PRIMERA PARTE
Programación con STM32Cube HAL

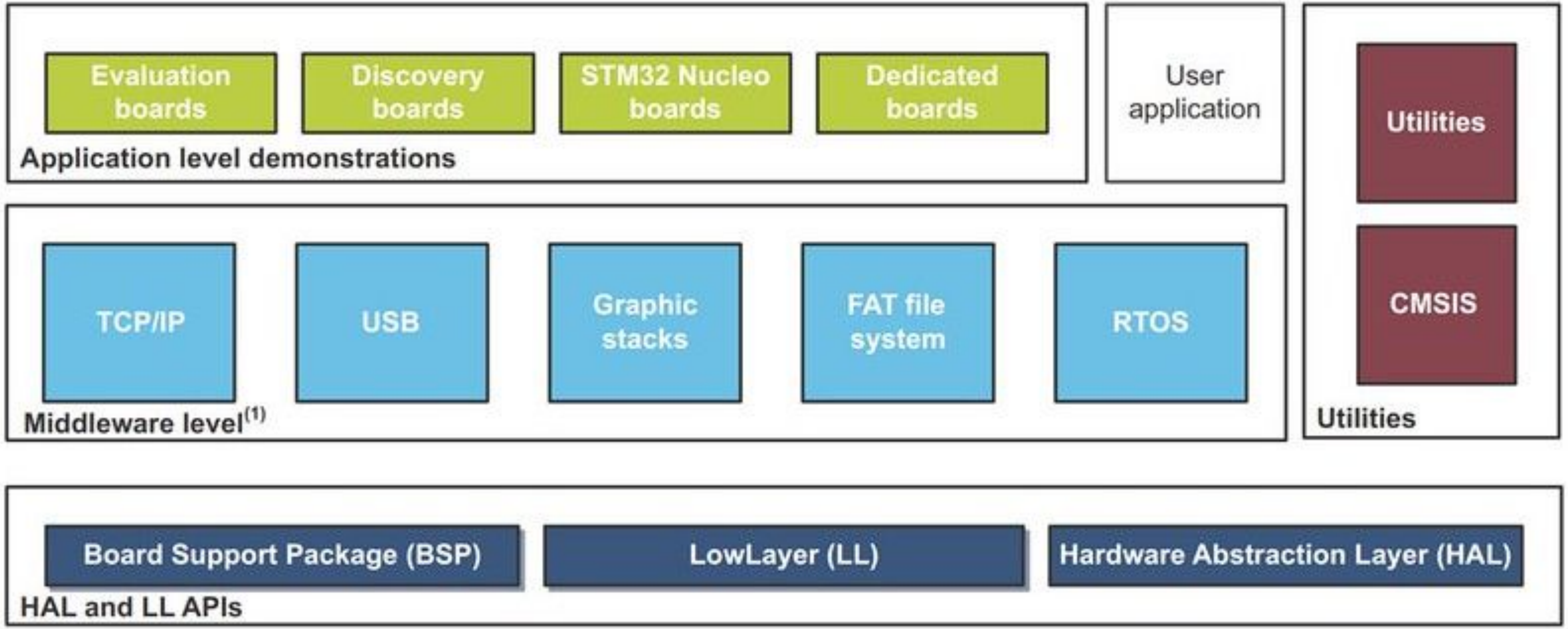
¿Qué es la HAL (Hardware Abstraction Layer)?

HAL es una capa de abstracción de hardware desarrollada por STMicroelectronics para simplificar la programación de sus microcontroladores STM32.

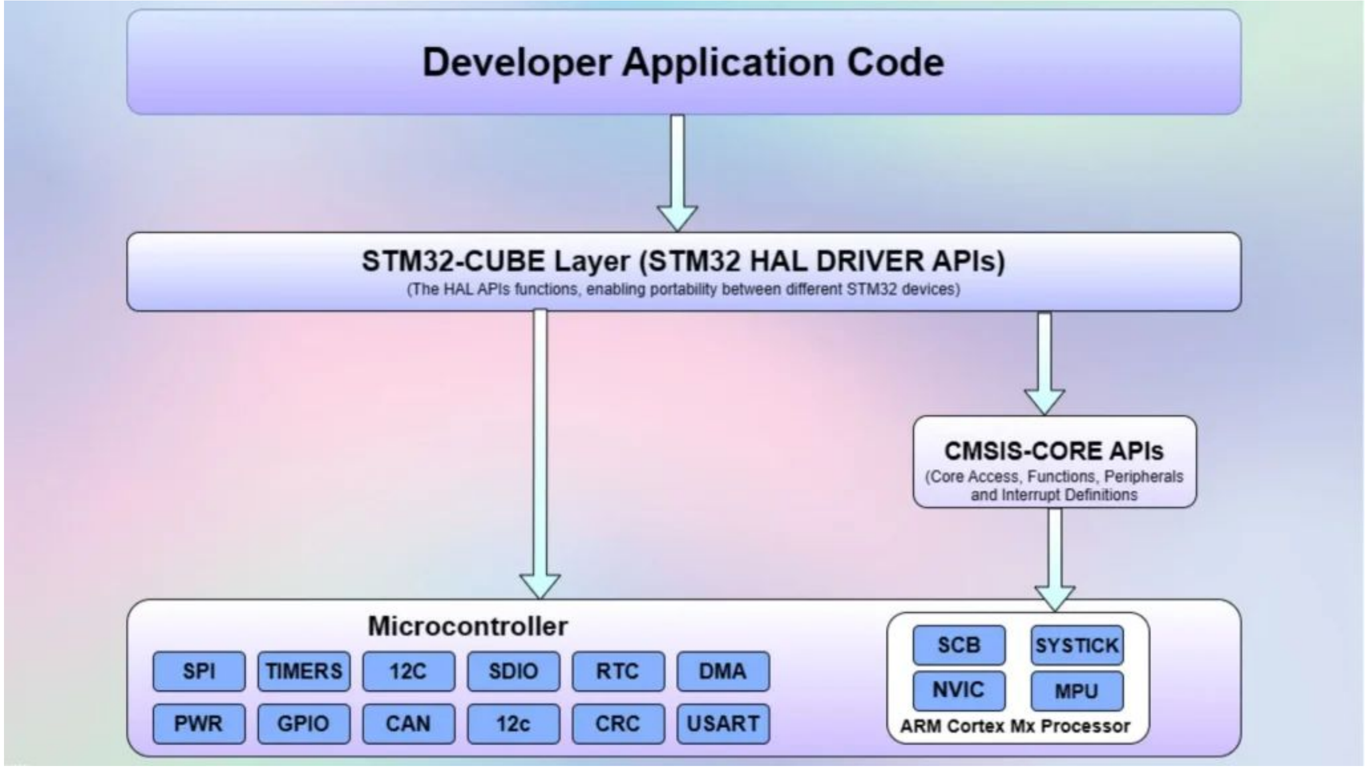
Permite a los desarrolladores interactuar con los periféricos del microcontrolador a través de una API sin necesidad de escribir directamente en registros, haciendo el código más portable y fácil de mantener.

Es parte de la plataforma **STM32Cube Embedded software**

¿Qué es la HAL (Hardware Abstraction Layer)?



¿Qué es la HAL (Hardware Abstraction Layer)?



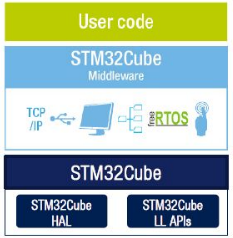
¿Qué es STM32Cube Embedded software?

STM32Cube - Embedded software

6

Introduction

- What is it ?
  - Full featured packages with drivers, USB, TCP/IP, Graphics, File system and RTOS
  - Set of common application programming interfaces, ensuring high portability inside whole STM32 family
  - Set of APIs directly based on STM32 peripheral registers
  - Set of initialization APIs functionally similar to the SPL block peripheral initialization functions
- Target Audience
  - Hardware Abstraction Layer (HAL) APIs: embedded system developers with a strong structured background. New customers looking for a fast way to evaluate STM32 and easy portability
  - Low-Layer (LL) APIs: low level embedded system developers, typically coming from an 8-bit background, used to assembly or C with little abstraction. Stronger focus on customers migrating from the SPL environment.



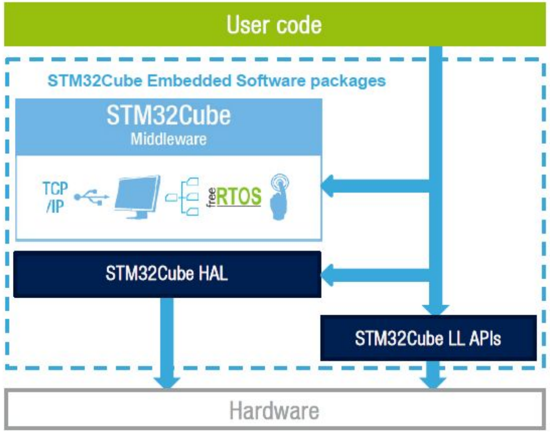
life.augmented

STM32Cube - Embedded software

- Three entry points for the user application:
  - Middleware stacks
  - HAL APIs
  - LL APIs

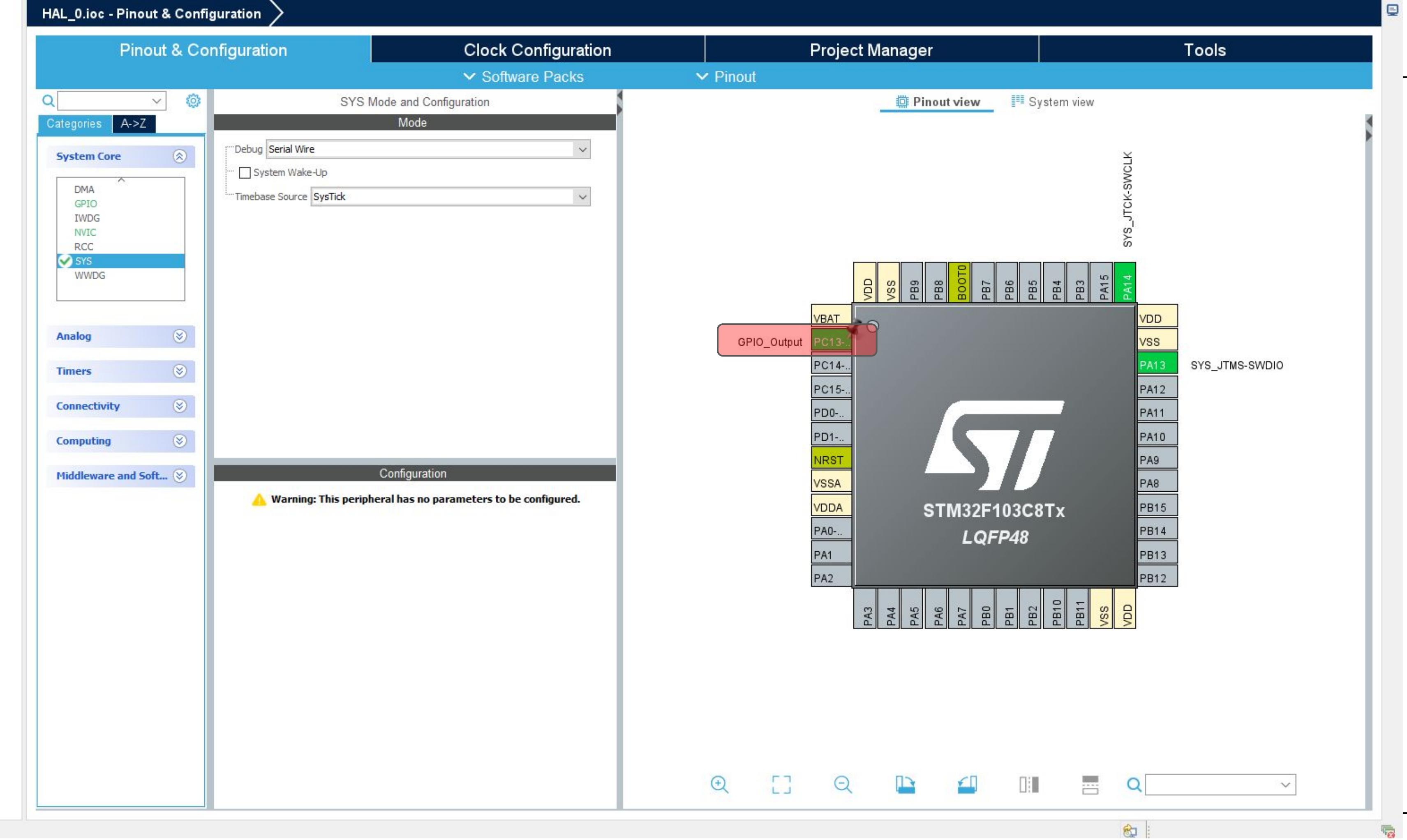
- Possible concurrent usage of HAL and LL
  - Limitation: LL cannot be used with HAL for the same peripheral instance. Impossible to run concurrent processes on the same IP using both APIs, but sequential use is allowed
  - Example of hybrid model:
    - Simpler static peripheral initialization with HAL
    - Optimized runtime peripheral handling with LL calls

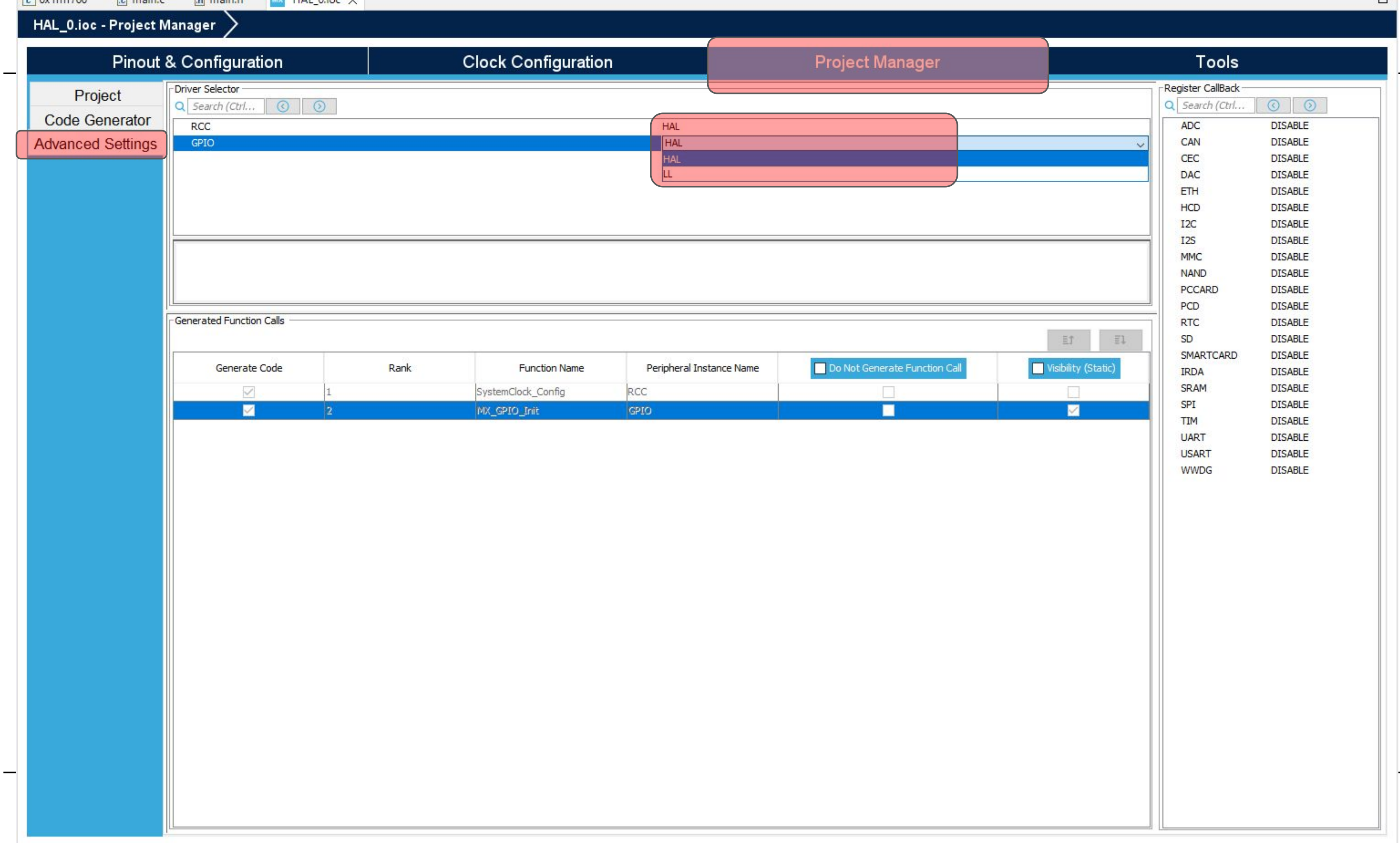
Architecture overview



E

¿


HAL_0.ioc - Pinout & Configuration
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Software Packs
Pinout
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
Connectivity
Computing
Middleware and Soft...
SYS Mode and Configuration
Mode
Debug
Serial Wire
System Wake-Up
Timebase Source
SysTick
Configuration
Warning: This peripheral has no parameters to be configured.
Pinout view
System view
SYS_JTCK-SWCLK
VDD
VSS
PB9
PB8
BOOT0
PB7
PB6
PB5
PB4
PB3
PA15
PA14
VBAT
GPIO_Output
PC13-..
PC14-..
PC15-..
PD0-..
PD1-..
NRST
VSSA
VDDA
PA0-..
PA1
PA2
VDD
VSS
PA13
SYS_JTMS-SWDIO
PA12
PA11
PA10
PA9
PA8
PB15
PB14
PB13
PB12
STM32F103C8Tx
LQFP48
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB10
PB11
VSS
VDD


HAL_0.ioc - Project Manager
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Project
Code Generator
Advanced Settings
Driver Selector
RCC
HAL
GPIO
HAL
HAL
LL
Generated Function Calls
Generate Code
Rank
Function Name
Peripheral Instance Name
Do Not Generate Function Call
Visibility (Static)
1
SystemClock_Config
RCC
2
MX_GPIO_Init
GPIO
Register CallBack
ADC DISABLE
CAN DISABLE
CEC DISABLE
DAC DISABLE
ETH DISABLE
HCD DISABLE
I2C DISABLE
I2S DISABLE
MMC DISABLE
NAND DISABLE
PCCARD DISABLE
PCD DISABLE
RTC DISABLE
SD DISABLE
SMARTCARD DISABLE
IRDA DISABLE
SRAM DISABLE
SPI DISABLE
TIM DISABLE
UART DISABLE
USART DISABLE
WWDG DISABLE

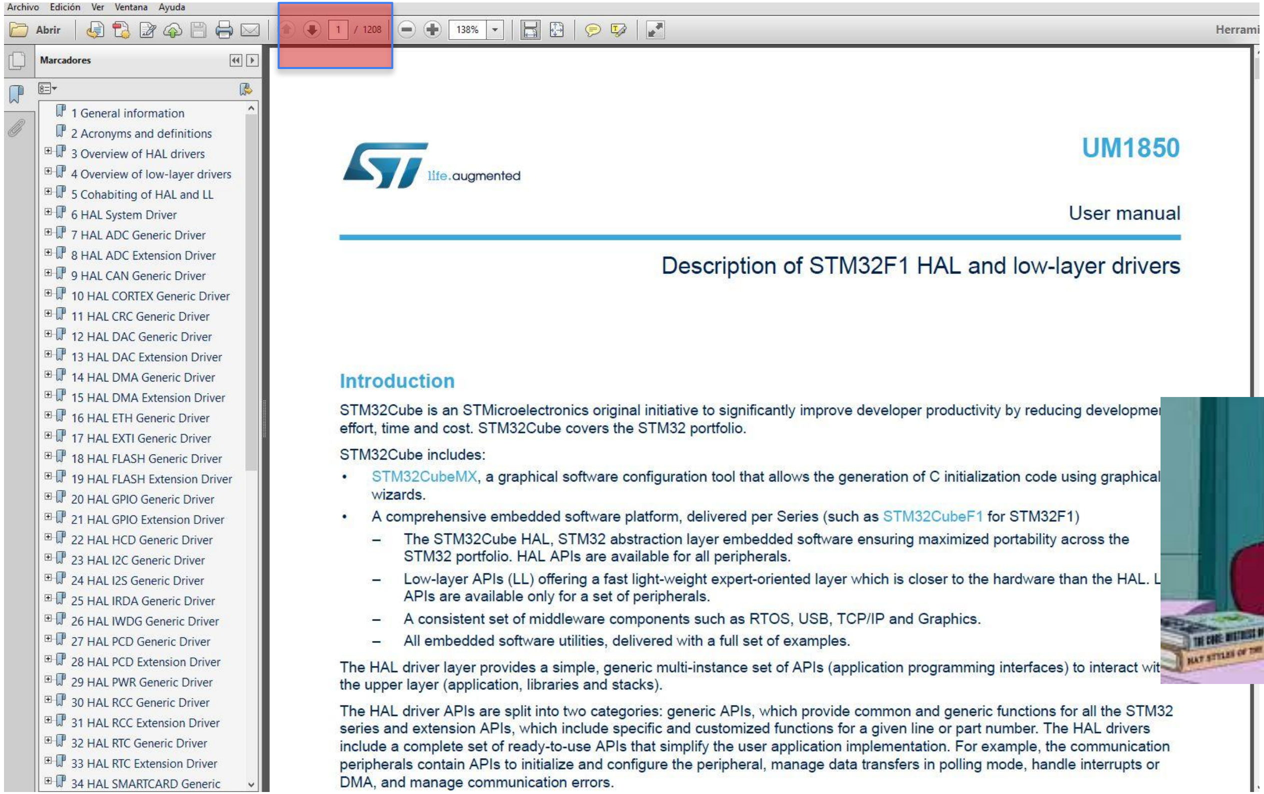
```c
#include "main.h"


void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{

 /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
 HAL_Init();

 /* Configure the system clock */
 SystemClock_Config();

 MX_GPIO_Init();

 while (1) { }
}
```

```c
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
   Error_Handler();
 }
 /** Initializes the CPU, AHB and APB buses clocks
 */
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
 {
   Error_Handler();
 }
}
```

```c
#include "main.h"


void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{

 /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
 HAL_Init();

 /* Configure the system clock */
 SystemClock_Config();

 MX_GPIO_Init();

 while (1) { }
}
```

```c
static void MX_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOC_CLK_ENABLE();
 __HAL_RCC_GPIOA_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
 /*Configure GPIO pin : PC13 */
 GPIO_InitStruct.Pin = GPIO_PIN_13;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
```

UM 1850 – manual de usuario:


Archivo Edición Ver Ventana Ayuda

Abrir 1 / 1208 138% Herrami

Marcadores

1 General information
2 Acronyms and definitions
3 Overview of HAL drivers
4 Overview of low-layer drivers
5 Cohabiting of HAL and LL
6 HAL System Driver
7 HAL ADC Generic Driver
8 HAL ADC Extension Driver
9 HAL CAN Generic Driver
10 HAL CORTEX Generic Driver
11 HAL CRC Generic Driver
12 HAL DAC Generic Driver
13 HAL DAC Extension Driver
14 HAL DMA Generic Driver
15 HAL DMA Extension Driver
16 HAL ETH Generic Driver
17 HAL EXTI Generic Driver
18 HAL FLASH Generic Driver
19 HAL FLASH Extension Driver
20 HAL GPIO Generic Driver
21 HAL GPIO Extension Driver
22 HAL HCD Generic Driver
23 HAL I2C Generic Driver
24 HAL I2S Generic Driver
25 HAL IRDA Generic Driver
26 HAL IWDG Generic Driver
27 HAL PCD Generic Driver
28 HAL PCD Extension Driver
29 HAL PWR Generic Driver
30 HAL RCC Generic Driver
31 HAL RCC Extension Driver
32 HAL RTC Generic Driver
33 HAL RTC Extension Driver
34 HAL SMARTCARD Generic

life.augmented

UM1850

User manual

Description of STM32F1 HAL and low-layer drivers

Introduction

STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing developme[...] effort, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as STM32CubeF1 for STM32F1)
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. HAL APIs are available for all peripherals.
  - Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. L[...] APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS, USB, TCP/IP and Graphics.
  - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact wit[...] the upper layer (application, libraries and stacks).

The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors.

Introducción al uso del API Manual

Las librerías HAL y LL son complementarias y cubren un amplio rango de requerimientos de aplicación:

- La **LL** ofrece APIs de bajo nivel (nivel registro), con **mejor optimización** pero menor portabilidad. Esto requiere un profundo conocimiento del MCU y los periféricos.
- La **HAL** ofrece APIs de alto nivel con **gran portabilidad entre distintos STM32**. Oculta al usuario la complejidad del MCU y los periféricos.
    - Es **multiinstancia**: una misma función permite controlar distintas instancias de un mismo tipo de periférico, cada instancia se identifica con una variable (**estructura handler**) que se pasa como parámetro por referencia a las funciones de la API
    - Los drivers de periféricos ofrecen (por lo general) **tres modos** de funcionamiento: **polling**, **interrupción** y **DMA**
    - En modo **polling** la API ofrece un esquema sistemático de **timeouts** para todas las acciones bloqueantes
    - Esquema de **callbacks** que permite ejecutar funciones de usuario de la capa de aplicación durante la **inicialización de periféricos**, **interrupciones**, **errores**
    - Mecanismo de **bloqueo** para prevenir acceso concurrente al HW compartido
    - Todas las funciones son **reentrantes**

Uso típico de periférico (PPP) con drivers HAL:

1. Definir variable handler (`PPP_HandleTypeDef handler;`) *
2. Asignar dirección de memoria base de los registros del periféricos (`handler.Instance`)
3. Asignar valores de inicialización (`handler.Init`)
4. Invocar función de inicialización (`HAL_PPP_Init(&handler,...)`)

—---

5. Usar el periférico con funciones específicas según la aplicación

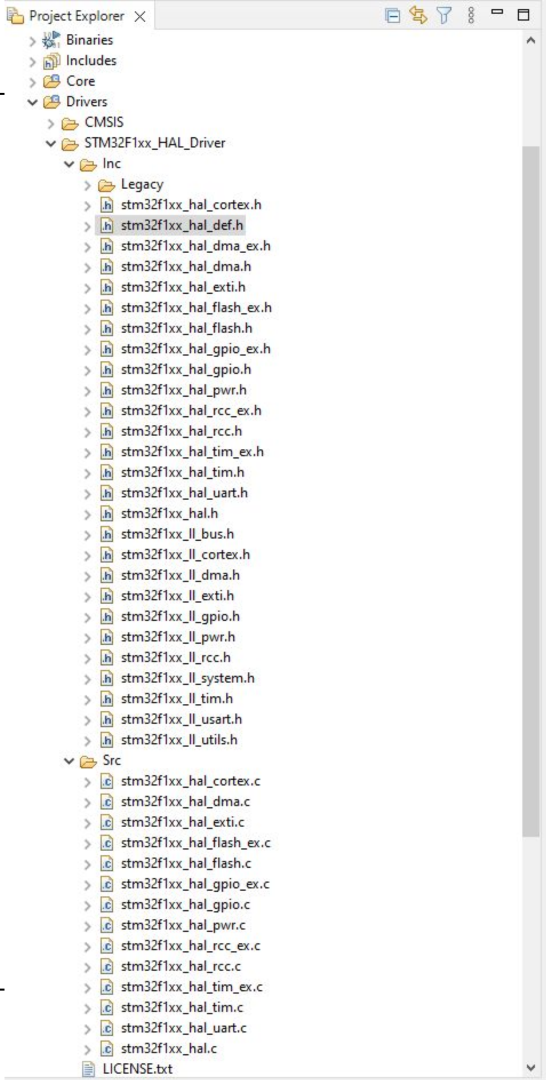
(`HAL_PPP_Accion(&handler,...)`,
`HAL_PPP_Accion_IT(&handler,...)`,
`HAL_PPP_Accion_DMA(&handler,...)`)

* Los siguientes periféricos son la excepción y la HAL los controla directamente por la dirección de memoria de los registros definida en la CMSIS

- GPIO
- SYSTICK
- NVIC
- PWR
- RCC
- FLASH

Organización de los **drivers** HAL:

File	Description
stm32f1xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f1xx_hal_adc.c, stm32f1xx_hal_irda.c, …*
stm32f1xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32f1xx_hal_adc.h,stm32f1xx_hal_irda.h, …*
stm32f1xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32f1xx_hal_adc_ex.c,stm32f1xx_hal_flash_ex.c, …*
stm32f1xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32f1xx_hal_adc_ex.h,stm32f1xx_hal_flash_ex.h, …*
stm32f1xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32f1xx_hal.h	stm32f1xx_hal.c header file
stm32f1xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32f1xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.



Esquema de los **drivers** HAL:

File	Description
stm32f1xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f1xx_hal_adc.c, stm32f1xx_hal_irda.c, …*
stm32f1xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32f1xx_hal_adc.h,stm32f1xx_hal_irda.h, …*
stm32f1xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32f1xx_hal_adc_ex.c,stm32f1xx_hal_flash_ex.c, …*
stm32f1xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32f1xx_hal_adc_ex.h,stm32f1xx_hal_flash_ex.h, …*
stm32f1xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32f1xx_hal.h	stm32f1xx_hal.c header file
stm32f1xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32f1xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

Individuales para cada periférico (o función)

Esquema de los **drivers** HAL:

File	Description
stm32f1xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f1xx_hal_adc.c, stm32f1xx_hal_irda.c, …*
stm32f1xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32f1xx_hal_adc.h,stm32f1xx_hal_irda.h, …*
stm32f1xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32f1xx_hal_adc_ex.c,stm32f1xx_hal_flash_ex.c, …*
stm32f1xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32f1xx_hal_adc_ex.h,stm32f1xx_hal_flash_ex.h, …*
stm32f1xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32f1xx_hal.h	stm32f1xx_hal.c header file
stm32f1xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32f1xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

Módulo ppal.

Esquema de los **drivers** HAL:

File	Description
stm32f1xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f1xx_hal_adc.c, stm32f1xx_hal_irda.c, …*
stm32f1xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32f1xx_hal_adc.h,stm32f1xx_hal_irda.h, …*
stm32f1xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32f1xx_hal_adc_ex.c,stm32f1xx_hal_flash_ex.c, …*
stm32f1xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32f1xx_hal_adc_ex.h,stm32f1xx_hal_flash_ex.h, …*
stm32f1xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32f1xx_hal.h	stm32f1xx_hal.c header file
stm32f1xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32f1xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

definiciones, macros, enums y tipos de datos usados en toda al HAL:

```c
typedef enum
{
  HAL_OK       = 0x00U,
  HAL_ERROR    = 0x01U,
  HAL_BUSY     = 0x02U,
  HAL_TIMEOUT  = 0x03U
} HAL_StatusTypeDef;
```

Archivos de **capa de aplicación**:

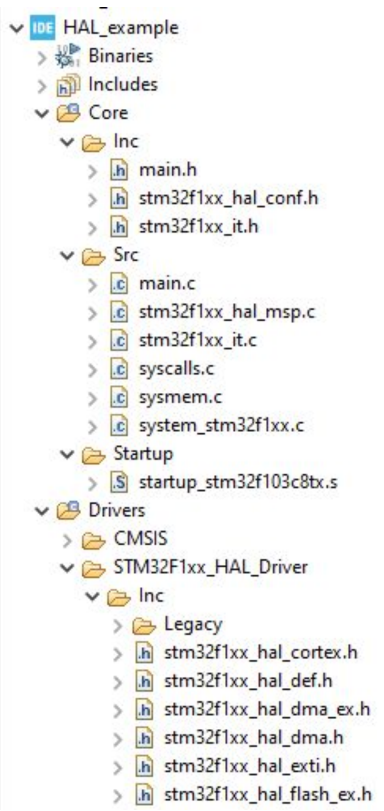


Table 3. User-application files

File	Description
system_stm32f1xx.c	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM.
startup_stm32f1xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32f1xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32f1xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32f1xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • Call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

Esquema general de los drivers HAL:

Estructuras de datos HAL

Cada controlador HAL puede contener las siguientes estructuras de datos:

- **Estructura de manejo de periféricos (handlers)**
  - **x ej.: `TIM_HandleTypeDef`, `ADC_HandleTypeDef`, `UART_HandleTypeDef`...**
- Estructura de inicialización y configuración
  - x ej: `GPIO_InitTypeDef`, `TIM_Base_InitTypeDef`, `UART_InitTypeDef`
- Estructura de proceso específico
  - x ej: `TIM_ClockConfigTypeDef`, `TIM_ClockConfigTypeDef`

Handlers

en main.c

1) Definir variable handler

```c
UART_HandleTypeDef huart1;
static void MX_USART1_UART_Init(void)
{
 /* USER CODE BEGIN USART1_Init 0 */
 /* USER CODE END USART1_Init 0 */
 /* USER CODE BEGIN USART1_Init 1 */
 /* USER CODE END USART1_Init 1 */
 huart1.Instance = USART1;
 huart1.Init.BaudRate = 115200;
 huart1.Init.WordLength = UART_WORDLENGTH_8B;
 huart1.Init.StopBits = UART_STOPBITS_1;
 huart1.Init.Parity = UART_PARITY_NONE;
 huart1.Init.Mode = UART_MODE_TX_RX;
 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart1.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart1) != HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN USART1_Init 2 */
 /* USER CODE END USART1_Init 2 */
}
```

en stm32f103xb.h (CMSIS)

```c
#define PERIPH_BASE           0x40000000UL /*!< Peripheral base address in the alias region */

#define APB2PERIPH_BASE   (PERIPH_BASE + 0x00010000UL)

#define USART1_BASE       (APB2PERIPH_BASE + 0x00003800UL)

#define USART1           ((USART_TypeDef *)USART1_BASE)
```

2) Asignar dirección de memoria base de los registros del periféricos

3) Asignar valores de inicialización

4) Invocar función de inicialización

Handlers

en stm32f1xx_hal_uart.h

```c
typedef struct
{
  USART_TypeDef                 *Instance;        /*!< UART registers base address        */
  UART_InitTypeDef              Init;             /*!< UART communication parameters      */
  const uint8_t                 *pTxBuffPtr;      /*!< Pointer to UART Tx transfer Buffer */
  uint16_t                      TxXferSize;       /*!< UART Tx Transfer size              */
  __IO uint16_t                 TxXferCount;      /*!< UART Tx Transfer Counter           */
  uint8_t                       *pRxBuffPtr;      /*!< Pointer to UART Rx transfer Buffer */
  uint16_t                      RxXferSize;       /*!< UART Rx Transfer size              */
  __IO uint16_t                 RxXferCount;      /*!< UART Rx Transfer Counter           */
  __IO HAL_UART_RxTypeTypeDef ReceptionType;      /*!< Type of ongoing reception          */
  __IO HAL_UART_RxEventTypeTypeDef RxEventType;   /*!< Type of Rx Event                   */
  DMA_HandleTypeDef             *hdmatx;          /*!< UART Tx DMA Handle parameters      */
  DMA_HandleTypeDef             *hdmarx;          /*!< UART Rx DMA Handle parameters      */
  HAL_LockTypeDef               Lock;             /*!< Locking object                     */
  __IO HAL_UART_StateTypeDef    gState;           /*!< UART state information related to global Handle management
                                                       and also related to Tx operations.
                                                       This parameter can be a value of @ref HAL_UART_StateTypeDef */
  __IO HAL_UART_StateTypeDef    RxState;          /*!< UART state information related to Rx operations.
                                                       This parameter can be a value of @ref HAL_UART_StateTypeDef */
  __IO uint32_t                 ErrorCode;        /*!< UART Error code                    */
} UART_HandleTypeDef;
```

Handlers

en stm32f1xx_hal_uart.h

```
typedef struct
{
  USART_TypeDef              *Instance;      /*!< UART registers base address        */
  UART_InitTypeDef           Init;           /*!< UART communication parameters      */
  const uint8_t              *pTxBuffPtr;    /*!< Pointer to UART Tx transfer Buffer */
  uint16_t                   TxXferSize;     /*!< UART Tx Transfer size              */
  __IO uint16_t              TxXferCount;    /*!< UART Tx Transfer Counter           */
  uint8_t                    *pRxBuffPtr;    /*!< Pointer to UART Rx transfer Buffer */
  uint16_t
  __IO uint16_t
  __IO HAL_UART_RxTypeTy
  __IO HAL_UART_RxEvent
  DMA_HandleTypeDef
  DMA_HandleTypeDef
  HAL_LockTypeDef
  __IO HAL_UART_StateTyp
                                                                                                       f */
  __IO HAL_UART_StateTypeDef    RxState;     /*!< UART state information related to Rx operations.
                                                  This parameter can be a value of @ref HAL_UART_StateTypeDef */
  __IO uint32_t              ErrorCode;      /*!< UART Error code                    */
} UART_HandleTypeDef;
```

en stm32f103xb.h (CMSIS)

```
typedef struct
{
  __IO uint32_t SR;         /*!< USART Status register,                   Address offset: 0x00 */
  __IO uint32_t DR;         /*!< USART Data register,                     Address offset: 0x04 */
  __IO uint32_t BRR;        /*!< USART Baud rate register,                Address offset: 0x08 */
  __IO uint32_t CR1;        /*!< USART Control register 1,                Address offset: 0x0C */
  __IO uint32_t CR2;        /*!< USART Control register 2,                Address offset: 0x10 */
  __IO uint32_t CR3;        /*!< USART Control register 3,                Address offset: 0x14 */
  __IO uint32_t GTPR;       /*!< USART Guard time and prescaler register, Address offset: 0x18 */
} USART_TypeDef;
```

Handlers

en stm32f1xx_hal_uart.h

```
typedef struct
{
  USART_TypeDef                 *Instance;
  UART_InitTypeDef              Init;
  const uint8_t                 *pTxBuffPtr;
  uint16_t                      TxXferSize;
  __IO uint16_t                 TxXferCount;
  uint8_t                       *pRxBuffPtr;
  uint16_t                      RxXferSize;
  __IO uint16_t                 RxXferCount;
  __IO HAL_UART_RxTypeTypeDef ReceptionType;
  __IO HAL_UART_RxEventTypeTypeDef RxEventType;
  DMA_HandleTypeDef             *hdmatx;
  DMA_HandleTypeDef             *hdmarx;
  HAL_LockTypeDef               Lock;
  __IO HAL_UART_StateTypeDef    gState;


  __IO HAL_UART_StateTypeDef    RxState;

  __IO uint32_t                 ErrorCode;
} UART_HandleTypeDef;
```

```
typedef struct
{
  uint32_t BaudRate;            /*!< This member configures the UART communication baud ra
                                     The baud rate is computed using the following formula
                                     - IntegerDivider = ((PCLKx) / (16 * (huart->Init.Baud
                                     - FractionalDivider = ((IntegerDivider - ((uint32_t)
  uint32_t WordLength;          /*!< Specifies the number of data bits transmitted or rece
                                     This parameter can be a value of @ref UART_Word_Lengt
  uint32_t StopBits;            /*!< Specifies the number of stop bits transmitted.
                                     This parameter can be a value of @ref UART_Stop_Bits
  uint32_t Parity;              /*!< Specifies the parity mode.
                                     This parameter can be a value of @ref UART_Parity
                                     @note When parity is enabled, the computed parity is
                                           at the MSB position of the transmitted data (9t
                                           the word length is set to 9 data bits; 8th bit
                                           word length is set to 8 data bits). */
  uint32_t Mode;                /*!< Specifies whether the Receive or Transmit mode is ena
                                     This parameter can be a value of @ref UART_Mode */
  uint32_t HwFlowCtl;           /*!< Specifies whether the hardware flow control mode is e
                                     This parameter can be a value of @ref UART_Hardware_F
  uint32_t OverSampling;        /*!< Specifies whether the Over sampling 8 is enabled or d
fPCLK/8).

                                     This parameter can be a value of @ref UART_Over_Sampl
                                     on STM32F100xx family, so OverSampling parameter shou

} UART_InitTypeDef;
```

Ejemplo 1 - “hola mundo” usando drivers HAL:

Imprimir en una terminal serie (UART) “Hola mundo\r\n” cada 10 segundos (SYSTICK).

Notes

- In the default implementation, this variable is incremented each 1ms in SysTick ISR.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

`HAL_Delay`

Function name

void HAL_Delay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on variable incremented.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

`HAL_GetTick`

Function name

uint32_t HAL_GetTick (void)

Function description

Provides a tick value in millisecond.

Return values

- **tick:** value

Notes

- This function is declared as __weak to be overwritten in case of other implementations in user file.

`HAL_GetTickPrio`

Function name

uint32_t HAL_GetTickPrio (void)

Function description

This function returns a tick priority.

Function description

UART MSP DeInit.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **None:**

`HAL_UART_Transmit`

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Sends an amount of data in blocking mode.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

`HAL_UART_Receive`

Function name

HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives an amount of data in blocking mode.

Parameters

Ejemplo 1 - “hola mundo” usando drivers HAL:

Imprimir en una terminal serie (UART) “Hola mundo\r\n” cada 10 segundos (SYSTICK).

```
while (1)
 {
       HAL_Delay(10000);
       HAL_UART_Transmit(&huart1, (uint8_t*)"Hola mundo\r\n", strlen("Hola mundo\r\n"), 100);
 }
```

```
void HAL_Delay(uint32_t Delay)
```

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

Ejemplo 2 - Interprete de comandos

Leer de un terminal serie un comando (terminado en ‘\r\n”)

- Si el comando es “LED ON” o “LED OFF” actuar en correspondencia con el LED on board
- Si el comando es “LED?” responder con un mensaje al terminal el estado del LED
- Si el comando es otro responder con un mensaje de error al terminal

Ejemplo 2 - Interprete de comandos

Leer de un terminal serie un comando (terminado en ‘\r\n”)

- Si el comando es “LED ON” o “LED OFF” actuar en correspondencia con el LED on board
- Si el comando es “LED?” responder con un mensaje al terminal el estado del LED
- Si el comando es otro responder con un mensaje de error al terminal

```
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)

HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size, uint32_t Timeout)

void HAL_GPIO_WritePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
```

```c
const size_t rx_buffer_size = 50;
 uint8_t rx_buffer[rx_buffer_size]; // Buffer para recibir comandos
 uint8_t rx_byte;
 uint16_t rx_index = 0;
 uint8_t estado = 0;
 while (1)
 {
         // Recibir un byte del terminal serie
         if (HAL_UART_Receive(&huart1, &rx_byte, 1, HAL_MAX_DELAY) == HAL_OK) {
                 rx_buffer[rx_index++] = rx_byte;
                 if(rx_index>=rx_buffer_size && rx_byte != '\n') rx_index=0;
                 if (rx_byte == '\n') {
                         rx_buffer[rx_index] = '\0';
                         if (strcmp((char*)rx_buffer, "LED ON\r\n") == 0 || strcmp((char*)rx_buffer, "LED ON\n") == 0 ) {
                                 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET); // Encender el LED
                                 estado = 1;
                         } else if (strcmp((char*)rx_buffer, "LED OFF\r\n") == 0 || strcmp((char*)rx_buffer, "LED OFF\n") == 0 ){
                                 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET); // Apagar el LED
                                 estado = 0;
                         } else if (strcmp((char*)rx_buffer, "LED?\r\n") == 0 || strcmp((char*)rx_buffer, "LED?\n") == 0 ) {
                                 if (estado == 0) {
                                         HAL_UART_Transmit(&huart1, (uint8_t*)"LED OFF\r\n", strlen("LED OFF\r\n"),HAL_MAX_DELAY);
                                 } else {
                                         HAL_UART_Transmit(&huart1, (uint8_t*)"LED ON\r\n", strlen("LED ON\r\n"), HAL_MAX_DELAY);
                                 }
                         } else {
                                 HAL_UART_Transmit(&huart1, (uint8_t*)"Error\r\n", strlen("Error\r\n"), HAL_MAX_DELAY);
                         }
                         rx_index = 0; // Reiniciar el índice del buffer de recepción
                 }
         }
 }
```

SEGUNDA PARTE
Manejo de interrupciones con la HAL

Manejo de interrupciones

- Cuando programamos embebidos la mayoría de los eventos que gestionamos son asincrónicos, generados por periféricos propios o bien el mundo exterior a nuestro sistema.
- Todos los microcontroladores gestionan interrupciones. Justamente la interrupción es un evento asincrónico que provoca un cambio de contexto de ejecución de código en base a prioridades.
- El Código que atiende la interrupción se denomina “rutina de servicio de interrupción” (ISR por sus siglas en inglés).
- Las interrupciones junto con sus prioridades permiten la gestión de tareas y forman la base de los sistemas operativos permitiendo los cambios de contexto.
- Las interrupciones pueden ser de hardware o software.
- Los Cortex- M tienen hardware dedicado para su gestión: el NVIC (Nested Vector Interrupt Controller)

Controlador NVIC

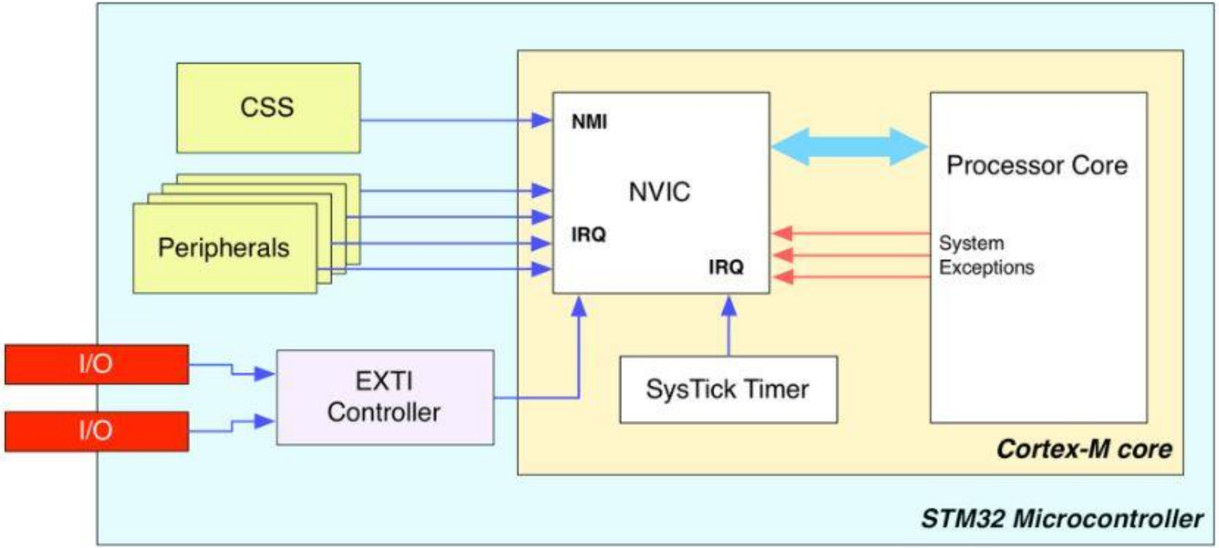


Figure 1: the relation between the NVIC controller, the Cortex-M core and the STM32 peripherals

Imagen extraída del libro “Mastering STM32” de Carmine Noviello.

Habilitación de interrupciones

Cuando un micro STM32 arranca, por default, solo están habilitadas las excepciones:

- Reset
- NMI: vinculada al CSS (clock security system). Es un periférico de autodiagnóstico que detecta fallas en el HSE.
- Hard Fault: excepción genérica relacionada a interrupciones de soft. Cuando otras excepciones están deshabilitadas actúa como recolector.

El resto de la excepciones e interrupciones se debe habilitar.
Para habilitar las IRQ la librería HAL dispone de la función:

```
void HAL_NVIC_EnableIRQ(IRQn_Type IRQn);
```

IRQn_Type es una enumeración de todas las excepciones e interrupciones del micro.

Habilitación de interrrupciones

Esta enumeración se define en la cmsis. Se encuentra en el archivo stm32f103xb.h

main.c | *stm32f1xx_it.c | stm32f103xb.h

```c

 /*!< Interrupt Number Definition */
typedef enum
{
/******  Cortex-M3 Processor Exceptions Numbers ***************************************************/
  NonMaskableInt_IRQn         = -14,    /*!< 2 Non Maskable Interrupt                             */
  HardFault_IRQn              = -13,    /*!< 3 Cortex-M3 Hard Fault Interrupt                     */
  MemoryManagement_IRQn       = -12,    /*!< 4 Cortex-M3 Memory Management Interrupt              */
  BusFault_IRQn               = -11,    /*!< 5 Cortex-M3 Bus Fault Interrupt                      */
  UsageFault_IRQn             = -10,    /*!< 6 Cortex-M3 Usage Fault Interrupt                    */
  SVCall_IRQn                 = -5,     /*!< 11 Cortex-M3 SV Call Interrupt                       */
  DebugMonitor_IRQn           = -4,     /*!< 12 Cortex-M3 Debug Monitor Interrupt                 */
  PendSV_IRQn                 = -2,     /*!< 14 Cortex-M3 Pend SV Interrupt                       */
  SysTick_IRQn                = -1,     /*!< 15 Cortex-M3 System Tick Interrupt                   */

/******  STM32 specific Interrupt Numbers *********************************************************/
  WWDG_IRQn                   = 0,      /*!< Window WatchDog Interrupt                            */
  PVD_IRQn                    = 1,      /*!< PVD through EXTI Line detection Interrupt            */
  TAMPER_IRQn                 = 2,      /*!< Tamper Interrupt                                     */
  RTC_IRQn                    = 3,      /*!< RTC global Interrupt                                 */
  FLASH_IRQn                  = 4,      /*!< FLASH global Interrupt                               */
  RCC_IRQn                    = 5,      /*!< RCC global Interrupt                                 */
  EXTI0_IRQn                  = 6,      /*!< EXTI Line0 Interrupt                                 */
  EXTI1_IRQn                  = 7,      /*!< EXTI Line1 Interrupt                                 */
  EXTI2_IRQn                  = 8,      /*!< EXTI Line2 Interrupt                                 */
  EXTI3_IRQn                  = 9,      /*!< EXTI Line3 Interrupt                                 */
  EXTI4_IRQn                  = 10,     /*!< EXTI Line4 Interrupt                                 */
  DMA1_Channel1_IRQn          = 11,     /*!< DMA1 Channel 1 global Interrupt                      */
  DMA1_Channel2_IRQn          = 12,     /*!< DMA1 Channel 2 global Interrupt                      */
  DMA1_Channel3_IRQn          = 13,     /*!< DMA1 Channel 3 global Interrupt                      */
  DMA1_Channel4_IRQn          = 14,     /*!< DMA1 Channel 4 global Interrupt                      */
  DMA1_Channel5_IRQn          = 15,     /*!< DMA1 Channel 5 global Interrupt                      */
  DMA1_Channel6_IRQn          = 16,     /*!< DMA1 Channel 6 global Interrupt                      */
  DMA1_Channel7_IRQn          = 17,     /*!< DMA1 Channel 7 global Interrupt                      */
  ADC1_2_IRQn                 = 18,     /*!< ADC1 and ADC2 global Interrupt                       */
  USB_HP_CAN1_TX_IRQn         = 19,     /*!< USB Device High Priority or CAN1 TX Interrupts       */
  USB_LP_CAN1_RX0_IRQn        = 20,     /*!< USB Device Low Priority or CAN1 RX0 Interrupts       */
  CAN1_RX1_IRQn               = 21,     /*!< CAN1 RX1 Interrupt                                   */
```

Habilitación de interrrupciones

- Esta disponible la función complementaria, para deshabilitar:

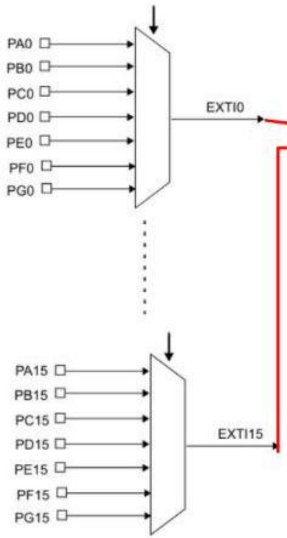
```
void HAL_NVIC_DisableIRQ(IRQn_Type IRQn);
```

- Es importante destacar que estas funciones son necesarias para habilitar o deshabilitar las interrupciones a nivel del controlador NVIC, pero por otra parte habrá que configurar el periférico correspondiente para que opere en modo interrupción.

- Por ejemplo, utilizando la función HAL_UART_Transmit_IT() Podemos configurar el periférico UART en modo interrupción, pero también será necesario habilitar la interrupción correspondiente llamando a la función HAL_NVIC_EnableIRQ().

Interrrupciones externas (EXTI)

- Los micros STM32 manejan un número variable de interrupciones externas conectadas al NVIC por medio del controlador EXTI. El número de líneas depende de la familia. En nuestro caso son 19 líneas:

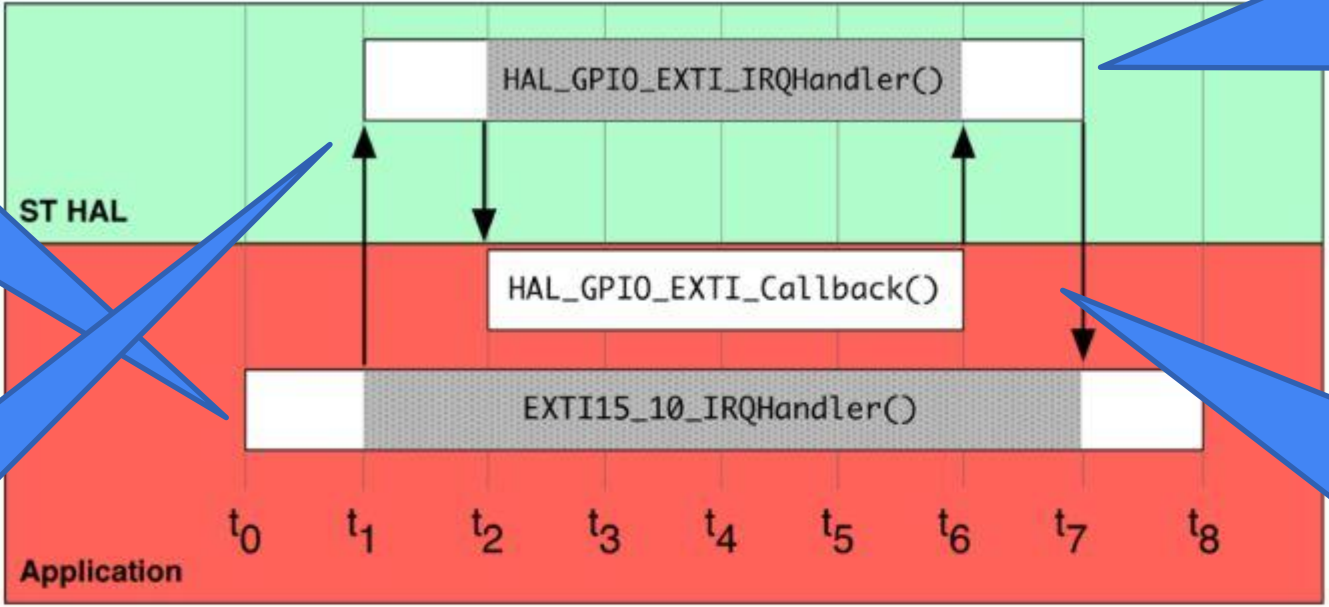


6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
41	48	settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4
42	49	settable	USBWakeup	USB wakeup from suspend through EXTI line interrupt	0x0000_00E8
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044

Gestión de interrupciones desde la HAL

1.Cuando se produce la interrupción se accede al EXTI15_10_IRQ Handler()

2. Desde el Handler de aplicación pasamos el control al HAL_GPIO_EXTI_IRQ Handler()



3. La HAL procesará la interrupción por nosotros. Invocará a la función de callback pasandolé la fuente de interrupción identificada

4. En la función de Callback() implementamos el código que dispara la interrupción

Imagen extraída del libro “Mastering STM32” de Carmine Noviello.

Gestión de interrupciones desde CubeMx

- Desde STM32CubeMX se pueden habilitar fácilmente las interrupciones y el mismo generará el código de la ISR.
- STM32CubeMX automáticamente agregará la ISR habilitada dentro del archivo src/stm32xxx_it.c.
- Solo necesitamos agregar la función de callback() correspondiente dentro de nuestra aplicación.

Ejemplo: Blink con Interrupción temporizada

Configurar un timer que genere una interrupción cada 200 ms. Programe el callback correspondiente para que invierta el estado del led onboard.

Ejemplo:

Configura callback
correspor

Parameters

- **htim:** TIM handle.
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

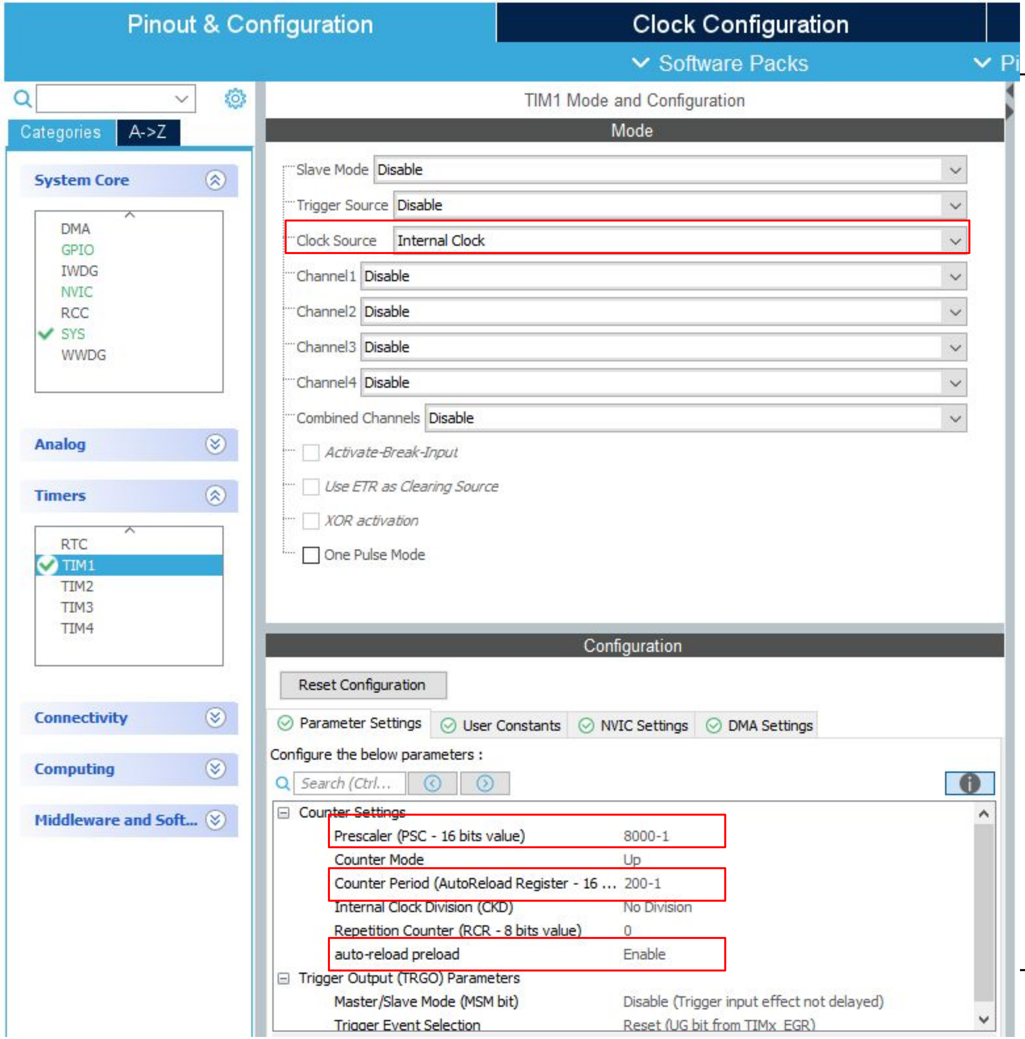
HAL_TIM_PeriodElapsedHalfCpltCallback

Function name

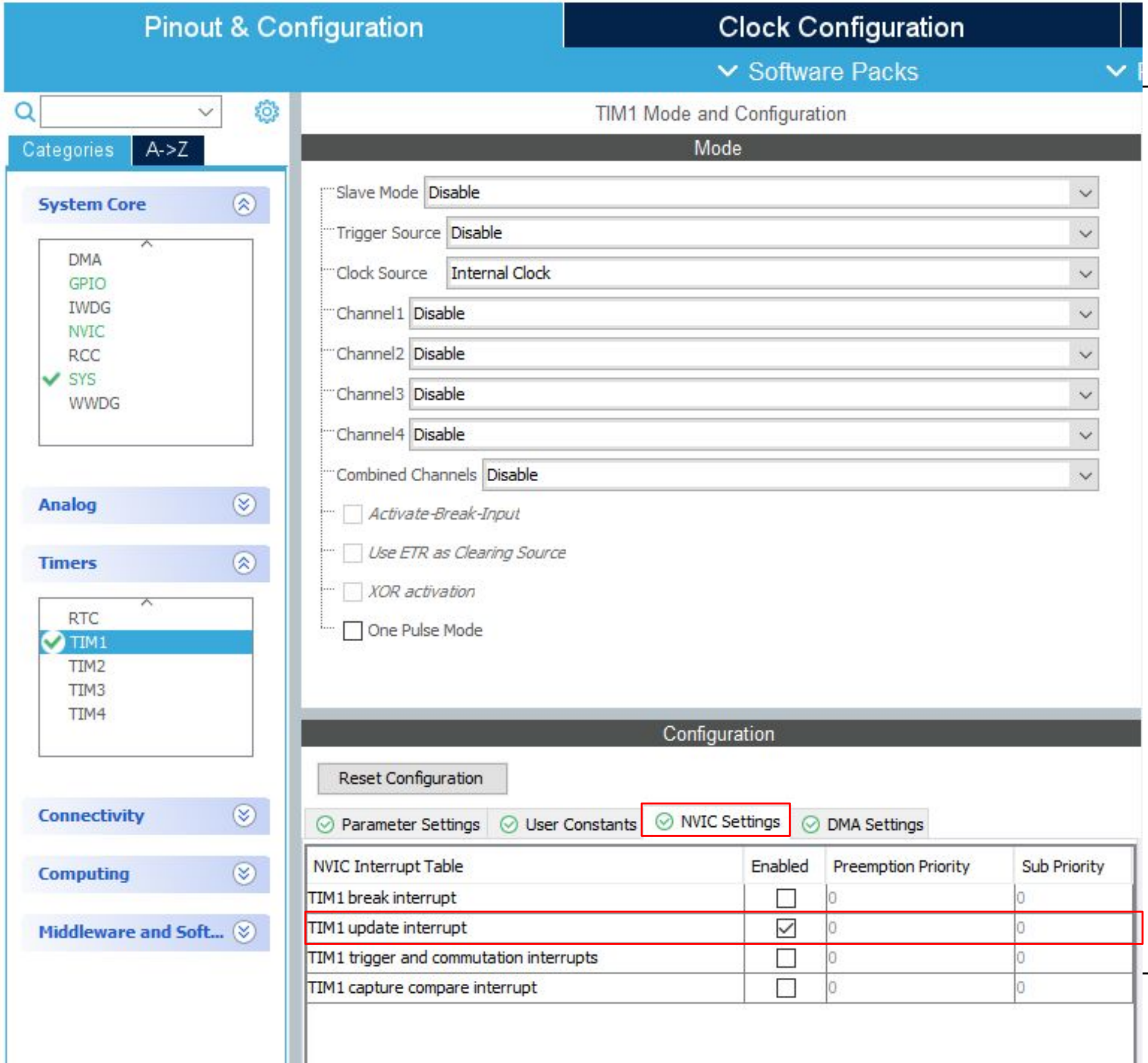
void HAL_TIM_PeriodElapsedHalfCpltCallback (TIM_HandleTypeDef * htim)

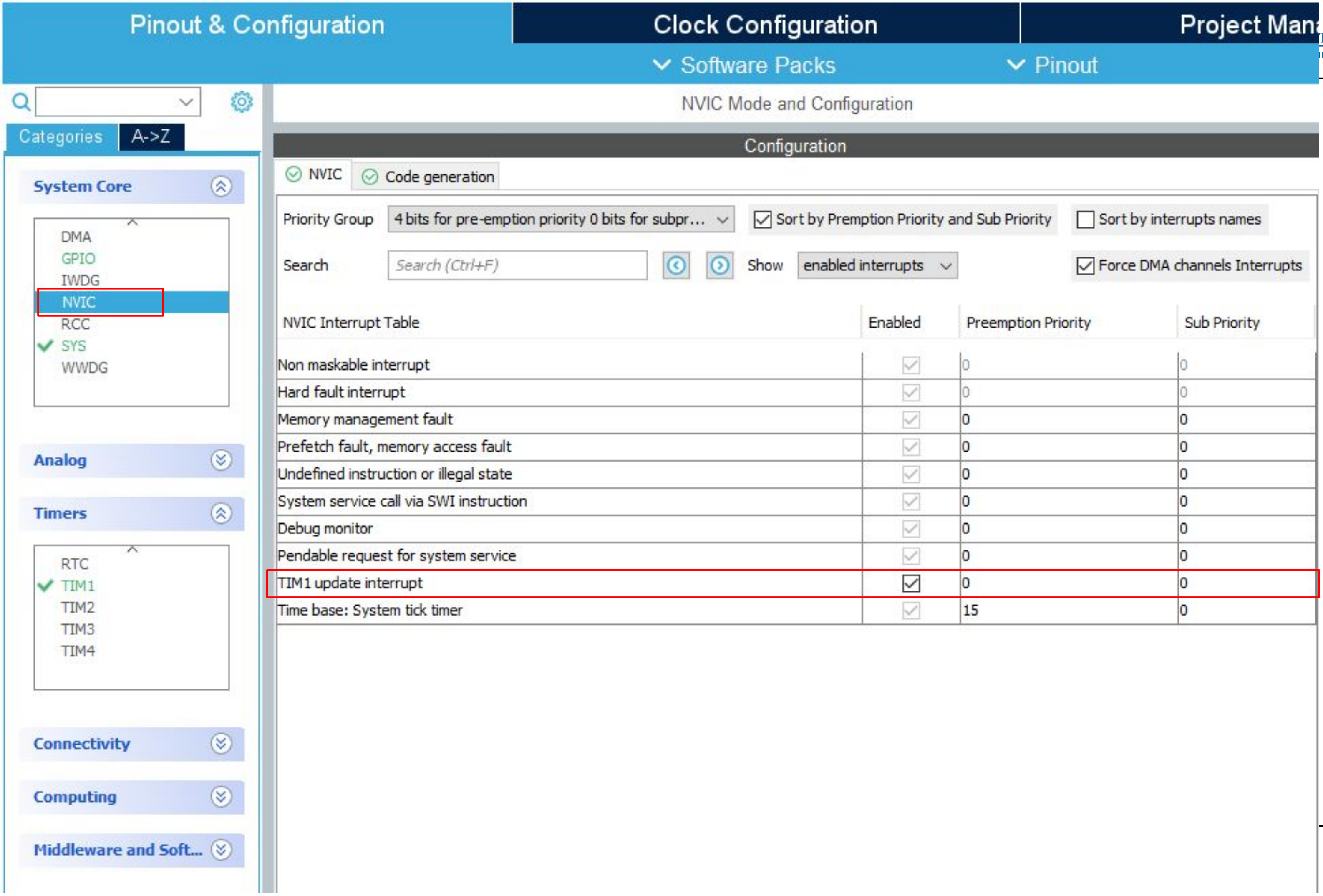
Function description

Period elapsed half complete callback in non-blocking mode.

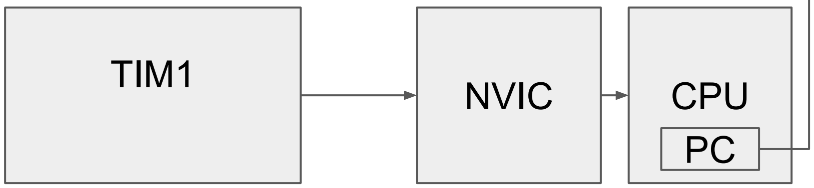


frecuencia = INT_CLK / (PRESCALER+1) / (AUTO_RELOAD +1)


Pinout & Configuration
Clock Configuration
Software Packs
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
RTC
TIM1
TIM2
TIM3
TIM4
Connectivity
Computing
Middleware and Soft...
TIM1 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 Disable
Channel3 Disable
Channel4 Disable
Combined Channels Disable
Activate-Break-Input
Use ETR as Clearing Source
XOR activation
One Pulse Mode
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
NVIC Interrupt Table
Enabled
Preemption Priority
Sub Priority
TIM1 break interrupt 0 0
TIM1 update interrupt 0 0
TIM1 trigger and commutation interrupts 0 0
TIM1 capture compare interrupt 0 0


Pinout & Configuration
Clock Configuration
Project Man
Software Packs
Pinout
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
RTC
TIM1
TIM2
TIM3
TIM4
Connectivity
Computing
Middleware and Soft...
NVIC Mode and Configuration
Configuration
NVIC
Code generation
Priority Group
4 bits for pre-emption priority 0 bits for subpr...
Sort by Premption Priority and Sub Priority
Sort by interrupts names
Search
Search (Ctrl+F)
Show
enabled interrupts
Force DMA channels Interrupts
NVIC Interrupt Table
Enabled
Preemption Priority
Sub Priority
Non maskable interrupt 0 0
Hard fault interrupt 0 0
Memory management fault 0 0
Prefetch fault, memory access fault 0 0
Undefined instruction or illegal state 0 0
System service call via SWI instruction 0 0
Debug monitor 0 0
Pendable request for system service 0 0
TIM1 update interrupt 0 0
Time base: System tick timer 15 0

```c
int main(void)
{
 ...
 HAL_TIM_Base_Start_IT(&htim1);
 while (1)
 { }
}
```

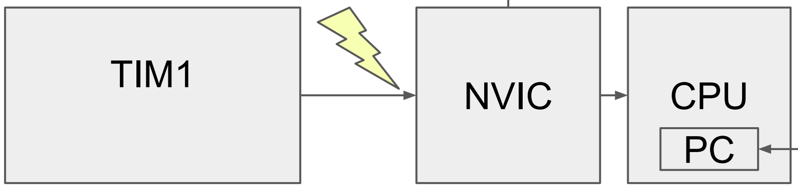


en main.c

```c
int main(void)
{
 ...
 HAL_TIM_Base_Start_IT(&htim1);
 while (1)
 { }
}
```

en startup_stmf103c8tx.s

```
/*********************************************
* The minimal vector table for a Cortex M3.  Note that the
proper constructs must be placed on this to ensure that it
ends up at physical address 0x0000.0000.
*********************************************/
 .section .isr_vector,"a",%progbits
 .type g_pfnVectors, %object
 .size g_pfnVectors, .-g_pfnVectors
g_pfnVectors:
 .word _estack
 .word Reset_Handler
 .word NMI_Handler
 .word HardFault_Handler
 .word MemManage_Handler
 .word BusFault_Handler
 .word UsageFault_Handler
 ...
 .word EXTI9_5_IRQHandler
 .word TIM1_BRK_IRQHandler
 .word TIM1_UP_IRQHandler
 ...
```



en main.c

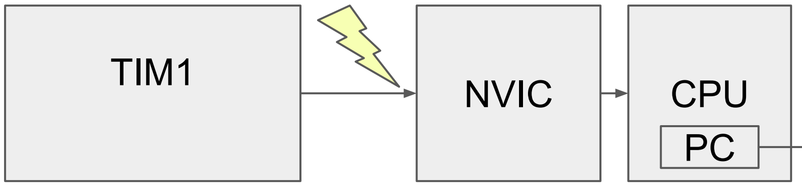
```c
int main(void)
{
 ...
 HAL_TIM_Base_Start_IT(&htim1);
 while (1)
 { }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{ HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);}
```

en stm32f1xx_it.c

```c
void TIM1_UP_IRQHandler(void)
{
 /* USER CODE BEGIN TIM1_UP_IRQn 0 */
 /* USER CODE END TIM1_UP_IRQn 0 */
 HAL_TIM_IRQHandler(&htim1);
 /* USER CODE BEGIN TIM1_UP_IRQn 1 */
 /* USER CODE END TIM1_UP_IRQn 1 */
}
```

en stm32f1xx_hal_tim.c

```c
void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim)
{...
HAL_TIM_PeriodElapsedCallback(htim);
...
}
```



```c
int main(void)
{
 ...
 HAL_TIM_Base_Start_IT(&htim1);
 while (1)
 { }
}
```