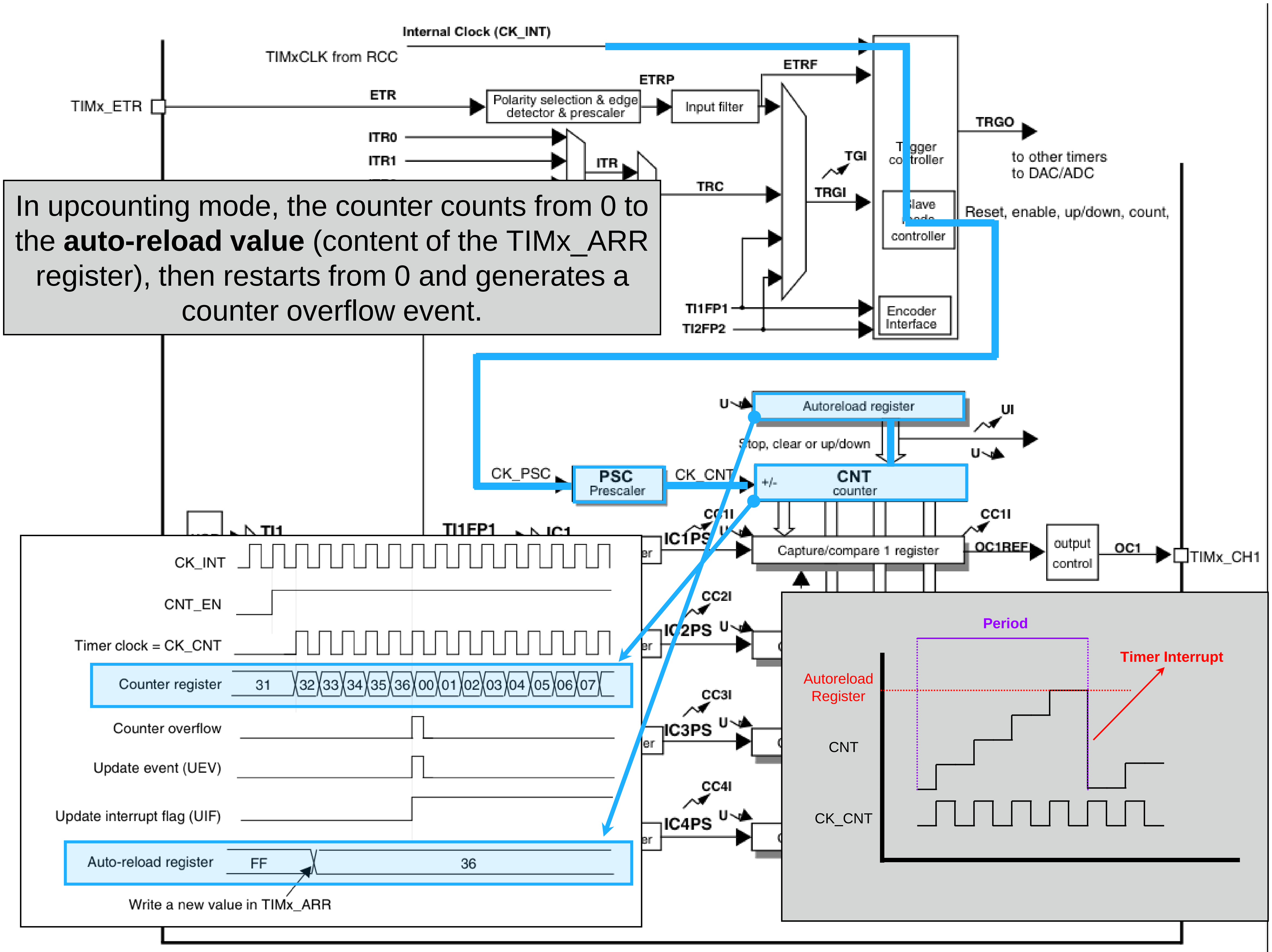
Timers

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.



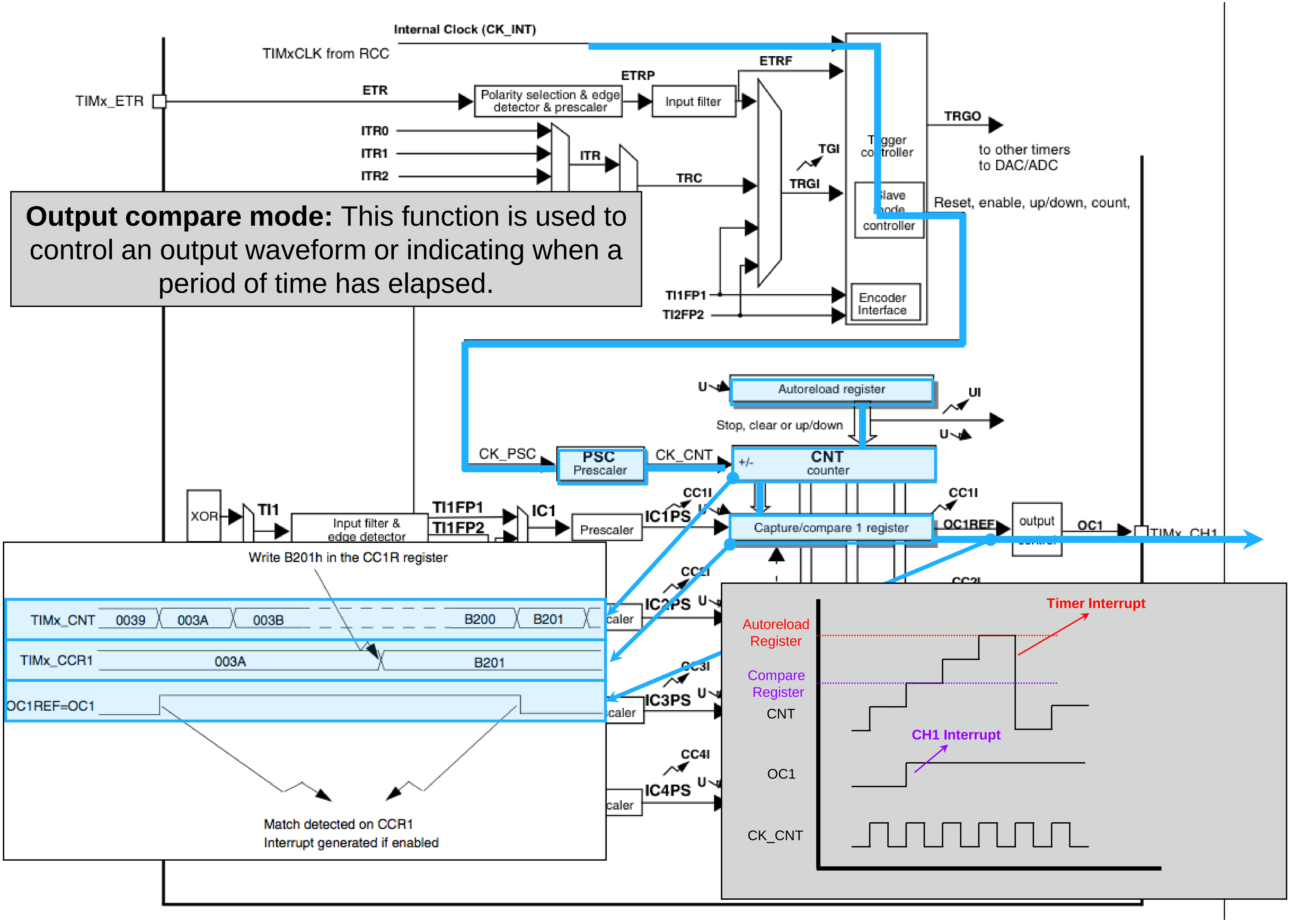
Timers

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.

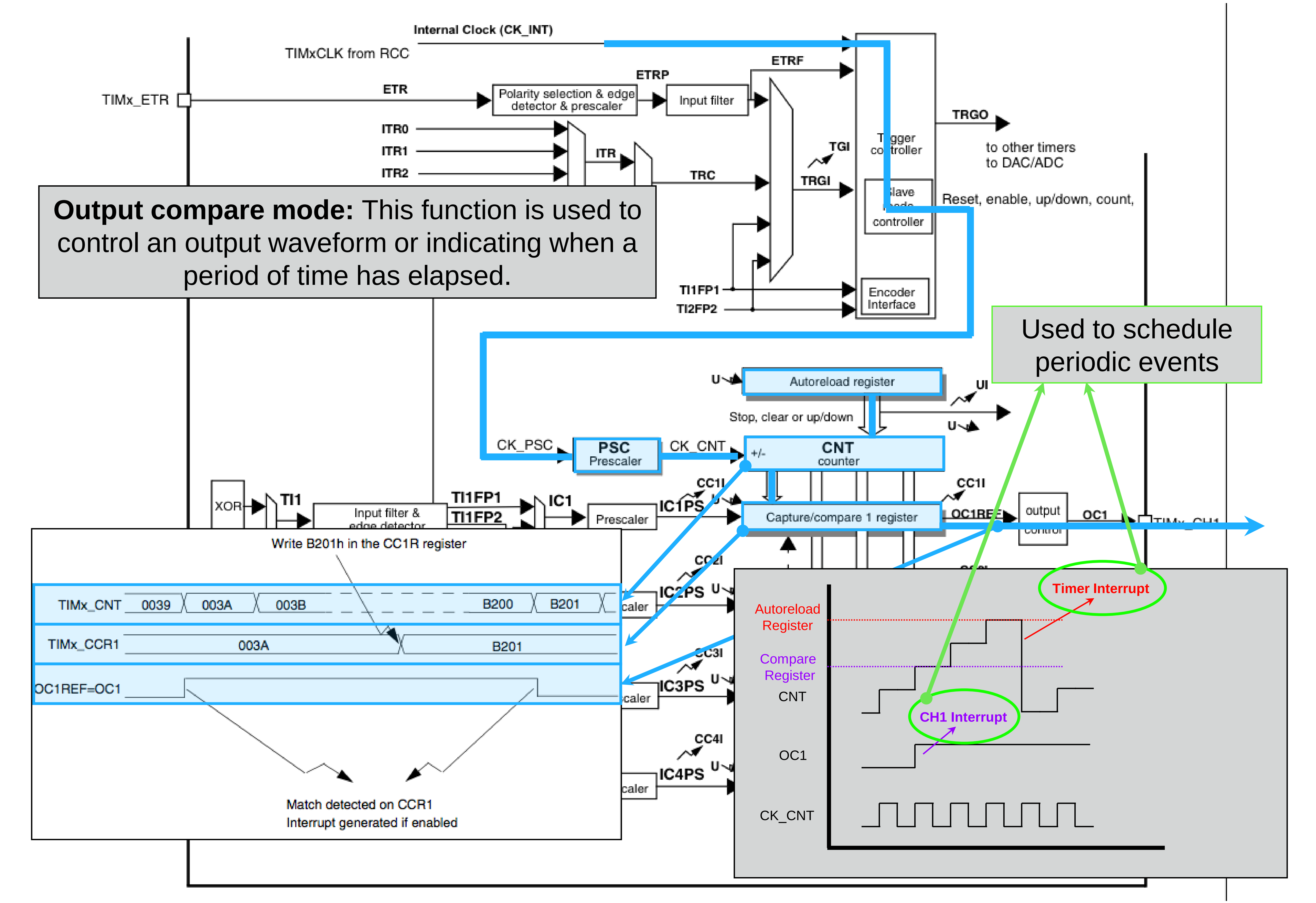


Timers

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



Timers



Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
Polarity selection & edge detector & prescaler
ETRP
Input filter
ETRF
ITR0
ITR1
ITR2
ITR
TRC
TGI
TRGI
Trigger controller
Slave mode controller
Encoder Interface
TI1FP1
TI2FP2
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
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
Autoreload register
UI
Stop, clear or up/down
CK_PSC
PSC
Prescaler
CK_CNT
CNT
counter
XOR
TI1
Input filter & edge detector
TI1FP1
TI1FP2
IC1
Prescaler
IC1PS
CC1I
Capture/compare 1 register
OC1REF
output control
OC1
Write B201h in the CC1R register
TIMx_CNT 0039 003A 003B B200 B201
TIMx_CCR1 003A B201
OC1REF=OC1
Match detected on CCR1
Interrupt generated if enabled
CC2I
IC2PS
CC3I
IC3PS
CC4I
IC4PS
Timer Interrupt
Autoreload Register
Compare Register
CNT
CH1 Interrupt
OC1
CK_CNT

Timers (**what**)

- I want a LED blinking at 1Hz 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 TIMER2 is TIM2_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 TIM2_CH1 (Timer 2 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_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

    /* Configure the GPIO_LED pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &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;

    /* Enable the TIM2 gloabal Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_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

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

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

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
uint16_t PrescalerValue = 0;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
```

Clock enable for the TIMER2

Timers (**how**)

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

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
uint16_t PrescalerValue = 0;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
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

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
uint16_t PrescalerValue = 0;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) (SystemCoreClock / 1000) - 1;
```

To set the prescaler we use the formula:

Prescaler = (SystemCoreClock / Fx) - 1

where Fx is the counter clock of the TIMER (CK_CNT) we want.
In this case we are setting the TIM2 counter clock to 1KHz

Timers (**how**)

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

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
uint16_t PrescalerValue = 0;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) (SystemCoreClock / 1000) - 1;
TIM_TimeBaseStructure.TIM_Period = 999;
```

Timers (**how**)

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

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
uint16_t PrescalerValue = 0;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
 TIM_TimeBaseStructure.TIM_ClockDivision = 0;
 TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) (SystemCoreClock / 1000) - 1;
TIM_TimeBaseStructure.TIM_Period = 999;

TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
```

As usual the init routine

Timers (**how**)

- We use the TIM2_CH1 (Timer 2 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 TIM2_CH1 (Timer 2 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 TIM2_CH1 (Timer 2 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 TIM2_CH1 (Timer 2 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.
The blinking frequency is then 1 Hz

Timers (**how**)

- We use the TIM2_CH1 (Timer 2 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(TIM2, &TIM_OCInitStructure);
```

Init as usual

Timers (**how**)

- We use the TIM2_CH1 (Timer 2 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(TIM2, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);
```

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

Timers (**how**)

- We use the TIM2_CH1 (Timer 2 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(TIM2, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);

TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE);
```

We are interested in the **interrupt** of the CHANNEL 1 of the TIMER2

Timers (**how**)

- We use the TIM2_CH1 (Timer 2 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(TIM2, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);

TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE);
TIM_Cmd(TIM2, ENABLE);
```

TIMER2 **enabled**

Timers (**how**)

- Handle the interrupt

```
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
    {
            TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
            GPIO_WriteBit(GPIOC, GPIO_Pin_9, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_9)));
    }
}
```

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.c

```
#include "stm32F10x.h“

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    /* Enable the TIM2 gloabal Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_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;

    /* Enable the GPIO_LED Clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

    /* Configure the GPIO_LED pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

Timers (**code**)

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

    uint16_t PrescalerValue = 0;

    LEDs_Configuration();
    NVIC_Configuration();
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    /* Compute the prescaler value */
    PrescalerValue = (uint16_t) (SystemCoreClock / 1000) - 1;

    /* Time base configuration */
    TIM_TimeBaseStructure.TIM_Period = 999;
    TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

    /* Output Compare Timing Mode configuration: Channel1 */
    /* Frozen - The comparison between the output compare register TIMx_CCR1 and the
    counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing
    base). */
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;

    /* OC1 signal is output on the corrisponding output pin */
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 0;

    /* OC1 active high */
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
            TIM_OC1Init(TIM2, &TIM_OCInitStructure);
    /* TIMx_CCR1 can be written at anytime, the new value is taken in account immediately */
    TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);
    TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE);

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

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

Timers (**code**)

stm32f10x_it.c

```
...
void TIM2_IRQHandler(void)
{
	if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
	{
		TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
		GPIO_WriteBit(GPIOC, GPIO_Pin_9, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_9)));
	}
}
...
```

Timers (**exercises**)

1. **Modify the blinking frequency to 2Hz, 5Hz, 0.5Hz, 0.1Hz**

2. **Make a LED blinking at 2Hz, the other one at 3Hz**

   ➞ Tip: use TIM_GetCapture1() and TIM_SetCompare1()

   ➞ Tip: you need to use two different channels

USE TIMERS!!
(do not change frequency
incrementing variables)

- **Use the button to modify the blinking frequency of the LEDs** (using timers)

- **Exercise (+) only one is mandatory (your choice)** **Exercise (++) is optional**
- **(+) Generate a 500Hz square wave in output from TIM2_CH1 pin** (check using the oscilloscope)

  ➞ Tips: You should use Toggle of the Output Compare mode

  ➞ More information: RM0041, STM32F10x Standard Peripherals Library, AN2581

- **(+) Generate a 50Hz square wave in output from TIM2_CH1 pin and a 1KHz square wave in output from TIM2_CH2.**

- **(++) Generate a PWM signal @ 25KHz duty cycle 10%**

Timers (**questions**)

1. **Explain the other possible values for the** TIM_OCMode **field of** TIM_OCInitTypeDef **structure** (see TIMx_CCMR1 register in RM0041)

2. **What** TIM_CounterMode_CenterAligned **is?** (in comparison to TIM_CounterMode_Up) **How does it work?**

3. **What happens in the code if we use a period = 65535?**

4. **What happens in the exercise n°2 if we omit** TIM_OC1PreloadConfig(...) **?**

5. **What ClockDivision is?**

- **(++) In the timer block diagram we said that CK_INT=24MHz. This is not always true. Why?** (check the clock tree). **What the frequency range for CK_INT is in the Discovery board? Write a program to check that CK_INT=24MHz.**