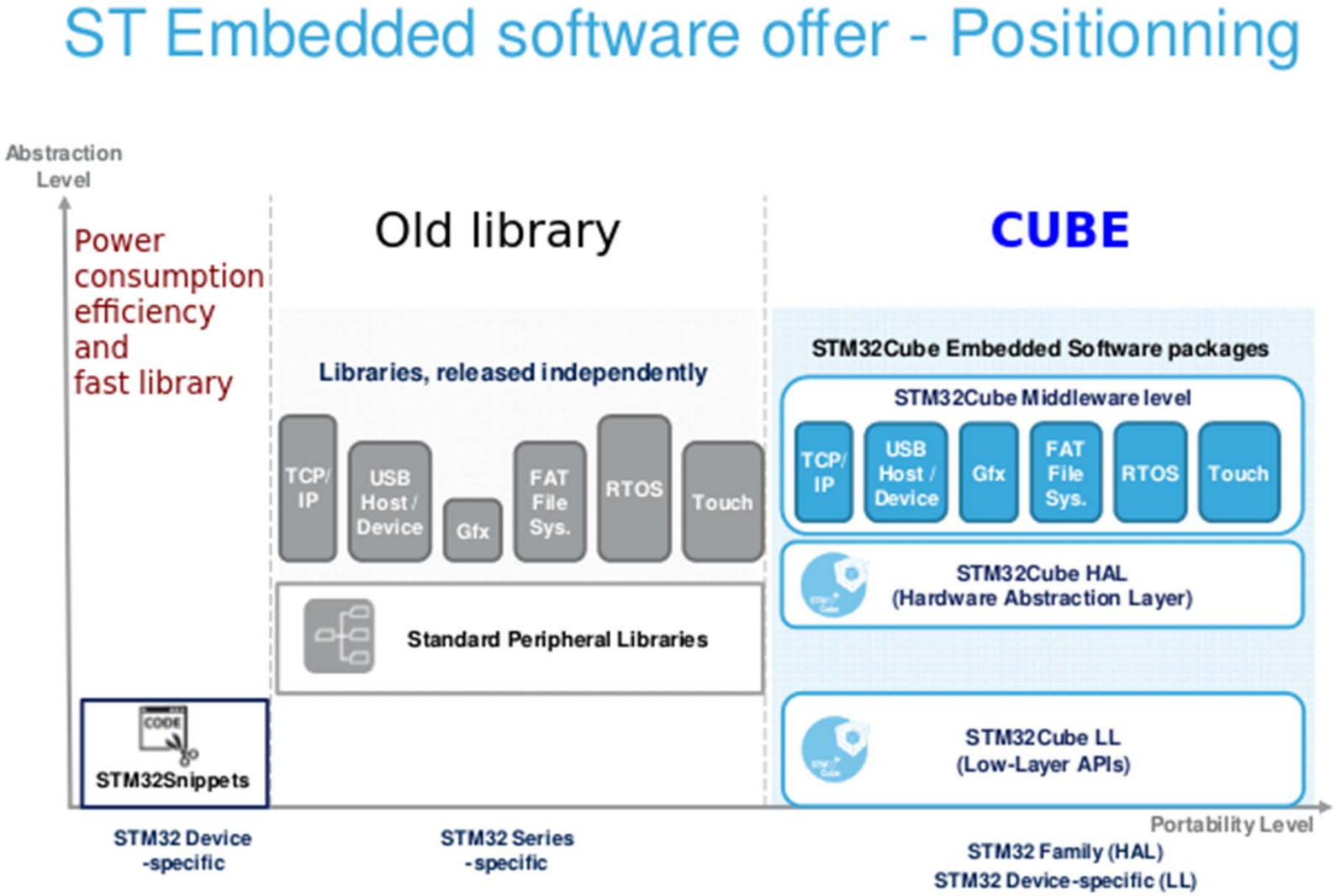
Hardware Abstraction & Low Level Layers for STM32

Problem to solve

- What is the best way to program a complex microprocessor-based system and its peripherals?
  - Directly using assembler (complex and device dependent)
  - Using code provided by the manufacturer for a specific device/model
  - Using software libraries with different optimizations levels
    - HAL and LL libraries are examples of this (STM32 CUBE)
  - In our course, we have decided to use the HAL provided by ST Microelectronics because it is the same library for all the 32-bit families

Three different approaches

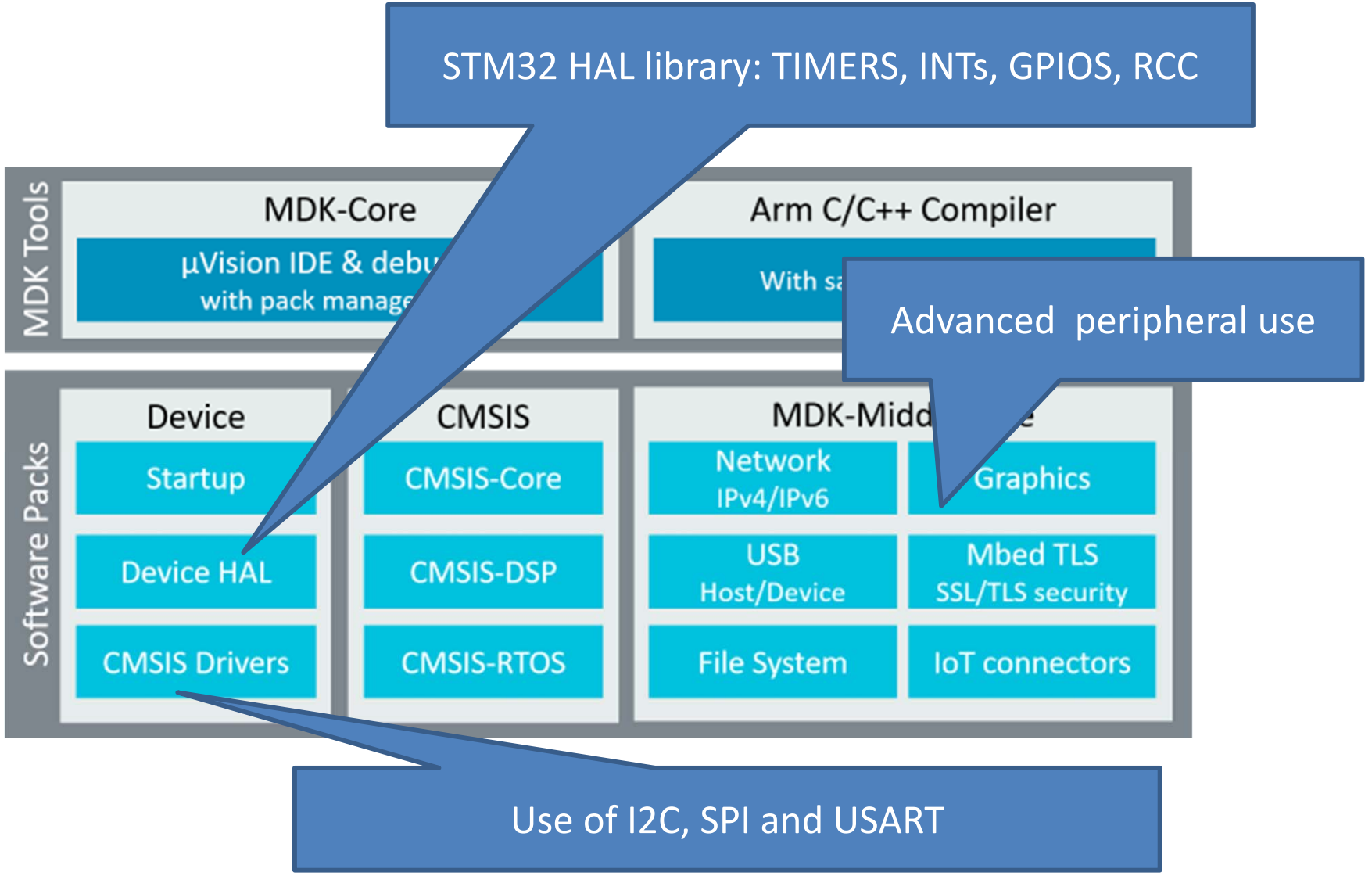


In Microprocessor Based Systems we will use the HAL API

ST Embedded software offer – Comparison

Offer		Portability	Optimization (Memory & Mips)	Easy	Readiness	Hardware coverage
STM32Snippets			+++			+
Standard Peripheral Library		++	++	+	++	+++
STM32 Cube	HAL API	+++	+	++	+++	+++
	LL APIs		+++			++

Development of applications for ARM 32 devices using KEIL-MDK



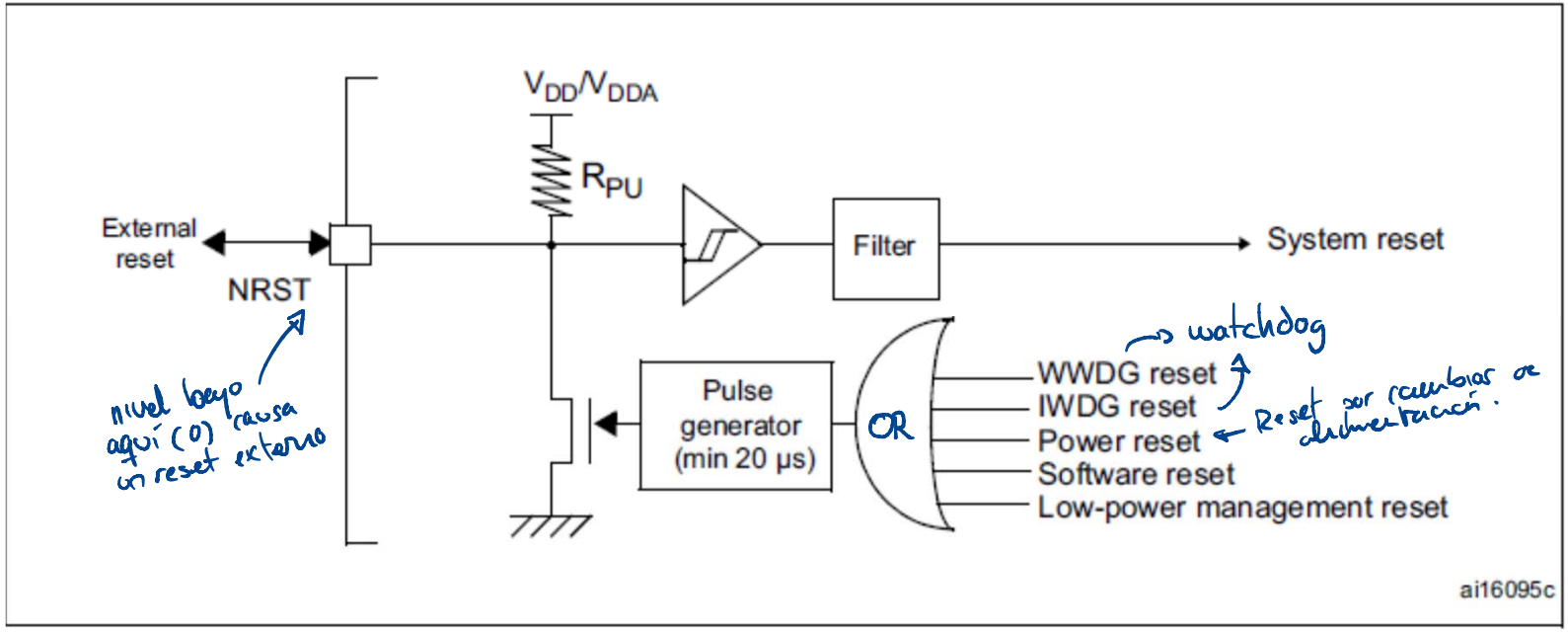
Software development in SBM HAL

- Keil Microvision for editing, compiling, linking, and debugging
- STM-32 HAL for configuring:
  - clocks and reset circuits (HAL RCC)
  - GPIO
  - Timers
  - Interrupts (NVIC and EXT1)
- CMSIS-Drivers for:
  - I2C and SPI
  - USART

RCC: Reset and Clock Control

Reset circuit

- Reset sources acts on NRST pin.
- Reset service routine is fixed at address 0x00000004

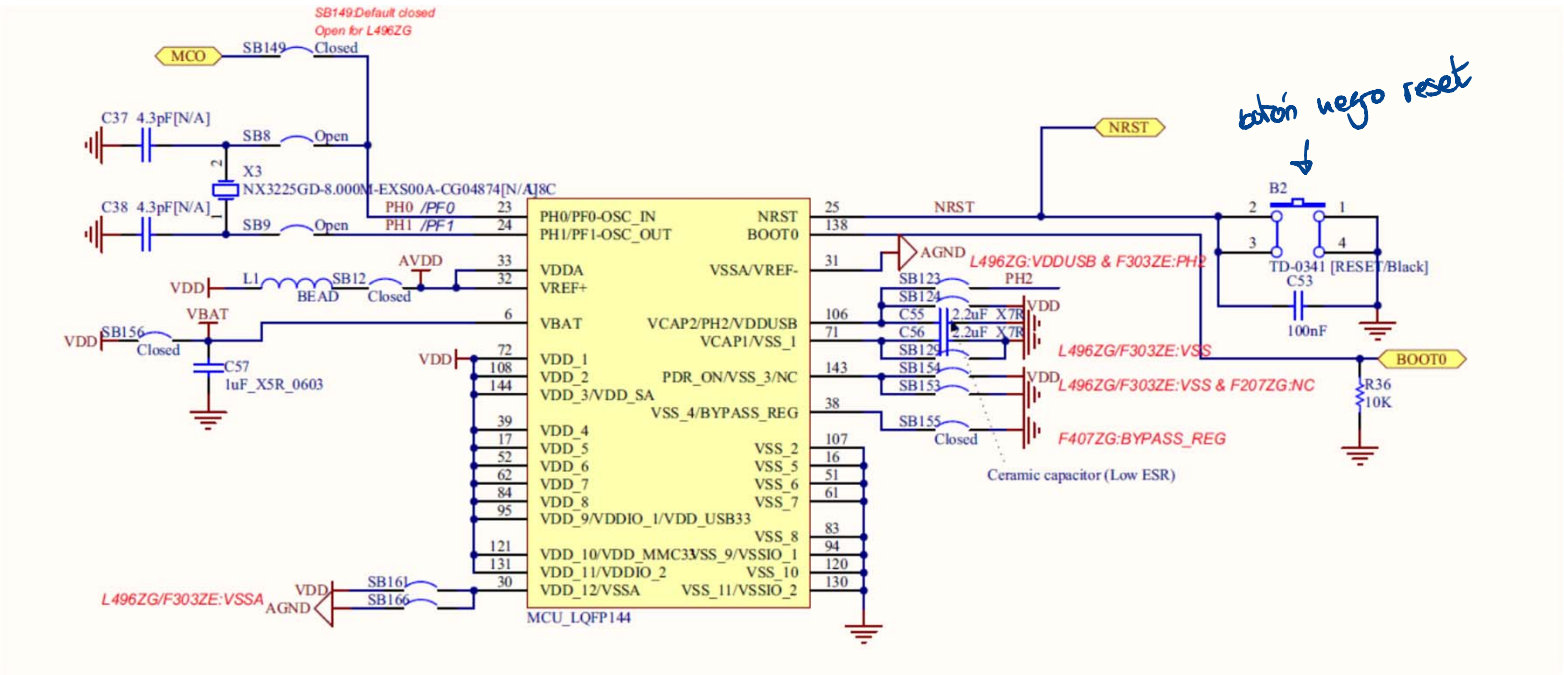


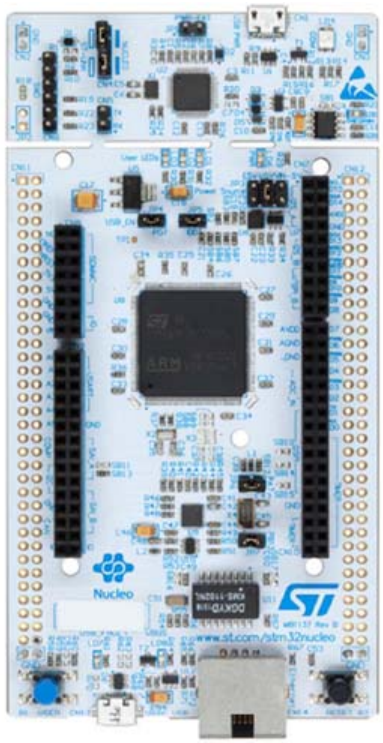
Reset circuit (3 types of reset sources)

- System Reset
  - Low level on external pin NRST
  - Window watchdog end of count condition (WWDG). Counter that must be refreshed within a specific time window. If not, the watchdog generates a system reset
  - Independent watchdog end of count (IWDG). Triggers a system reset if a counter reaches a given timeout value (it uses the RC oscillator)
  - Software reset
  - Low power management reset
    - Reset when entering standby mode
    - Reset when entering in Stop mode
- Power reset
  - Power-on/down reset (POR/PDR), brownout reset (BOR)
- Backup domain reset

NUCLEO-144 with STM32F429ZI

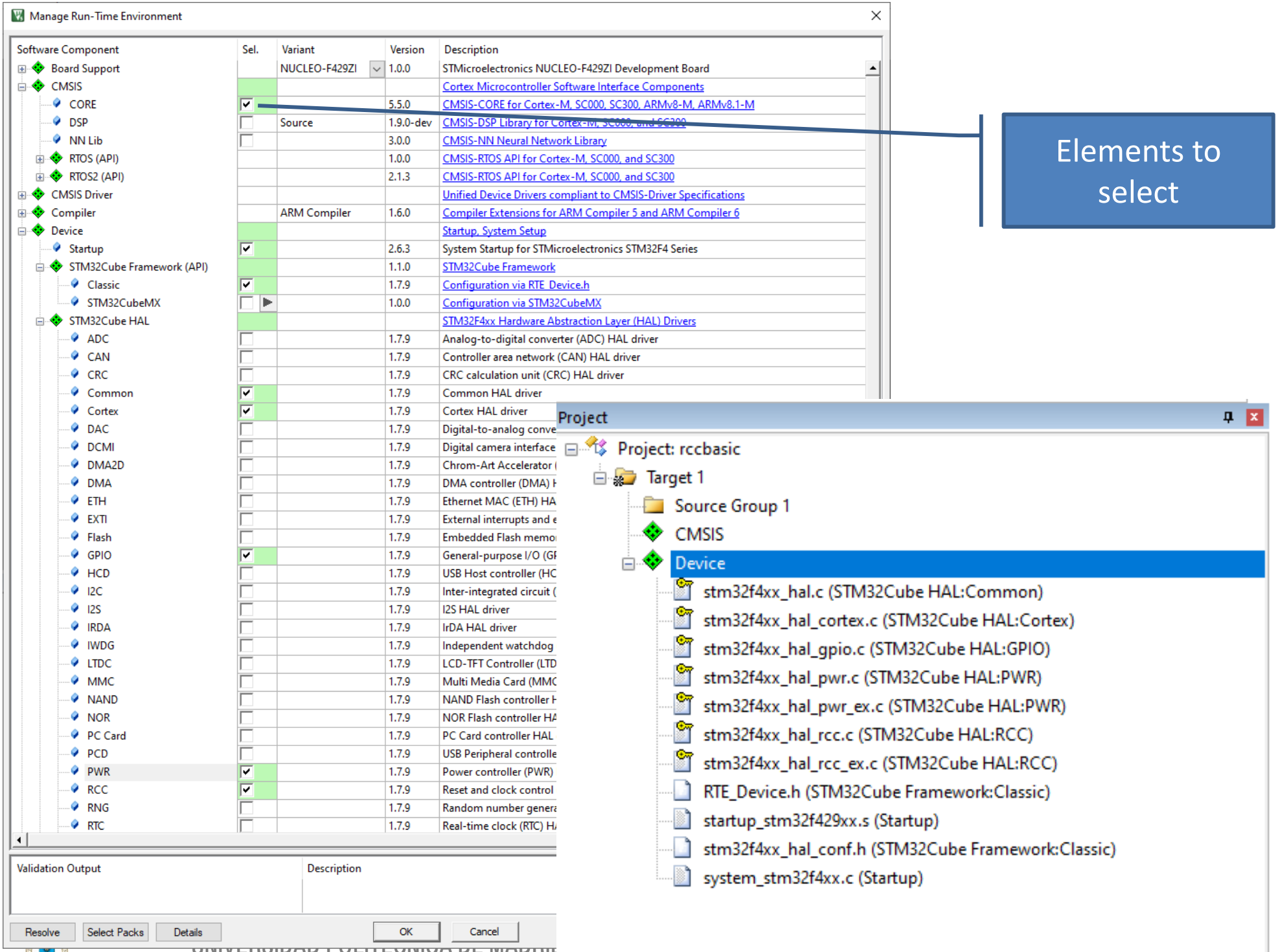
- B2 Button: generates reset on NRST signal
- Connection detail in NUCLEO-144 with STM32F429ZI (144 pins)





Creation of a default project with Keil Microvision

- Run Keil Microvision
- New Project
- Select the ST device
- Select the minimum software elements (next slide)
- Press OK
- Inspect the project created


Manage Run-Time Environment
Software Component
Sel.
Variant
Version
Description
Board Support
NUCLEO-F429ZI
1.0.0
STMicroelectronics NUCLEO-F429ZI Development Board
CMSIS
Cortex Microcontroller Software Interface Components
CORE
5.5.0
CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M, ARMv8.1-M
DSP
Source
1.9.0-dev
NN Lib
3.0.0
CMSIS-NN Neural Network Library
RTOS (API)
1.0.0
CMSIS-RTOS API for Cortex-M, SC000, and SC300
RTOS2 (API)
2.1.3
CMSIS Driver
Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler
ARM Compiler
1.6.0
Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device
Startup, System Setup
Startup
2.6.3
System Startup for STMicroelectronics STM32F4 Series
STM32Cube Framework (API)
1.1.0
STM32Cube Framework
Classic
1.7.9
Configuration via RTE_Device.h
STM32CubeMX
1.0.0
Configuration via STM32CubeMX
STM32Cube HAL
STM32F4xx Hardware Abstraction Layer (HAL) Drivers
ADC
Analog-to-digital converter (ADC) HAL driver
CAN
Controller area network (CAN) HAL driver
CRC
CRC calculation unit (CRC) HAL driver
Common
Common HAL driver
Cortex
Cortex HAL driver
DAC
DCMI
DMA2D
DMA
ETH
EXTI
Flash
GPIO
HCD
I2C
I2S
IRDA
IWDG
LTDC
MMC
NAND
NOR
PC Card
PCD
PWR
RCC
RNG
RTC
Validation Output
Description
Resolve
Select Packs
Details
OK
Cancel
Elements to select
Project
Project: rccbasic
Target 1
Source Group 1
CMSIS
Device
stm32f4xx_hal.c (STM32Cube HAL:Common)
stm32f4xx_hal_cortex.c (STM32Cube HAL:Cortex)
stm32f4xx_hal_gpio.c (STM32Cube HAL:GPIO)
stm32f4xx_hal_pwr.c (STM32Cube HAL:PWR)
stm32f4xx_hal_pwr_ex.c (STM32Cube HAL:PWR)
stm32f4xx_hal_rcc.c (STM32Cube HAL:RCC)
stm32f4xx_hal_rcc_ex.c (STM32Cube HAL:RCC)
RTE_Device.h (STM32Cube Framework:Classic)
startup_stm32f429xx.s (Startup)
stm32f4xx_hal_conf.h (STM32Cube Framework:Classic)
system_stm32f4xx.c (Startup)

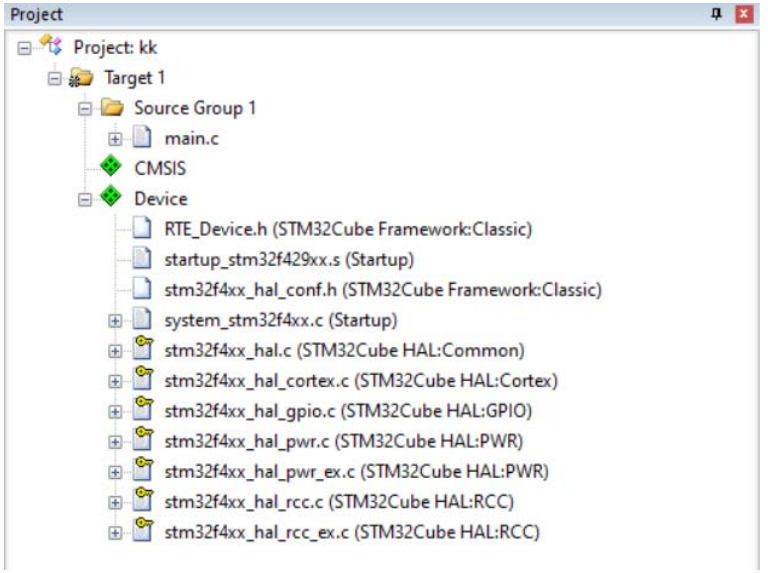
Reset handler (in ARM assembly)

- See file startup_stm32f429xx.s
- After a hardware reset the Cortex Microprocessor starts the execution of the Reset Handler. The address of this handler is stored at address 0x00000004
- It calls first to "SystemInit" function and then calls "__main" (the C main() function)
- SystemInit is defined in "system_stm32f4xx.c" file (startup package)

```
; Reset handler
Reset_Handler PROC
              EXPORT Reset_Handler [WEAK]
    IMPORT SystemInit
    IMPORT __main
        LDR R0, =SystemInit
        BLX R0 LDR R0, =__main
        BX R0
        ENDP
```

inicializa el sistema (proporcionado por ARM)

se manda el programa a main

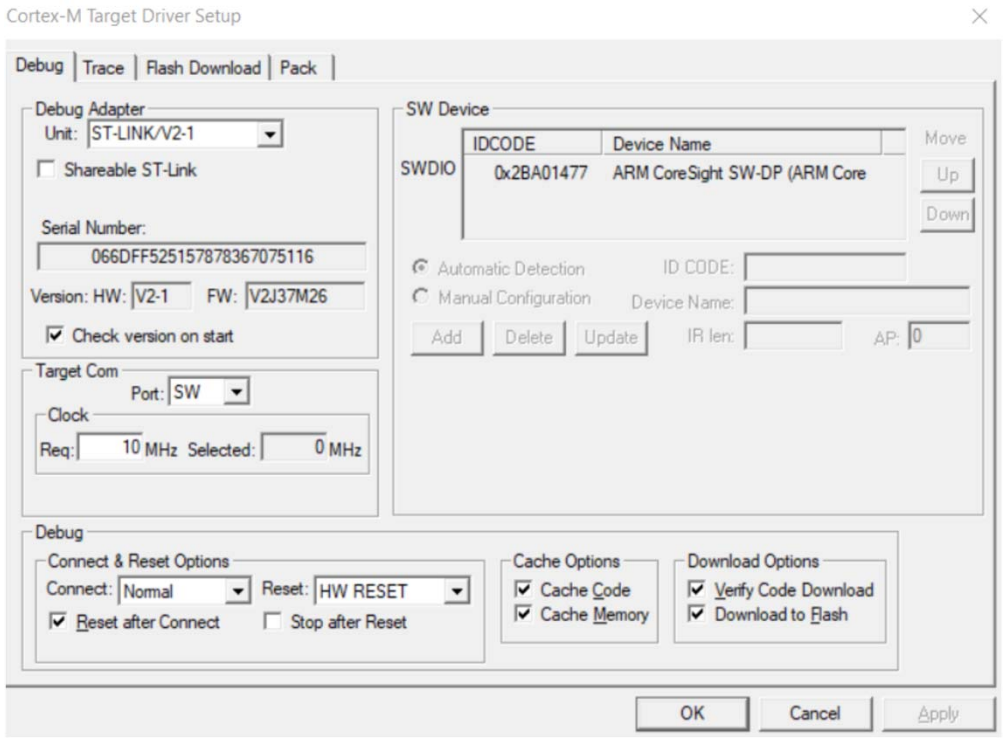


Reset Handler

- During the execution of the reset handler
  - The processor is using de internal clock 16 MHz
  - The different PINs are in the default state
- The user/developer has to configure the processor and the peripherals. This should be done in the main function
  - the first function to call in the main is “HAL_Init()” to gain access to the HAL Library

(Keil) Debug Connect Options

- Normal: stops CPU at the current executed instructions after connecting
- With Pre-reset: applies a hardware reset before connecting the device
- Under Reset: holds the hardware reset active while connecting to the device
- Without stop: connects and disconnects without explicitly stopping the CPU



(Keil) Debug Reset options

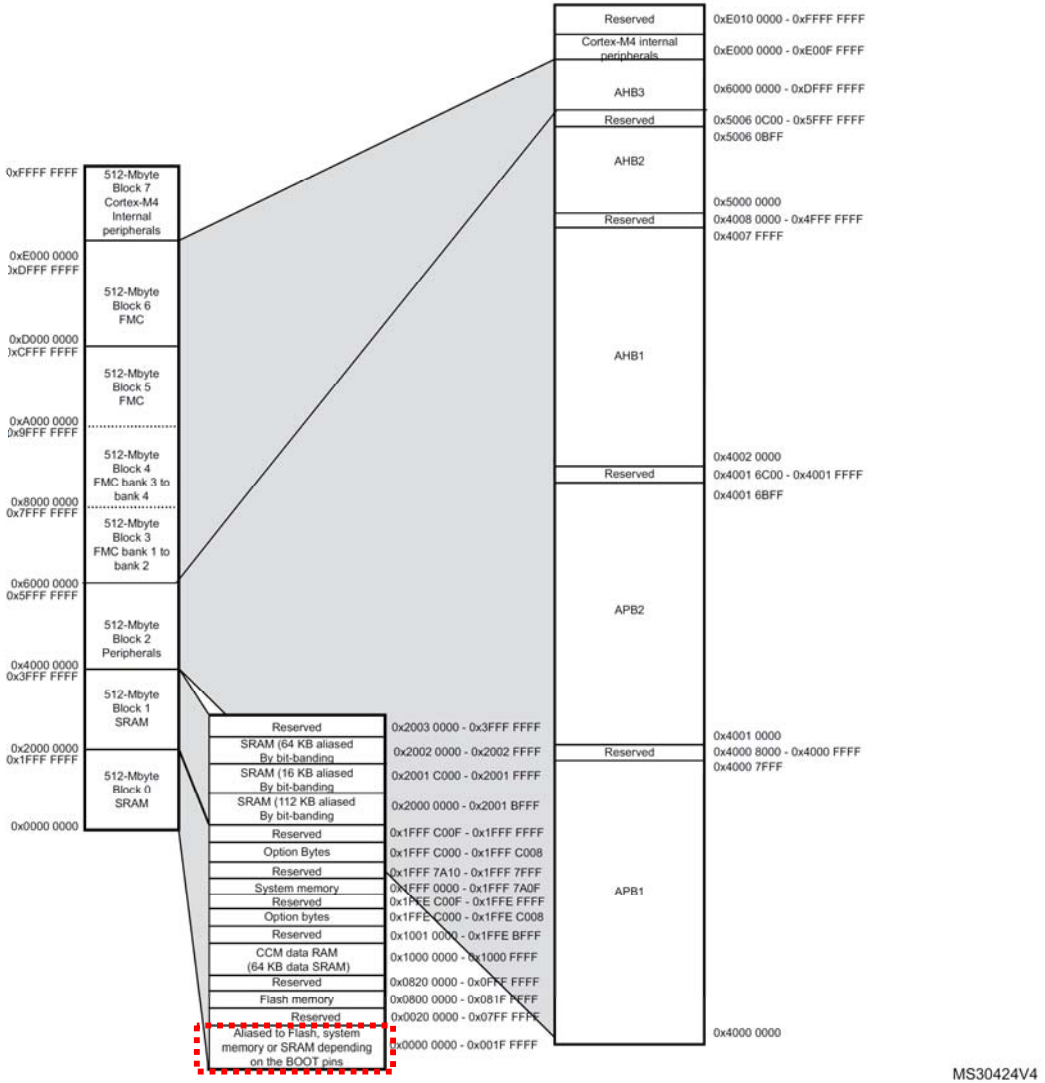
- Reset after connect: enables or disables the operation selected in the Reset drop-down list
- HW Reset: asserts the hardware reset signal
- SYSRESETREQ: performs a software reset (Cortex M and peripherals are reset)
- VECTRESET: Set the VECTRESET bit and only the cortex M is reset

Memory MAP and boot mode

Table 2. Boot modes

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space

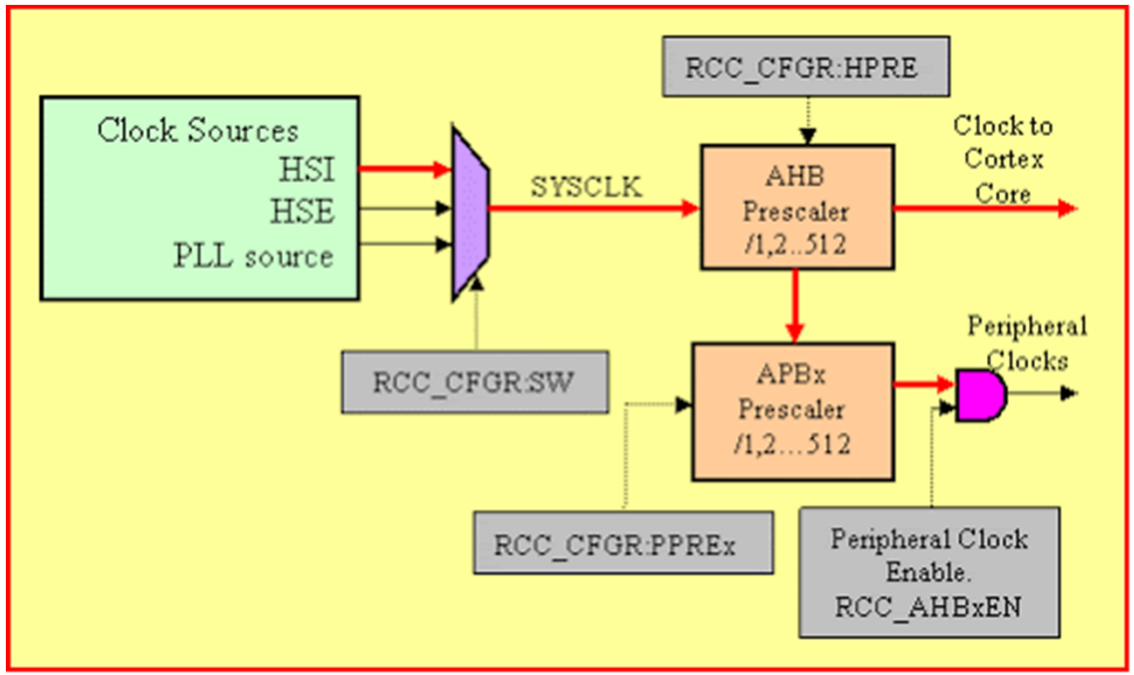
- NUCLEO 144:
  Default BOOT0=0



MS30424V4

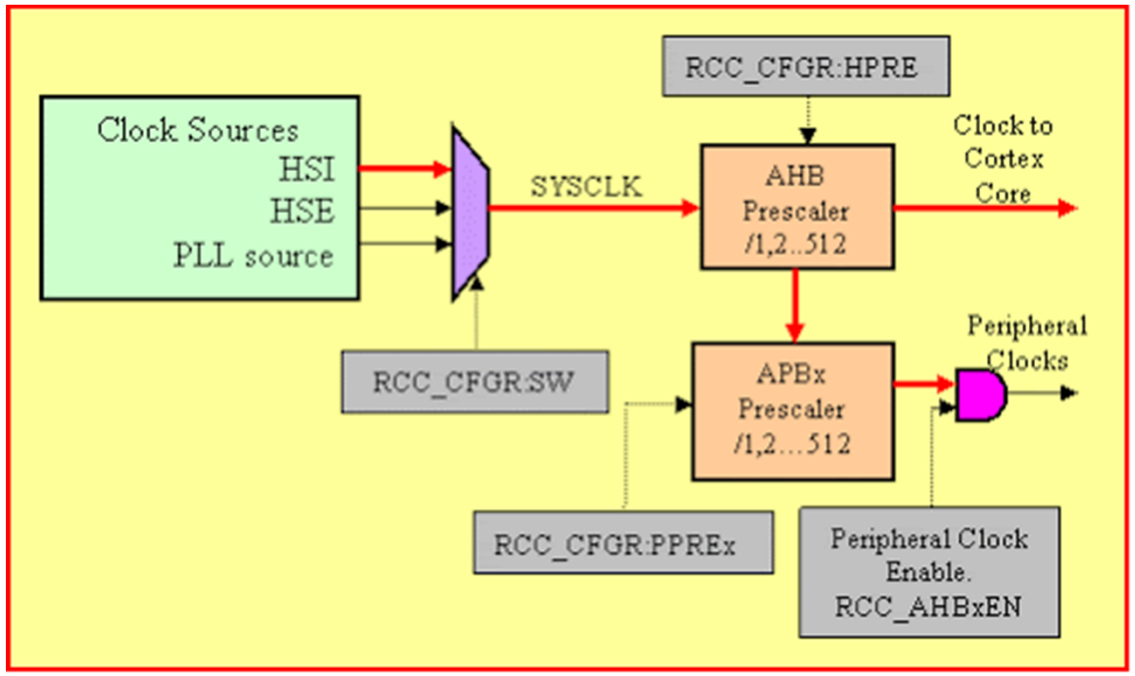
RCC: Clock Control

Basic clock circuitry for the STM32Fxxx



- The STM32F microcontroller system clock can come from one of three sources:
  - The high-speed internal clock (HSI)
  - The high-speed external clock(HSE)
  - The phase locked loop (PLL) clock -> multiplicar el reloj
- The RCC_CR (CLOCK CONTROL) and RCC_CFGR (CLOCK CONFIGURATION) registers are used to select and enable the clock source

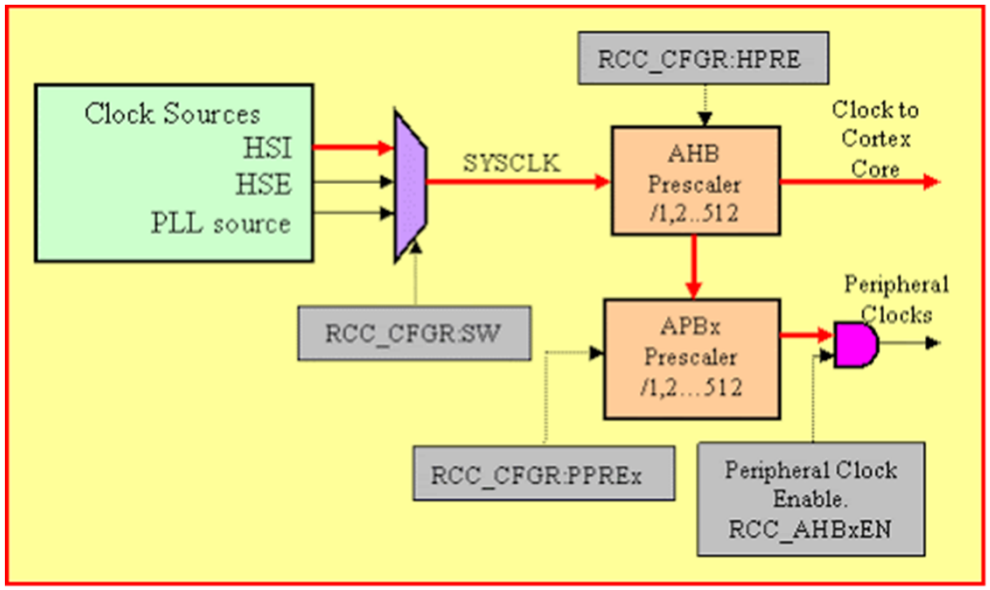
Basic clock circuitry for the STM32Fxxx



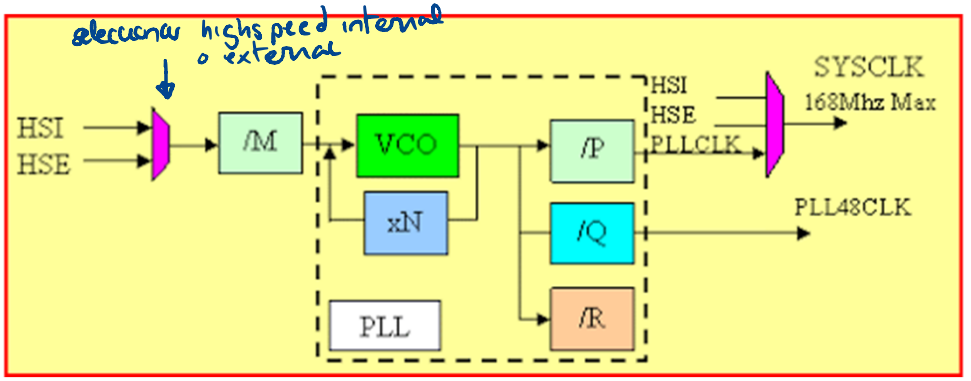
- After resetting the HSI (High-speed internal clock) is enabled
- HSE (High-speed external clock)

Basic clock circuitry for the STM32Fxxx

- Bus prescalers and peripheral clocks

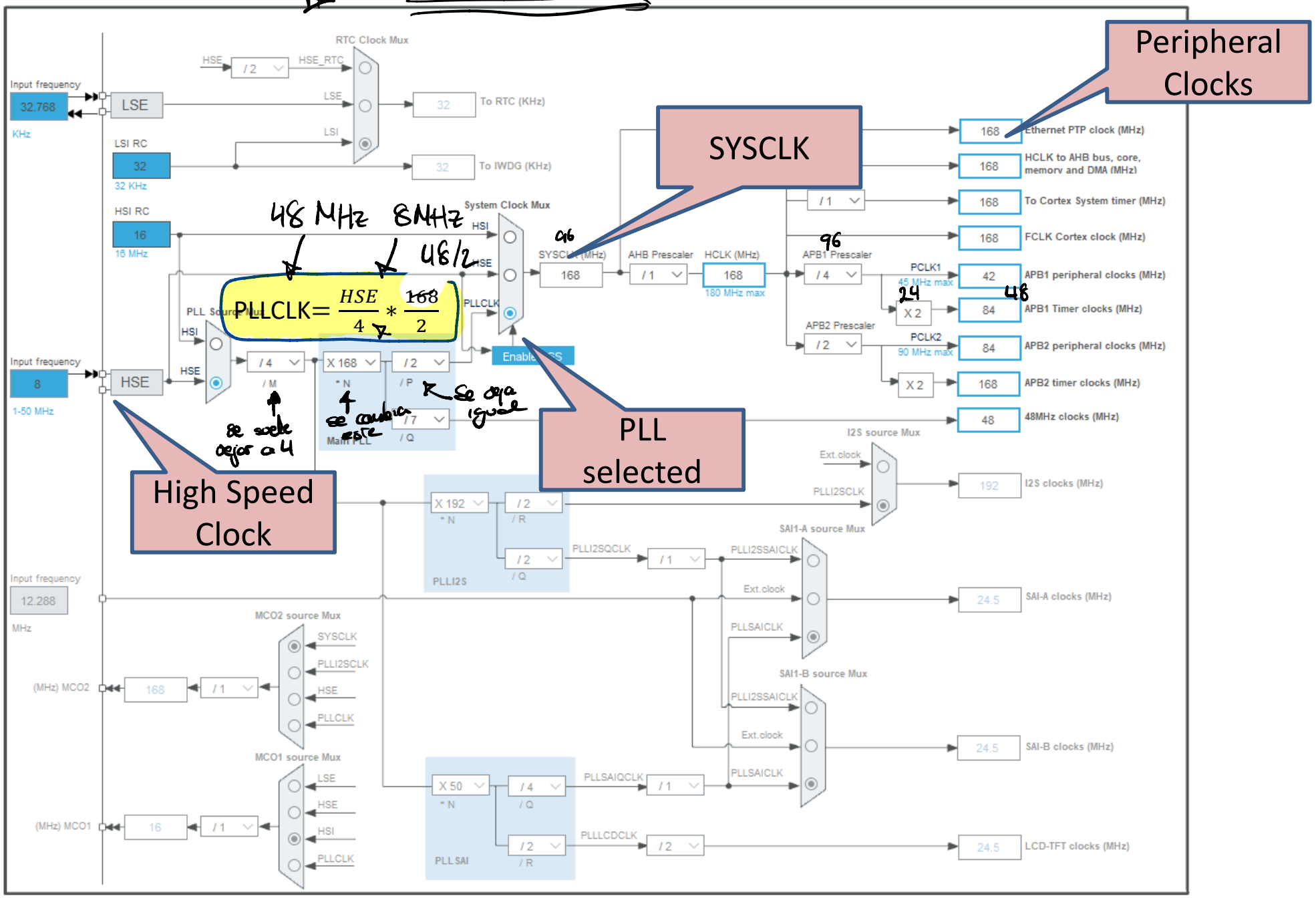


- Using the Phase Locked loop



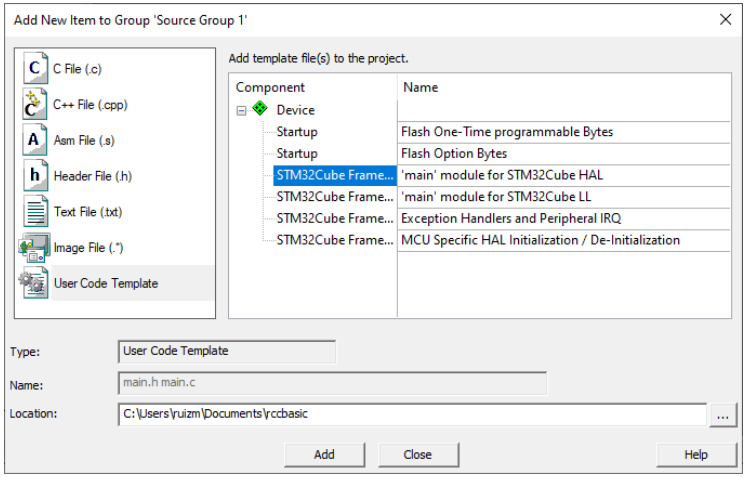
Calcular frecuencia de salida en función de los valores de xN y /P

STM32 LPX


Peripheral Clocks
SYSCLK
PLL selected
High Speed Clock
RTC Clock Mux
Input frequency
32.768
KHz
LSE
LSI RC
32
32 KHz
HSI RC
16
16 MHz
HSE
/ 2
HSE_RTC
LSE
LSI
To RTC (KHz)
To IWDG (KHz)
System Clock Mux
HSI
HSE
PLLCLK
SYSCLK (MHz)
168
AHB Prescaler
/ 1
HCLK (MHz)
168
180 MHz max
APB1 Prescaler
/ 4
PCLK1
45 MHz max
X 2
APB2 Prescaler
/ 2
PCLK2
90 MHz max
X 2
PLL Source Mux
/ 4
/ M
X 168
* N
/ 2
/ P
/ 7
/ Q
Main PLL
Input frequency
8
1-50 MHz
HSE
Enable CSS
168
Ethernet PTP clock (MHz)
168
HCLK to AHB bus, core, memory and DMA (MHz)
168
To Cortex System timer (MHz)
168
FCLK Cortex clock (MHz)
42
APB1 peripheral clocks (MHz)
84
APB1 Timer clocks (MHz)
84
APB2 peripheral clocks (MHz)
168
APB2 timer clocks (MHz)
48
48MHz clocks (MHz)
I2S source Mux
Ext.clock
PLLI2SCLK
192
I2S clocks (MHz)
X 192
* N
/ 2
/ R
/ 2
/ Q
PLLI2S
PLLI2SQCLK
/ 1
SAI1-A source Mux
PLLI2SSAICLK
Ext.clock
PLLSAICLK
24.5
SAI-A clocks (MHz)
SAI1-B source Mux
PLLI2SSAICLK
Ext.clock
PLLSAICLK
24.5
SAI-B clocks (MHz)
Input frequency
12.288
MHz
MCO2 source Mux
SYSCLK
PLLI2SCLK
HSE
PLLCLK
(MHz) MCO2
168
/ 1
MCO1 source Mux
LSE
HSE
HSI
PLLCLK
(MHz) MCO1
16
/ 1
X 50
* N
/ 4
/ Q
/ 2
/ R
PLL SAI
PLLSAIQCLK
/ 1
PLLLCDCLK
/ 2
24.5
LCD-TFT clocks (MHz)
48 MHz
8MHz
48/2
96
96
24
48
168
168
se coge igual
se queda asi
se cambia este

In the Keil Microvision Project

- In “Source Group 1”->Add New Item (User Code Template)
- Select Device->’main’ module for STM32Cube HAL-> Add

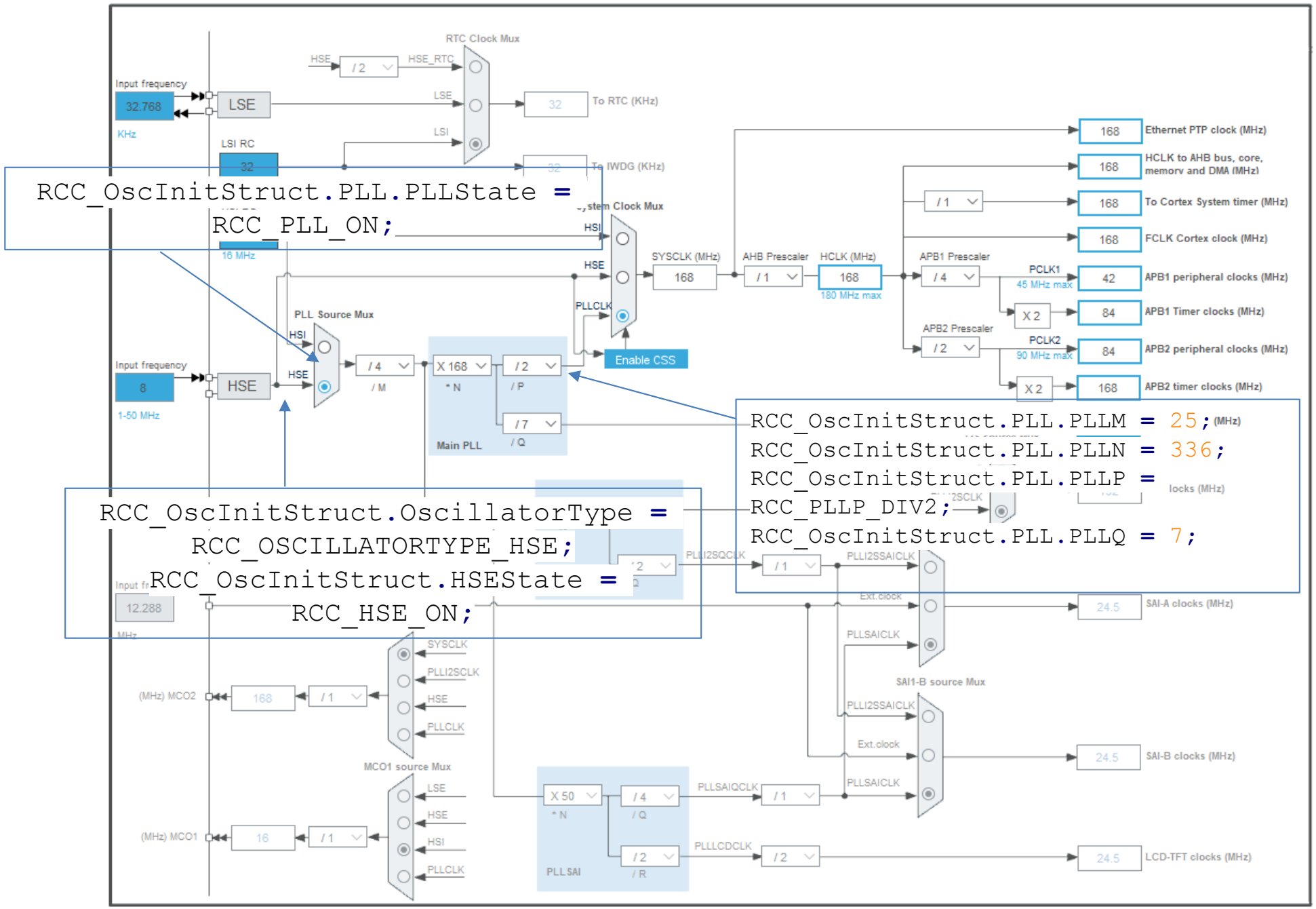


- Display the content of “main.c”
- Inspect the content of SystemClock_Config function

HSE_VALUE = 8000000; ← definirlo para asignar el valor y que lo conozca el micro.

Clock configuration I

```
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;

  /* Enable Power Control clock */
  __HAL_RCC_PWR_CLK_ENABLE();
  ……..
  /* Enable HSE Oscillator and activate PLL with HSE as source
*/
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
```

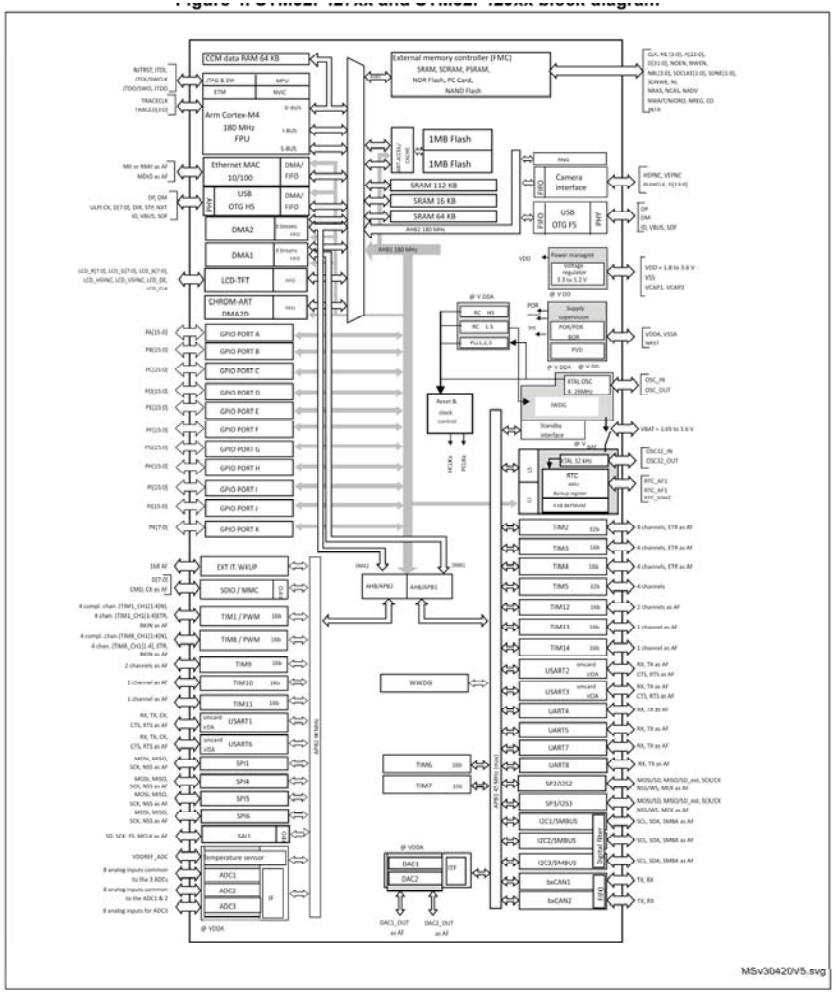

RCC_OscInitStruct.PLL.PLLState =
RCC_PLL_ON;
RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState =
RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLM = 25; (MHz)
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP =
RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
RTC Clock Mux
Input frequency
32.768
KHz
LSE
LSI RC
32
To RTC (KHz)
To IWDG (KHz)
HSE
/ 2
HSE_RTC
LSI
System Clock Mux
HSI
HSE
PLLCLK
SYSCLK (MHz)
168
AHB Prescaler
/ 1
HCLK (MHz)
168
180 MHz max
Enable CSS
PLL Source Mux
Input frequency
8
1-50 MHz
/ 4
/ M
X 168
* N
/ 2
/ P
/ 7
/ Q
Main PLL
APB1 Prescaler
/ 4
PCLK1
45 MHz max
X 2
APB2 Prescaler
/ 2
PCLK2
90 MHz max
168
Ethernet PTP clock (MHz)
HCLK to AHB bus, core, memory and DMA (MHz)
To Cortex System timer (MHz)
FCLK Cortex clock (MHz)
42
APB1 peripheral clocks (MHz)
84
APB1 Timer clocks (MHz)
84
APB2 peripheral clocks (MHz)
APB2 timer clocks (MHz)
Input frequency
12.288
MHz
PLLI2SQCLK
PLLI2SSAICLK
Ext.clock
PLLSAICLK
24.5
SAI-A clocks (MHz)
SAI1-B source Mux
SAI-B clocks (MHz)
MCO2
(MHz) MCO2
SYSCLK
PLLI2SCLK
HSE
PLLCLK
MCO1 source Mux
MCO1
(MHz) MCO1
LSE
HSI
X 50
PLLSAIQCLK
PLLLCDCLK
PLL SAI
/ R
LCD-TFT clocks (MHz)

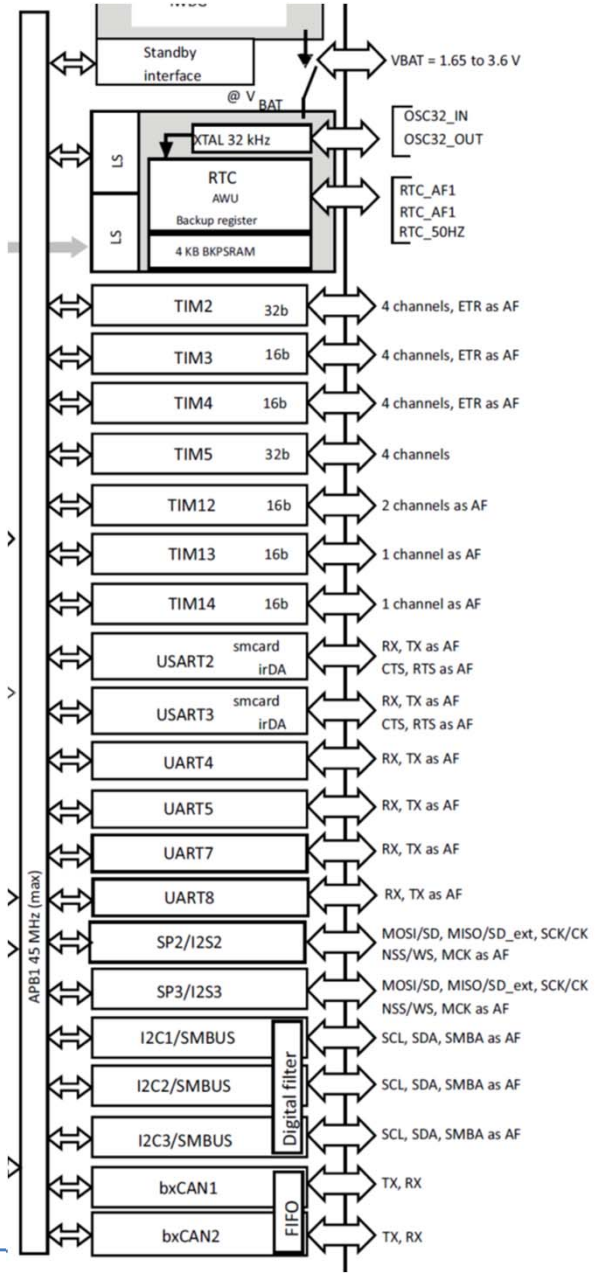
```
  if(HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
     clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  if(HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }

..
}
```

La variable SystemCoreClock contiene el valor del reloj del sistema

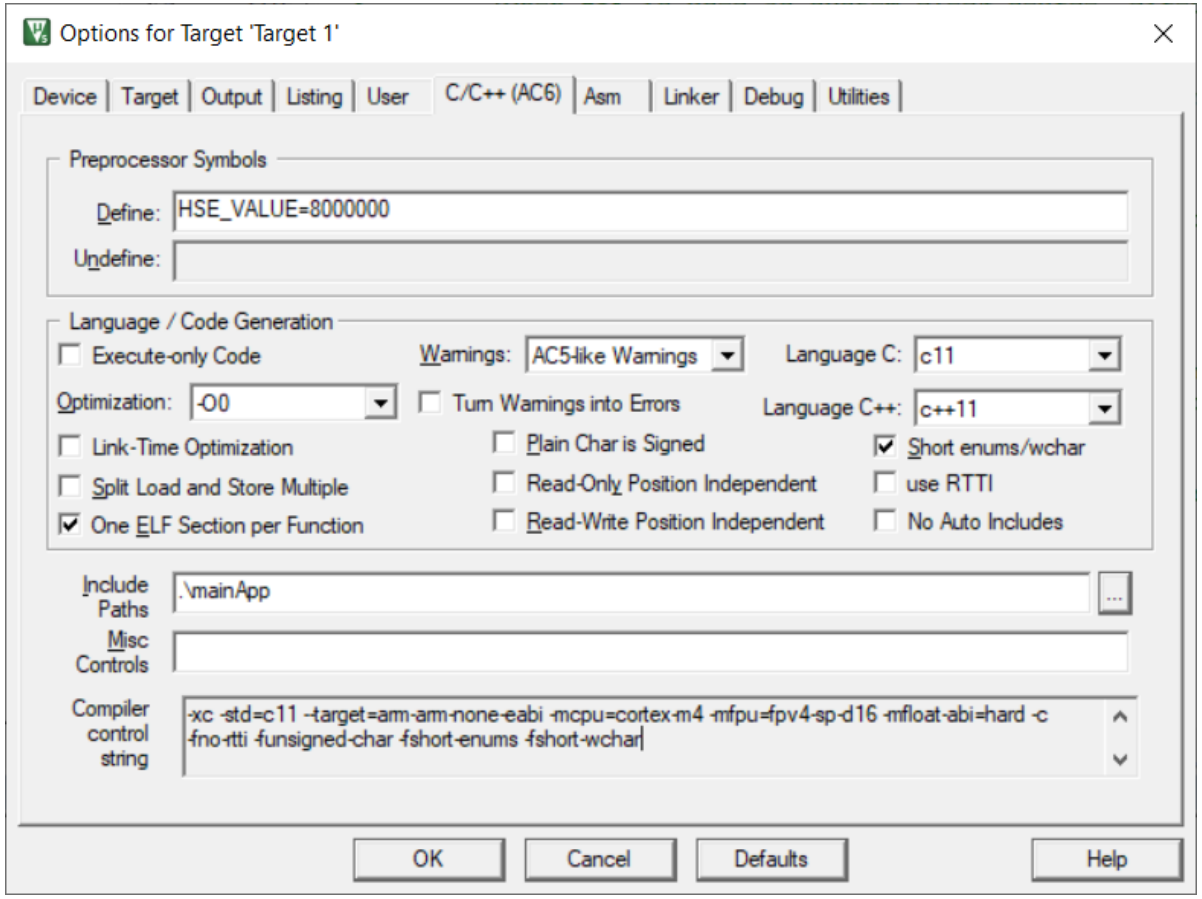
Clock distribution for peripherals and other HW elements

- 32f429 Datasheet



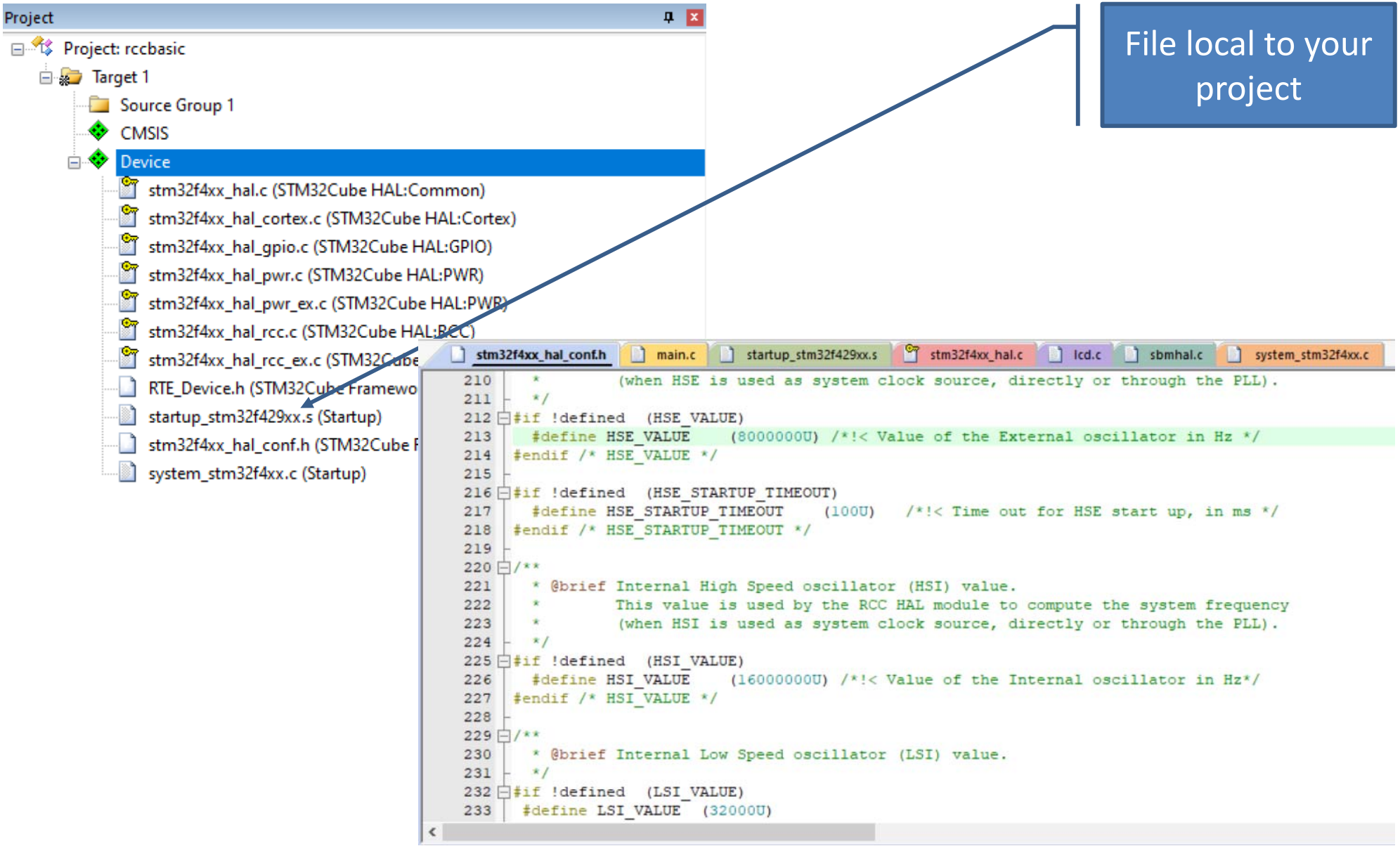


Defining the HSE_VALUE in Microvision



- This is equivalent to:
  - #define HSE_VALUE 8000000

HAL parameters “customization”

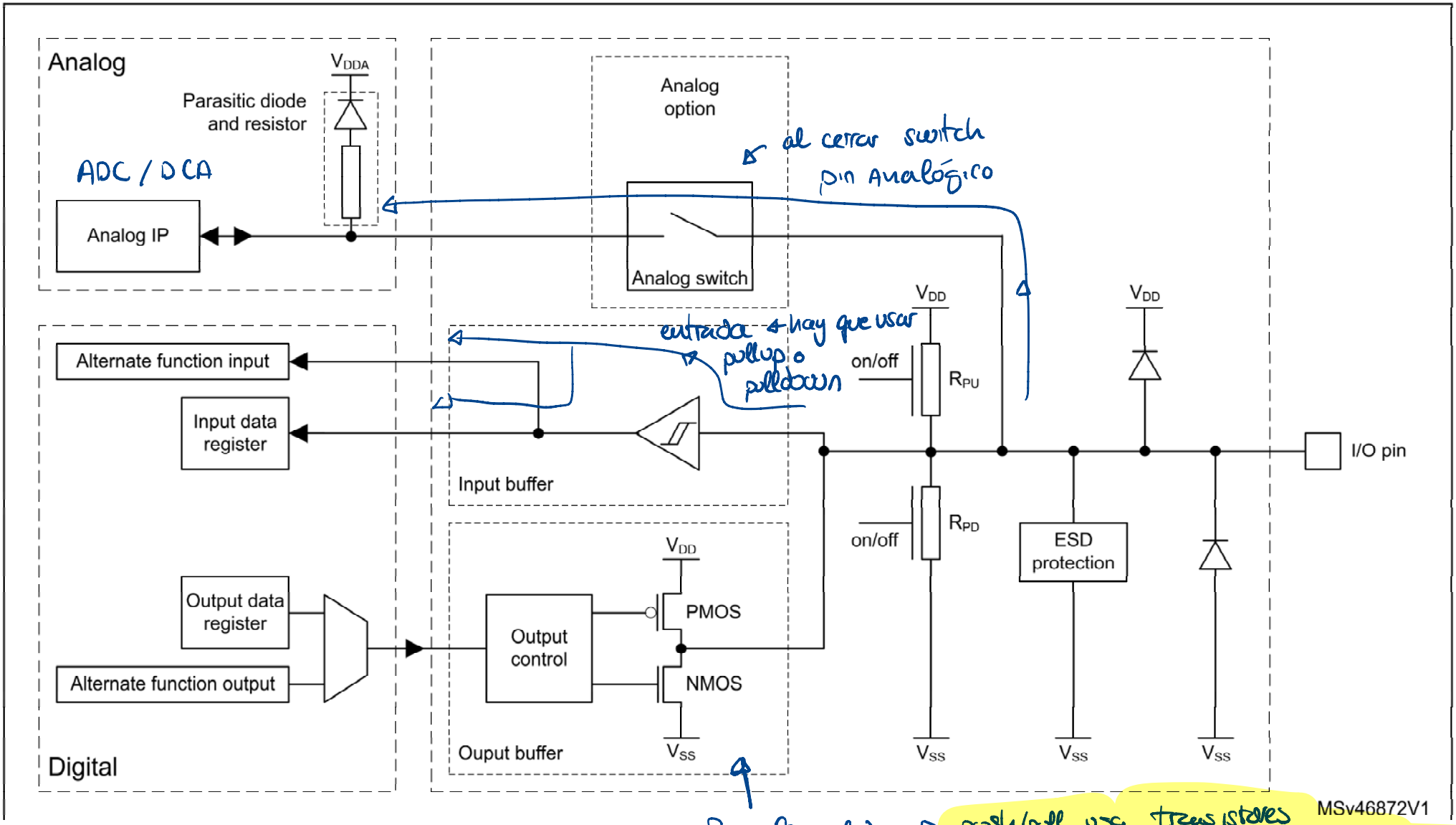


General Purpose Input/Output (GPIO)

Multifunction Pins

- A pin in the microcontroller can have multiple functions:
  - Digital I/O pin (input or output)
  - Specific peripheral function pin (for example the output of a hardware timer or an input clock signal) (Alternate function) ← nombre en STM
  - Analog input to connect to an ADC or DAC
  - The pin can be configured in Open Drain, Pull-up or pull-down, high speed or low speed.

Multifunction PINs



Configuration (using registers)

a.- Configuration process

1.- GPIO port mode register (GPIOx_MODER)
2.- GPIO port output type register (GPIOx_OTYPER)
3.- GPIO port output speed register (GPIOx_OSPEEDR)
4.- GPIO port pull-up/pull-down register (GPIOx_PUPDR)

MODER(i) [1:0]	OTYPER(i)	OSPEEDR(i) [B:A]	PUPDR(i) [1:0]	I/O configuration	
01	0	SPEED [B:A]	0 0	GP output	PP
	0		0 1	GP output	PP + PU
	0		1 0	GP output	PP + PD
	0		1 1	Reserved	
	1		0 0	GP output	OD
	1		0 1	GP output	OD + PU
	1		1 0	GP output	OD + PD
	1		1 1	Reserved (GP output OD)	
10	0	SPEED [B:A]	0 0	AF	PP
	0		0 1	AF	PP + PU
	0		1 0	AF	PP + PD
	0		1 1	Reserved	
	1		0 0	AF	OD
	1		0 1	AF	OD + PU
	1		1 0	AF	OD + PD
	1		1 1	Reserved	
00	x	x x	0 0	Input	Floating
	x	x x	0 1	Input	PU
	x	x x	1 0	Input	PD
	x	x x	1 1	Reserved (input floating)	
11	x	x x	0 0	Input/output	Analog
	x	x x	0 1	Reserved	
	x	x x	1 0		
	x	x x	1 1		

Source: ST. RM0090 Reference manual

Direct Programming of GPIO ← no se hace

This code shows how manage I/O access using pointers.

```
volatile uint32_t *pGPIOA_MODER = 0;
volatile uint32_t *pGPIOA_ODR = 0;

pGPIOA_MODER = (uint32_t*)0x48000000; // Address of the GPIOA MODER register
pGPIOA_ODR = (uint32_t*)(0x48000000 + 0x14); // Address of the GPIOA ODR register

// Before use a peripheral, it must be enabled and connected to the AHB1 bus
// This will be done using the HAL

*pGPIOA_MODER = *pGPIOA_MODER | 0x04; // Sets MODER[3:2] = 0x1 and configure like Output
*pGPIOA_ODR = *pGPIOA_ODR | 0x02; // Sets PA1 high
```

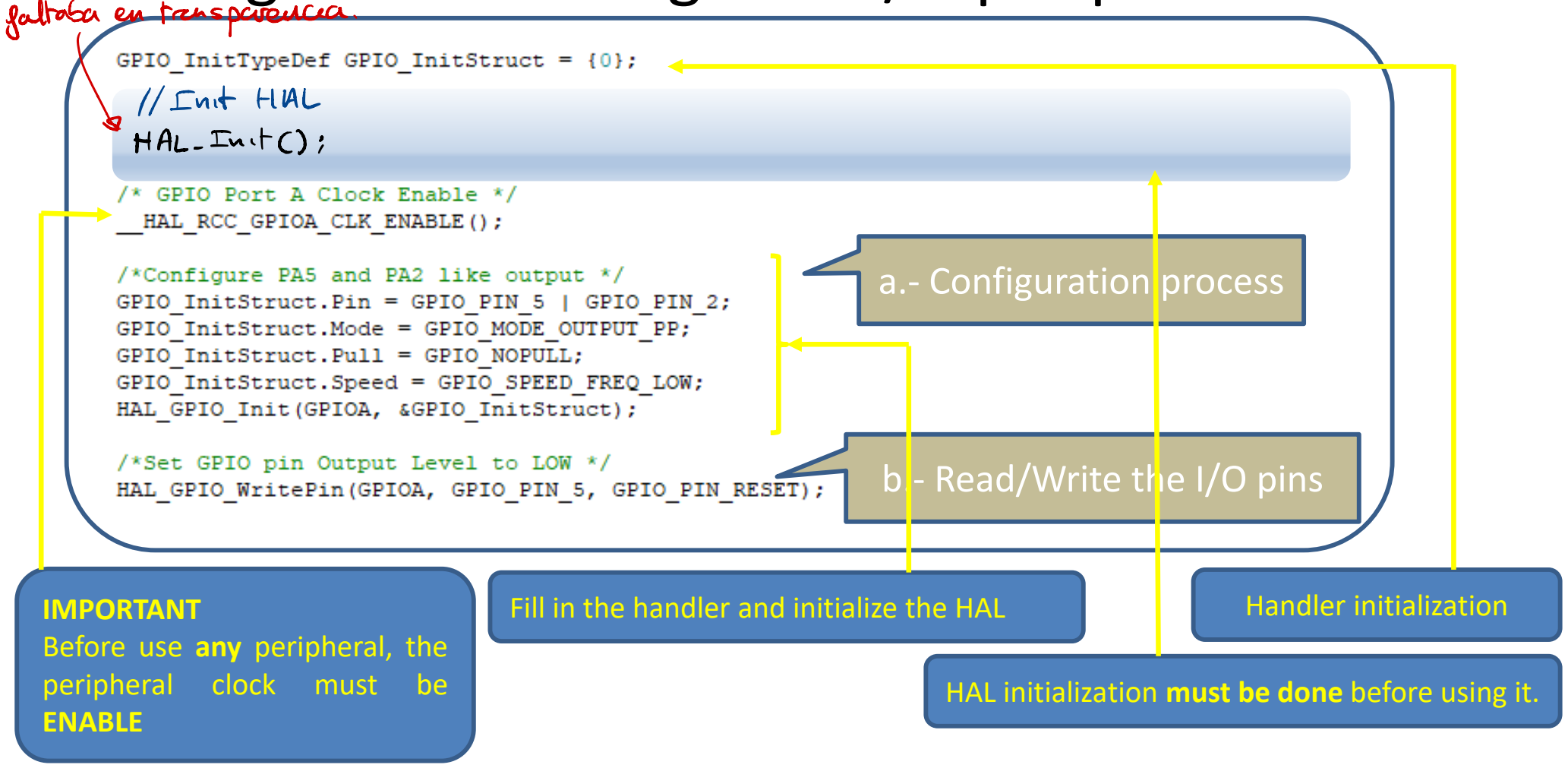
Pointer declaration and initialization

Assign pointer specific peripheral address

Access peripheral:
a.- Configure pin.
b.- Access pin.

Programming GPIO

- Using HAL to manage the I/O peripheral.



HAL GPIOFunctions and Handlers

- Initialization and de-initialization

```
void              HAL_GPIO_Init(GPIO_TypeDef  *GPIOx, GPIO_InitTypeDef *GPIO_Init);
void              HAL_GPIO_DeInit(GPIO_TypeDef  *GPIOx, uint32_t GPIO_Pin);
```

```
GPIO_PinState     HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void              HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);
void              HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void              HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin);
void              HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
```

See: stm32l4xx_hal_gpio.h

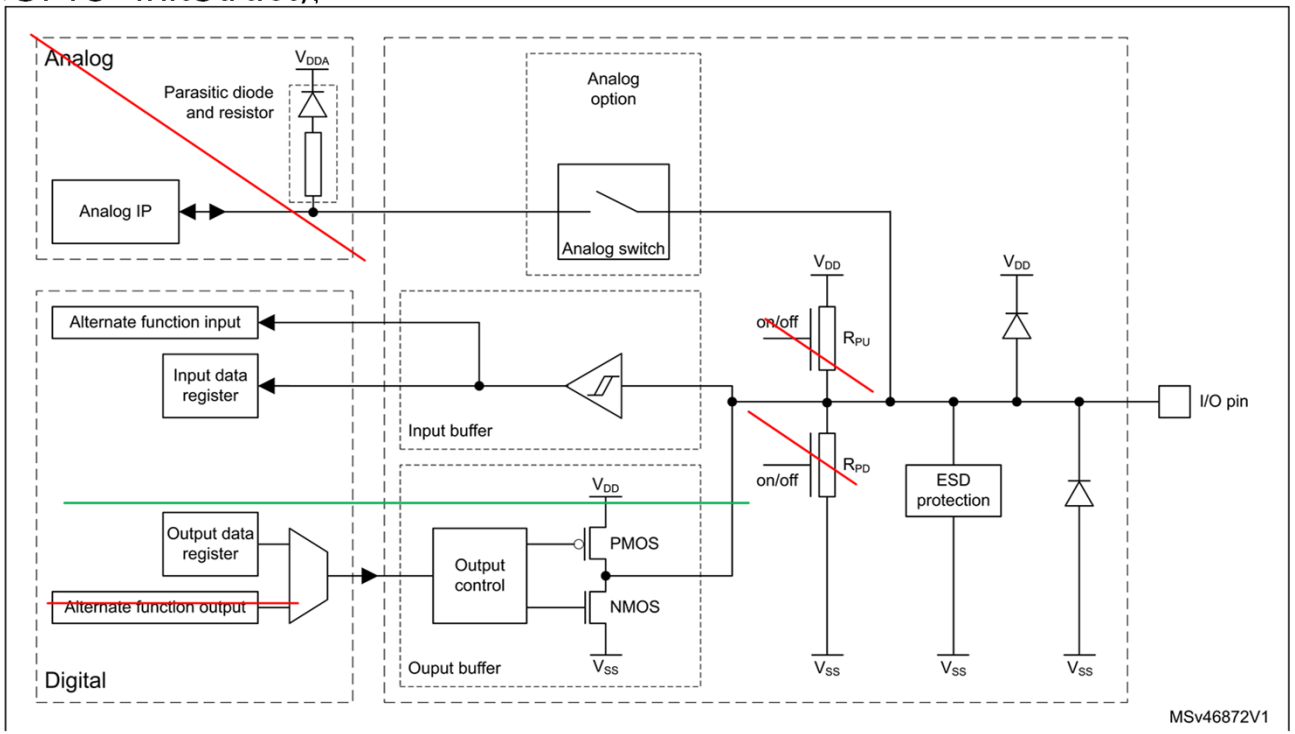
```
typedef struct {
        volatile uint32_t MODER;
        volatile uint32_t OTYPER;
        volatile uint32_t OSPEEDR;
        volatile uint32_t PUPDR;
        volatile uint32_t IDR;
        volatile uint32_t ODR;
        volatile uint32_t BSRR;
        volatile uint32_t LCKR;
        volatile uint32_t AFR[2];
        volatile uint32_t BRR;
} GPIO_TypeDef;
```

```
typedef struct {
        uint32_t Pin;
        uint32_t Mode;
        uint32_t Pull;
        uint32_t Speed;
        uint32_t Alternate;
} GPIO_InitTypeDef;
```

```
typedef enum
{
  GPIO_PIN_RESET = 0U,
  GPIO_PIN_SET
}GPIO_PinState;
```

Examples (Digital output)

```
GPIO_InitTypeDef GPIO_InitStruct = {0};
__HAL_RCC_GPIOB_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```



Examples (digital input - interrupt)

```
GPIO_InitTypeDef GPIO_InitStruct = {0};
__HAL_RCC_GPIOF_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_6;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP; //GPIO_PULLDOWN //GPIO_NOPULL
HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
```

digital input

```
GPIO_InitTypeDef GPIO_InitStruct = {0};
__HAL_RCC_GPIOC_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

interrupt con subida flanco

Examples (Analog)

```
__HAL_RCC_GPIOA_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Examples (Alternate Function)

Table 12. STM32F427xx and STM32F429xx alternate function mapping

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 A	PA0	-	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	-	ETH_MII_CRS	-	-	-	EVENTOUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_RTS	UART4_RX	-	-	ETH_MII_RX_CLK/ETH_RMII_REF_CLK	-	-	-	EVENTOUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX	-	-	-	ETH_MDIO	-	-	-	EVENTOUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	-	USART2_RX	-	-	OTG_HS_ULPI_D0	ETH_MII_COL	-	-	LCD_B5	EVENTOUT
	PA4	-	-	-	-	-	SPI1_NSS	SPI3_NSS/I2S3_WS	USART2_CK	-	-	-	-	OTG_HS_	DCMI_	LCD_	EVENTOUT
	PA5	-	TIM2_CH1/TIM2_ETR	-	TIM8_CH1N	-	SPI1_SCK	-	-	-	-	-	-	-	-	-	-
	PA6	-	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	-	SPI1_MISO	-	-	-	-	-	-	-	-	-	-
	PA7	-	TIM1_CH1N	TIM3_CH2	TIM8_CH1N	-	SPI1_MOSI	-	-	-	-	-	-	-	-	-	-
	PA8	MCO1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_CK	-	-	-	-	-	-	-	-
	PA9	-	TIM1_CH2	-	-	I2C3_SMBA	-	-	USART1_TX	-	-	-	-	-	-	-	-
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_RX	-	-	ID	-	-	D1	-	TOUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_CTS	-	CAN1_RX	OTG_FS_DM	-	-	-	LCD_R4	EVENTOUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_RTS	-	CAN1_TX	OTG_FS_	-	-	-	LCD_R5	EVEN

para usar función alternativa

```
__HAL_RCC_GPIOA_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
__HAL_RCC_GPIOA_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Conclusions

- Some STM32 microcontroller pins can be configured for different functions
  - Use HAL to configure the PIN in the initialization of your system
  - Check the reference manual to see the Alternate Functions (to be used for timers, SPI, I2C, etc)
  - Review the HAL_GPIO code in Keil environment

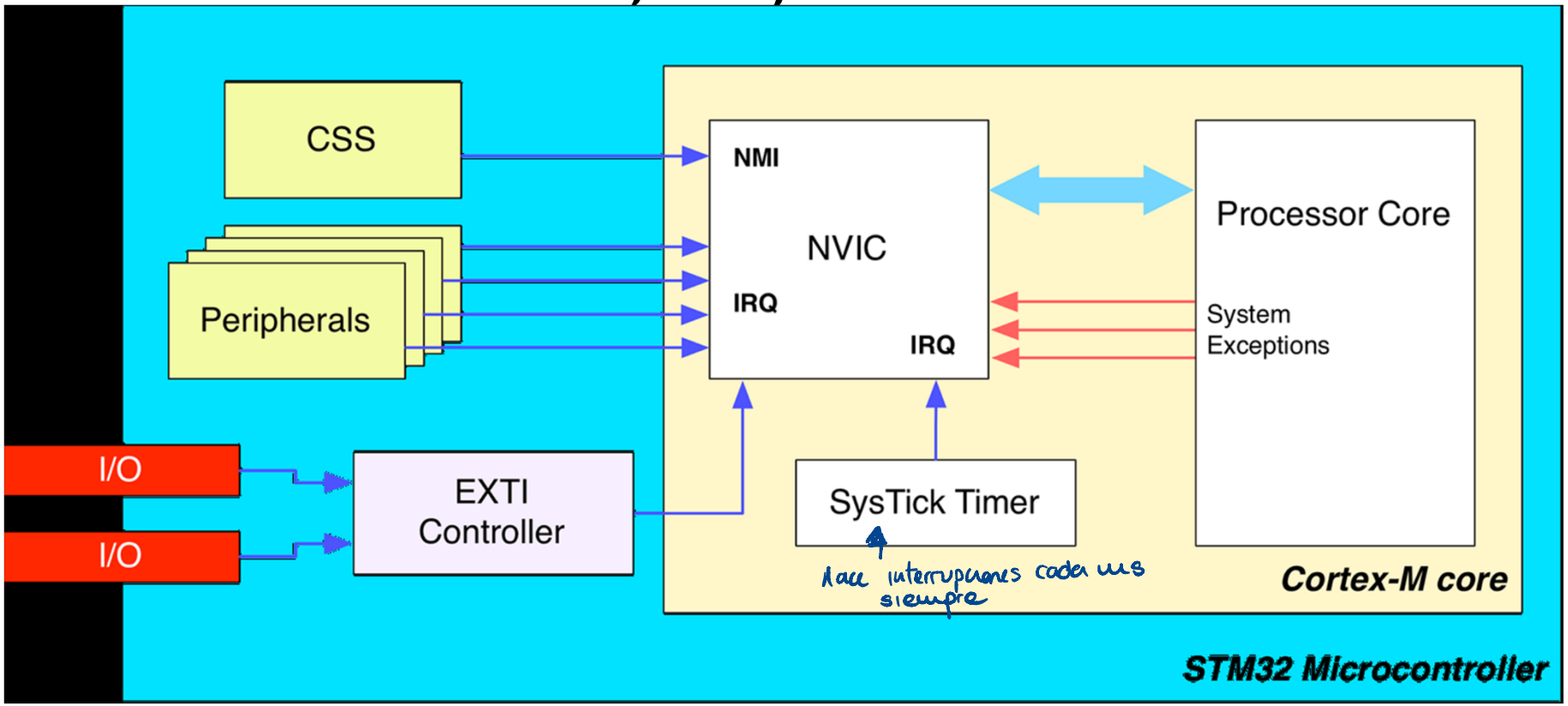
Interrupts

Interrupts

- Mechanism to handle asynchronous events.
- When an interrupt happens, the microcontroller does several actions:
  - Save the current software context and program counter
  - Jump to the Interrupt Service Routine
- When the ISR finishes, the software context is restored and continue with program execution
- The ARM Cortex-M family provides a unit to manage the interrupts. **(Nested Vectored Interrupt Controller, NVIC)**

Nested Vectored Interrupt Controller (NVIC)

- The NVIC manages the interrupts and the exceptions (from internal peripherals, from external GPIO lines, etc)



Source: Mastering STM32. Carmine Noviello

Nested Vectored Interrupt Controller (NVIC)

- The processor knows where to jump when an event happens using the vector table (ordered by priority)

Number	Exception type	Priority[a]	Function
1	Reset	-3	Reset
2	NMI	-2	Non-Maskable Interrupt
3	Hard Fault	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled.
4	Memory Management[c]	Configurable[b]	MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present.
5	Bus Fault[c]	Configurable	Pre-fetch fault, memory access fault, and other address/memory related.
6	Usage Fault[c]	Configurable	Usage fault, such as Undefined instruction executed or illegal state transition attempt.
7-10	-	-	RESERVED
11	SVCall	Configurable	System service call with SVC instruction.
12	Debug Monitor[c]	Configurable	Debug monitor – for software based debug.
13	-	-	RESERVED
14	PendSV	Configurable	Pending request for system service.
15	SysTick	Configurable	System tick timer has fired.
16-[47/240][d]	IRQ	Configurable	IRQ Input

Exceptions are managed like interrupts too

Source: Mastering STM32. Carmine Noviello

Nested Vectored Interrupt Controller (NVIC)

- The processor knows where to jump when an event happens using the vector table

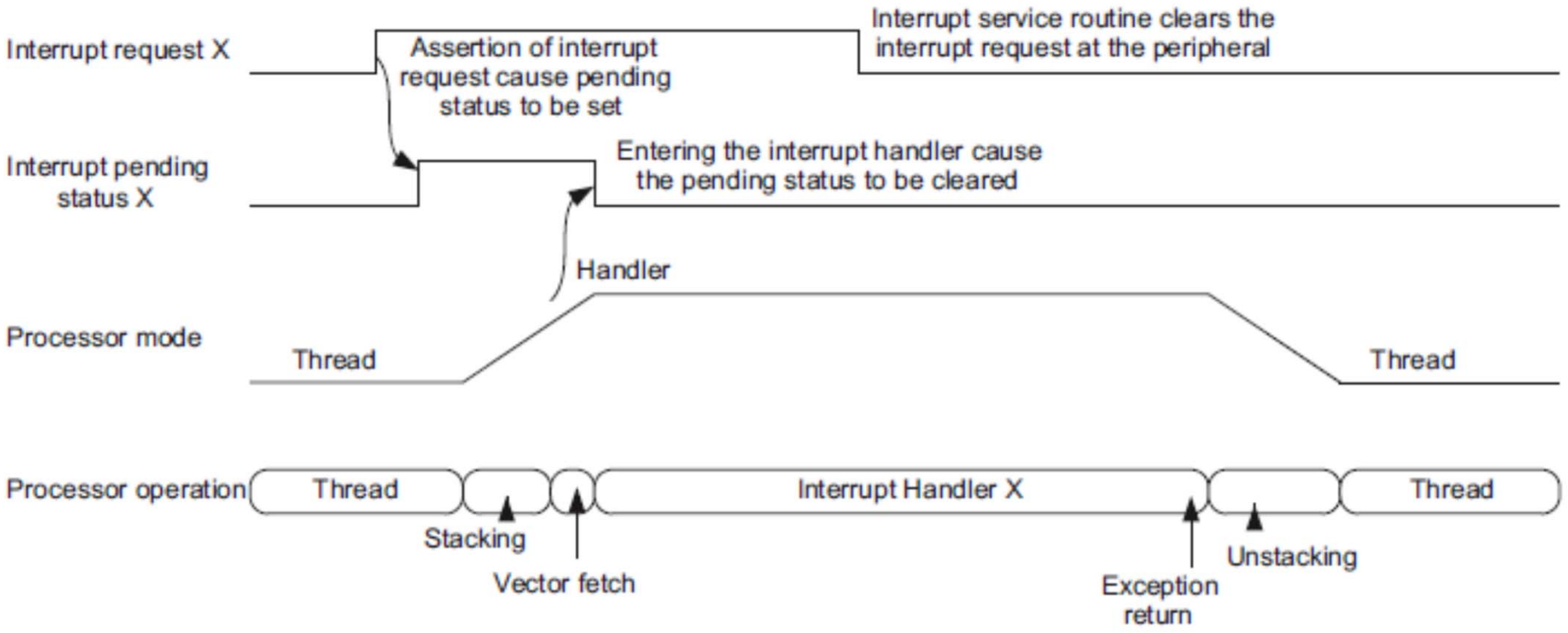
Number	Exception type	Priority[a]	Function
1	Reset	-3	Reset
2	NMI	-2	Non-Maskable Interrupt
3	Hard Fault	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled.
4	Memory Management[c]	Configurable[b]	MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present.
5	Bus Fault[c]	Configurable	Pre-fetch fa related.
6	Usage Fault[c]	Configurable	Usage fau transition a
7-10	-	-	RESERVED
11	SVCall	Configurable	System serv
12	Debug Monitor[c]	Configurable	Debug mon
13	-	-	RESERVED
14	PendSV	Configurable	Pending req
15	SysTick	Configurable	System tick
16-[47/240][d]	IRQ	Configurable	IRQ Input

startup_stm32f429xx.s file

```
; Vector Table Mapped to Address 0 at Reset
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  __Vectors_End
                EXPORT  __Vectors_Size

__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     MemManage_Handler         ; MPU Fault Handler
                DCD     BusFault_Handler          ; Bus Fault Handler
                DCD     UsageFault_Handler        ; Usage Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     DebugMon_Handler          ; Debug Monitor Handler
                DCD     0                         ; Reserved
                DCD     PendSV_Handler            ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler

                ; External Interrupts
                DCD     WWDG_IRQHandler                   ; Window WatchDog
                DCD     PVD_IRQHandler                    ; PVD through EXTI Line detection
                DCD     TAMP_STAMP_IRQHandler             ; Tamper and TimeStamps through the EXTI line
                DCD     RTC_WKUP_IRQHandler               ; RTC Wakeup through the EXTI line
```

Interrupt Lifecycle

- Interrupt lifecycle example



- The ISR routine must fulfil this criterium:
  - Reduce execution time avoiding loops and intensive operations

Using Interrupts

- After reset, all interrupts are disable, except Reset, NMI and Hard Fault
- Enabling an interrupt IRQ using HAL:
  - **void** HAL_NVIC_EnableIRQ (IRQn_Type_ IRQn);

IRQn_Type is an enumerated defined in the stm32f429xx.h file

- Disabling an interrupt:
  - **void** HAL_NVIC_DisableIRQ (IRQn_Type_IRQn);

- Obviously the peripheral must be configured to generate interrupts

Using Interrupts

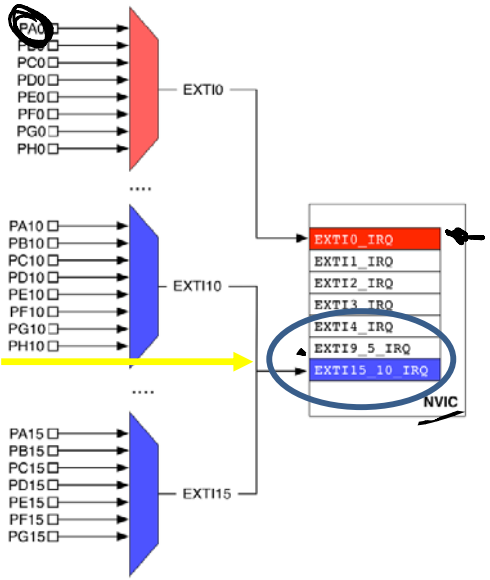
- An ISR **must** be defined
  - First, inside the ISR the associate pending bit **must** be cleared
  - Later the interrupt code is executed

Using Interrupts

This code shows how manage an interrupt for an external GPIO
A switch is connected to PIN 13 PORT C

```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
   __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
   // ISR Body;
}
```

1.- Enable Interrupt



PIN 13 PORT C

stm32f4xx_hal_cortex.c | stm32f4xx_hal_cortex.h | stm32f429xx.h | startup_stm32f429xx.s

```
 * @brief STM32F4XX Interrupt Number Definition, according to the selected device
 *        in @ref Library_configuration_section
 */
typedef enum
{
/******  Cortex-M4 Processor Exceptions Numbers ***********************************
  NonMaskableInt_IRQn        = -14,    /*!< 2 Non Maskable Interrupt
  MemoryManagement_IRQn      = -12,    /*!< 4 Cortex-M4 Memory Management Interrupt
  BusFault_IRQn              = -11,    /*!< 5 Cortex-M4 Bus Fault Interrupt
  UsageFault_IRQn            = -10,    /*!< 6 Cortex-M4 Usage Fault Interrupt
  SVCall_IRQn                = -5,     /*!< 11 Cortex-M4 SV Call Interrupt
  DebugMonitor_IRQn          = -4,     /*!< 12 Cortex-M4 Debug Monitor Interrupt
  PendSV_IRQn                = -2,     /*!< 14 Cortex-M4 Pend SV Interrupt
  SysTick_IRQn               = -1,     /*!< 15 Cortex-M4 System Tick Interrupt
/******  STM32 specific Interrupt Numbers *****************************************
  WWDG_IRQn                  = 0,      /*!< Window WatchDog Interrupt
  PVD_IRQn                   = 1,      /*!< PVD through EXTI Line detection Interrupt
                                 .....     .....
  USART1_IRQn                = 37,     /*!< USART1 global Interrupt
  USART2_IRQn                = 38,     /*!< USART2 global Interrupt
  USART3_IRQn                = 39,     /*!< USART3 global Interrupt
  EXTI15_10_IRQn             = 40,     /*!< External Line[15:10] Interrupts
  RTC_Alarm_IRQn             = 41,     /*!< RTC Alarm (A and B) through EXTI Line Interrupt
  OTG_FS_WKUP_IRQn           = 42,     /*!< USB OTG FS Wakeup through EXTI line interrupt

                                 .....     .....

  LTDC_IRQn                  = 88,     /*!< LTDC global Interrupt
  LTDC_ER_IRQn               = 89,     /*!< LTDC Error global Interrupt
  DMA2D_IRQn                 = 90      /*!< DMA2D global Interrupt
} IRQn_Type;
```

Using Interrupts

This code shows how manage interrupt for an external GPIO.
A switch is connected to PIN 13 PORT C

```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
   __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
   // ISR Body;
}
```

2.- Coding ISR

3.- **Clear** pending flag

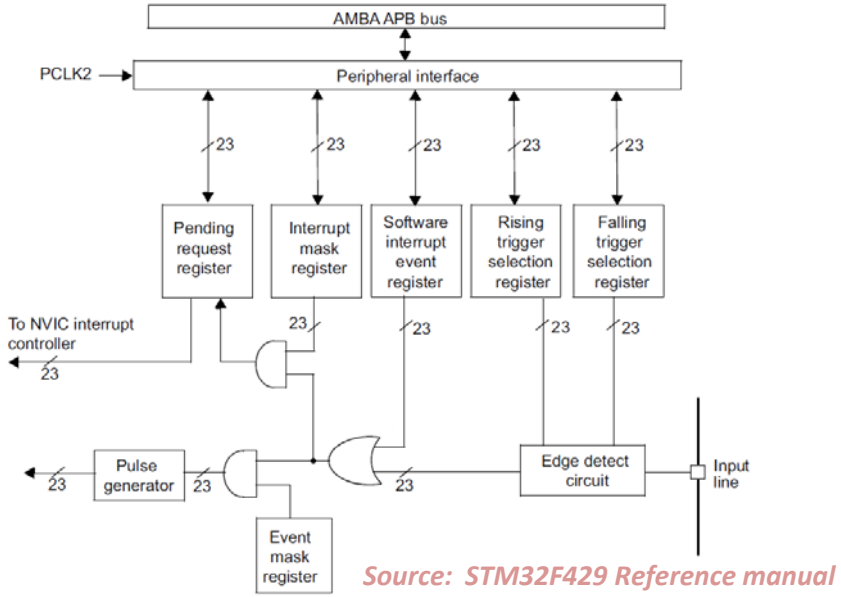
main.c* | stm32f4xx_hal_cortex.c | stm32f4xx_hal_cortex.h | stm32f429xx.h | startup_stm32f429xx.s

```
232
233  Default_Handler PROC
234
235                 EXPORT  WWDG_IRQHandler                 [WEAK]
236                 EXPORT  PVD_IRQHandler                  [WEAK]
237                 EXPORT  TAMP_STAMP_IRQHandler           [WEAK]
238                 EXPORT  RTC_WKUP_IRQHandler             [WEAK]
239                 EXPORT  FLASH_IRQHandler                [WEAK]
240                 EXPORT  RCC_IRQHandler                  [WEAK]
241                 EXPORT  EXTI0_IRQHandler                [WEAK]
242                 EXPORT  EXTI1_IRQHandler                [WEAK]
243                 EXPORT  EXTI2_IRQHandler                [WEAK]
244                 EXPORT  EXTI3_IRQHandler                [WEAK]
245                 EXPORT  EXTI4_IRQHandler                [WEAK]
246                 EXPORT  DMA1_Stream0_IRQHandler         [WEAK]
247                 EXPORT  DMA1_Stream1_IRQHandler         [WEAK]
248                 EXPORT  DMA1_Stream2_IRQHandler         [WEAK]
249                 EXPORT  DMA1_Stream3_IRQHandler         [WEAK]
250                 EXPORT  DMA1_Stream4_IRQHandler         [WEAK]
251                 EXPORT  DMA1_Stream5_IRQHandler         [WEAK]
252                 EXPORT  DMA1_Stream6_IRQHandler         [WEAK]
253                 EXPORT  ADC_IRQHandler                  [WEAK]
254                 EXPORT  CAN1_TX_IRQHandler              [WEAK]
255                 EXPORT  CAN1_RX0_IRQHandler             [WEAK]
256                 EXPORT  CAN1_RX1_IRQHandler             [WEAK]
257                 EXPORT  CAN1_SCE_IRQHandler             [WEAK]
258                 EXPORT  EXTI9_5_IRQHandler              [WEAK]
259                 EXPORT  TIM1_BRK_TIM9_IRQHandler        [WEAK]
260                 EXPORT  TIM1_UP_TIM10_IRQHandler        [WEAK]
261                 EXPORT  TIM1_TRG_COM_TIM11_IRQHandler   [WEAK]
                    .....   .....
273                 EXPORT  USART2_IRQHandler               [WEAK]
274                 EXPORT  USART3_IRQHandler               [WEAK]
                    EXPORT  EXTI15_10_IRQHandler            [WEAK]
276                 EXPORT  RTC_Alarm_IRQHandler            [WEAK]
277                 EXPORT  OTG_FS_WKUP_IRQHandler          [WEAK]
278                 EXPORT  TIM8_BRK_TIM12_IRQHandler       [WEAK]
```

Using Interrupts

This code shows how manage interrupt for an external GPIO.
A switch is connected to PIN 13 PORT C

```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
  // ISR Body;
}
```



Source: STM32F429 Reference manual

¡¡¡¡ IMPORTANT !!!!
Do not forget to configure the peripheral to work in interrupt MODE

Configure the peripheral before interrupts are used

```
  __HAL_RCC_GPIOC_CLK_ENABLE();

/*Configure GPIO pin : PC13 - USER BUTTON */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

See: stm32l4xx_hal_gpio.h

Using Interrupts

- HAL interrupt Model.

```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
  // ISR Body;
}
```

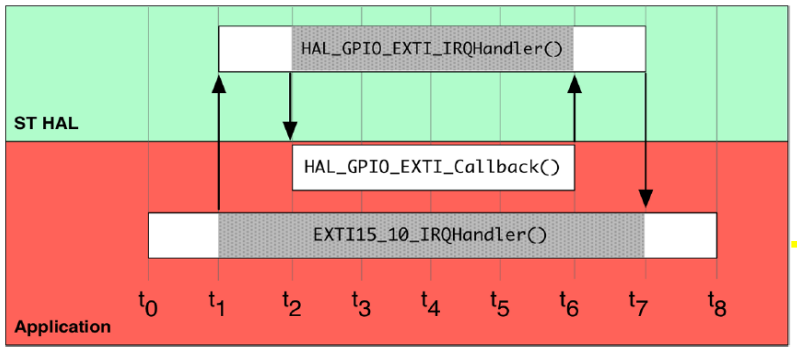
The peripheral interrupt flag is cleared

Use this method!

```
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);


void EXTI15_10_IRQHandler(void) {
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

    //ISR Body

}
```

HAL provides a higher degree of abstraction

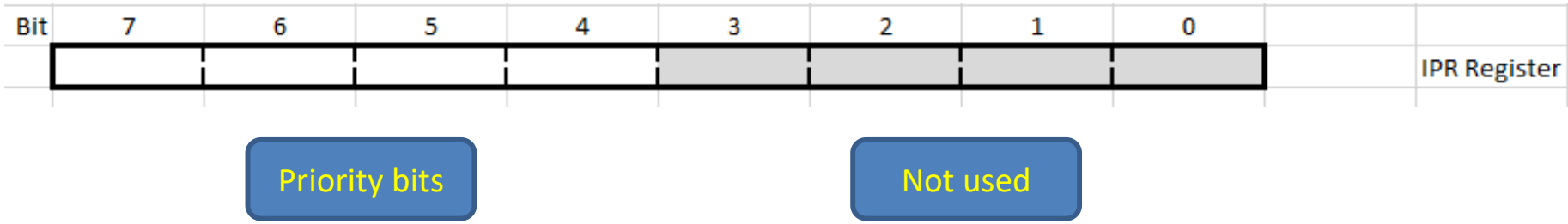
Look for the function in the file stm32F4xx_hal_YYY.c
With YYY for specific peripheral.



Source: Mastering STM32. Carmine Noviello

Interrupts Priority

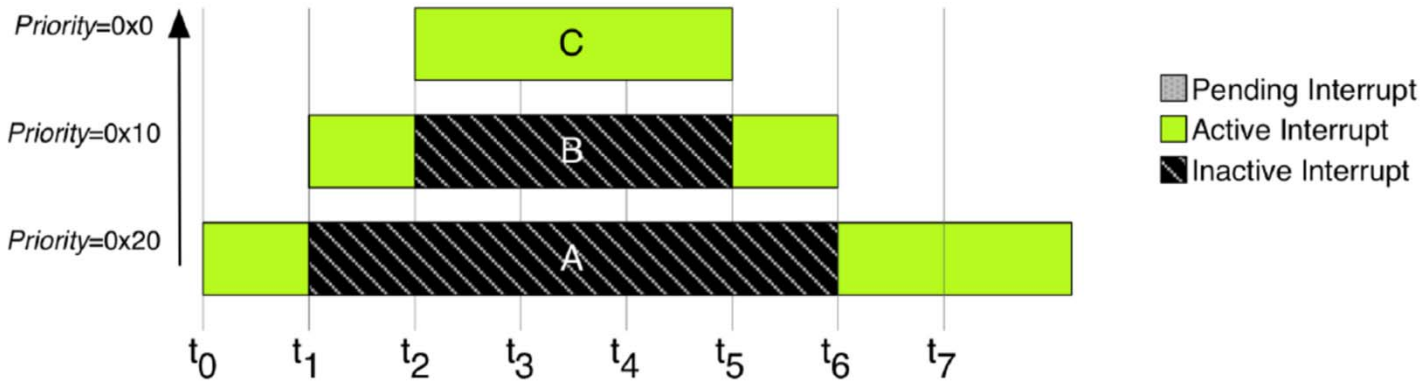
- ARM Cortex-M architecture has the capability to assign priority to interrupts
- Priority is handled using an eight-bit register (M3/M4/M7)



16 priority levels 0x00, 0x10, 0x20 ----- 0xD0, 0xE0, 0xF0.

Higher priority **Lower priority**

Preemption of interrupts (concept)



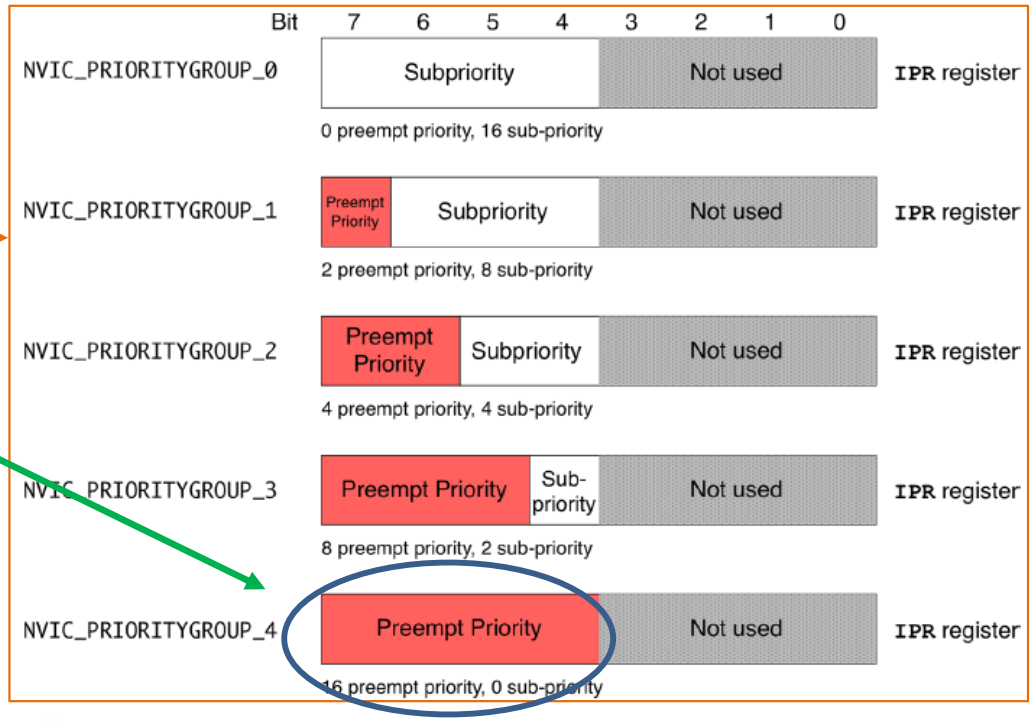
Interrupts Priority

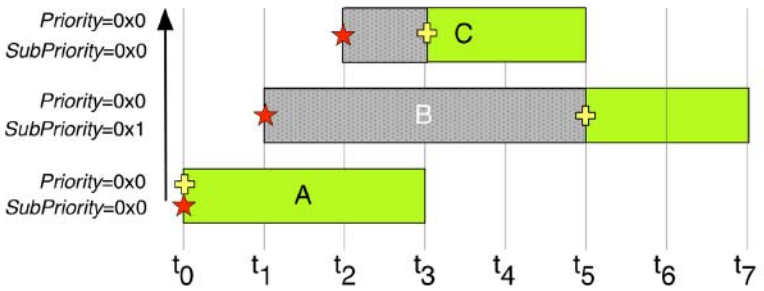
- Priority / Subpriority

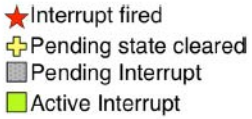
Scheme: Preemption Priority / Sub-priority
Five working groups can be configured

Bits AIRCR in System Control Block register let assign a Priority / Subpriority scheme.

Source: Mastering STM32. Carmine Noviello







HAL_Init() **function configures NVIC_PRIORITYGROUP_4**

```
void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup);
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority);
```

HAL Functions and Handlers

Initialization and de-initialization

```
void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup);
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority);
void HAL_NVIC_EnableIRQ(IRQn_Type IRQn);
void HAL_NVIC_DisableIRQ(IRQn_Type IRQn);
void HAL_NVIC_SystemReset(void);
```

Peripheral Control

```
uint32_t HAL_NVIC_GetPriorityGrouping(void);
void HAL_NVIC_GetPriority(IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t* pPreemptPriority, uint32_t* pSubPriority);
uint32_t HAL_NVIC_GetPendingIRQ(IRQn_Type IRQn);
void HAL_NVIC_SetPendingIRQ(IRQn_Type IRQn);
void HAL_NVIC_ClearPendingIRQ(IRQn_Type IRQn);
uint32_t HAL_NVIC_GetActive(IRQn_Type IRQn);
```

IRQn_Type see stm32f29xx.h file

PriorityGroup see stm32f4xx_hal_cortex.h

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

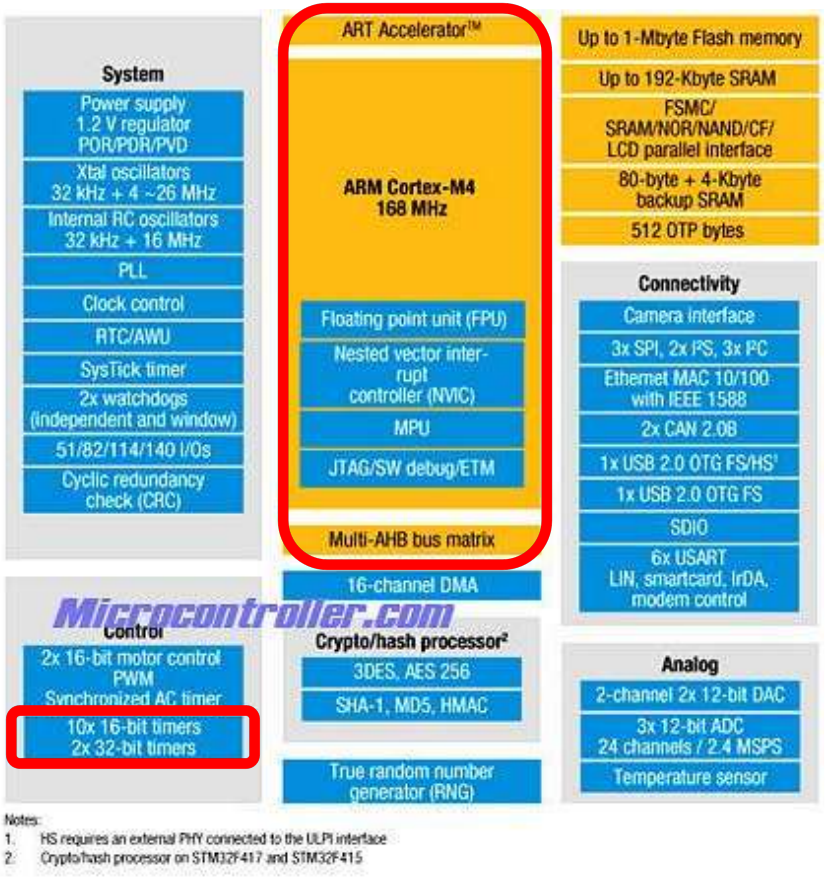
Timers. What are they for?

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.)

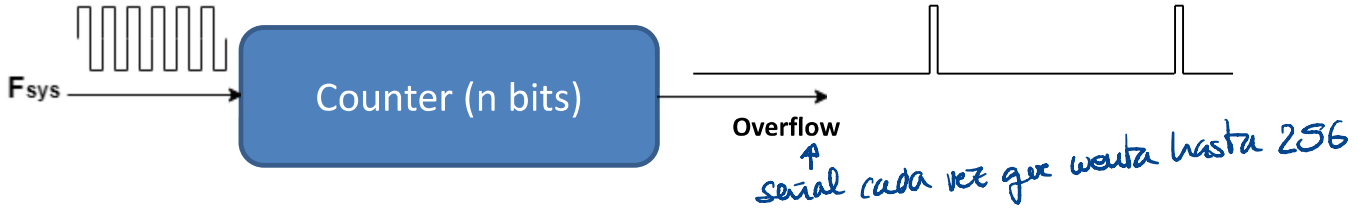
Timers in microcontrollers

- 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.

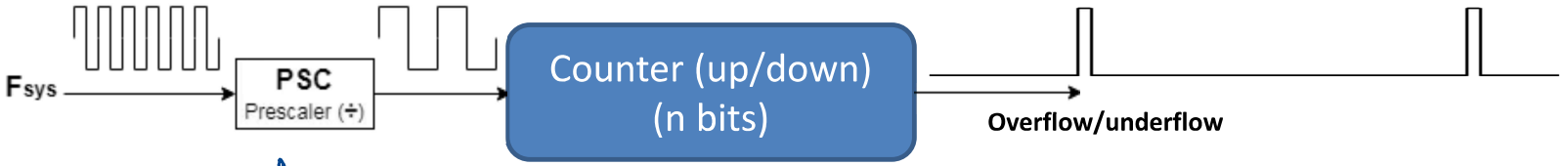
Example:

- n = 8 bits
- Fsys = 1000 Hz

→ Overflow every 256 (2^8) ms

Timers in microcontrollers

- Basic timer structure (with prescaler):



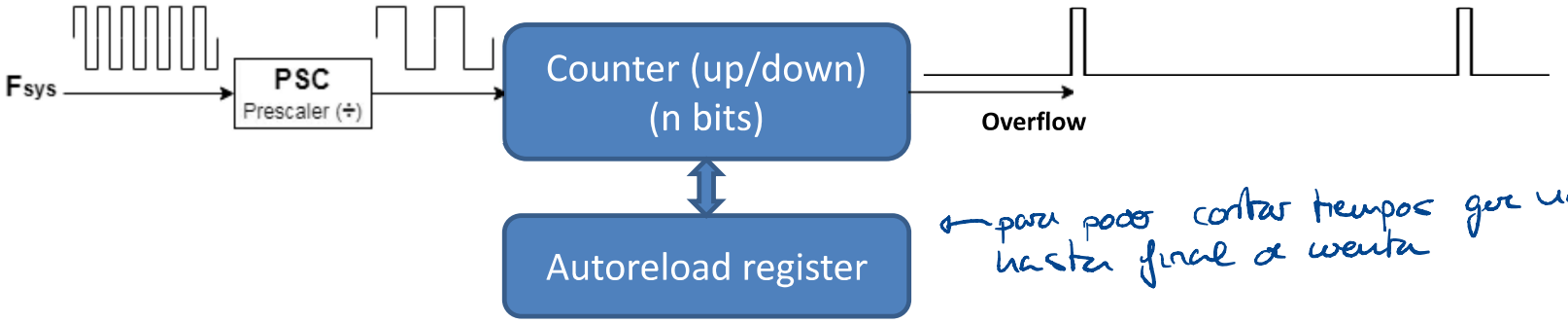
Usually, more functionalities are implemented in hardware, like a prescaler (PSC) to divide the input clock frequency by a selectable value.

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:

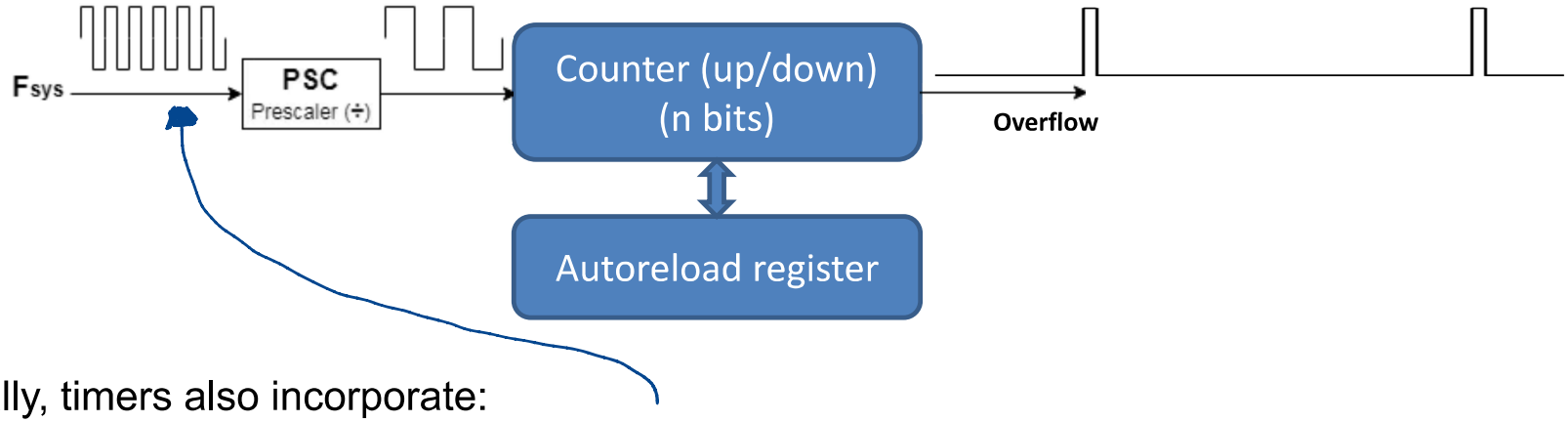
Fsys÷PSC → 250 Hz (input freq counter)

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 I2S 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**

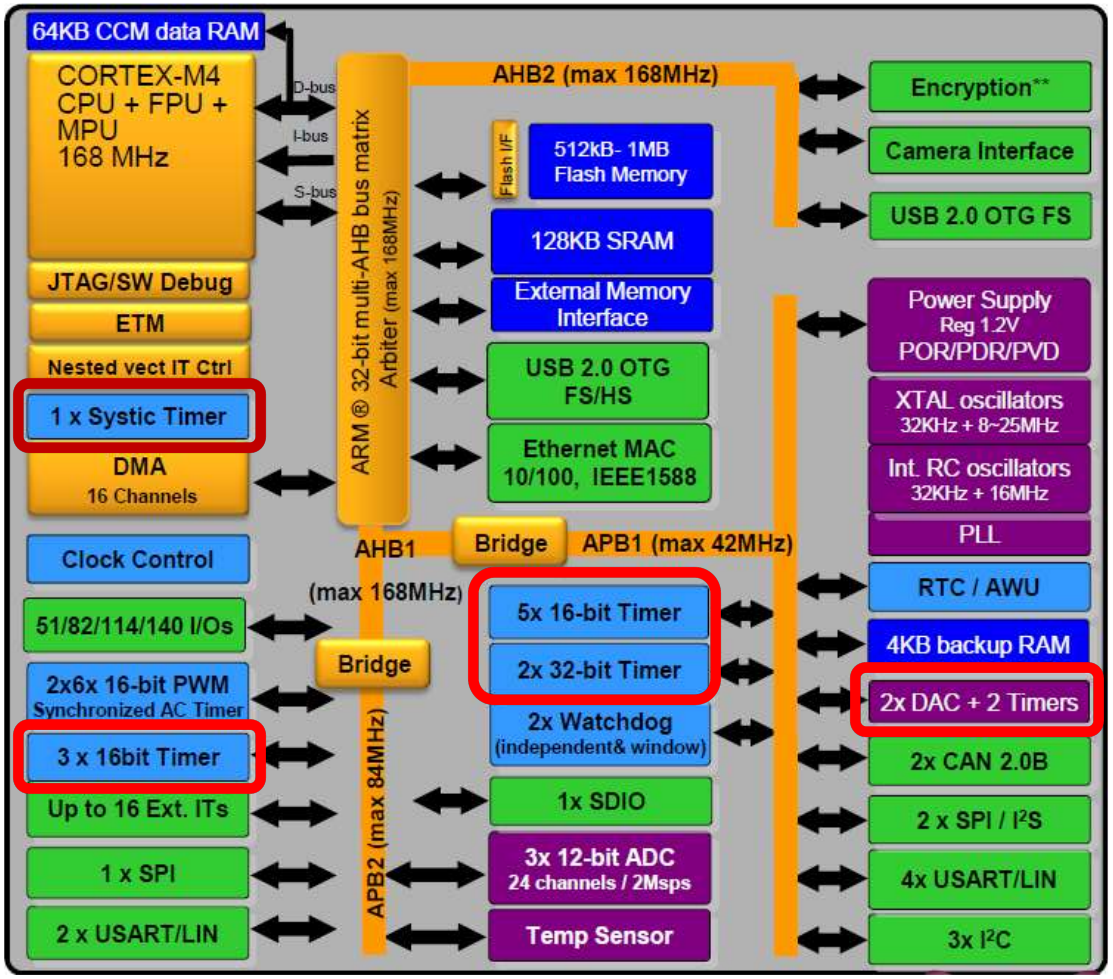
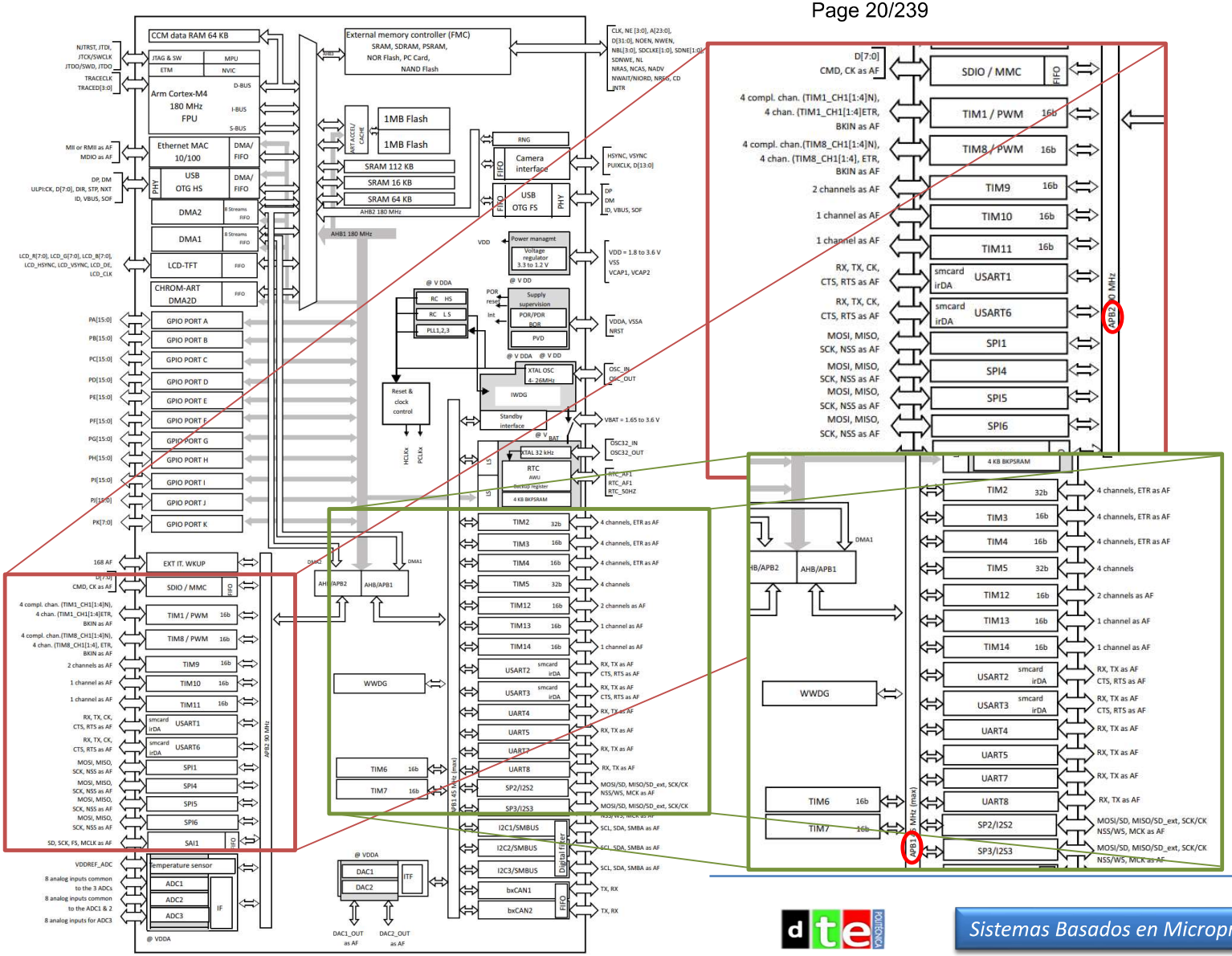


Figure 4. STM32F427xx and STM32F429xx block diagram

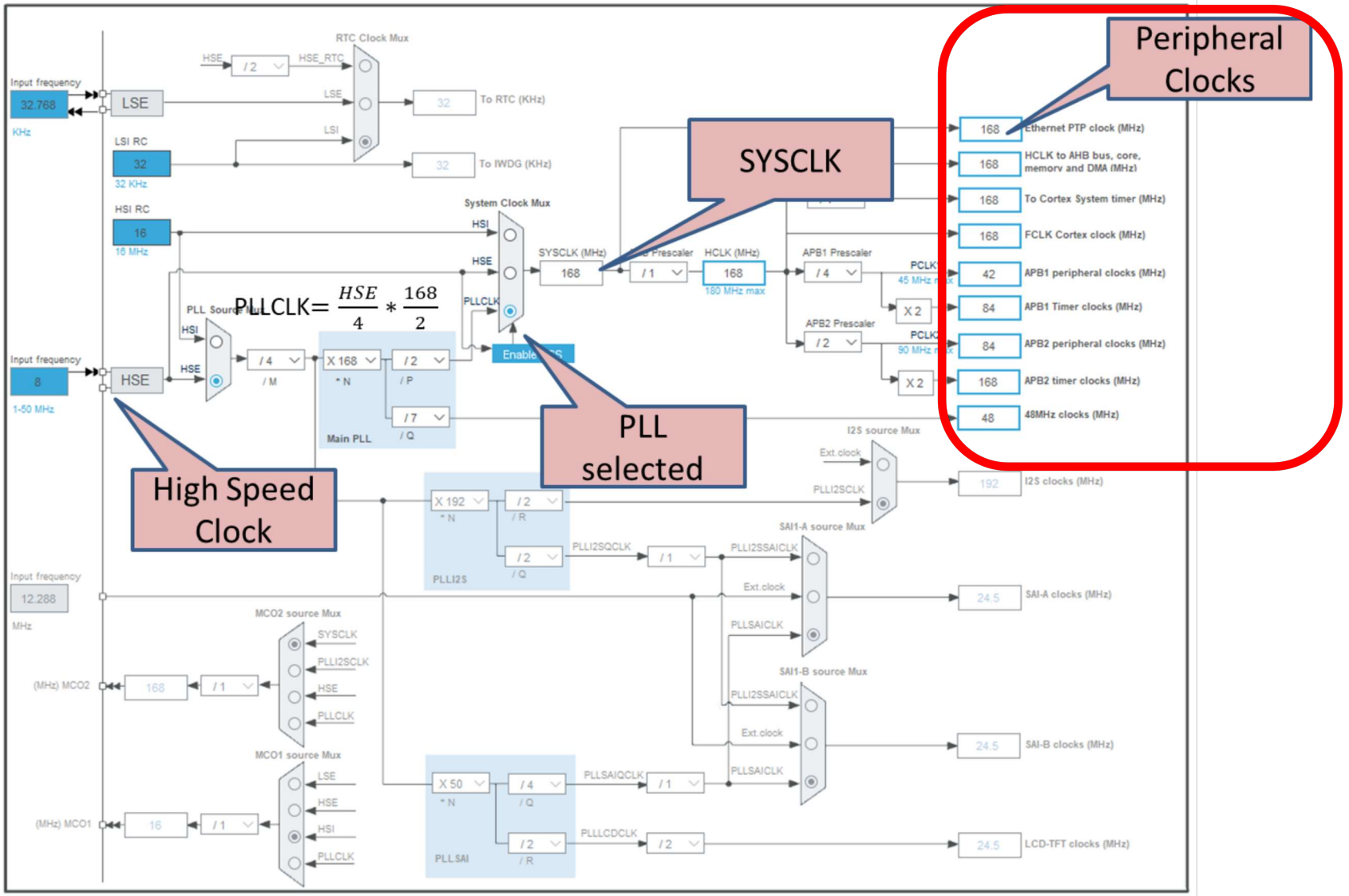


[STM32F429-DATASHEET](#)
Page 20/239

STM32F4xx Block Diagram

Timer clocks?

STM32F4xx Clock Diagram



Timers on STM32F4

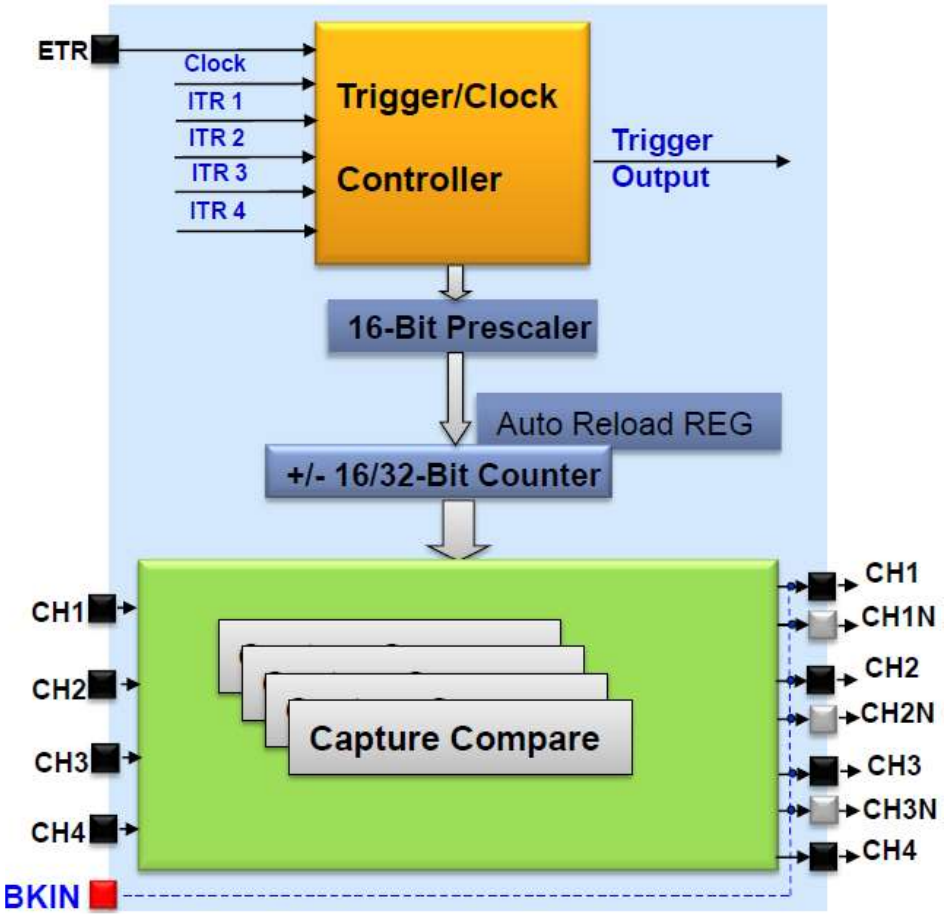
On board there are following timers available:

- **2x advanced 16bit** timers (TIM1,8)
- **2x general purpose 32bit** timers (TIM2,5)
- **8x general purpose 16bit** timers (TIM3,4,9,10..14) ← se usan estos
- **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)

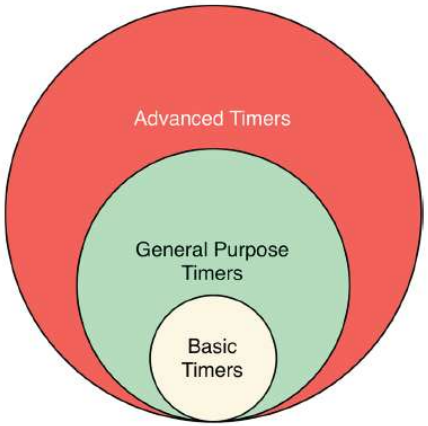
Application benefits

- 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

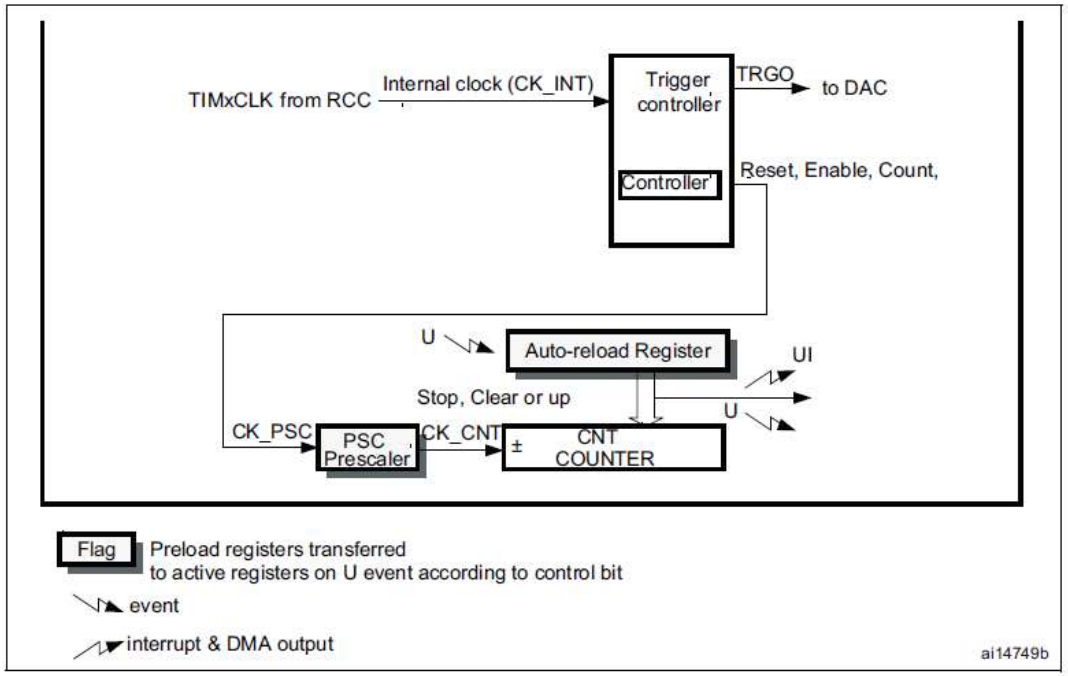
Timer type		STM32 F04x /F070x6 /F03x (exclu-ding /F030x8 and /F030x)	STM32 F030xB /F030x8 /F05x /F09x /F07x (exclu-ding F070x6)	STM32 F101 /F102 /F103 lines XL density (xF, xG)	STM32 F101 /F102 /F103 /F105 /F107 lines up to high-density (x4-xE)	STM32 F100 value line	STM32 F2 /F4 (exclu-ding /F401, /F411, /F410)	STM32 F401 /F411 /F410	STM32 F30X /F3x8 (exclu-ding /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

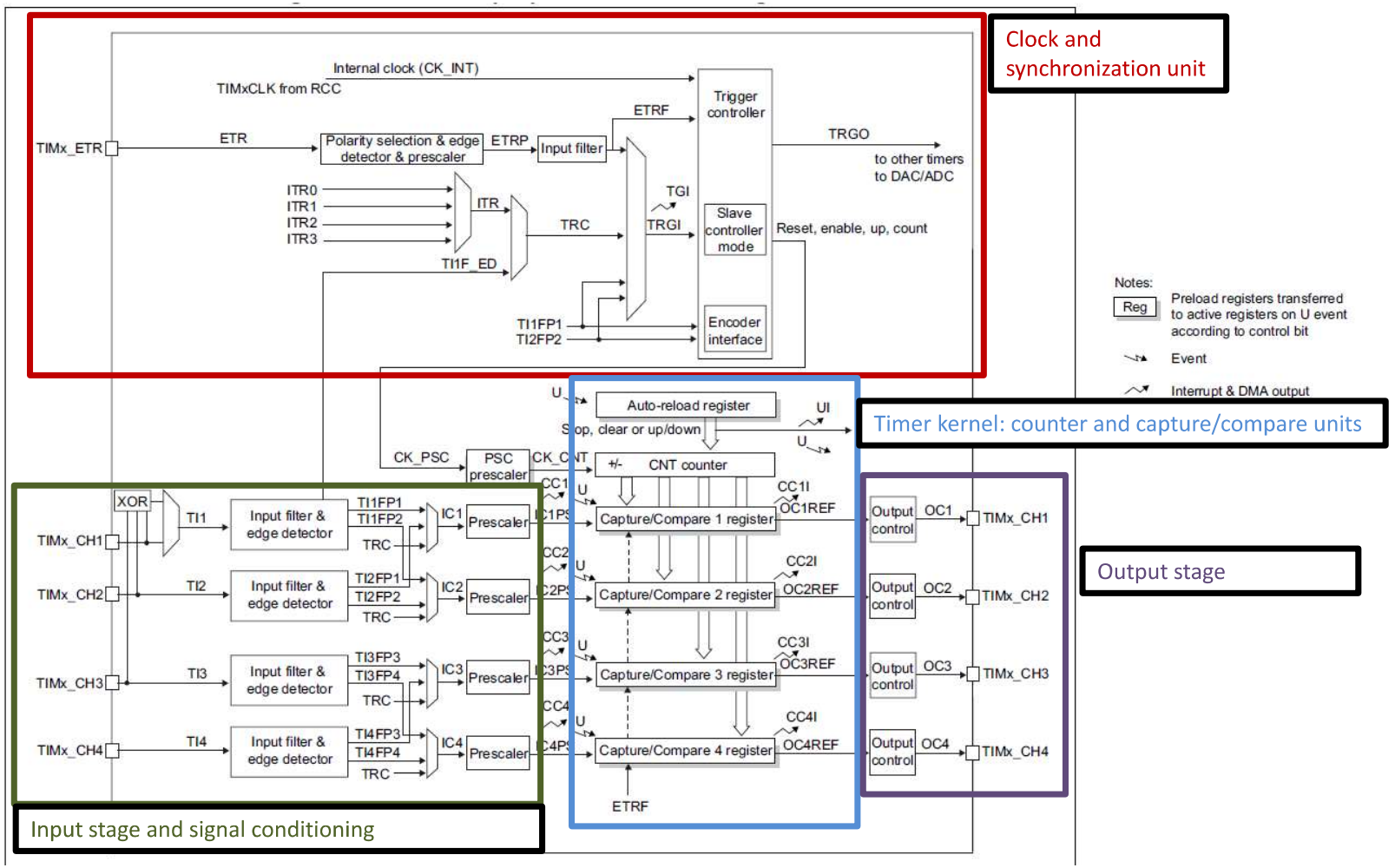


General-purpose timers (TIM2 to TIM5)

- 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

2900

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

TIM_HandleTypeDef

stm32f4xx_hal_tim.h

Timer to be set up

Basic parameters to be configured

```
/**
  * @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;
```

stm32f4xx_hal_tim.h

TIM_Base_InitTypeDef

Timer structures

```
/**
  * @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)

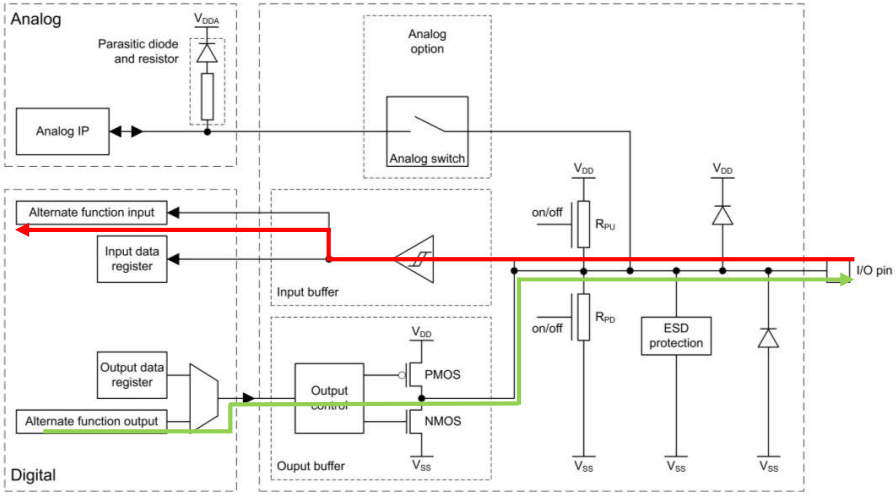
By default, counter UP

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

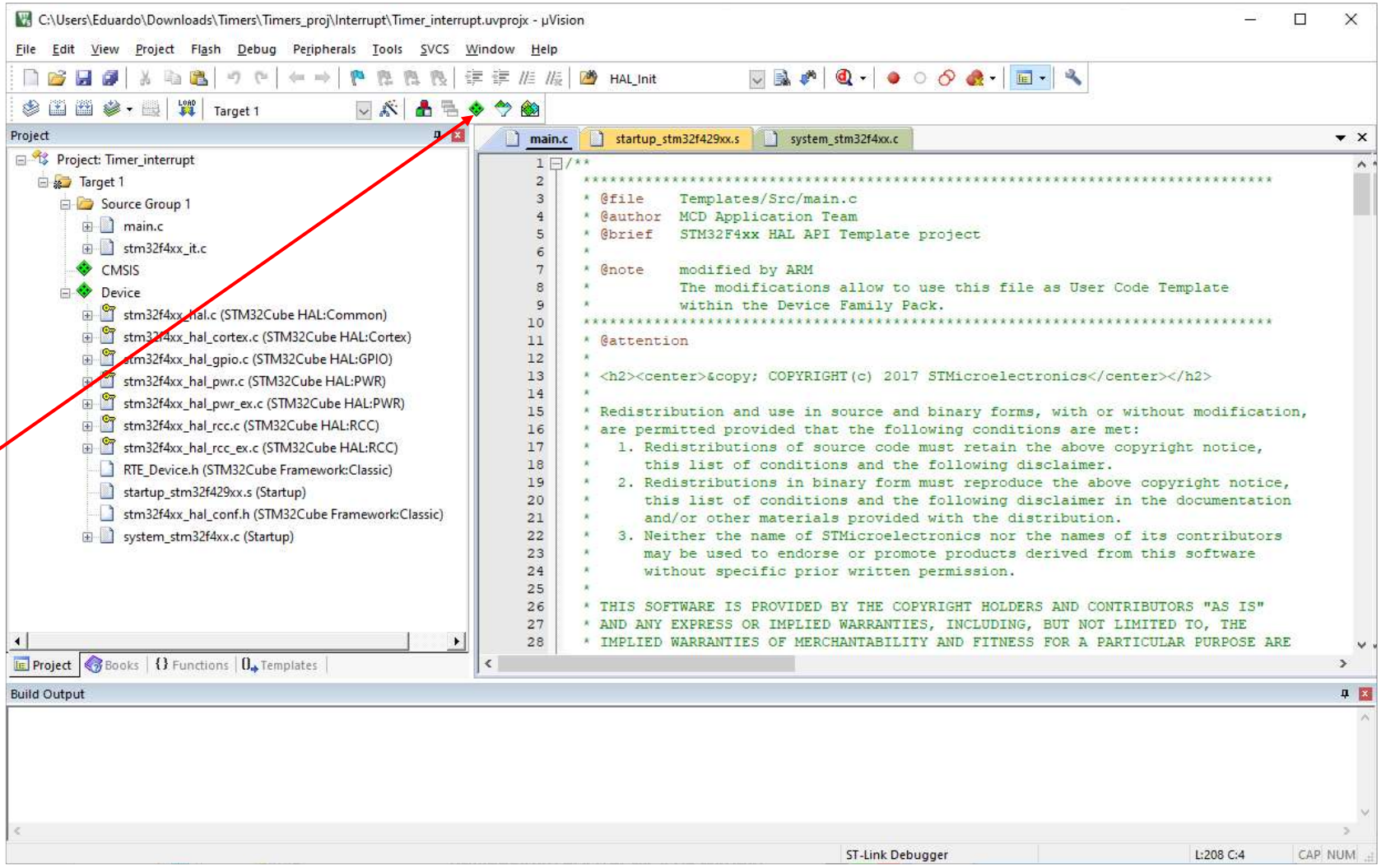
1. Create a new Keil project including the HAL Timer
2. Use the Timer 7 (Basic Timer) to generate a periodic interrupt every 500ms
3. Toggle the LED_1 in the Timer 7 ISR

11. Timers

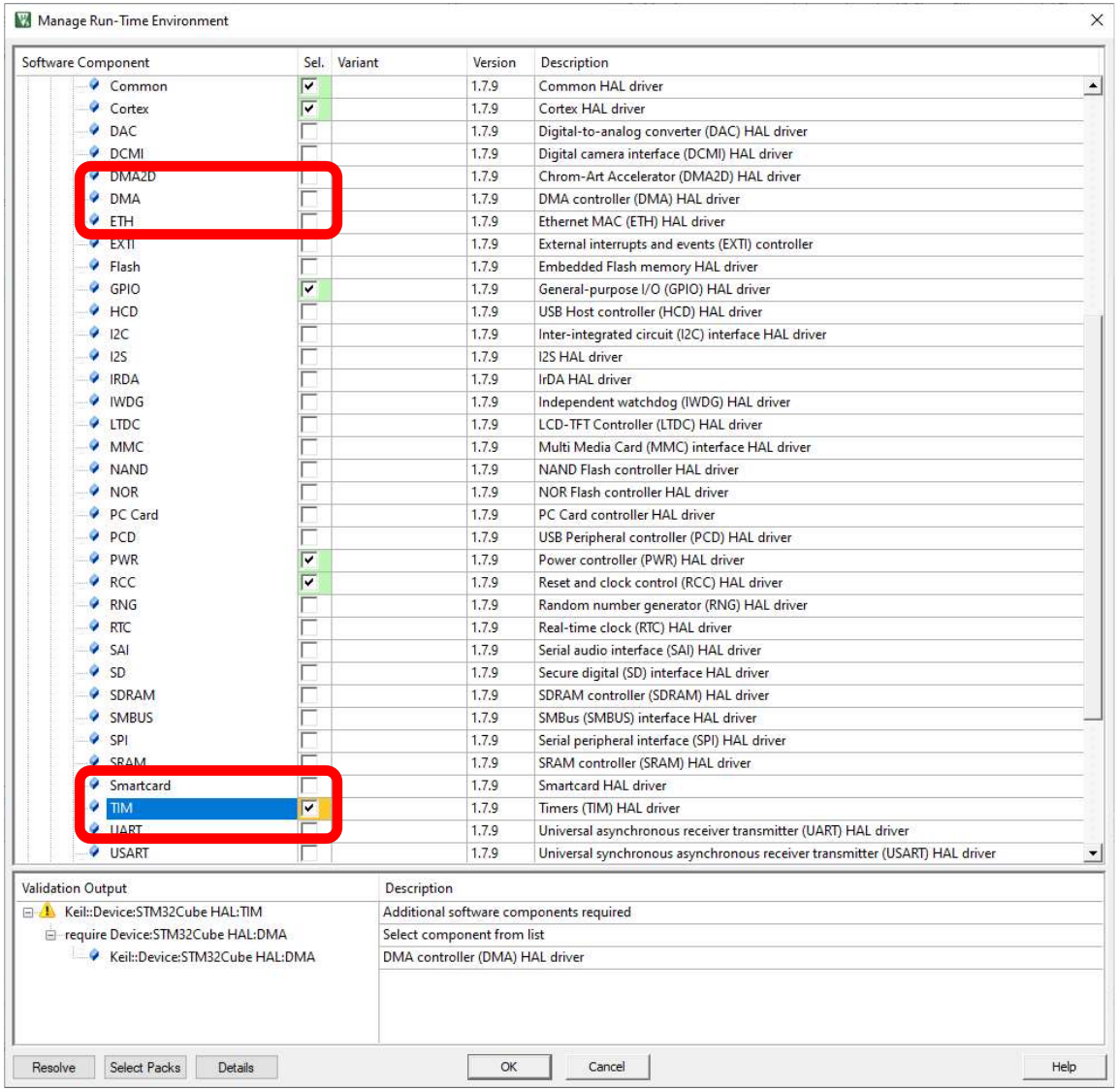
11.2.1 Using Timers in *Interrupt Mode*

How to configure a Keil project

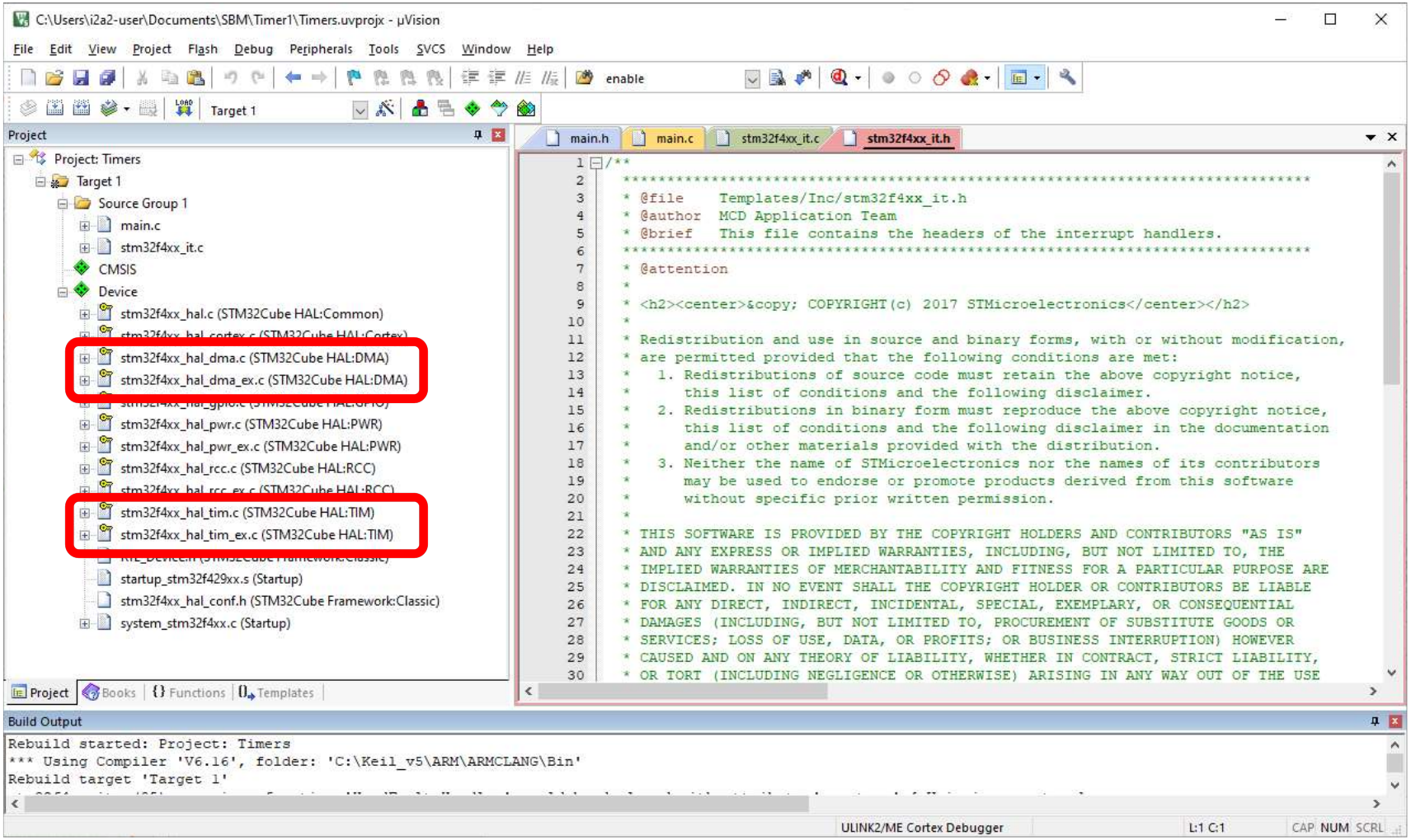
Add component software



How to configure a Keil project



How to configure a Keil project



Example 1. Timer interrupts

```
/* 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

```
  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

```
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

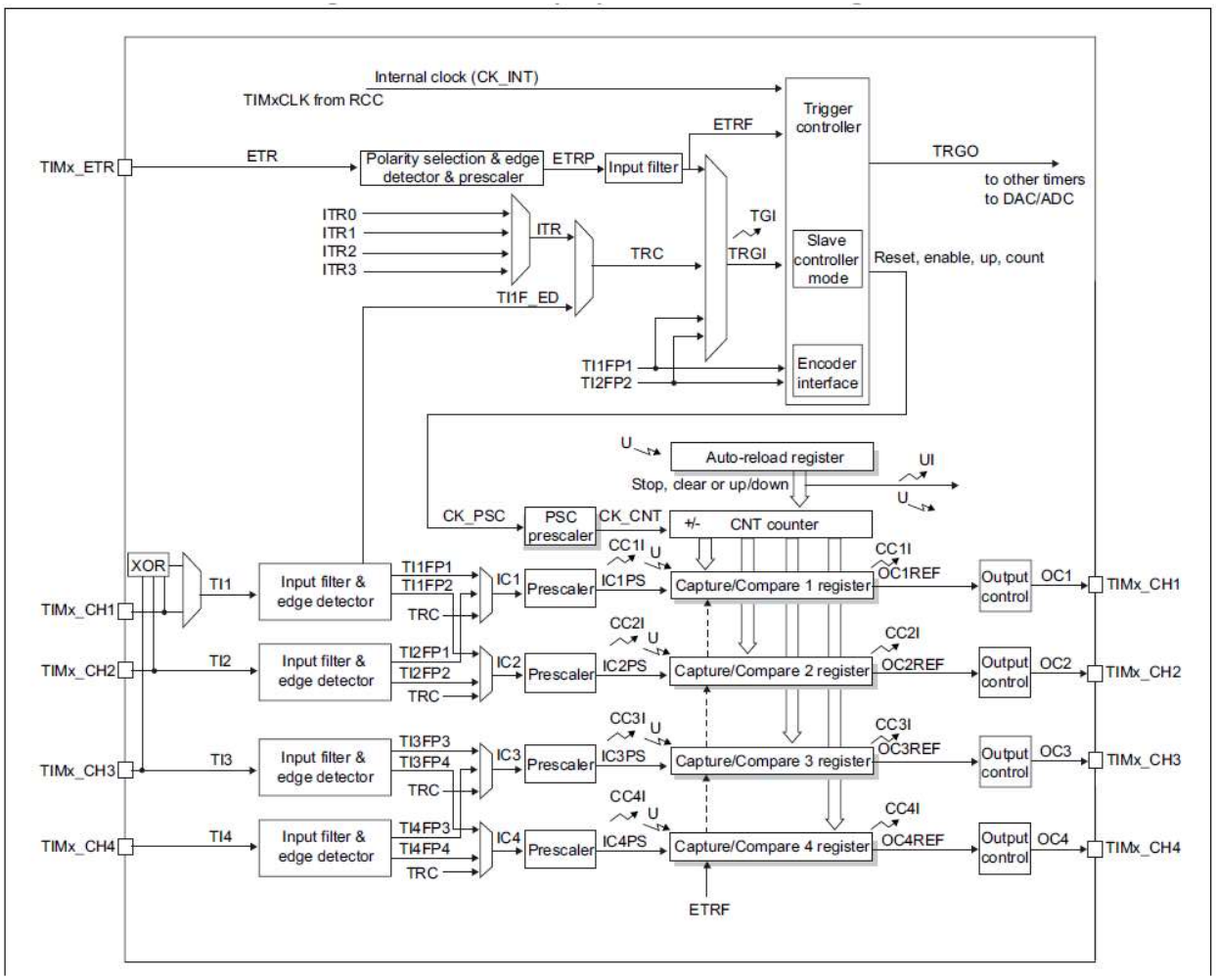
1. Create a new Keil project including the HAL Timer
2. Use Timer 2 (General-Purpose Timer) to generate a 2 kHz frequency square signal
3. **Generation of the square signal by the timer hardware, without software intervention**

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

```
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);
}
```

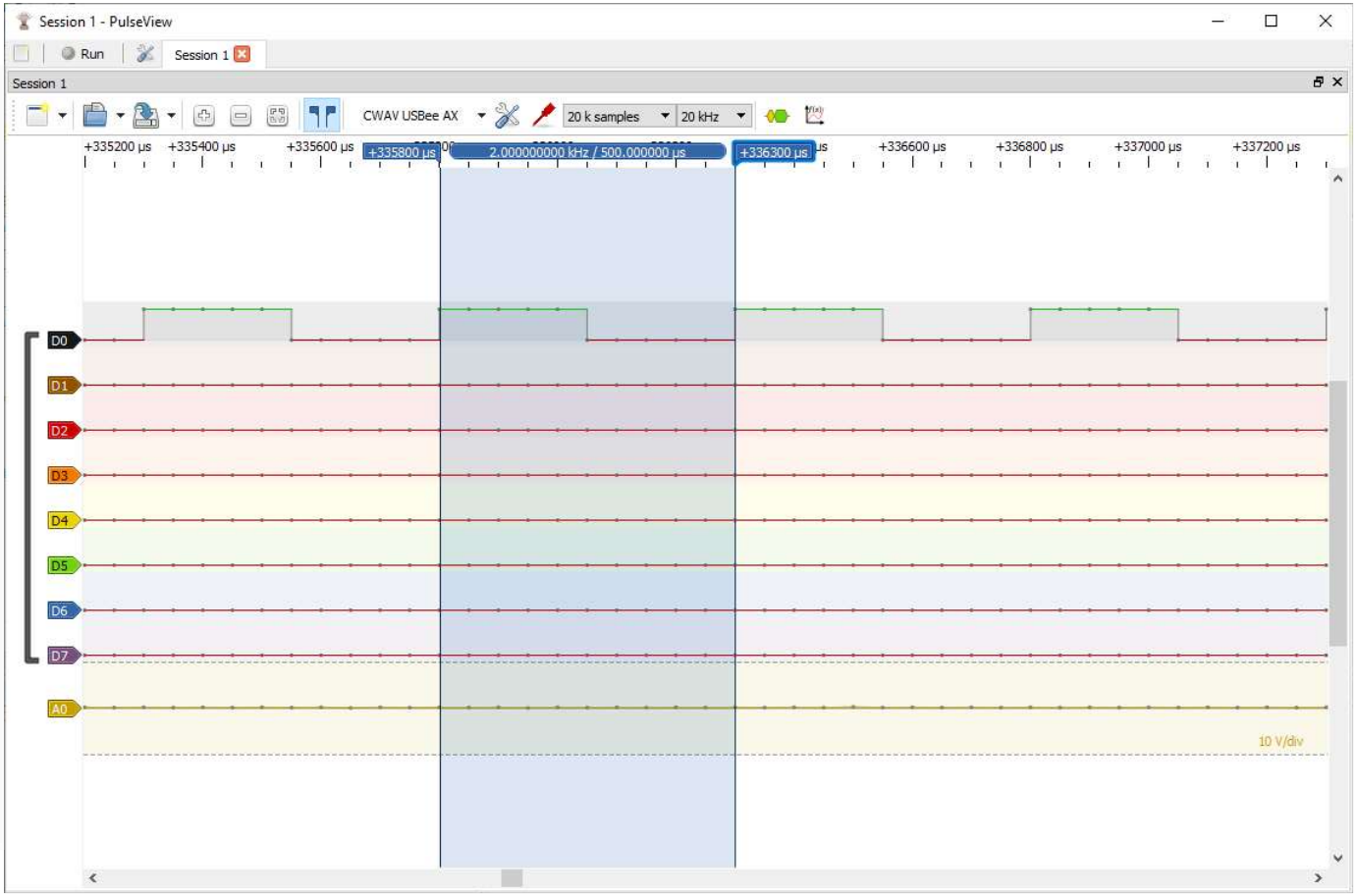
Example 2. Using Timers to generate a square signal

```
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);

}
```

Example 2. Using Timers to generate a square signal

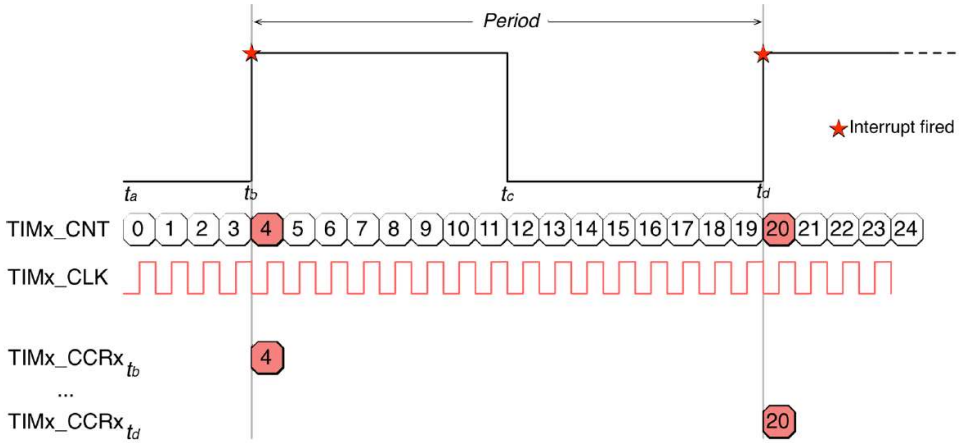
```
void TIM7_IRQHandler(void) {
// Pass the control to HAL, which processes the IRQ
  HAL_TIM_IRQHandler(&htim7);
}

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

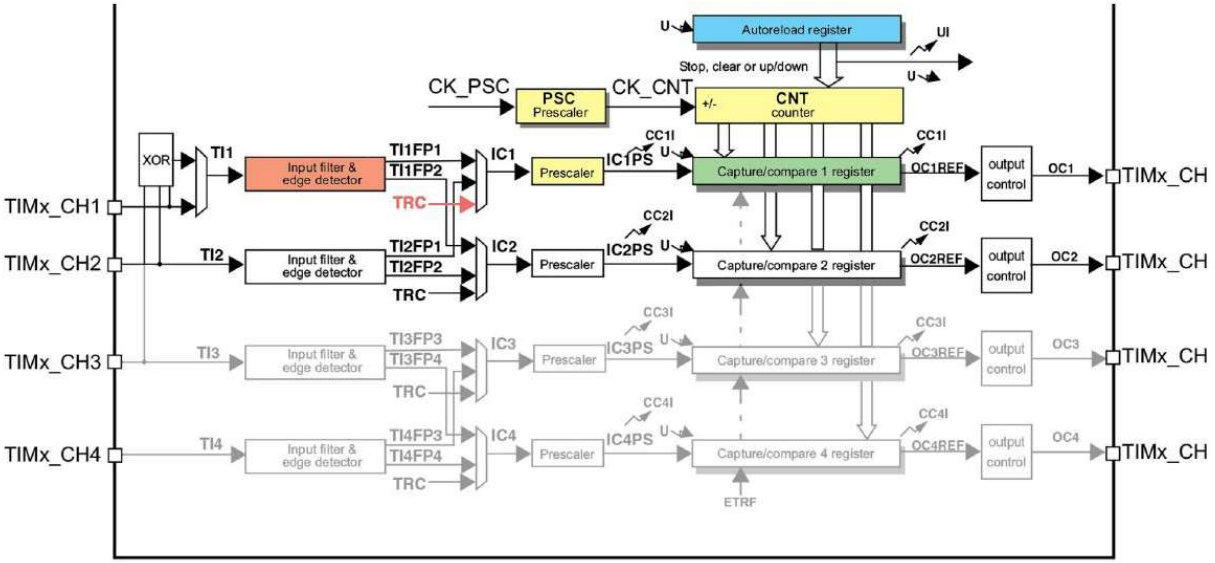
Example 2. Using Timers to generate a square signal



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
- …