STM32F429
Introduction to Timers

Outline

1. Basic concept about timers in microcontrollers
2. Timers in the STM32F4xx devices
3. Types of timers
    a. Basic
    b. General purpose
    c. Advanced
4. Basic functionality of general-purpose timers
    a. Time Base Generator
    b. Output Compare Mode
    c. Input capture mode
    d. Pulse-Width Generation
5. Working with timers. HAL TIM Generic Driver
6. Configuring the Keil project
7. Examples

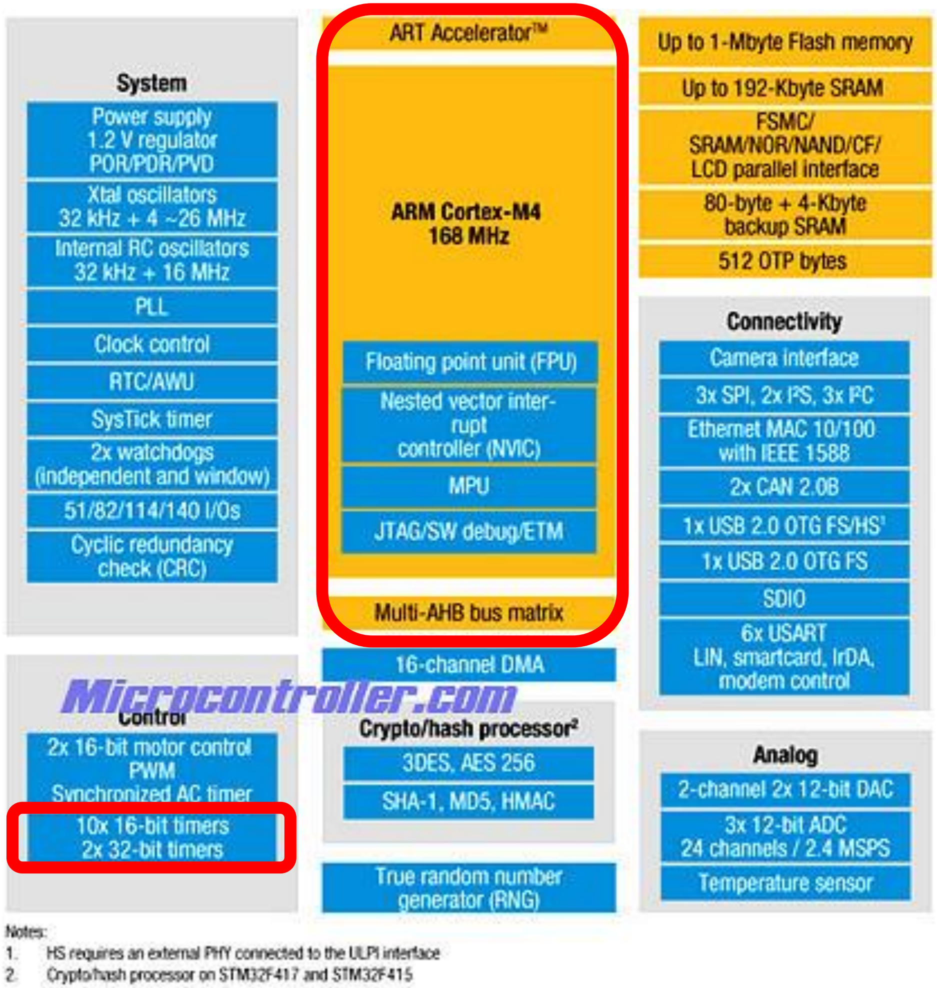
Timers. What are they for?

Timers in microcontrollers

- Main goals:
  - Scheduling events (periodical interrupts, delays, timeouts, etc.)
  - Measurement of external signals (period of a square signal, width of a pulse, external events, etc.)
  - Digital output generation (pulses, square signals, PWM signals –motor control, intensity lights, buzzers…-, etc.)

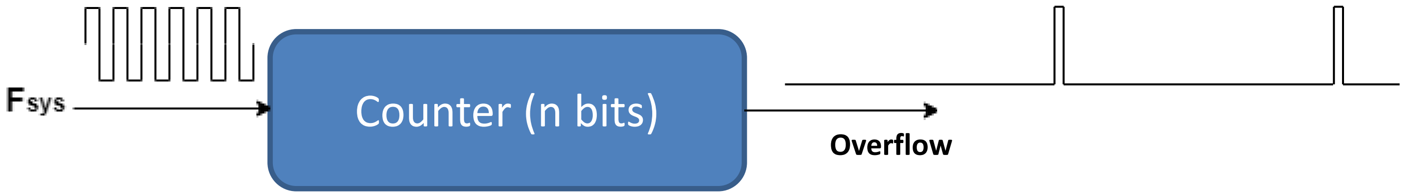
Timers in microcontrollers

- In a microcontroller, the Timers are implemented in a hardware independent from the CPU
  - ➢Timers provide **accurate timing** based on **dedicated hardware**.
  - ➢Timing is not provided by software (the software only controls the operating parameters).



Timers in microcontrollers

- Basic timer structure:



A Timer Module in its most basic form is a digital logic circuit that counts every clock cycle. This allows generating fixed timing events.

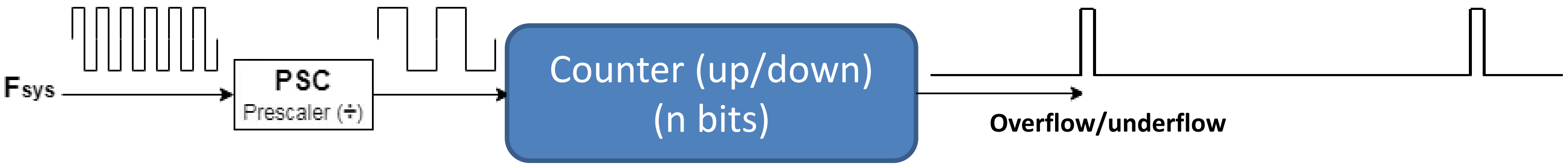
Example:

- n = 8 bits
- Fsys = 1000 Hz

→ Overflow every 256 (2^8) ms

Timers in microcontrollers

- Basic timer structure (with prescaler):



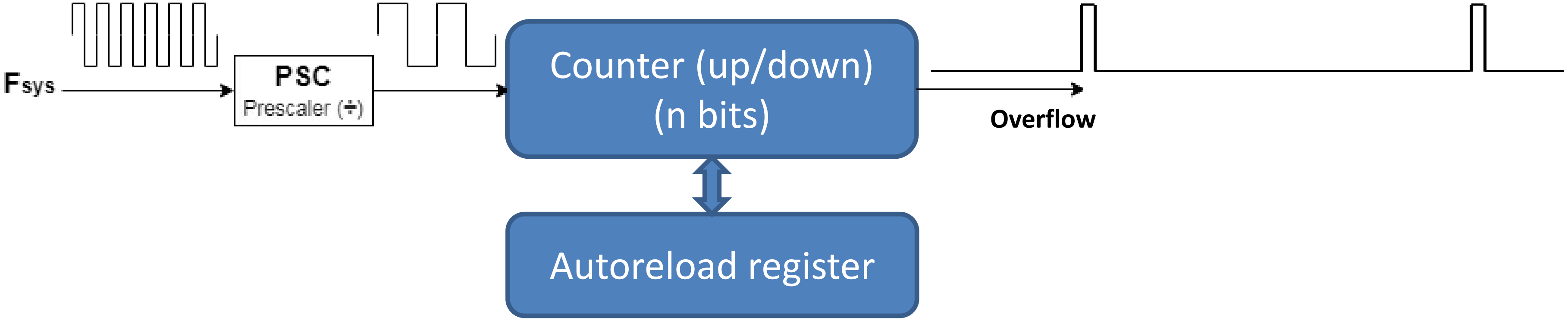
Usually, more functionalities are implemented in hardware, like a prescaler (PSC) to divide the input clock frequency by a selectable value.

Example:

- n = 8 bits
- Fsys = 1000 Hz → Overflow every 1024 ms
- Prescaler = 4

Timers in microcontrollers

- Basic timer structure (with prescaler and autoreload register):



An auto-reload register allows the timer to generate overflow (and reset) in a different count value than the maximum one.

Example:

- n = 8 bits (up)
- Fsys = 1000 Hz
- Prescaler = 4
- Autoreload register = 200

→

Overflow timing:

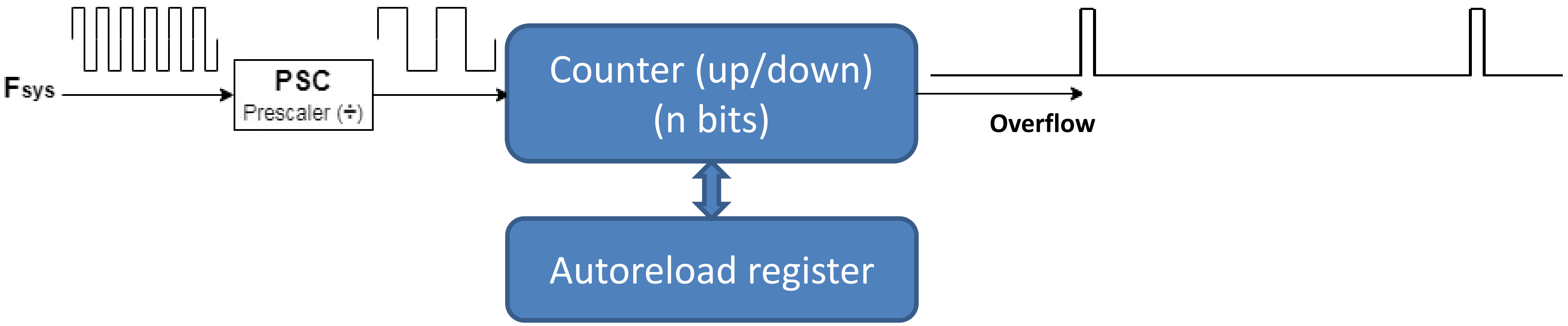
Fsys÷PSC → 250 Hz (input freq counter)

1/250 Hz → 4 ms

Overflow every: 200*4 = 800 ms

Timers in microcontrollers

- Generic timer structure:



Usually, timers also incorporate:

- Selectable clock sources, including external clock sources.
- Input capture registers to measure external events, signal frequency, time between events and so on.
- Output compare registers to generate pulses, square signals, PWM signals, etc.

STM32F4xx Block Diagram

- **Cortex-M4 w/ FPU, MPU and ETM**
- **Memory**
- Up to 1MB Flash memory
- 192KB RAM (including 64KB CCM data RAM
- FSMC up to 60MHz

- **New application specific peripherals**
- USB OTG HS w/ ULPI interface
- Camera interface
- HW Encryption**: DES, 3DES, AES 256-bit, SHA-1 hash, RNG.

- **Enhanced peripherals**
- USB OTG Full speed
- ADC: 0.416µs conversion/2.4Msps, up to 7.2Msps in interleaved triple mode
- ADC/DAC working down to 1.8V
- Dedicated PLL for I^{2}S precision
- Ethernet w/ HW IEEE1588 v2.0
- 32-bit RTC with calendar
- 4KB backup SRAM in VBAT domain
- **2 x 32bit and 8 x 16bit Timers**
- high speed USART up to 10.5Mb/s
- high speed SPI up to 37.5Mb/s

- **RDP (JTAG fuse)**
- **More I/Os in UFBGA 176 package**

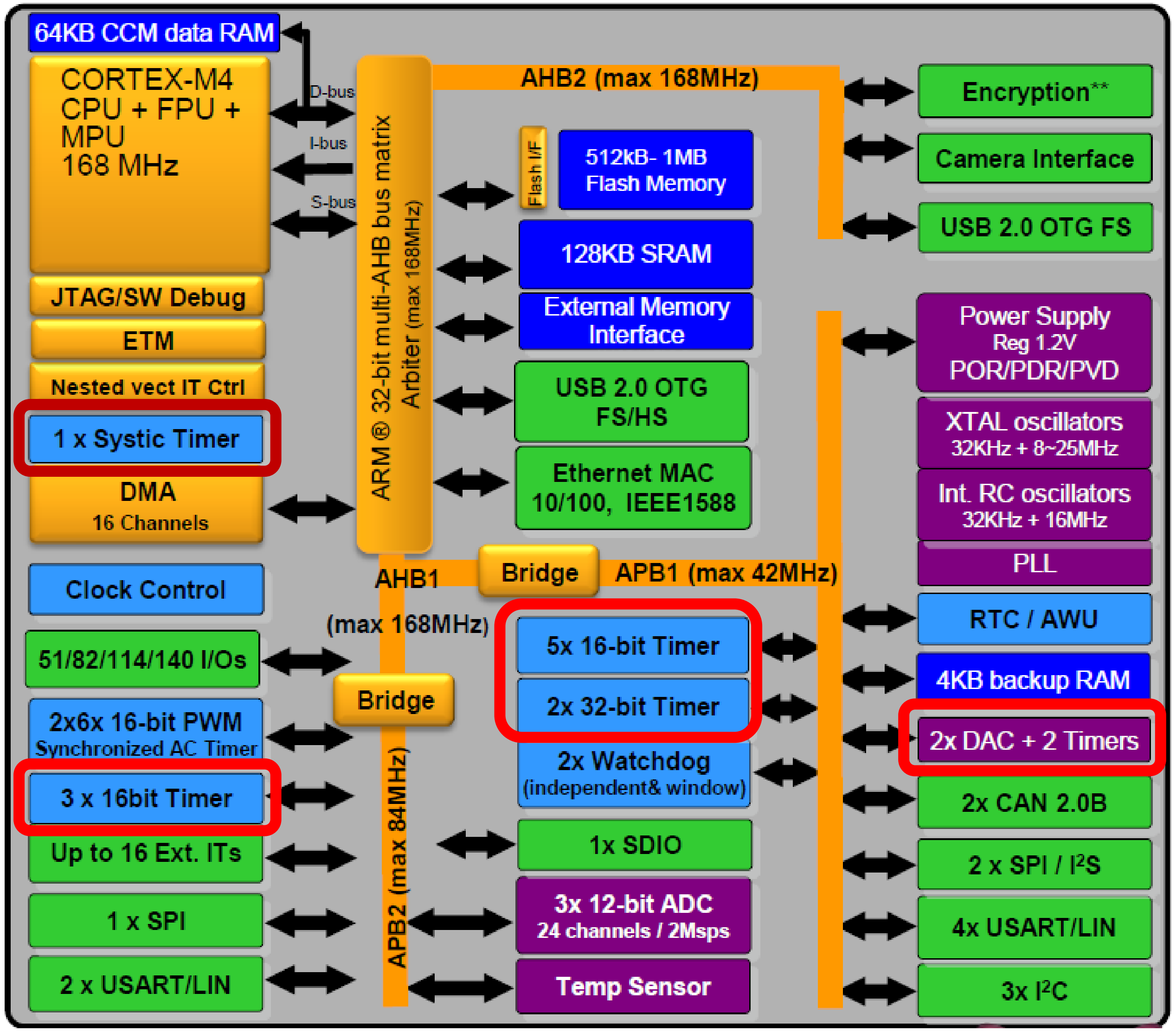
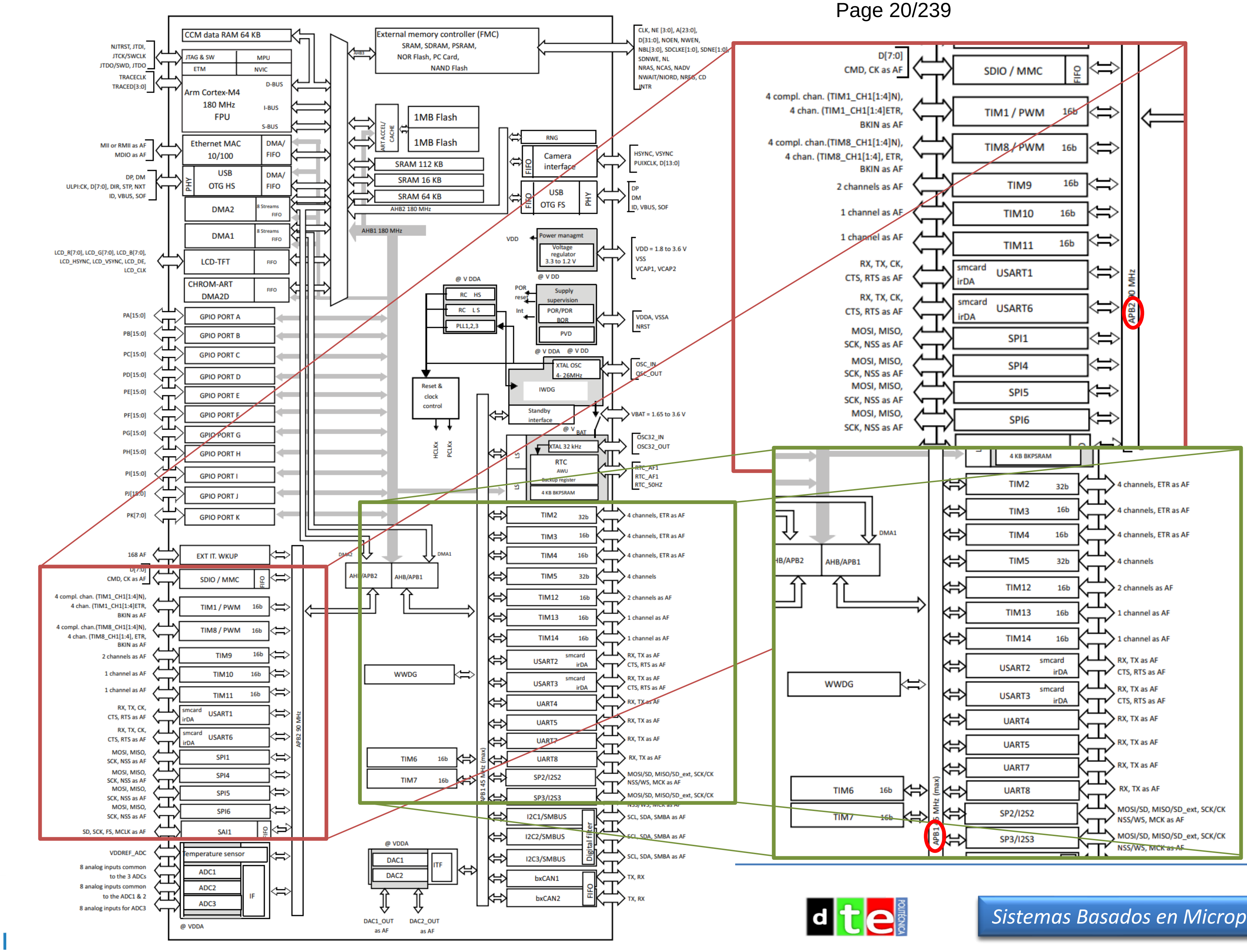


Figure 4. STM32F427xx and STM32F429xx block diagram

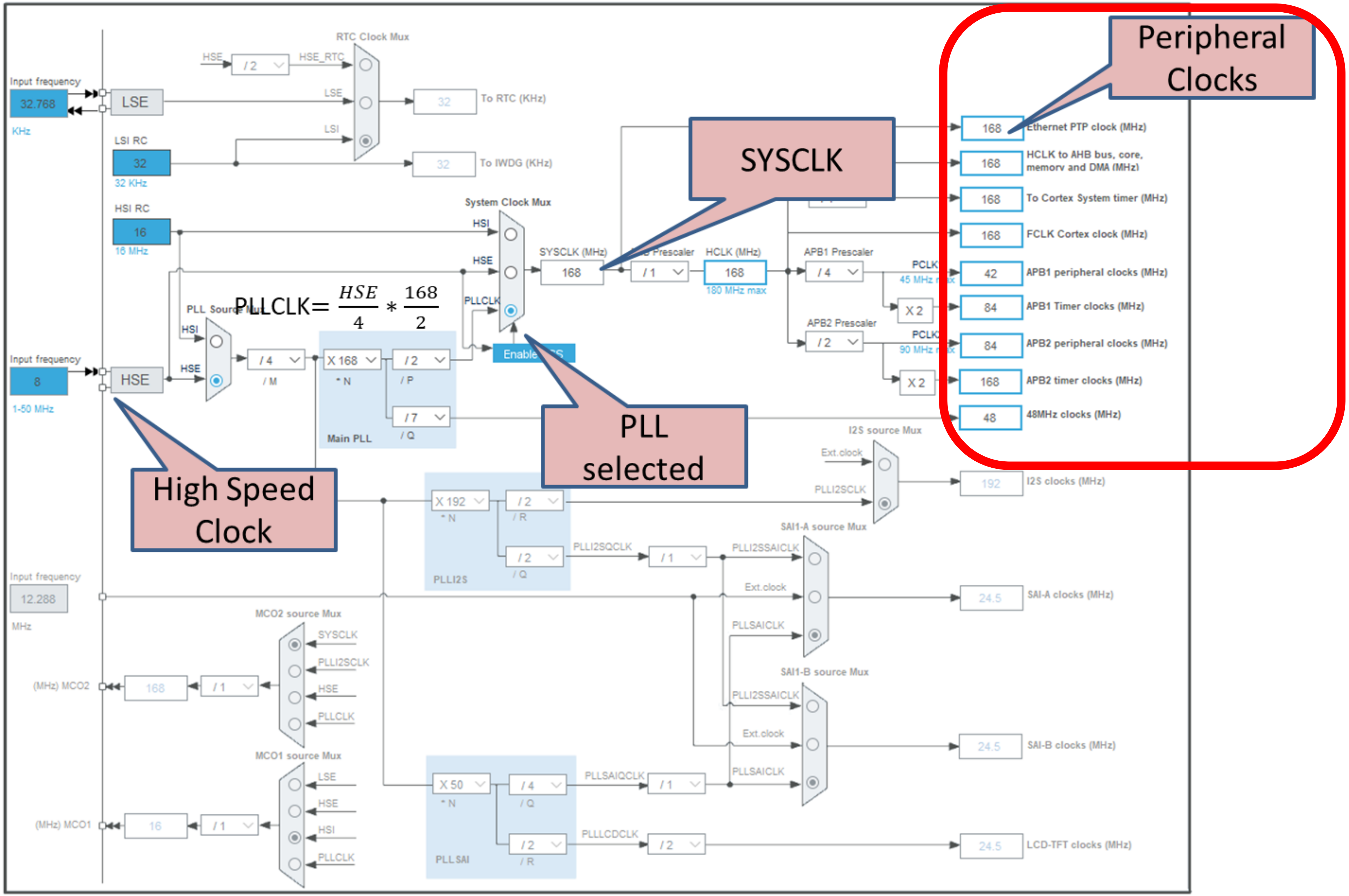
STM32F429-DATASHEET
Page 20/239



STM32F4xx Block Diagram

Timer clocks?

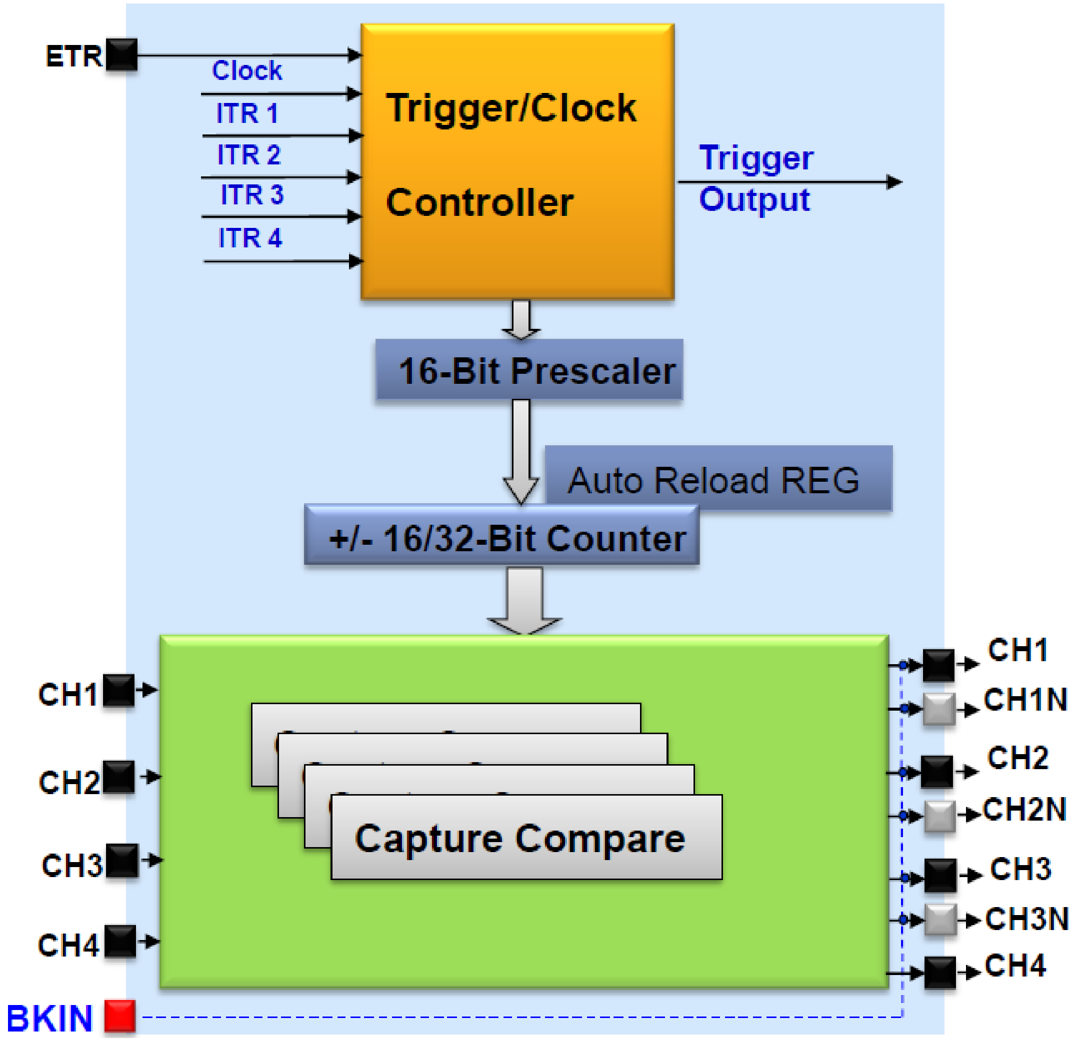
STM32F4xx Clock Diagram



Timers on STM32F4

On board there are following timers available:

- **2x advanced 16bit** timers (TIM1,8)
- **2x general purpose 32bit** timers (TIM2,5)
- **8x general purpose 16bit** timers (TIM3,4,9,10..14)
- **2x simple (basic) 16bit** timers for DAC (TIM6,7)
- **1x 24bit system timer** (SysTick)



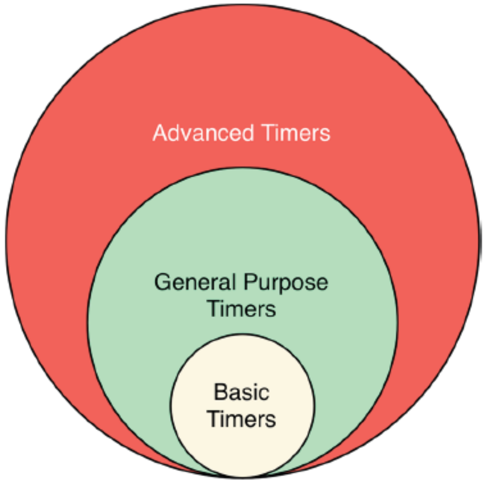
- Multiple timer units providing timing resources
  - Internally (triggers, time base)
  - Externally, for output or input:
    - For waveform generation (PWM)
    - For signal monitoring or measurement (frequency or timing)

Application benefits

- Versatile operating modes reducing CPU burden and minimizing interfacing circuitry needs
- A single architecture for all timer instances offers scalability and ease-of-use
- Also fully featured for motor control and digital power conversion applications

Timers on STM32F429

- STM32F429 has various built-in timers outlined as follows:
  - **Basic timers** are used either as time-base timers or for triggering the DAC peripheral. These timers do not have any input/output capabilities.
  - **General-purpose timers** can be used by any application for output comparison (timing and delay generation), one-pulse mode, input capture (for external signal frequency measurement), and sensor interface (encoder, hall sensor). Obviously, a general-purpose timer can be used as time base generator, like a *basic timer*. Timers from this category provide four-programmable input/output channels.
    - **1-channel/2-channels**: they are two subgroups of general-purpose timers providing only one/two input/output channel.
  - **Advanced timers**: these timers have the most features. In addition to general purpose functions, they include several features related to motor control and digital power conversion applications: three complementary signals with deadtime insertion and emergency shut-down input.



Timer availability in STM32Fx products

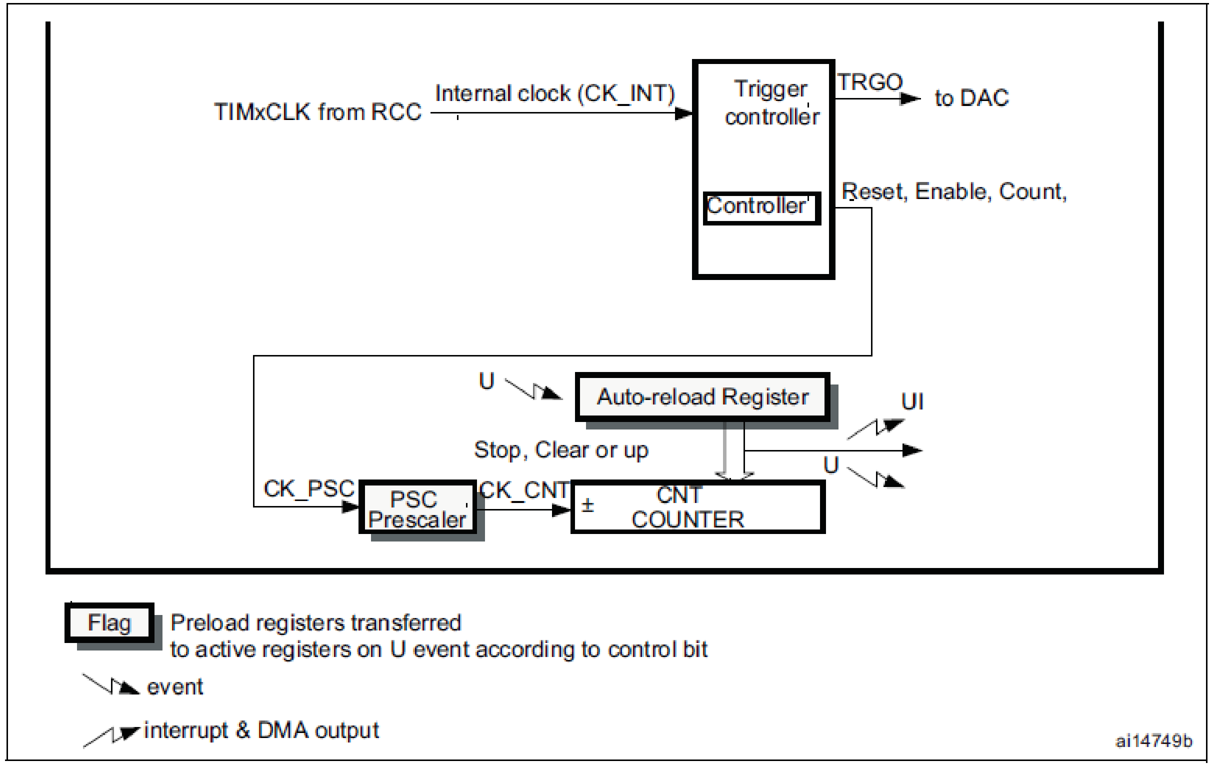
Timer type		STM32 F04x /F070x6 /F03x (excluding /F030x8 and /F030x)	STM32 F030xB /F030x8 /F05x /F09x /F07x (excluding F070x6)	STM32 F101 /F102 /F103 lines XL density (xF, xG)	STM32 F101 /F102 /F103 /F105 /F107 lines up to high-density (x4-xE)	STM32 F100 value line	STM32 F2 /F4 (excluding /F401, /F411, /F410)	STM32 F401 /F411 /F410	STM32 F30X /F3x8 (excluding /F378)	STM32 F37x	STM32 F334	STM32 F31x	STM32 F7 Series
Advanced		TIM1	TIM1	TIM1[1] TIM8[1]	TIM1[1] TIM8[1]	TIM1	TIM1 TIM8	TIM1	TIM1 TIM8[1] TIM20[1]	-	TIM1	TIM1 TIM8[1]	TIM1 TIM8
General purpose	32-bit	TIM2	TIM2	-	-	-	TIM2 TIM5	TIM2[1] TIM5	TIM2	TIM2 TIM5	TIM2	TIM2	TIM2 TIM5
	16-bit	TIM3	TIM3	TIM2 TIM3 TIM4 TIM5	TIM2 TIM3 TIM4[1] TIM5[1]	TIM2 TIM3 TIM4 TIM5[1]	TIM3 TIM4	TIM3[1] TIM4[1]	TIM3[1] TIM4[1] TIM19[1]	TIM3 TIM4 TIM19	TIM3	TIM3 TIM4	TIM3 TIM4
Basic		-	TIM6 TIM7[1]	TIM6 TIM7	TIM6[1] TIM7[1]	TIM6 TIM7	TIM6 TIM7	TIM6[1]	TIM6 TIM7[1]	TIM6 TIM7 TIM18	TIM6 TIM7	TIM6 TIM7[1]	TIM6 TIM7
1 channel		TIM14	TIM14	TIM10 TIM11 TIM13 TIM14	-	TIM13[1] TIM14[1]	TIM10 TIM11 TIM13 TIM14	TIM10[1] TIM11	-	TIM13 TIM14	-	-	TIM10 TIM11 TIM13 TIM14
2-channel		-	-	TIM9 TIM12	-	TIM12[1]	TIM9 TIM12	TIM9	-	TIM12	-	-	TIM9 TIM12
2-channel with complementary output		-	TIM15	-	-	TIM15	-	-	TIM15	TIM15	TIM15	TIM15	-
1-channel with complementary output		TIM16 TIM17	TIM16 TIM17	-	-	TIM16 TIM17	-	-	TIM16 TIM17	TIM16 TIM17	TIM16 TIM17	TIM16 TIM17	-
Low-power timer		-	-	-	-	-	-	LPTIM1[1]	-	-	-	-	LPTIM1
High-resolution timer		-	-	-	-	-	-	-	-	-	HRTIM	-	-

The most relevant feature of each timer category

Timer Type	Counter resolution	Counter type	DMA	Channels	Complimentary channels	Synchronization	
						Master	Slave
Advanced	16-bit	up, down and center aligned	Yes	4	3	Yes	Yes
General purpose	16/32-bit	up, down and center aligned	Yes	4	0	Yes	Yes
Basic	16-bit	up	Yes	0	0	Yes	No
1-channel	16-bit	up	No	1	0	Yes (OC signal)	No
2-channels	16-bit	up	No	2	0	Yes	Yes
1-channel with one complementary output	16-bit	up	Yes	1	1	Yes (OC signal)	No
2-channel with one complementary output	16-bit	up	Yes	2	1	Yes	Yes
High-resolution	16-bit	up	Yes	10	10	Yes	Yes
Low-power	16-bit	up	No	1	0	No	No

Basic timers (TIM6 and TIM7)

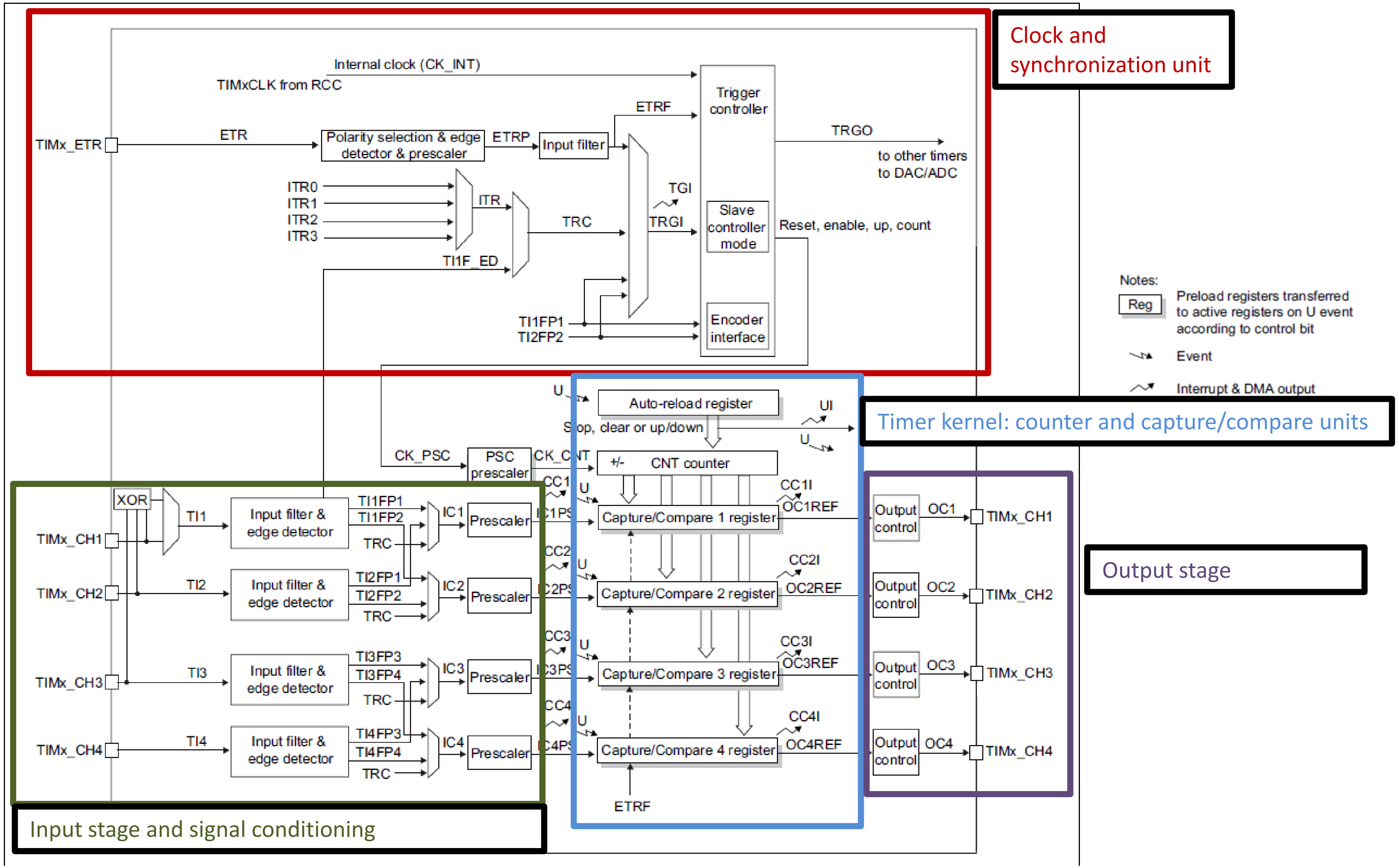
- Basic timer (TIM6 and TIM7) features:
  - 16-bit auto-reload UP counter
  - 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536
  - Synchronization circuit to trigger the DAC
  - Interrupt/DMA generation on the update event: counter overflow
  - No inputs, no outputs



General-purpose timers (TIM2 to TIM5)

- General-purpose TIM<x> timer features:
  - 16-bit (TIM3 and TIM4) or 32-bit (TIM2 and TIM5) up, down, up/down auto-reload counter.
  - 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536.
  - Up to 4 independent channels for:
    - **Input capture**
    - **Output compare**
    - **PWM generation (Edge- and Center-aligned modes)**
    - **One-pulse mode output**
  - Synchronization circuit to control the timer with external signals and to interconnect several timers.
  - Interrupt/DMA generation on the following events:
    - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
    - Trigger event (counter start, stop, initialization, or count by internal/external trigger)
    - Input capture
    - Output compare
  - Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
  - Trigger input for external clock or cycle-by-cycle current management

General-purpose timers (TIM2 to TIM5)



General purpose timers. Main uses.

- **Time Base Generator**. With internal and external clock sources (Basic timers only use internal clock sources).
- **Input capture mode.** The input capture mode offered by general purpose and advanced timers allows to compute the frequency of external signals applied to each one of the 4 channels that these timers provide.
- **Output Compare Mode**. The output compare is a mode offered by general purpose and advanced timers that allows to control the status of output channels when the channel compare register matches with the timer counter register.
- **Pulse-Width Generation**. General purpose timers allows to generate PWM signals (for motor control, LED dimming, power conversion, etc.)
- One Pulse Mode.
- Encoder Mode.
- Hall Sensor Mode.

Working with timers. HAL TIM Generic Driver. Registers structures

TIM_HandleTypeDef

TIM_HandleTypeDef is defined in the stm32f4xx_hal_tim.h

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***
- ***__IO HAL_TIM_ChannelStateTypeDef ChannelState***
- ***__IO HAL_TIM_ChannelStateTypeDef ChannelNState***
- ***__IO HAL_TIM_DMABurstStateTypeDef DMABurstState***

TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32f4xx_hal_tim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

See **UM1725.** User manual. *Description of STM32F4 HAL and low-layer drivers*.
Chapter 68. HAL TIM Generic Driver.

See also **stm32f4xx_hal_tim.h**

TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32f4xx_hal_tim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32f4xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32f4xx_hal_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Working with timers. TIM driver API

Time Base functions

- *HAL_TIM_Base_Init*
- *HAL_TIM_Base_DeInit*
- *HAL_TIM_Base_MspInit*
- *HAL_TIM_Base_MspDeInit*
- *HAL_TIM_Base_Start*
- *HAL_TIM_Base_Stop*
- *HAL_TIM_Base_Start_IT*
- *HAL_TIM_Base_Stop_IT*
- *HAL_TIM_Base_Start_DMA*
- *HAL_TIM_Base_Stop_DMA*

TIM Input Capture functions

- *HAL_TIM_IC_Init*
- *HAL_TIM_IC_DeInit*
- *HAL_TIM_IC_MspInit*
- *HAL_TIM_IC_MspDeInit*
- *HAL_TIM_IC_Start*
- *HAL_TIM_IC_Stop*
- *HAL_TIM_IC_Start_IT*
- *HAL_TIM_IC_Stop_IT*
- *HAL_TIM_IC_Start_DMA*
- *HAL_TIM_IC_Stop_DMA*

TIM PWM functions

- *HAL_TIM_PWM_Init*
- *HAL_TIM_PWM_DeInit*
- *HAL_TIM_PWM_MspInit*
- *HAL_TIM_PWM_MspDeInit*
- *HAL_TIM_PWM_Start*
- *HAL_TIM_PWM_Stop*
- *HAL_TIM_PWM_Start_IT*
- *HAL_TIM_PWM_Stop_IT*
- *HAL_TIM_PWM_Start_DMA*
- *HAL_TIM_PWM_Stop_DMA*

TIM Output Compare functions

- *HAL_TIM_OC_Init*
- *HAL_TIM_OC_DeInit*
- *HAL_TIM_OC_MspInit*
- *HAL_TIM_OC_MspDeInit*
- *HAL_TIM_OC_Start*
- *HAL_TIM_OC_Stop*
- *HAL_TIM_OC_Start_IT*
- *HAL_TIM_OC_Stop_IT*
- *HAL_TIM_OC_Start_DMA*
- *HAL_TIM_OC_Stop_DMA*

TIM Callbacks functions

- *HAL_TIM_PeriodElapsedCallback*
- *HAL_TIM_PeriodElapsedHalfCpltCallback*
- *HAL_TIM_OC_DelayElapsedCallback*
- *HAL_TIM_IC_CaptureCallback*
- *HAL_TIM_IC_CaptureHalfCpltCallback*
- *HAL_TIM_PWM_PulseFinishedCallback*
- *HAL_TIM_PWM_PulseFinishedHalfCpltCallback*
- *HAL_TIM_TriggerCallback*
- *HAL_TIM_TriggerHalfCpltCallback*
- *HAL_TIM_ErrorCallback*

See **UM1725.** User manual. *Description of STM32F4 HAL and low-layer drivers*. 68. HAL TIM Generic Driver.
See **stm32f4xx_hal_tim.h** and **stm32f4xx_hal_tim.c**

Timer modes

- The HAL provides three ways to use timers: ***polling***, ***interrupt*** and ***DMA*** *mode*. For this reason, the HAL provides three distinct functions to start/stop a timer:
  - HAL_TIM_Base_Start()
  - HAL_TIM_Base_Start_IT()
  - HAL_TIM_Base_Start_DMA()

  - HAL_TIM_IC_Start()
  - HAL_TIM_IC_Start_IT()
  - HAL_TIM_IC_Start_DMA()

  - HAL_TIM_OC_Start()
  - HAL_TIM_OC_Start_IT()
  - HAL_TIM_OC_Start_DMA()

  - …

Timer structures

stm32f4xx_hal_tim.h

TIM_HandleTypeDef

Timer to be set up

Basic parameters to be configured

```c
/**
  * @brief  TIM Time Base Handle Structure definition
  */
#if (USE_HAL_TIM_REGISTER_CALLBACKS == 1)
typedef struct __TIM_HandleTypeDef
#else
typedef struct
#endif /* USE_HAL_TIM_REGISTER_CALLBACKS */
{
  TIM_TypeDef                 *Instance;     /*!< Register base address             */
  TIM_Base_InitTypeDef        Init;          /*!< TIM Time Base required parameters */
  HAL_TIM_ActiveChannel       Channel;       /*!< Active channel                    */
  DMA_HandleTypeDef           *hdma[7];      /*!< DMA Handlers array
                                                  This array is accessed by a @ref DMA_Handle_index */
  HAL_LockTypeDef             Lock;          /*!< Locking object                    */
  __IO HAL_TIM_StateTypeDef   State;         /*!< TIM operation state               */

#if (USE_HAL_TIM_REGISTER_CALLBACKS == 1)
  void (* Base_MspInitCallback)(struct __TIM_HandleTypeDef *htim);              /*!< TIM Base Msp Init Callback                          */
  void (* Base_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim);            /*!< TIM Base Msp DeInit Callback                        */
  void (* IC_MspInitCallback)(struct __TIM_HandleTypeDef *htim);                /*!< TIM IC Msp Init Callback                            */
  void (* IC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim);              /*!< TIM IC Msp DeInit Callback                          */
  void (* OC_MspInitCallback)(struct __TIM_HandleTypeDef *htim);                /*!< TIM OC Msp Init Callback                            */
  void (* OC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim);              /*!< TIM OC Msp DeInit Callback                          */
  void (* PWM_MspInitCallback)(struct __TIM_HandleTypeDef *htim);               /*!< TIM PWM Msp Init Callback                           */
  void (* PWM_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim);             /*!< TIM PWM Msp DeInit Callback                         */
  void (* OnePulse_MspInitCallback)(struct __TIM_HandleTypeDef *htim);          /*!< TIM One Pulse Msp Init Callback                     */
  void (* OnePulse_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim);        /*!< TIM One Pulse Msp DeInit Callback                   */
  void (* Encoder_MspInitCallback)(struct __TIM_HandleTypeDef *htim);           /*!< TIM Encoder Msp Init Callback                       */
  void (* Encoder_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim);         /*!< TIM Encoder Msp DeInit Callback                     */
  void (* HallSensor_MspInitCallback)(struct __TIM_HandleTypeDef *htim);        /*!< TIM Hall Sensor Msp Init Callback                   */
  void (* HallSensor_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim);      /*!< TIM Hall Sensor Msp DeInit Callback                 */
  void (* PeriodElapsedCallback)(struct __TIM_HandleTypeDef *htim);             /*!< TIM Period Elapsed Callback                         */
  void (* PeriodElapsedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim);     /*!< TIM Period Elapsed half complete Callback           */
  void (* TriggerCallback)(struct __TIM_HandleTypeDef *htim);                   /*!< TIM Trigger Callback                                */
  void (* TriggerHalfCpltCallback)(struct __TIM_HandleTypeDef *htim);           /*!< TIM Trigger half complete Callback                  */
  void (* IC_CaptureCallback)(struct __TIM_HandleTypeDef *htim);                /*!< TIM Input Capture Callback                          */
  void (* IC_CaptureHalfCpltCallback)(struct __TIM_HandleTypeDef *htim);        /*!< TIM Input Capture half complete Callback            */
  void (* OC_DelayElapsedCallback)(struct __TIM_HandleTypeDef *htim);           /*!< TIM Output Compare Delay Elapsed Callback           */
  void (* PWM_PulseFinishedCallback)(struct __TIM_HandleTypeDef *htim);         /*!< TIM PWM Pulse Finished Callback                     */
  void (* PWM_PulseFinishedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim); /*!< TIM PWM Pulse Finished half complete Callback       */
  void (* ErrorCallback)(struct __TIM_HandleTypeDef *htim);                     /*!< TIM Error Callback                                  */
  void (* CommutationCallback)(struct __TIM_HandleTypeDef *htim);               /*!< TIM Commutation Callback                            */
  void (* CommutationHalfCpltCallback)(struct __TIM_HandleTypeDef *htim);       /*!< TIM Commutation half complete Callback              */
  void (* BreakCallback)(struct __TIM_HandleTypeDef *htim);                     /*!< TIM Break Callback                                  */
#endif /* USE_HAL_TIM_REGISTER_CALLBACKS */
} TIM_HandleTypeDef;
```

Timer structures

TIM_Base_InitTypeDef

```
/**
  * @brief  TIM Time base Configuration Structure definition
  */
typedef struct
{
  uint32_t Prescaler;         /*!< Specifies the prescaler value used to divide the TIM clock.
                                   This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF */

  uint32_t CounterMode;       /*!< Specifies the counter mode.
                                   This parameter can be a value of @ref TIM_Counter_Mode */

  uint32_t Period;            /*!< Specifies the period value to be loaded into the active
                                   Auto-Reload Register at the next update event.
                                   This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.  */

  uint32_t ClockDivision;     /*!< Specifies the clock division.
                                   This parameter can be a value of @ref TIM_ClockDivision */

  uint32_t RepetitionCounter;  /*!< Specifies the repetition counter value. Each time the RCR downcounter
                                    reaches zero, an update event is generated and counting restarts
                                    from the RCR value (N).
                                    This means in PWM mode that (N+1) corresponds to:
                                        - the number of PWM periods in edge-aligned mode
                                        - the number of half PWM period in center-aligned mode
                                     GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
                                     Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF. */

  uint32_t AutoReloadPreload;  /*!< Specifies the auto-reload preload.
                                   This parameter can be a value of @ref TIM_AutoReloadPreload */
} TIM_Base_InitTypeDef;
```

Please note that the maximum value is 65535 (0xFFFF)

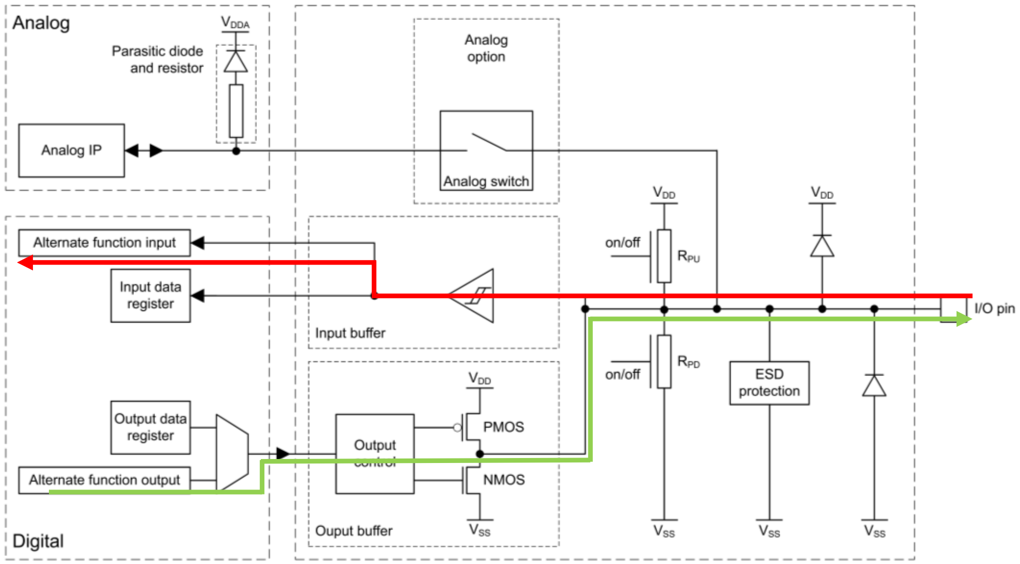
By default, counter UP

Auto-reload value

How to use the HAL TIM driver

1. Initialize the TIM low-level resources by implementing the following functions depending on the selected feature: Time base, output compare, input capture, PWM, …

2. Initialize the TIM low level resources:
   1. Enable the TIM interface clock using **__HAL_RCC_TIMx_CLK_ENABLE();**
   2. TIM pins configuration
      - Enable the clock for the TIM GPIOs using **__HAL_RCC_GPIOx_CLK_ENABLE();**
      - Configure these TIM pins **in Alternate function mode**



3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: **HAL_TIM_ConfigClockSource**. The clock configuration should be done before any start function.

How to use the HAL TIM driver

4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver (for example: **HAL_TIM_Base_Init** to use the Timer to generate a simple time base; or **HAL_TIM_OC_Init** and **HAL_TIM_OC_ConfigChannel**: to use the Timer to generate an Output Compare signal).

5. Activate the TIM peripheral using one of the start functions depending on the feature used (for example: **HAL_TIM_Base_Start_IT()**).

6. Enable, if needed, the peripheral IRQ (for example: **HAL_NVIC_EnableIRQ(TIM7_IRQn)**)

7. If interrupts are configured, place and codify the IRQ handler and the Callback function (put it in the **stm32f4xx_it.c** file).

```
void TIM7_IRQHandler(void) {
// Pass the control to HAL, which processes the IRQ
  HAL_TIM_IRQHandler(&htim7);
 }

 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
   if(htim->Instance == TIM7)
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
 }
```

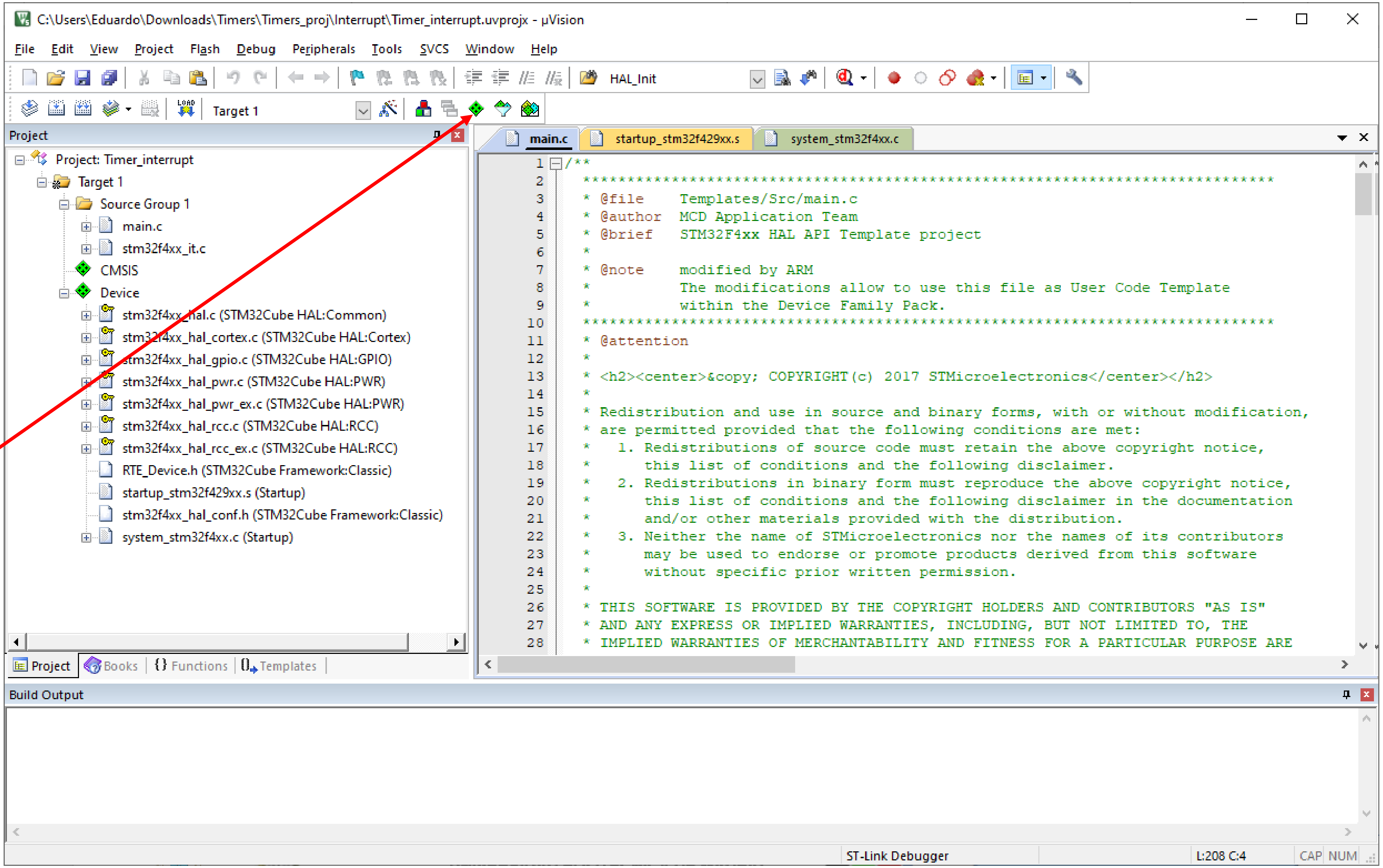
Example 1. Using Timers to generate periodic interruptions

1. Create a new Keil project including the HAL Timer
2. Use the Timer 7 (Basic Timer) to generate a periodic interrupt every 500ms
3. Toggle the LED_1 in the Timer 7 ISR

11. Timers

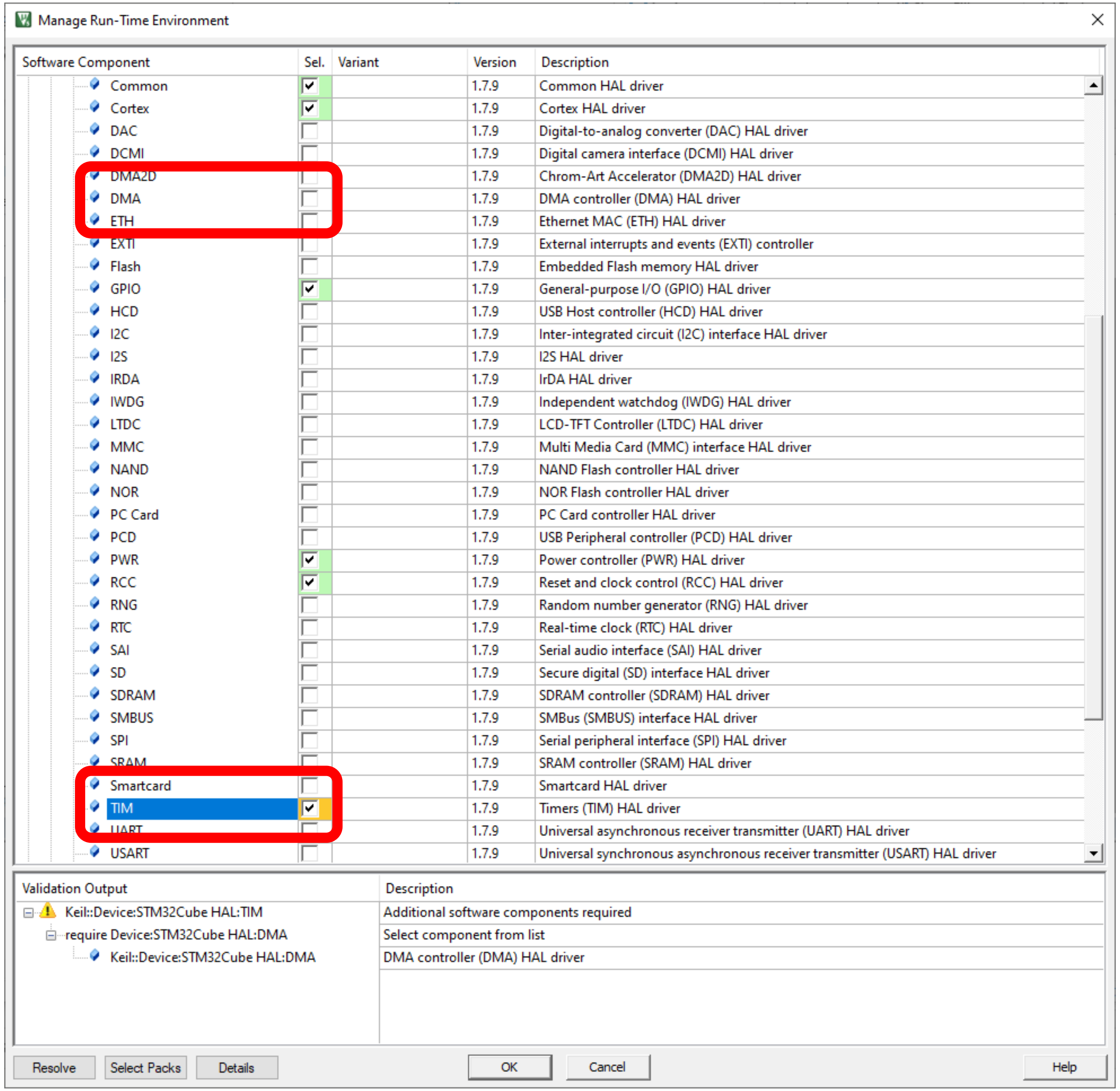
11.2.1 Using Timers in *Interrupt Mode*

How to configure a Keil project

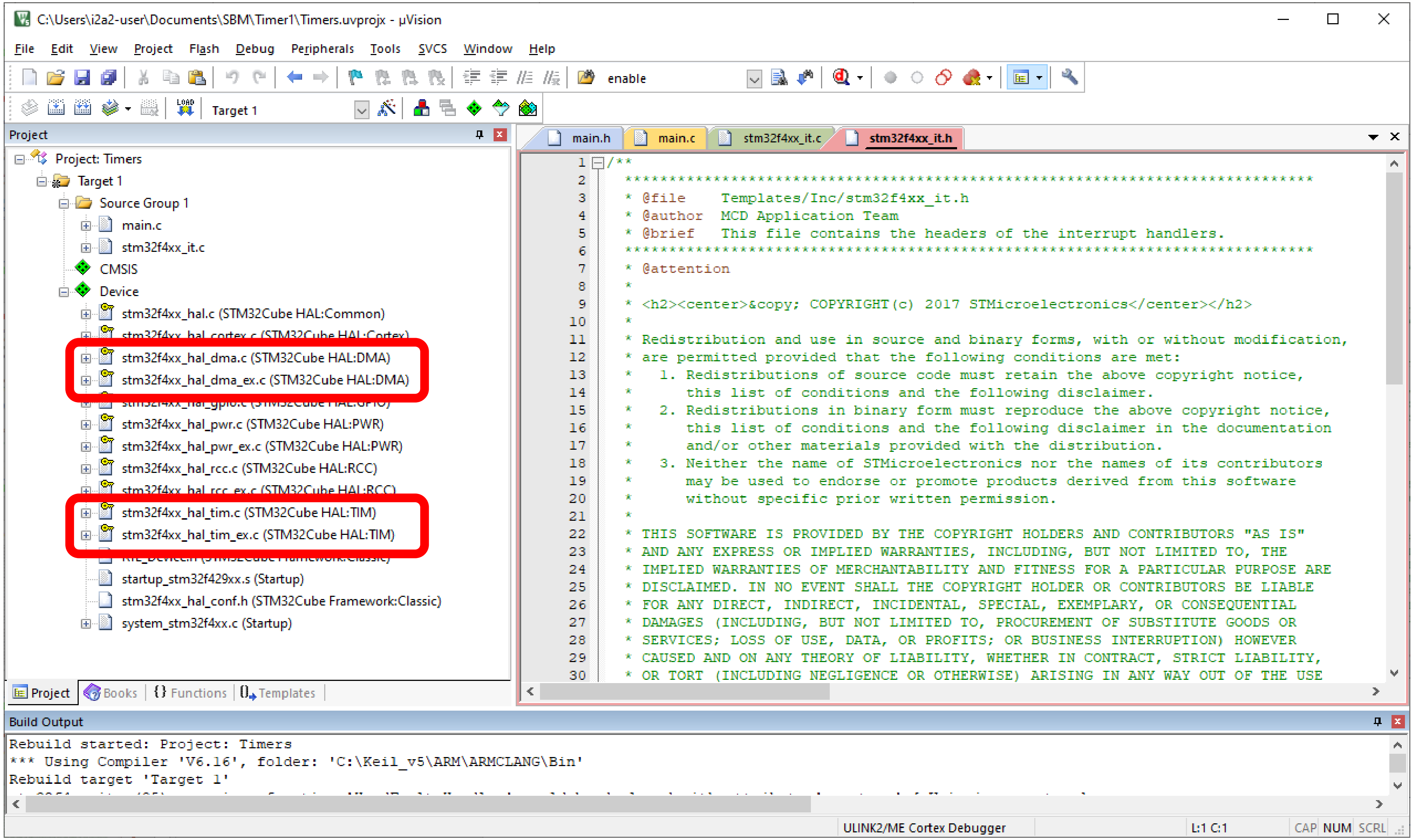


Add component software

How to configure a Keil project



How to configure a Keil project



Example 1. Timer interrupts

```c
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
static void SystemClock_Config(void);
static void Error_Handler(void);
void LED_Init(void);

/* Private functions ---------------------------------------------------------*/
/**
  * @brief  Main program
  * @param  None
  * @retval None
  */

TIM_HandleTypeDef htim7;

int main(void)
{

  /* STM32F4xx HAL library initialization:
       - Confi
  gure the Flash prefetch, Flash preread and Buffer caches
       - Systick timer is configured by default as source of time base, but user
             can eventually implement his proper time base source (a general purpose
```

Example 1. Timer interrupts

```c
  LED_Init();

  htim7.Instance = TIM7;
  htim7.Init.Prescaler = 47999; //48MHz/48000 = 1000Hz (assuming APB timer clock is 48MHz. You should check this point!!)
  htim7.Init.Period = 499; //1000HZ / 500 = 2Hz = 0.5s

  HAL_NVIC_EnableIRQ(TIM7_IRQn);   //Enable the peripheral IRQ
  __HAL_RCC_TIM7_CLK_ENABLE();     //Enable the TIM7 peripheral

  HAL_TIM_Base_Init(&htim7); //Configure the timer
  HAL_TIM_Base_Start_IT(&htim7); //Start the timer

  /* Infinite loop */
  while (1)
  {
  }
}
```

Example 1. Timer interrupts

```c
void TIM7_IRQHandler(void) {
// Pass the control to HAL, which processes the IRQ
  HAL_TIM_IRQHandler(&htim7);
 }

 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
   if(htim->Instance == TIM7)
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
 }
```

Example 2. Using Timers to generate a square signal

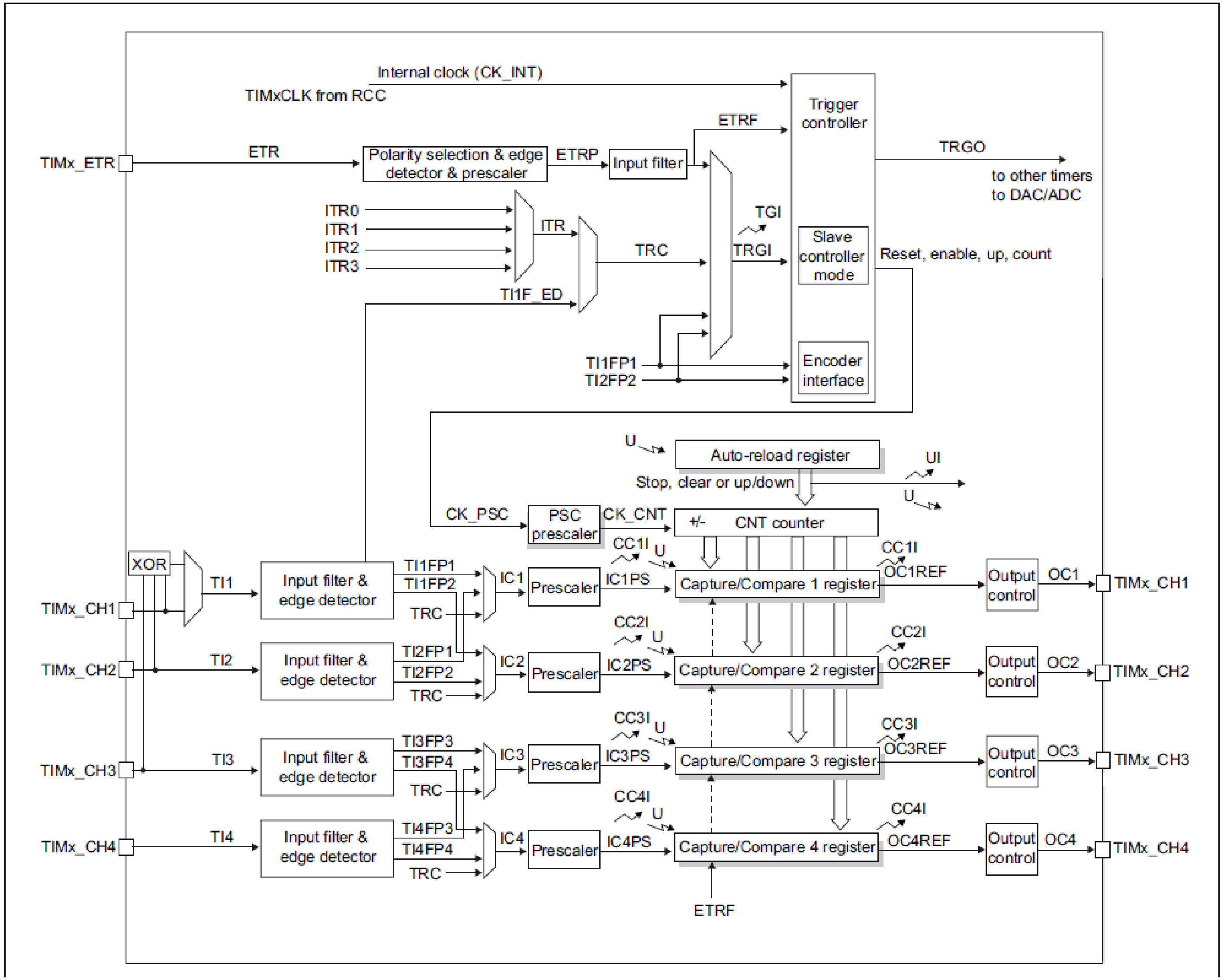
1. Create a new Keil project including the HAL Timer
2. Use Timer 2 (General-Purpose Timer) to generate a 2 kHz frequency square signal
3. **Generation of the square signal by the timer hardware, <u>without software intervention</u>**

11. Timers

11.3.6 Output Compare Mode

Example 2. Using Timers to generate a square signal

1. Identify possible outputs for Timer 2
2. Configure GPIO output as an alternate function
3. Configure Timer 2 for OC



Example 2. Using Timers to generate a square signal

Table 12. STM32F427xx and STM32F429xx alternate function mapping (continued)

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/2/3/4/5/6	SPI2/3/SAI1	SPI3/USART1/2/3	USART6/UART4/5/7/8	CAN1/2/TIM12/13/14/LCD	OTG2_HS/OTG1_FS	ETH	FMC/SDIO/OTG2_FS	DCMI	LCD	SYS
Port B	PB11	-	TIM2_CH4	-	-	I2C2_SDA	-	-	USART3_RX	-	-	OTG_HS_ULPI_D4	ETH_MII_TX_EN/ETH_RMII_TX_EN	-	-	LCD_G5	EVENTOUT
	PB12	-	TIM1_BKIN	-	-	I2C2_SMBA	SPI2_NSS/I2S2_WS	-	USART3_CK	-	CAN2_RX	OTG_HS_ULPI_D5	ETH_MII_TXD0/ETH_RMII_TXD0	OTG_HS_ID	-	-	EVENTOUT
	PB13	-	TIM1_CH1N	-	-	-	SPI2_SCK/I2S2_CK	-	USART3_CTS	-	CAN2_TX	OTG_HS_ULPI_D6	ETH_MII_TXD1/ETH_RMII_TXD1	-	-	-	EVENTOUT
								I2S2ext_SD	USART3_RTS	-	TIM12_CH1	-	-	OTG_HS_DM	-	-	EVENTOUT
								-	-	-	TIM12_CH2	-	-	OTG_HS_DP	-	-	EVENTOUT
								-	-	-	-	OTG_HS_ULPI_STP	-	FMC_SDNWE	-	-	EVENTOUT
								-	-	-	-	-	ETH_MDC	-	-	-	EVENTOUT
								I2S2ext_SD	-	-	-	OTG_HS_ULPI_DIR	ETH_MII_TXD2	FMC_SDNE0	-	-	EVENTOUT
								-	-	-	-	OTG_HS_ULPI_NXT	ETH_MII_TX_CLK	FMC_SDCKE0	-	-	EVENTOUT
								-	-	-	-	-	ETH_MII_RXD0/ETH_RMII_RXD0	-	-	-	EVENTOUT
								-	-	-	-	-	ETH_MII_RXD1/ETH_RMII_RXD1	-	-	-	EVENTOUT
								-	-	USART6_TX	-	-	-	SDIO_D6	DCMI_D0	LCD_HSYNC	EVENTOUT
								I2S3_MCK	-	USART6_RX	-	-	-	SDIO_D7	DCMI_D1	LCD_G6	EVENTOUT

STM32F427xx STM32F429xx

Pinouts and pin description

```c
static void initPIN_OUTPUT(void) {

    GPIO_InitTypeDef GPIO_InitStruct;

    /* Enable clock to GPIO-B */
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /* Set GPIOB Pin */
    GPIO_InitStruct.Pin         =   GPIO_PIN_11;
    GPIO_InitStruct.Mode        =   GPIO_MODE_AF_PP;
    GPIO_InitStruct.Alternate   =   GPIO_AF1_TIM2;

    /* Init GPIOB Pins */
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
```

Example 2. Using Timers to generate a square signal

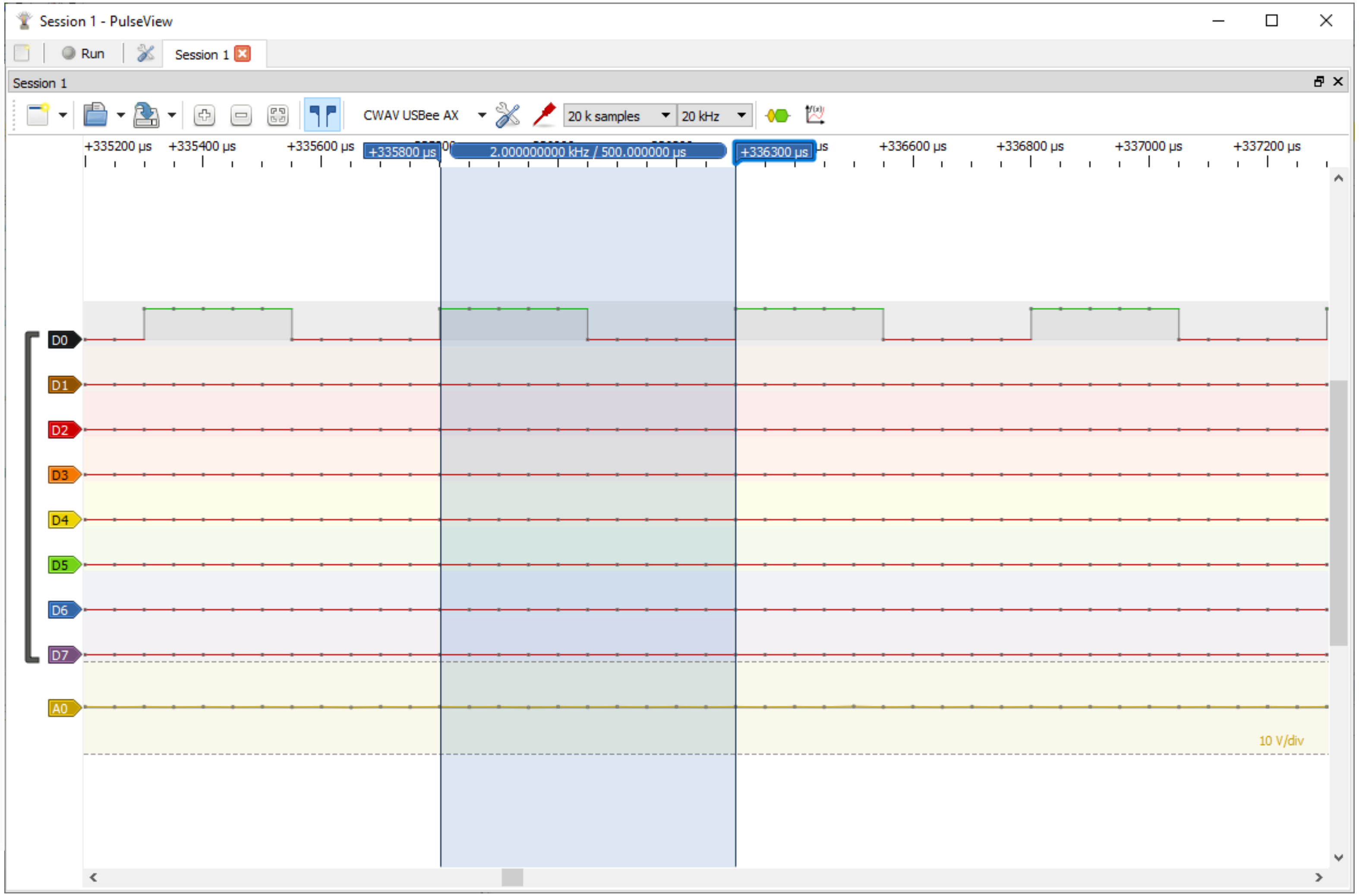
```c
static void initTimer(void) {

    TIM_OC_InitTypeDef TIM_Channel_InitStruct;

    /* Enable clock to Timer-2 */
    __HAL_RCC_TIM2_CLK_ENABLE();

    /***************************************************
                   Timer-2 Configuration:
    ****************************************************/

    tim2.Instance = TIM2;
    tim2.Init.Prescaler   = 0; // no prescaler (TIM2 clk 84 MHz)
    tim2.Init.Period = 20999;  // para una freq. de 2 KHz hay que hacer toggle cada 250 us --> p = 21000 ciclos
    HAL_TIM_OC_Init(&tim2);

    /* Finally initialize Timer-2, for Output  */

    TIM_Channel_InitStruct.OCMode        =   TIM_OCMODE_TOGGLE;
    TIM_Channel_InitStruct.OCPolarity    =   TIM_OCPOLARITY_HIGH;
    TIM_Channel_InitStruct.OCFastMode    =   TIM_OCFAST_DISABLE;
    HAL_TIM_OC_ConfigChannel(&tim2, &TIM_Channel_InitStruct, TIM_CHANNEL_4);

    HAL_TIM_OC_Start(&tim2, TIM_CHANNEL_4);

}
```

Considera que es Output y a la vez compare: Inicializado a nivel alto, a próx ciclo compara y hace toggle. NOS LIBRAMOS DE INTERRUPT. Su funcionamiento normal es estar alternando.

Example 2. Using Timers to generate a square signal

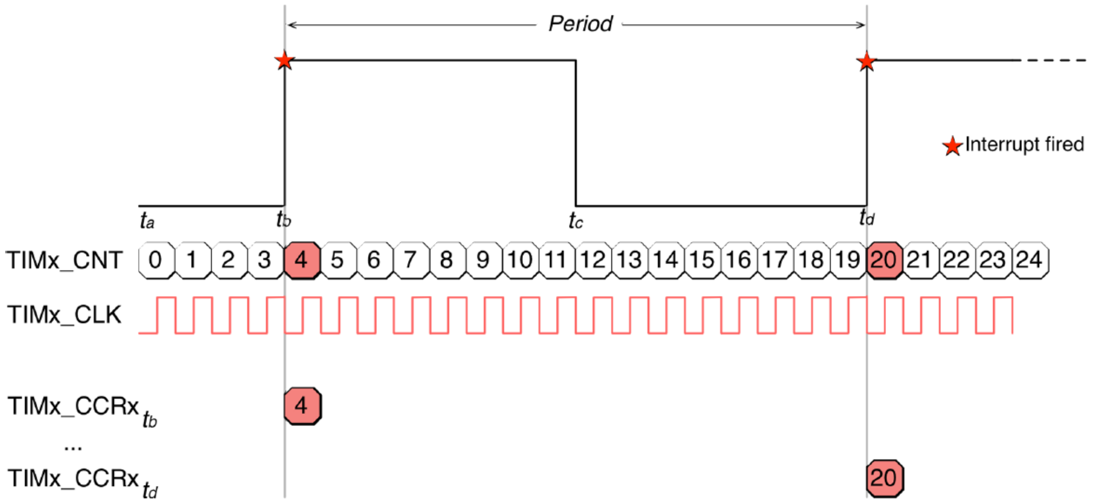
```
void TIM7_IRQHandler(void) {
// Pass the control to HAL, which processes the IRQ
  HAL_TIM_IRQHandler(&htim7);
 }

 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
   if(htim->Instance == TIM7)
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
 }
```

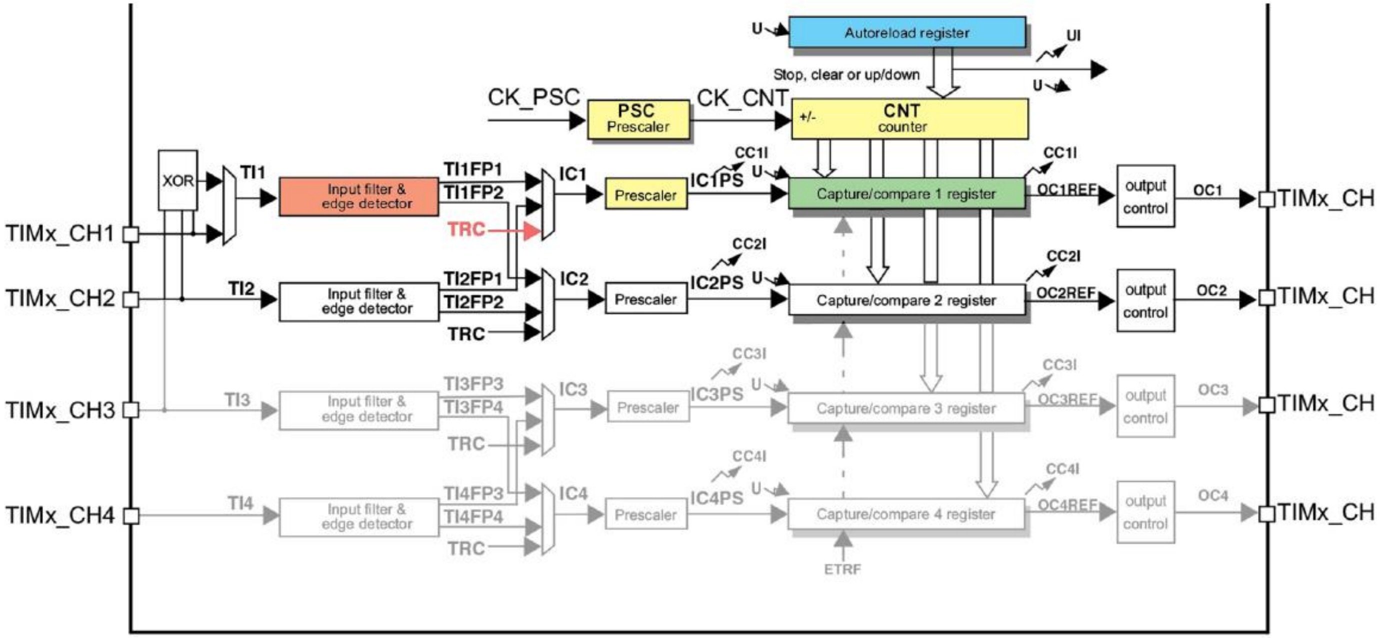
Example 2. Using Timers to generate a square signal



Example 3. Using Timers to measure the frequency of an external signal

11.3.5 Input Capture Mode





Example 4. Using Timers to generate a PWM signal

11.3.7 Pulse-Width Generation

- control the output voltage
- dimming of LEDs
- motor control
- power conversion
- …