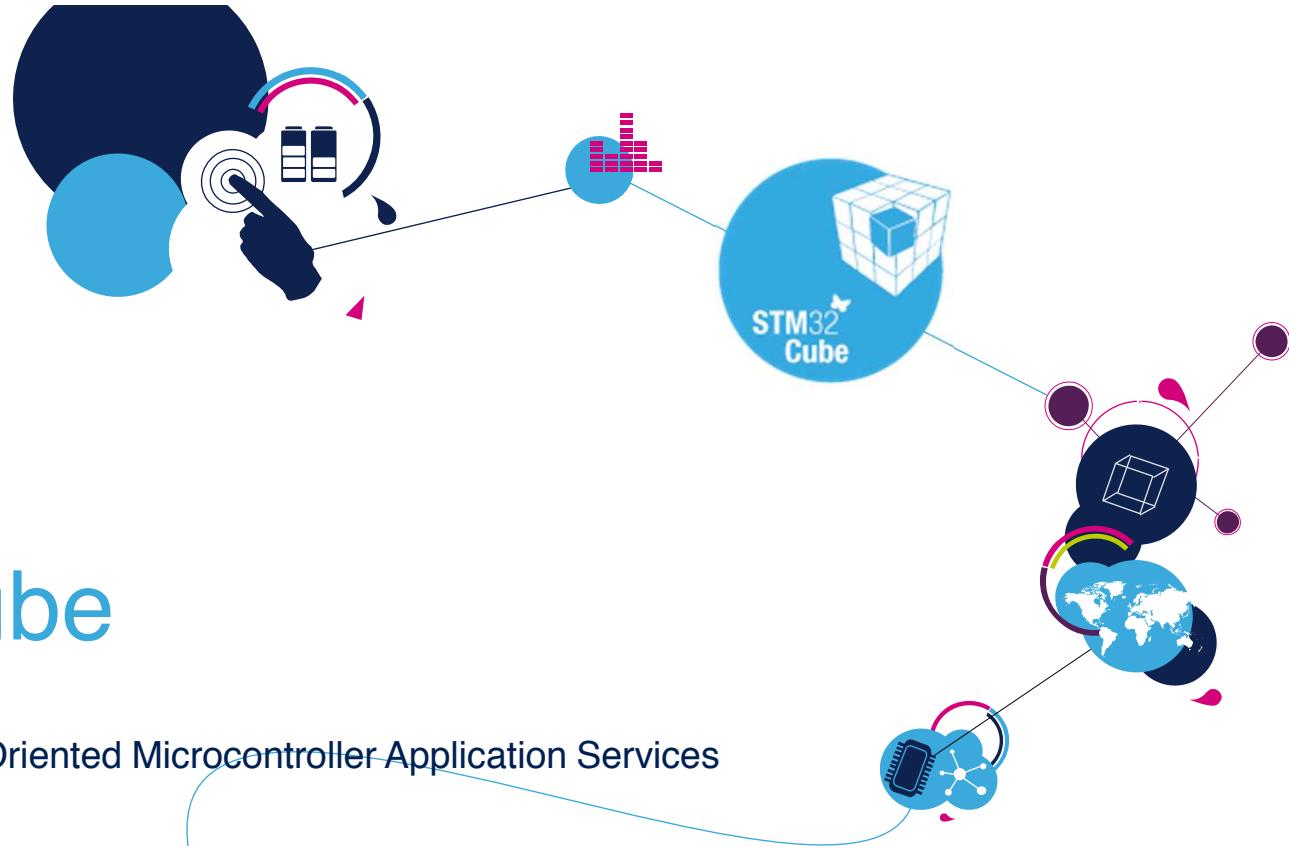


STM32Cube

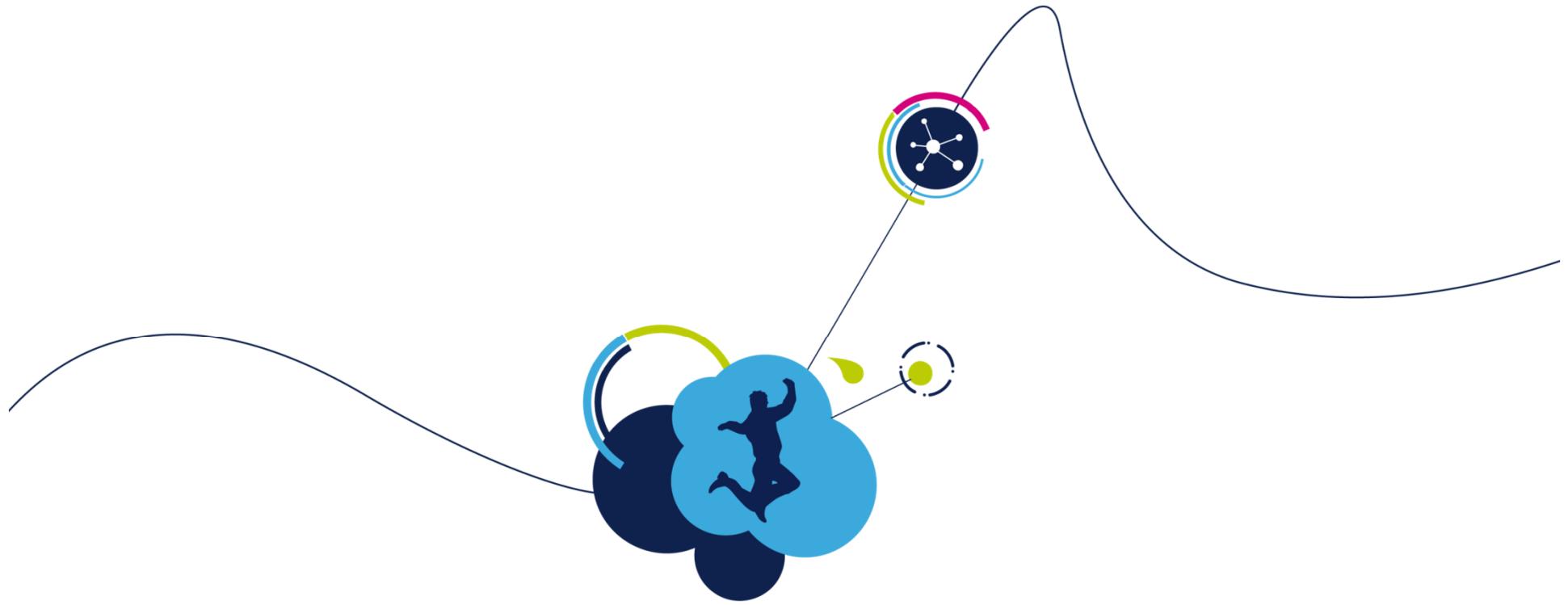
T.O.M.A.S – Technically Oriented Microcontroller Application Services
v0.01



- Installation of STM32CubeMX
- Fast CubeMX features
- HAL overview
- Basic of CubeMX tool and HAL basics
 - CubeMX configuration and project generation
 - First HAL project generated by CubeMX
 - Project structure
- Interrupts in CubeMX and HAL
- DMA in CubeMX and HAL
- Communication peripheries in CubeMX and HAL

- FreeRTOS
 - How it works on STM32
 - Memory management
 - Tasks
 - Queues
 - Semaphores
 - Multiplexers
 - Software timers
 - Hooks

- USB
 - How work on STM32 and with Cube HAL
 - Simple CubeMX project generation
 - USB class with CubeMX drivers
- Ethernet
 - Simple server/client example



STM32CubeMX installation

A

CubeMX install

6

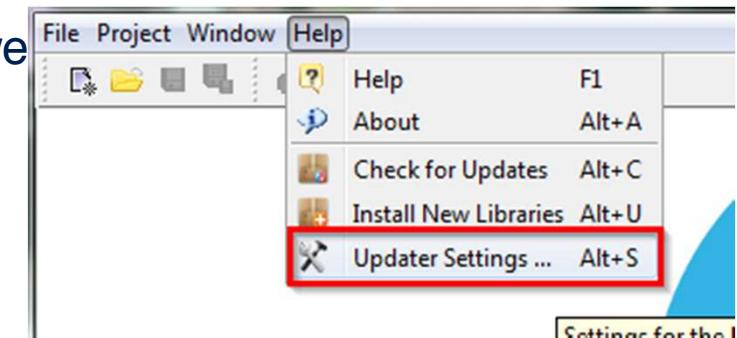
- CubeMX tool
 - http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242?s_searchtype=partnumber
- The CubeMX tool need java
 - Please check if you have last java on your pc, for sure 32bit and 64bit version
- Optionally you can download the Cube packages for STM32 device if you don't want to download them thro CubeMX
 - [STM32CubeL0](#)
 - [STM32CubeL1](#)
 - [STM32CubeF0](#)
 - STM32CubeF1
 - [STM32CubeF2](#)
 - [STM32CubeF3](#)
 - [STM32CubeF4](#)

A

CubeMX install

7

- Install the CubeMX
- After installation run CubeMX
- In case you download the package from web where they need to be stored
- MENU>Help>Updater Settings...
- You will see where is the repository folder
 - Default is C:/User/Acc_name/STM32Cube/Repository/
- You need to download STM32 packages into this folder

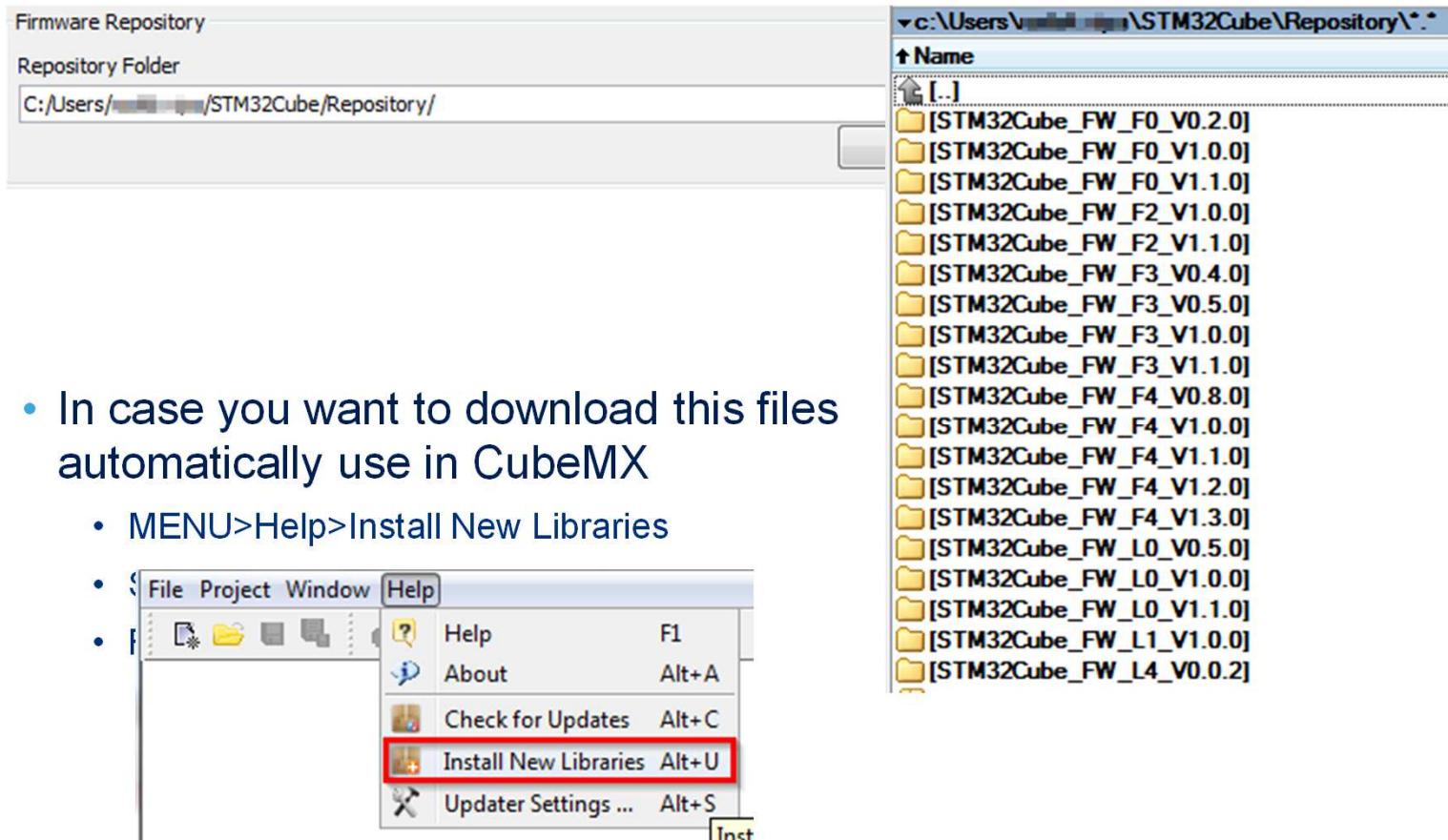


A

CubeMX install

8

- The comparison of the CubeMX repository settings and structure in this folder



A

CubeMX install

9

- For the code generation the CubeMX use the package from the Repository folder
- The CubeMX can generate the code for some GUI
 - Keil
 - IAR
 - Atollic
- For the debugging is necessity to have the ST-Link drivers
 - [STSW-LINK003](#) driver for Win XP/Vista/7
 - [STSW-LINK006](#) driver for Win 8
- For driver installation you will need the **Admin rights** on your PC



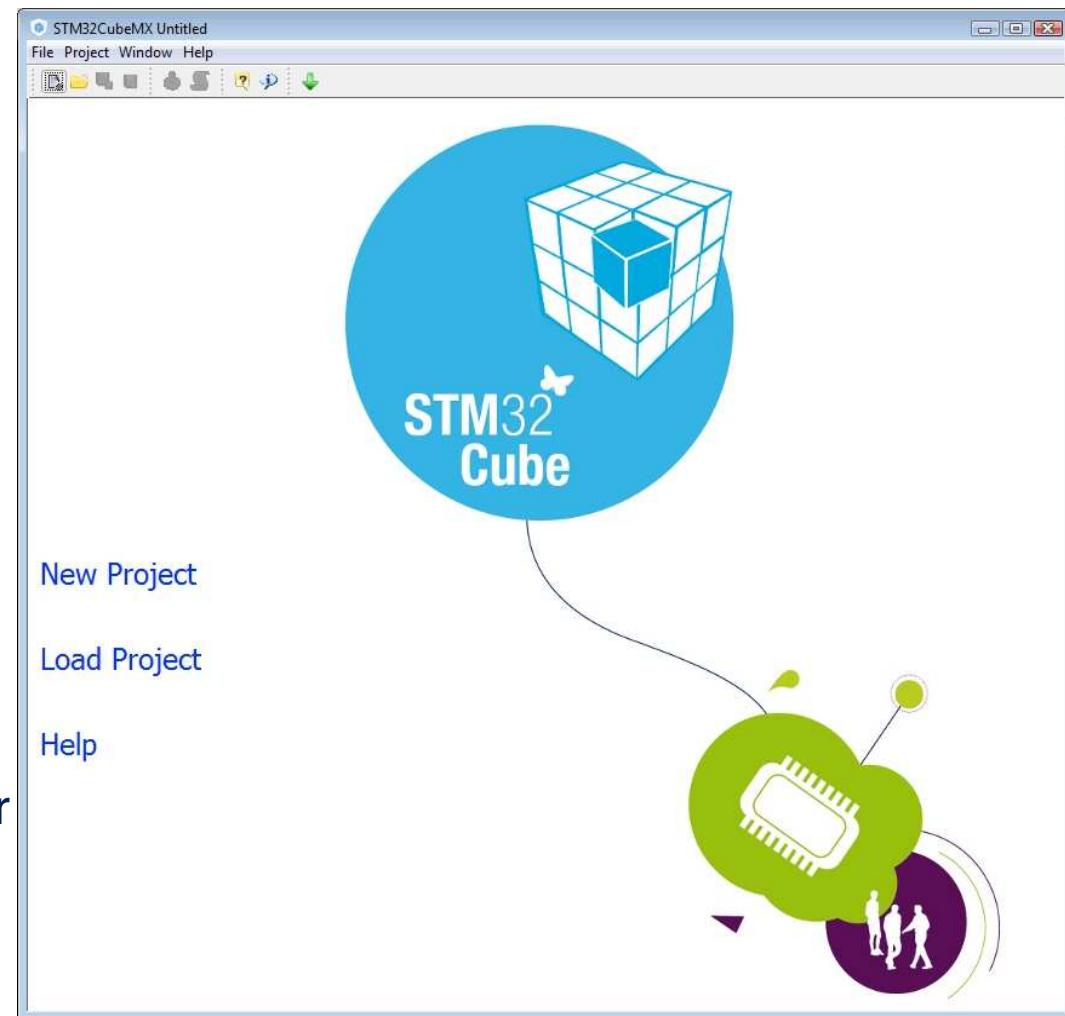
STM32CubeMX presentation

STM32Cube: STM32CubeMX

11

Step by step:

- MCU selector
- Pinout configuration
- Clock tree initialization
- Peripherals and middleware parameters
- Code generation
- Power consumption calculator

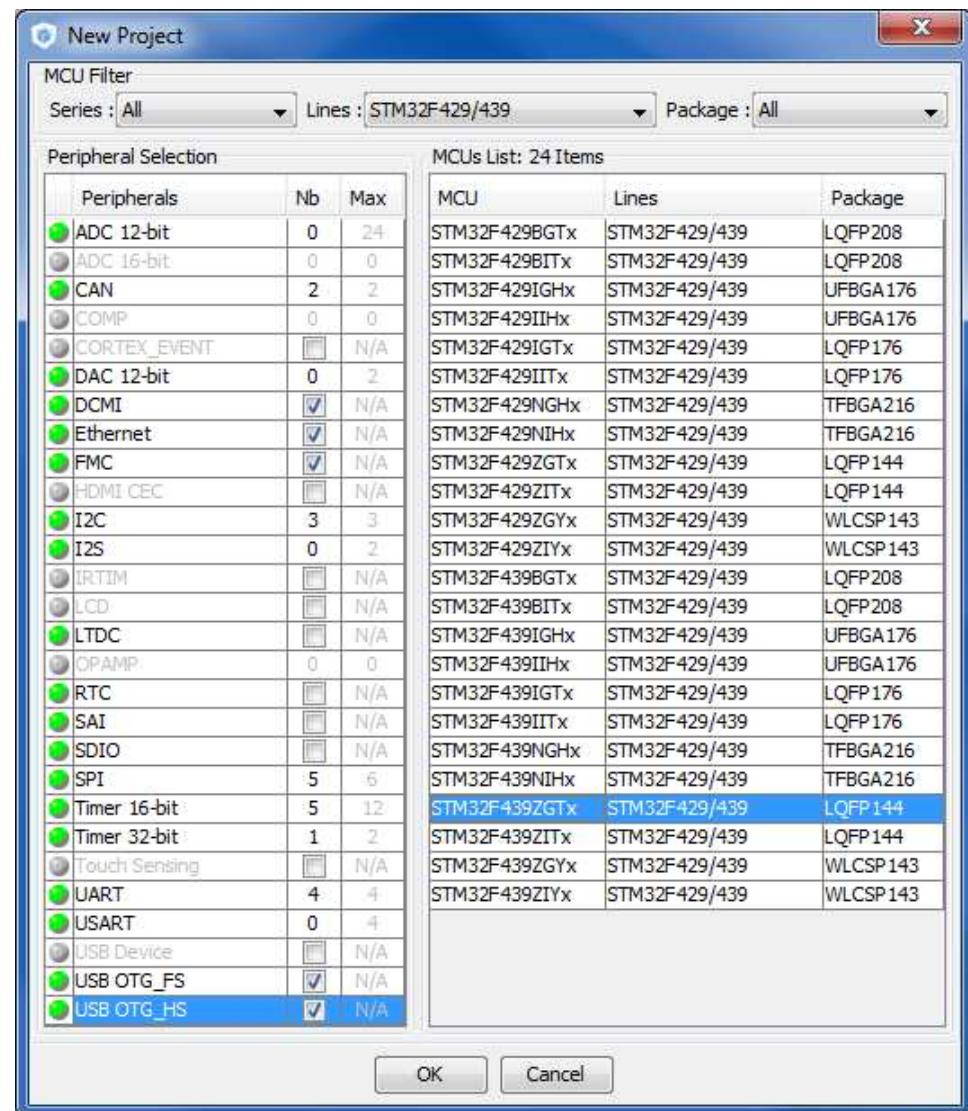


STM32CubeMX: MCU Selector

12

Easy Optional filtering:

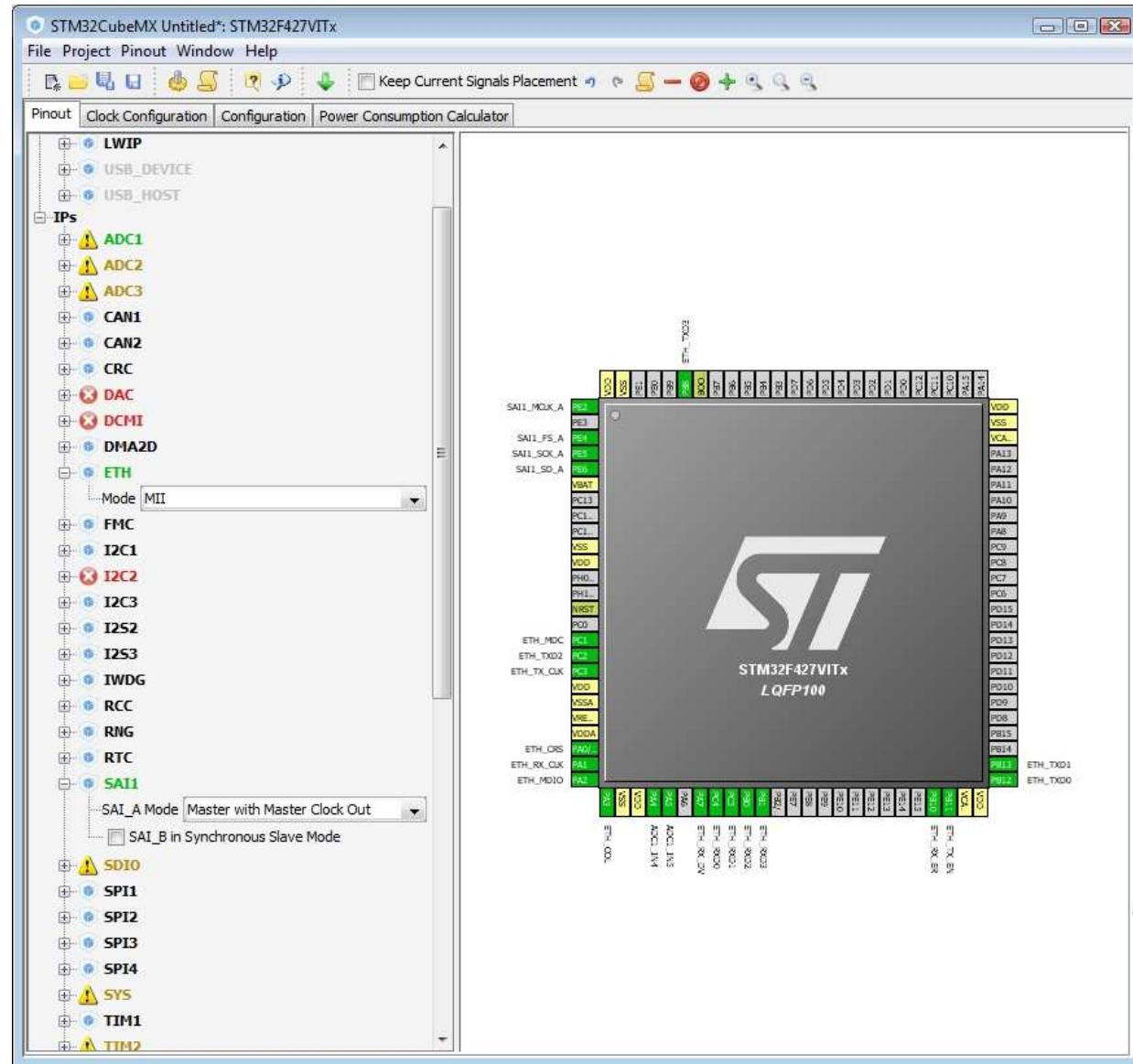
- Series
- Line
- Package
- Peripherals



STM32CubeMX: Pinout configuration

13

- Pinout from:
 - Peripheral tree
 - Manually
- Automatic signal remapping
- Management of dependencies between peripherals and/or middleware (FatFS, LWIP, ...)



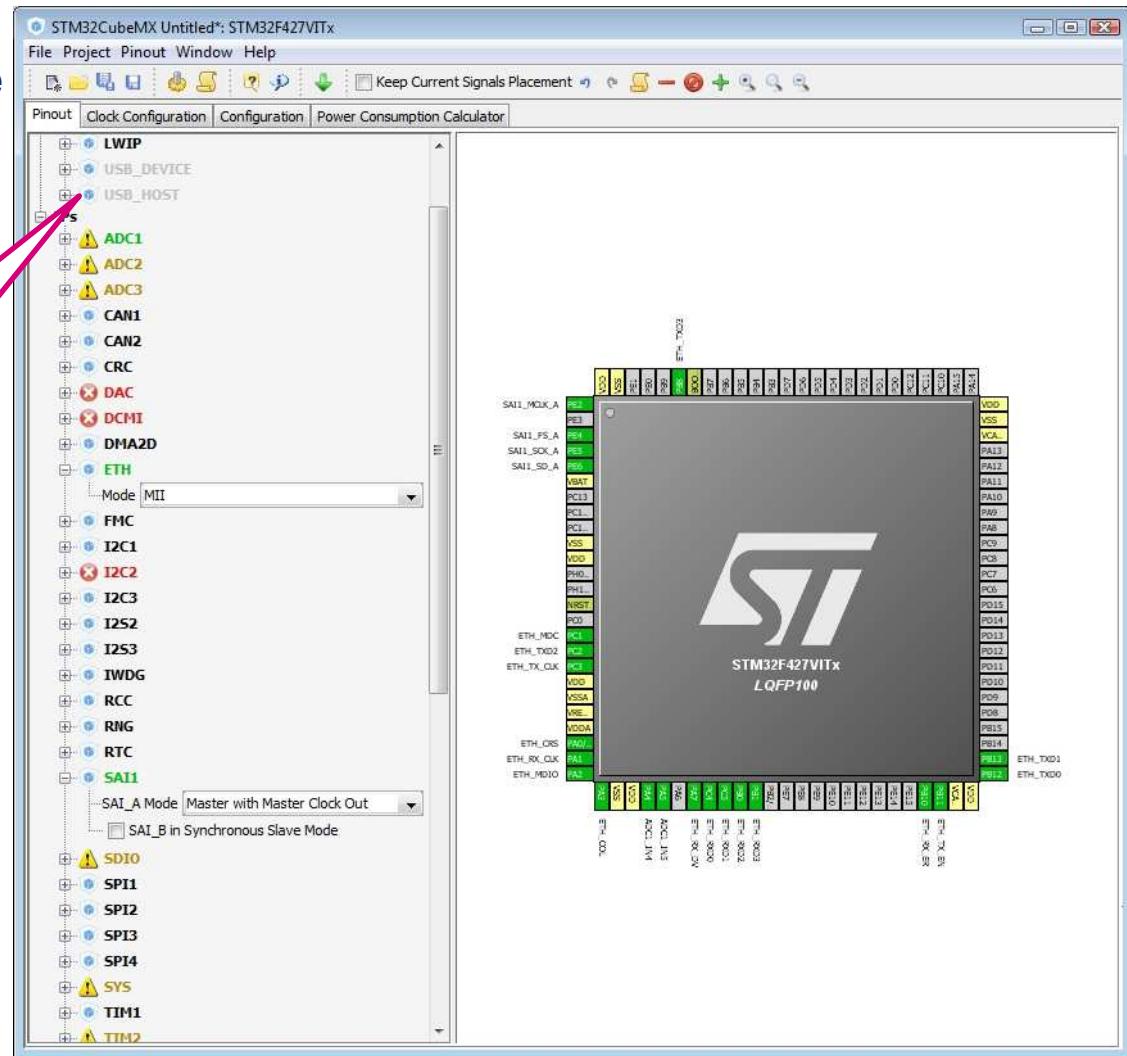
STM32CubeMX: Pinout configuration

14

- Different possible states for a peripheral modes

- Dimmed:
the mode is not available because
it requires another mode to be set
(just put the mouse on top of the
dimmed mode to see why)

Dimmed:
The additional periphery must be selected

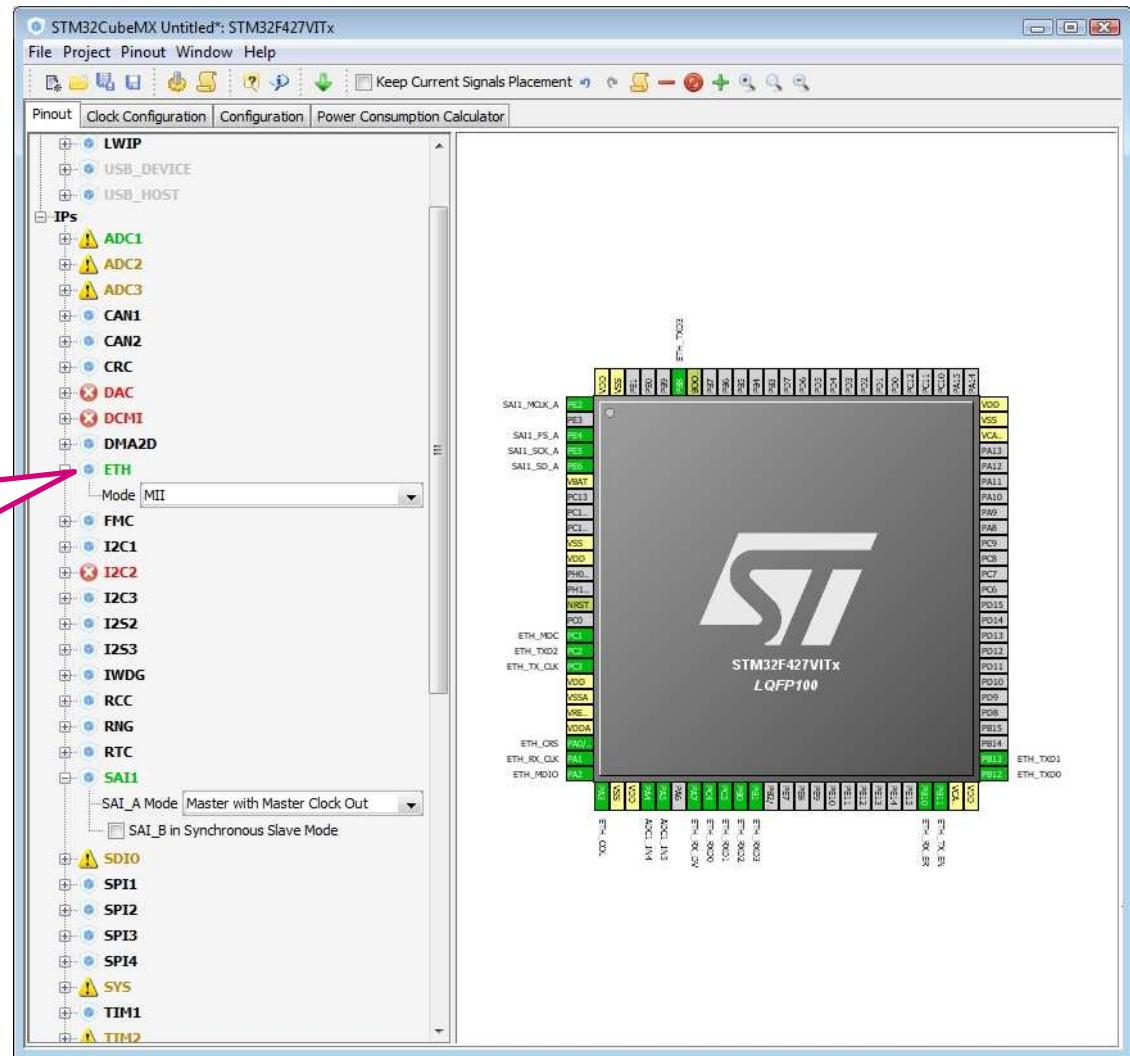


STM32CubeMX: Pinout configuration

15

- Different possible states for a peripheral modes
 - Green:
Periphery is assigned to pinout

Green:
Periphery will
be functional



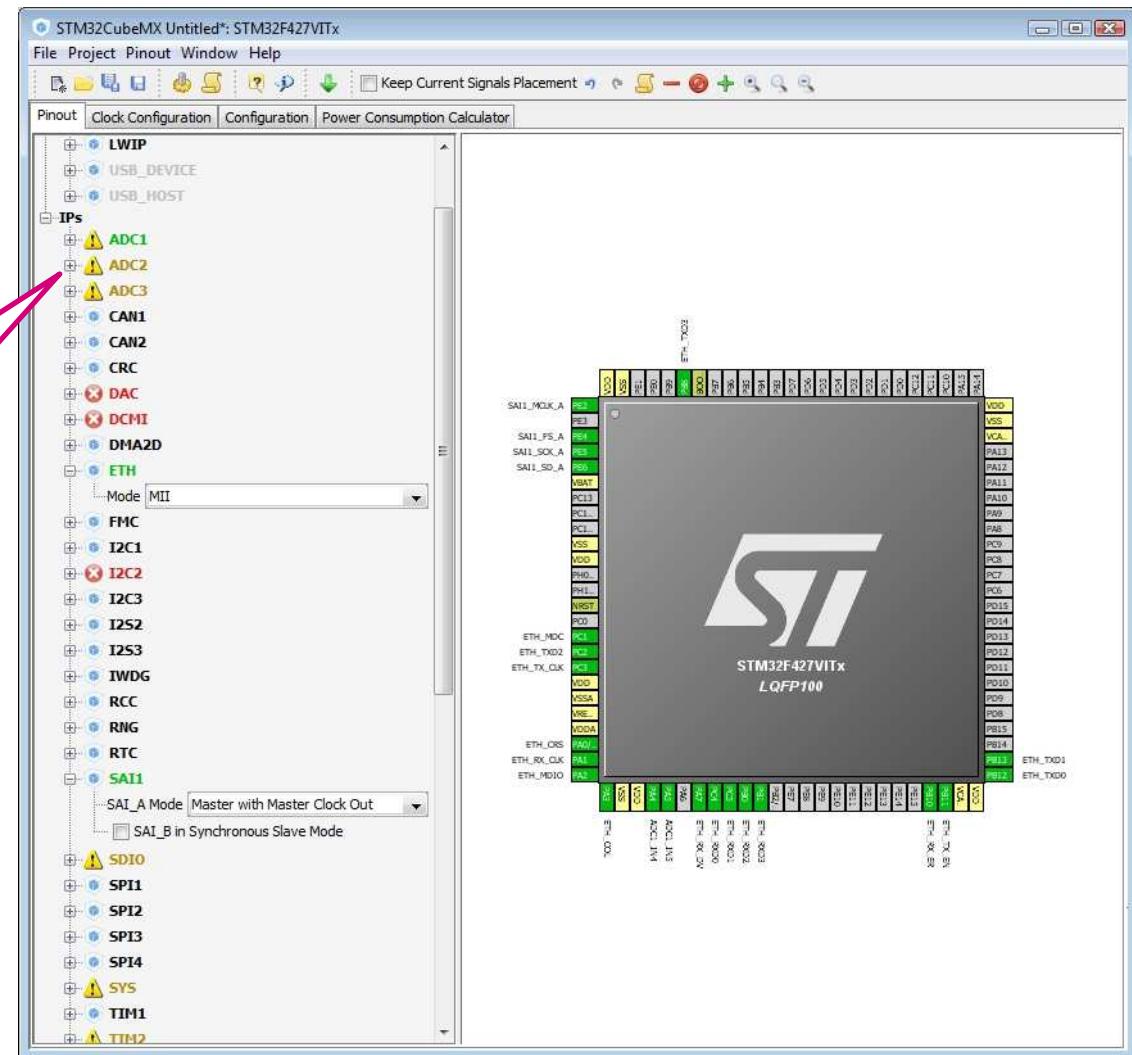
STM32CubeMX: Pinout configuration

16

- Different possible states for a peripheral modes

- Yellow:
Only some functionalities of periphery can be used

Yellow:
On ADC only
some channels
can be used



STM32CubeMX: Pinout configuration

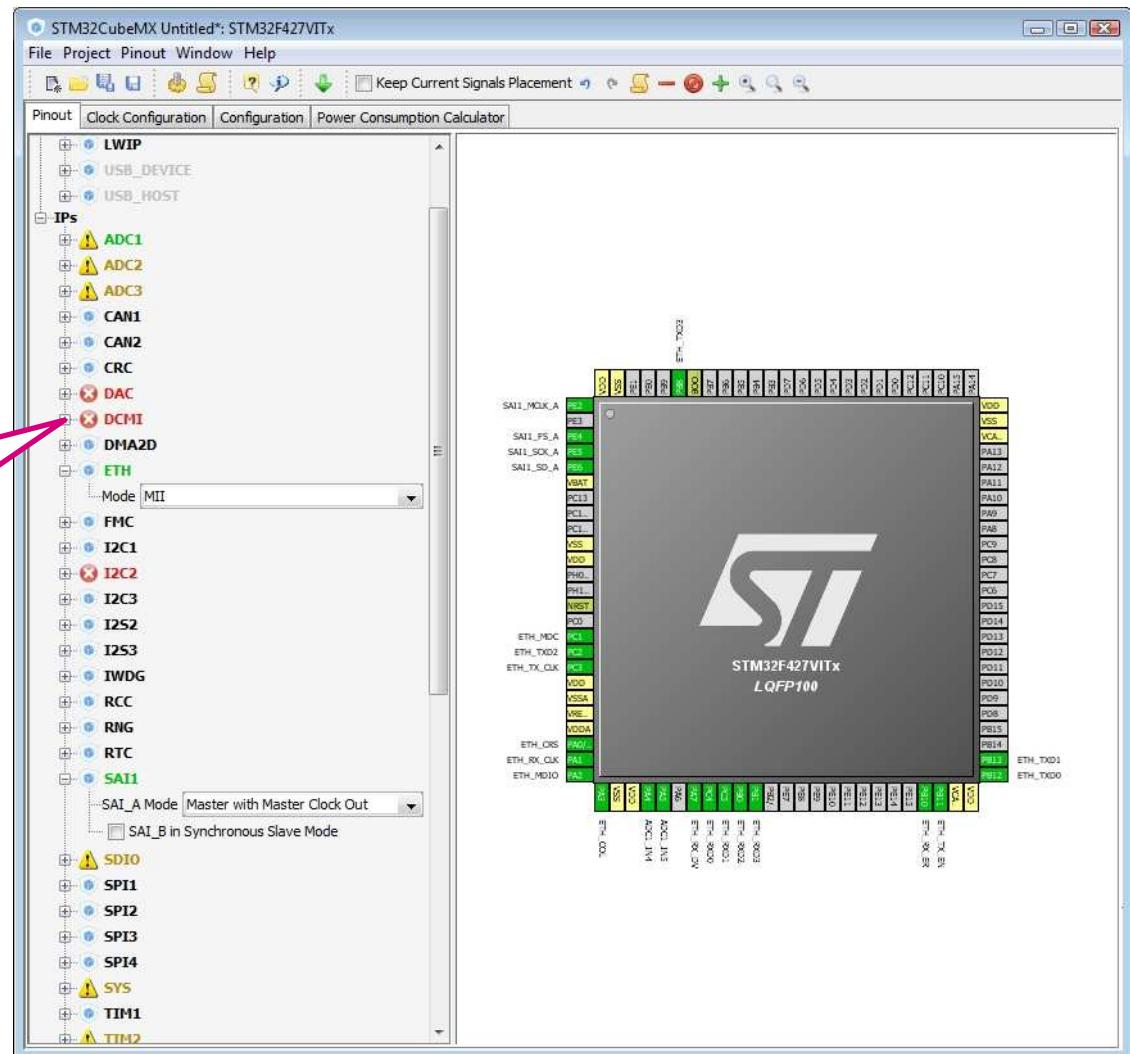
17

- Different possible states for a peripheral modes

- Red:

Signals required for this mode
can't be mapped on the pinout
(see tooltip to see conflicts)

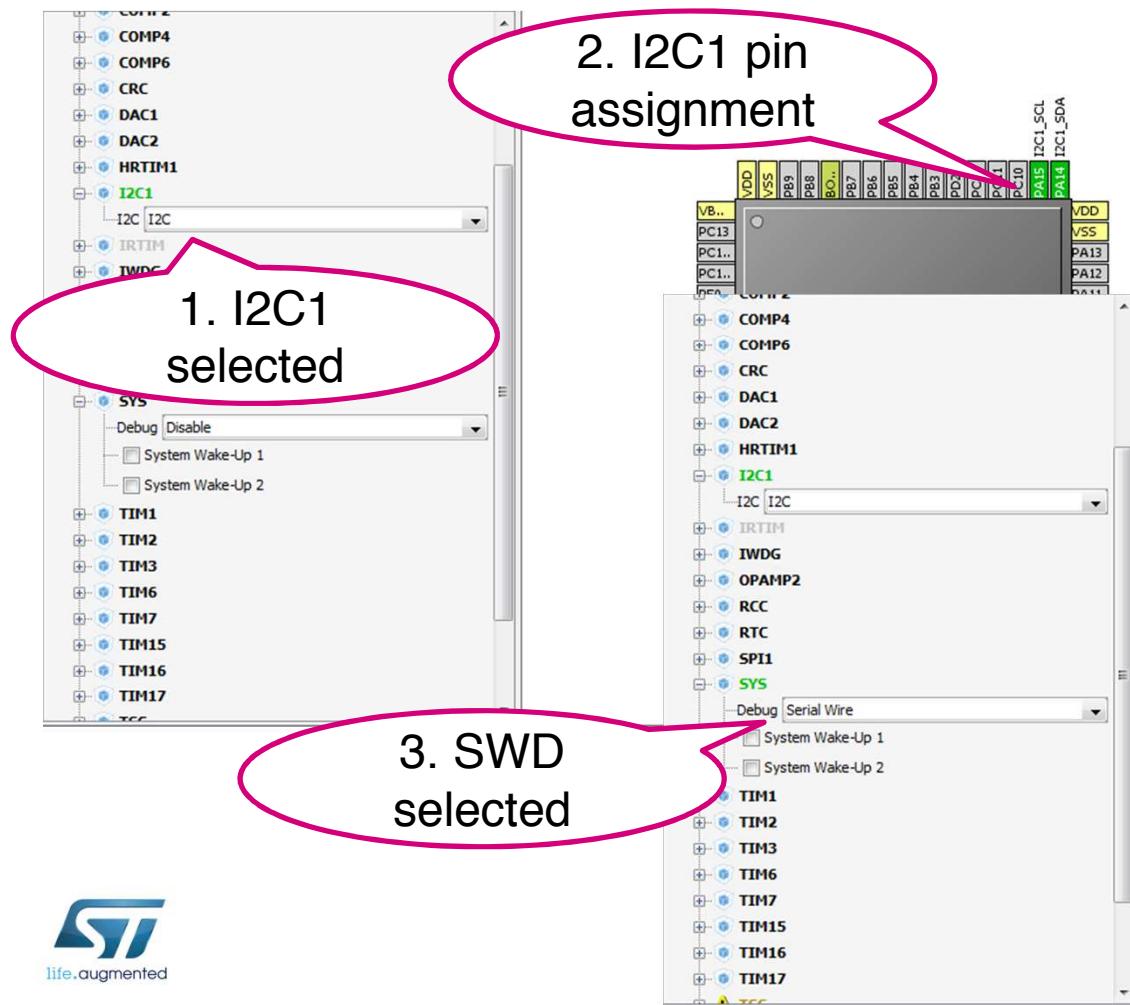
Red:
Periphery
cannot be used
in this pinout
setup



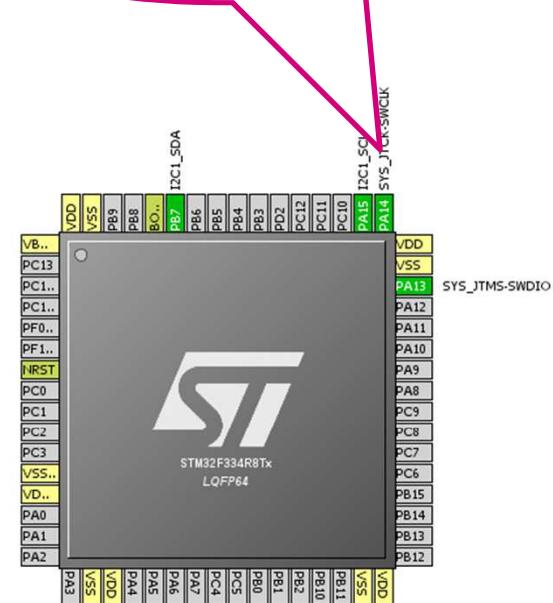
STM32CubeMX: Pinout configuration

18

- Keep User Placement renamed to Keep Current Signal Placement and is unchecked by default



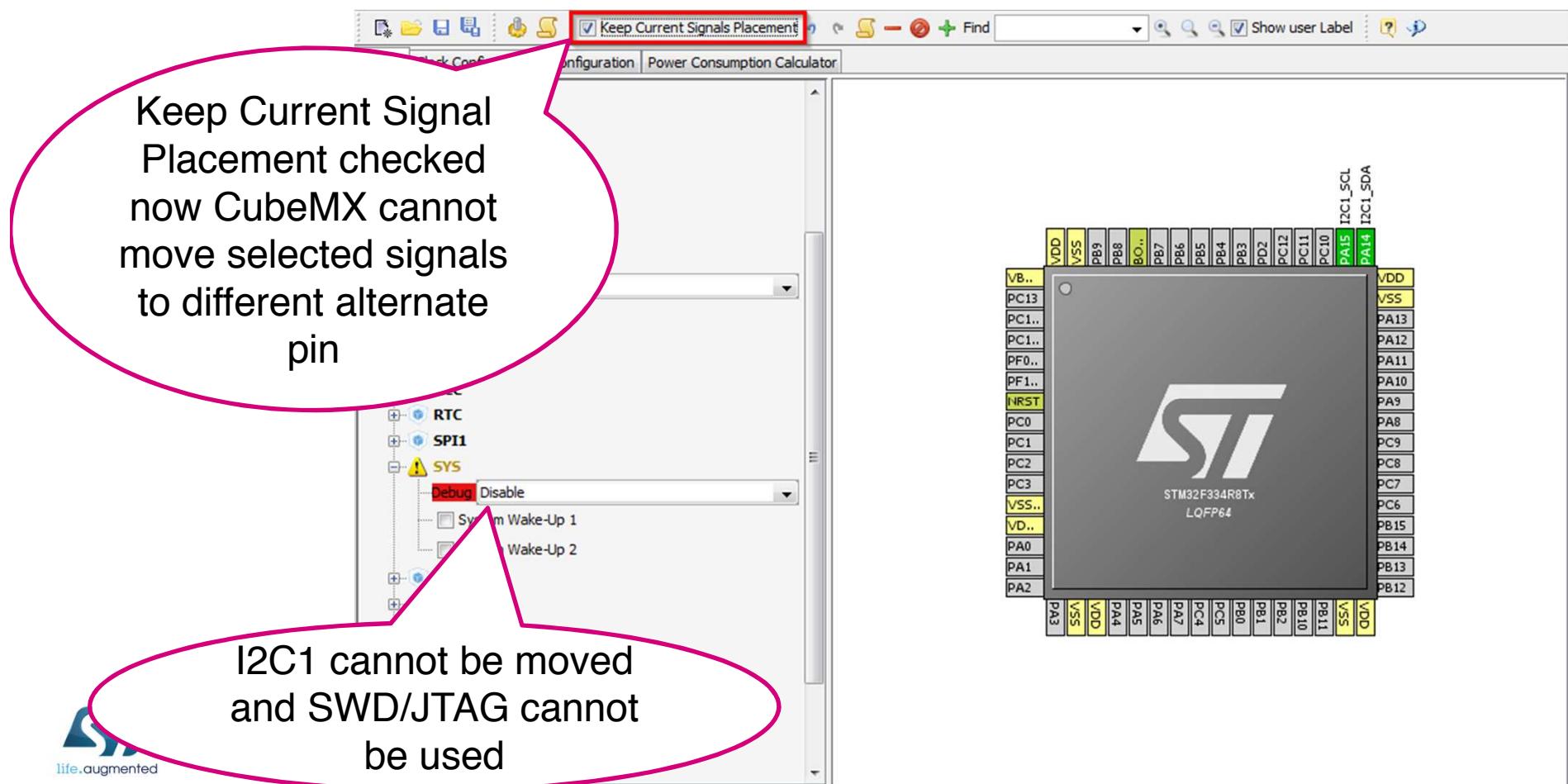
4. Pin conflict
between I2C1 and
SWD. I2C1_SDA
moved to alternative
position



STM32CubeMX: Pinout configuration

19

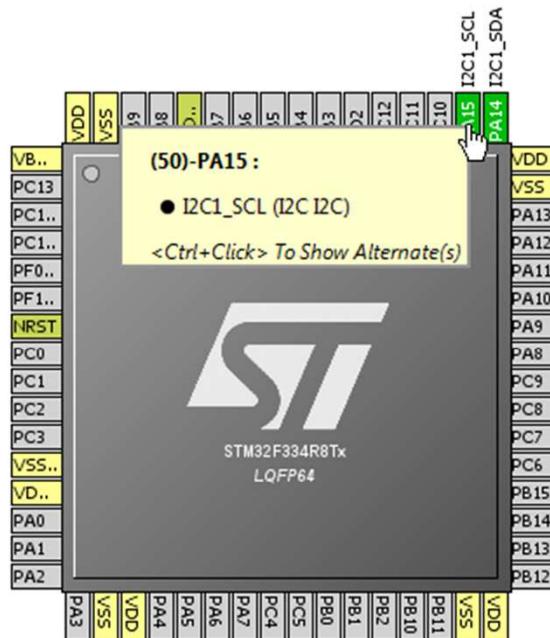
- Keep User Placement renamed to Keep Current Signal Placement and is unchecked by default



STM32CubeMX: Pinout configuration

20

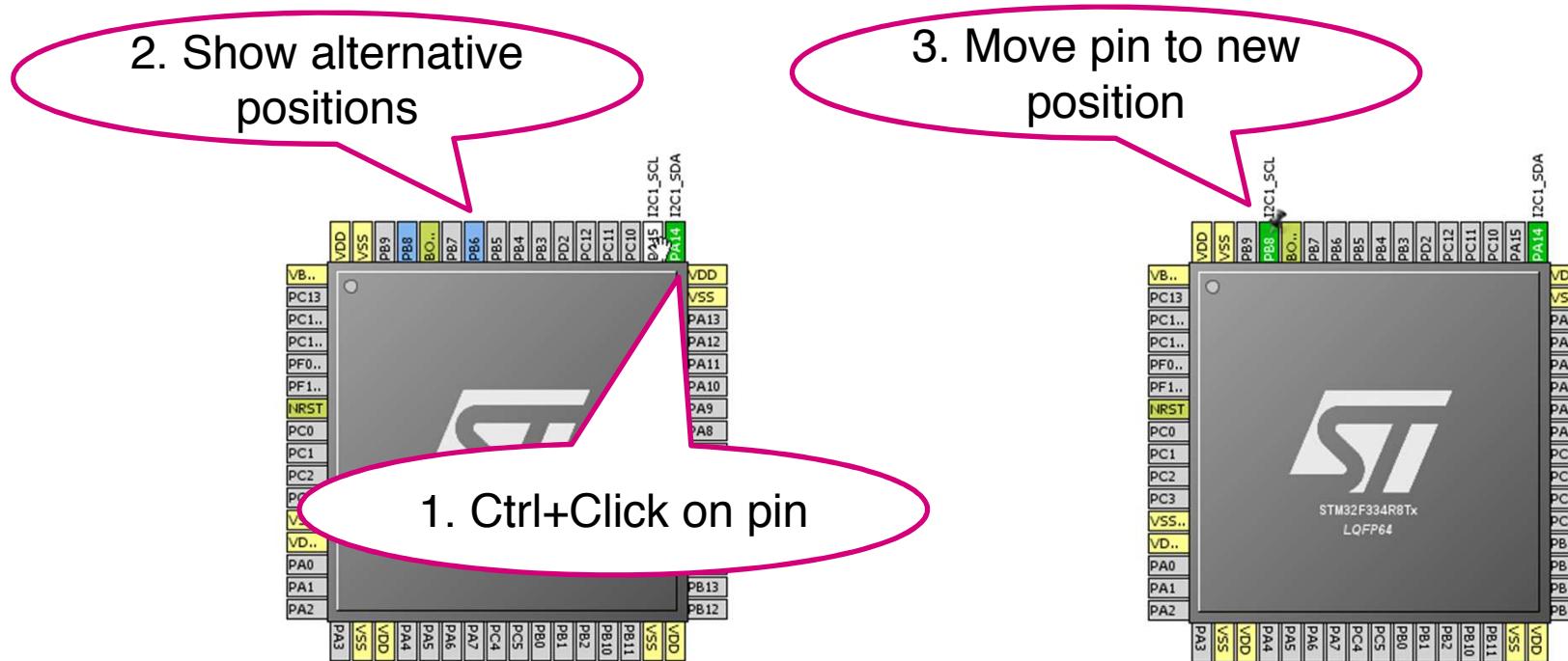
- Signals can be set/moved directly from the pinout view



STM32CubeMX: Pinout configuration

21

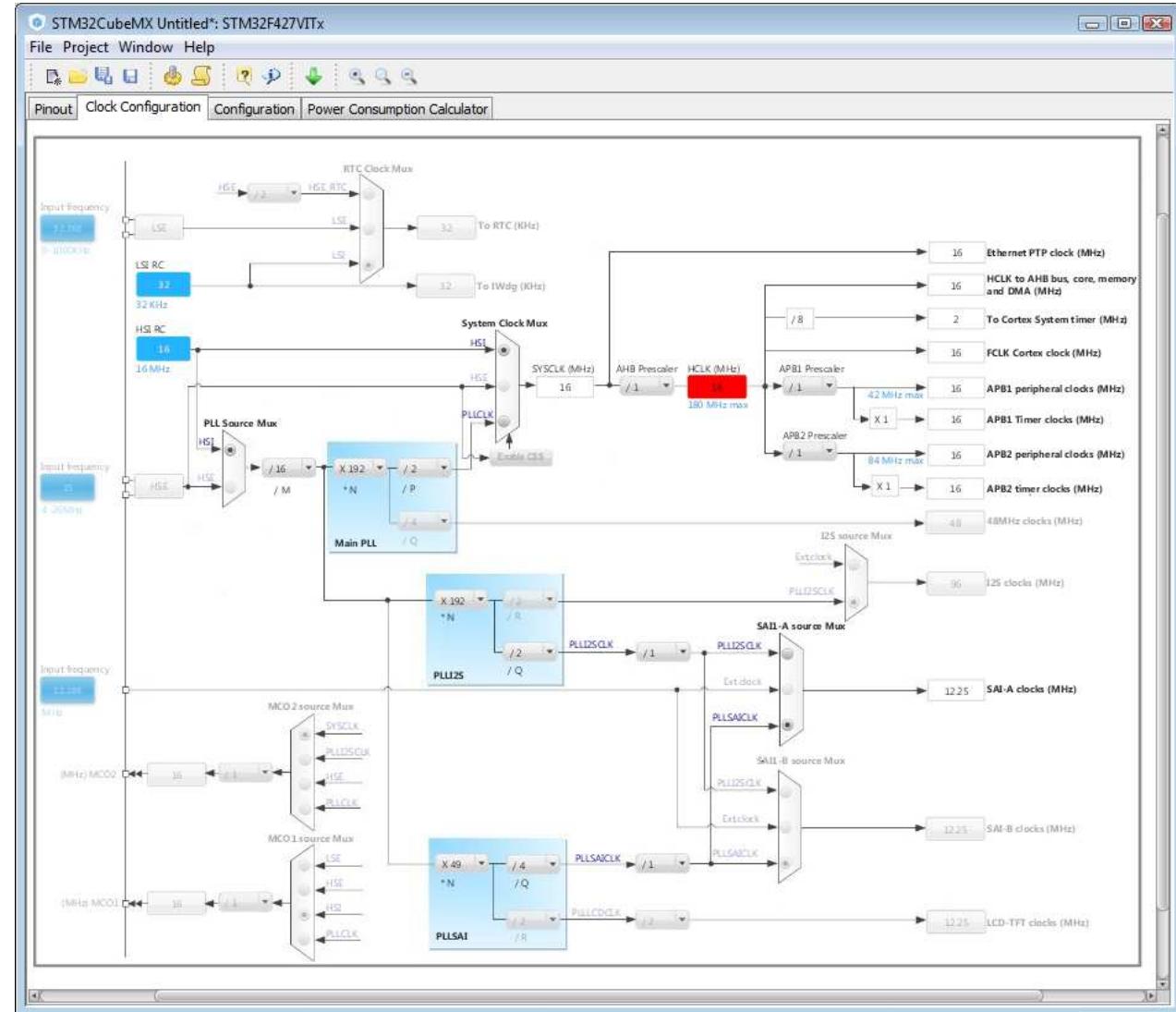
- Signals can be set/moved directly from the pinout view
 - To see alternate pins for a signal Ctrl+Click on the signal, you can then drag and drop the signal to the new pin (keep pressing the Ctrl key)



STM32CubeMX: Clock tree

22

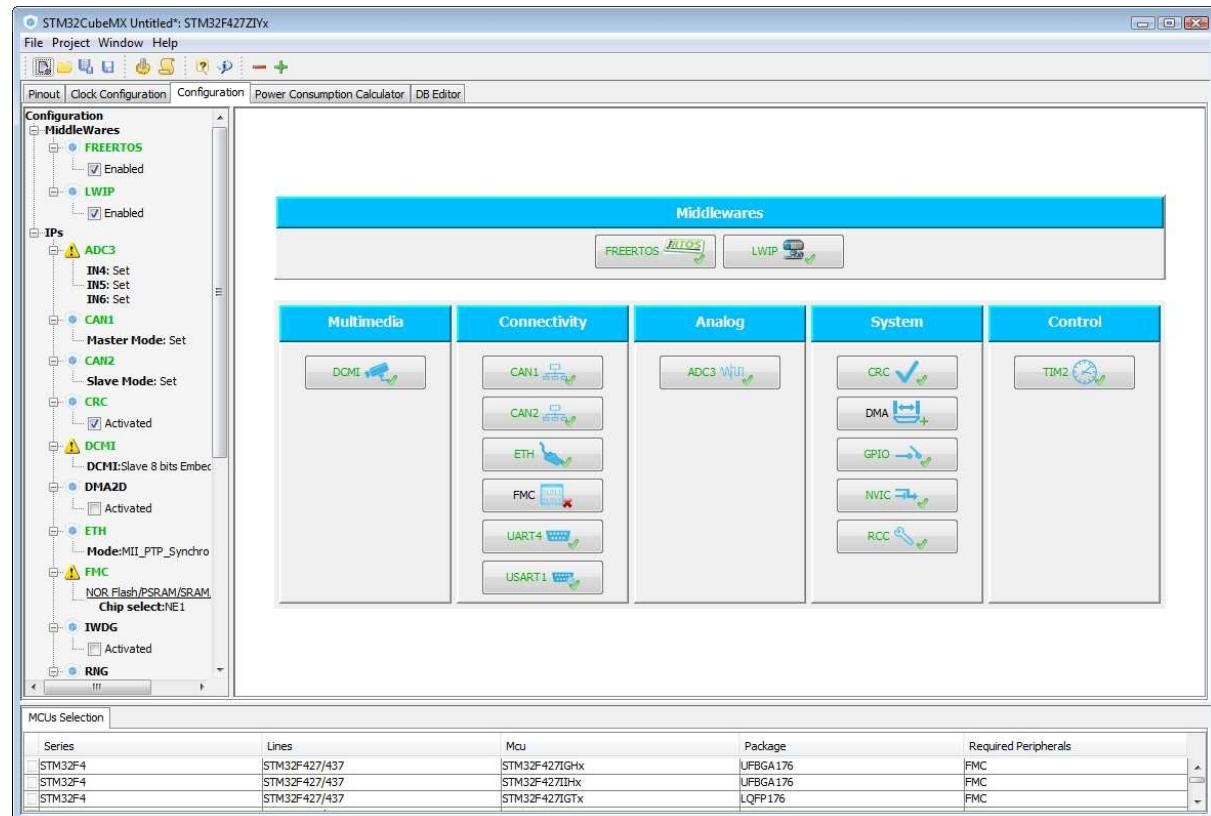
- Immediate display of all clock values
- Management of all clock constraints
- Highlight of errors



STM32CubeMX: Peripheral and middleware configuration

23

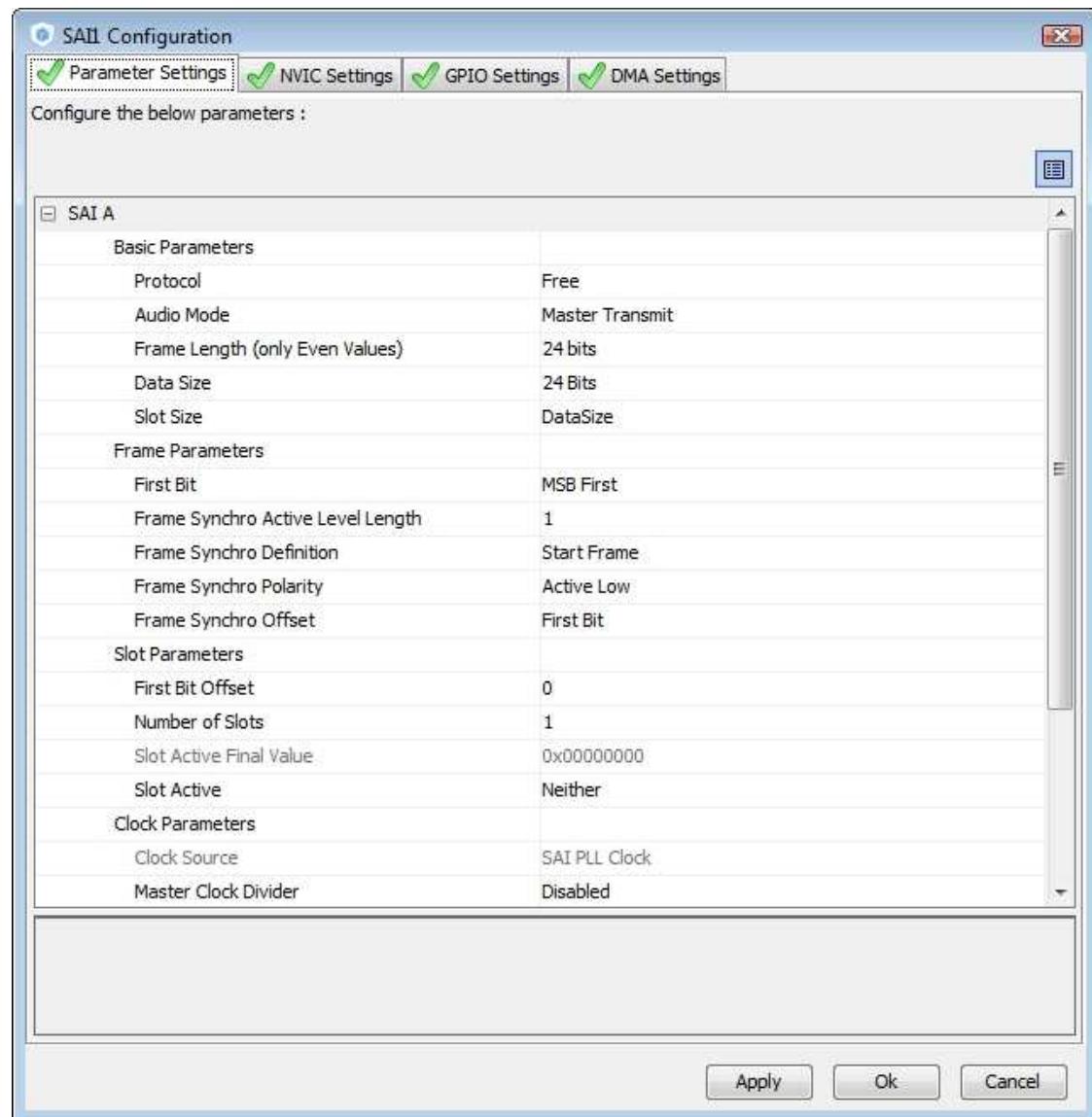
- Global view of used peripherals and middleware
- Highlight of configuration errors
 - + Not configured
 - ✓ OK
 - ✗ Error
- Read only tree view on the left with access to IPs / Middleware having no impact on the pinout



STM32CubeMX: Peripheral and middleware configuration

24

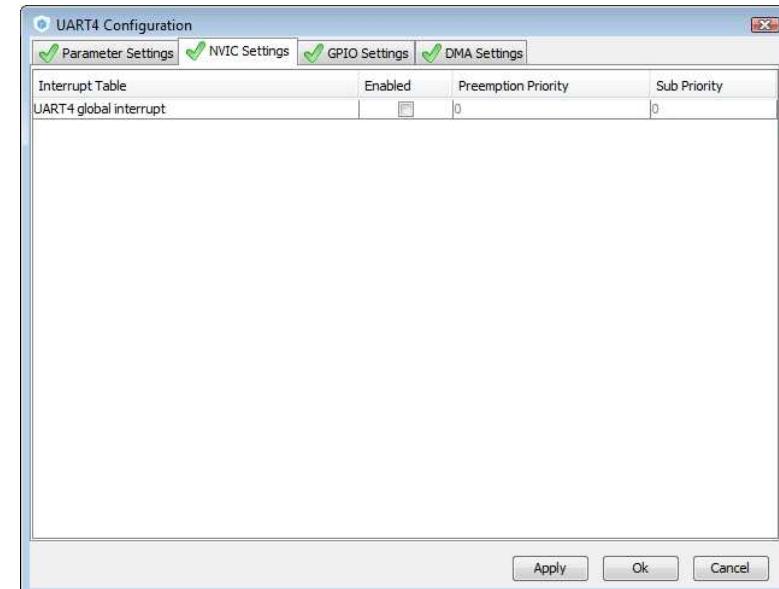
- Parameters with management of dependencies and constraints
- Interrupts
- GPIO
- DMA



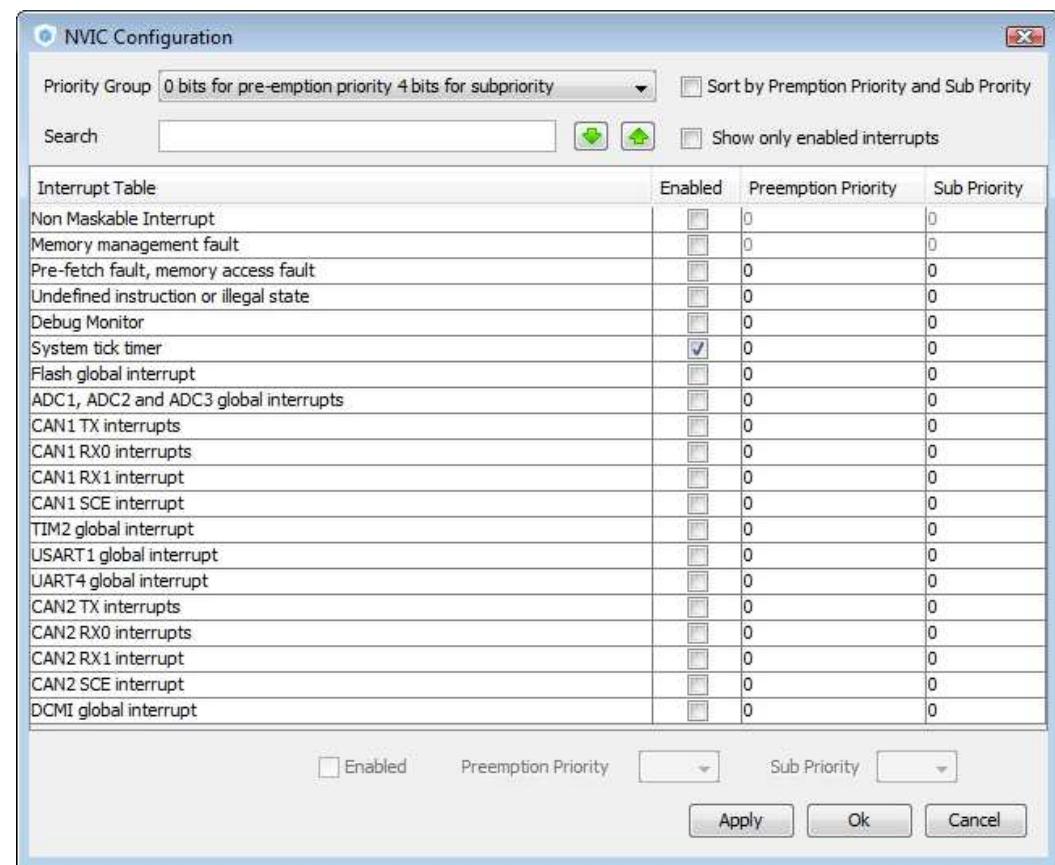
STM32CubeMX: Peripheral and middleware configuration

25

- Manage Interruptions
 - priorities can only be set in the NVIC global view
- Manage GPIO parameters
- Manage DMA
 - Configure all the parameters of the DMA request
 - Runtime parameters (start address, ...) are not managed



- Manage all interruptions
- Manage priorities and sort by priorities
- Search for a specific interrupt in the list

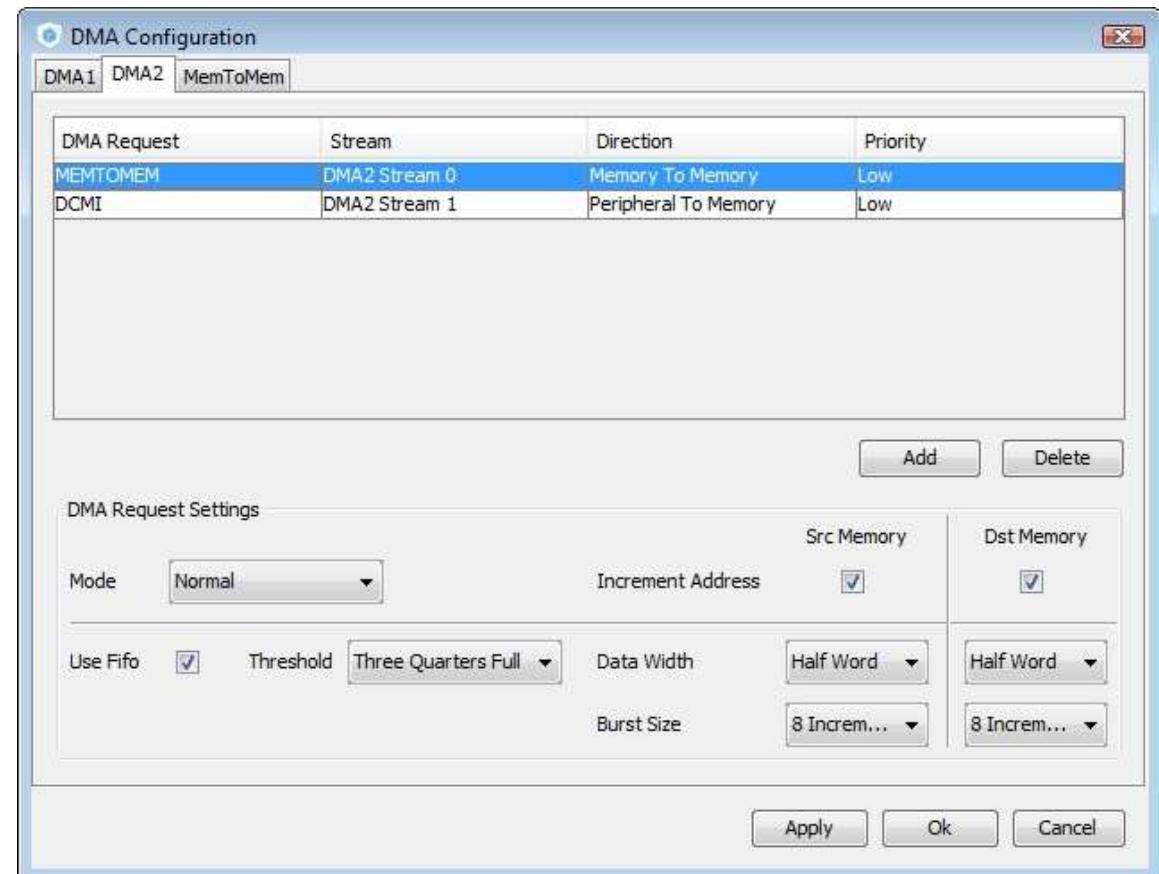


The screenshot shows the 'NVIC Configuration' dialog box. At the top, there is a dropdown menu for 'Priority Group' set to '0 bits for pre-emption priority 4 bits for subpriority'. There are also checkboxes for 'Sort by Preemption Priority and Sub Priority' and 'Show only enabled interrupts'. Below this is a search bar and two green arrow buttons. The main area is a table titled 'Interrupt Table' with three columns: 'Enabled', 'Preemption Priority', and 'Sub Priority'. The table lists various interrupt types, each with a checkbox in the 'Enabled' column and numerical values in the other two columns. The 'System tick timer' interrupt is currently selected, indicated by a checked checkbox in its 'Enabled' column.

Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non Maskable Interrupt	<input type="checkbox"/>	0	0
Memory management fault	<input type="checkbox"/>	0	0
Pre-fetch fault, memory access fault	<input type="checkbox"/>	0	0
Undefined instruction or illegal state	<input type="checkbox"/>	0	0
Debug Monitor	<input type="checkbox"/>	0	0
System tick timer	<input checked="" type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
ADC1, ADC2 and ADC3 global interrupts	<input type="checkbox"/>	0	0
CAN1 TX interrupts	<input type="checkbox"/>	0	0
CAN1 RX0 interrupts	<input type="checkbox"/>	0	0
CAN1 RX1 interrupt	<input type="checkbox"/>	0	0
CAN1 SCE interrupt	<input type="checkbox"/>	0	0
TIM2 global interrupt	<input type="checkbox"/>	0	0
USART1 global interrupt	<input type="checkbox"/>	0	0
UART4 global interrupt	<input type="checkbox"/>	0	0
CAN2 TX interrupts	<input type="checkbox"/>	0	0
CAN2 RX0 interrupts	<input type="checkbox"/>	0	0
CAN2 RX1 interrupt	<input type="checkbox"/>	0	0
CAN2 SCE interrupt	<input type="checkbox"/>	0	0
DCMI global interrupt	<input type="checkbox"/>	0	0

At the bottom, there are buttons for 'Enabled', 'Preemption Priority', 'Sub Priority', 'Apply', 'Ok', and 'Cancel'.

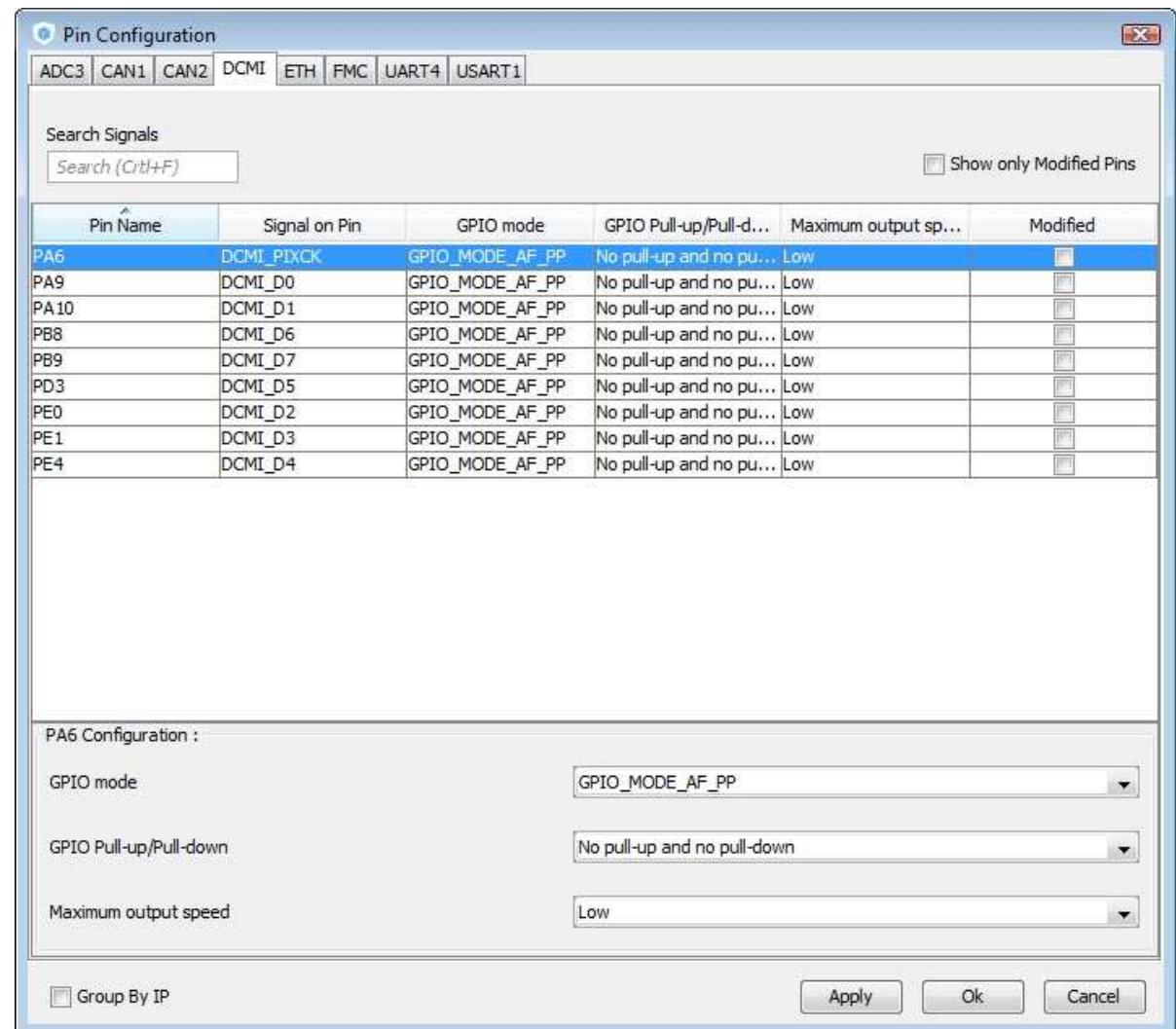
- Manage All DMA requests including Memory to Memory
- Set Direction and priority
- Set specific parameters



GPIO Panel

28

- Most of the GPIO parameters are set by default to the correct value
- You may want to change the maximum output speed
- You can select multiple pin at a time to set the same parameter



STM32CubeMX: Code generation

29

- Generation of all the C initialization code
- Automatic integration with partners toolchains
- User code can be added in dedicated sections and will be kept upon regeneration
- Required library code is automatically copied or referenced in the project (updater)

```
main.c
22  ****
23  */
24  /* Includes -
25  #include "stm32f4xx_hal.h"
26  #include "cmsis_os.h"
27  #include "lwip.h"
28  #include "usb_device.h"
29
30  /* Define structures */
31  ADC_HandleTypeDef hadc1;
32
33
34  /* USER CODE BEGIN 0 */
35
36  /* USER CODE END 0 */
37  /* Private function prototypes -----
38  static void SystemClock_Config(void);
39  static void StartThread(void const * argument);
40  static void MX_GPIO_Init(void);
41  static void MX_ADC1_Init(void);
42  static void MX_NVIC_Init(void);
43
44  int main(void)
45 {
46  /* USER CODE BEGIN 1 */
47
48  /* USER CODE END 1 */
49  /* MCU Configuration-
50  /* Reset of all peripherals, Initializes the Flash interface
51  HAL_Init();
52  /* Configure the system clock */
```

STM32CubeMX: Updater

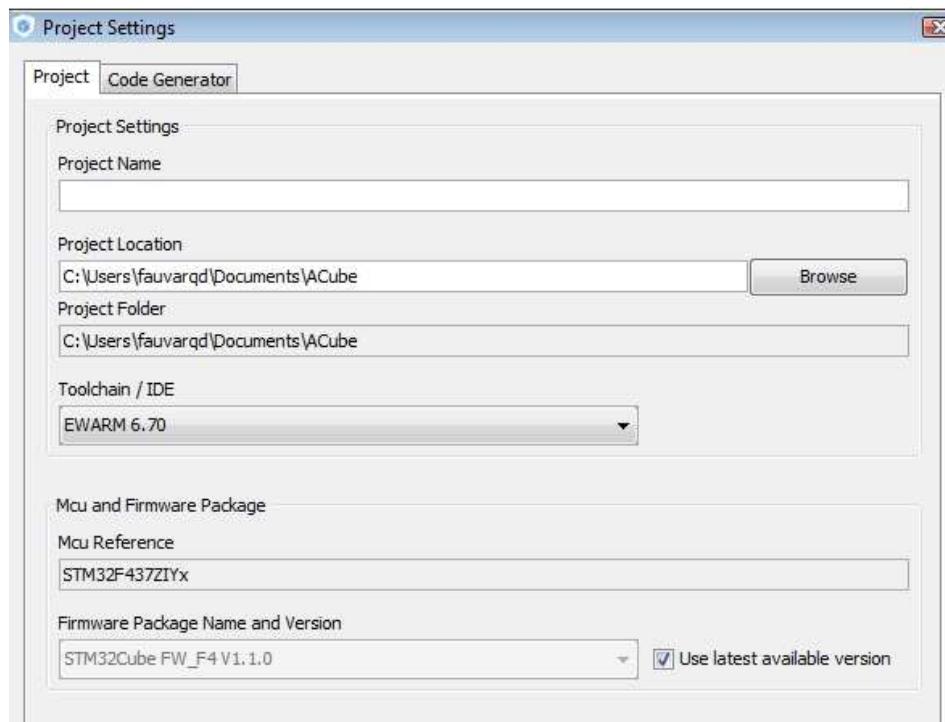
30

- Help->Updater settings
 - Choose location of STM32CubeFx firmware libraries repository
 - Choose manual or automatic check
 - Set Connection proxy
 - Inside ST use Ips5.sgp.st.com port 8080 with your windows login name and password
- Help->Install new libraries : Manage the content of the library repository
 - Click on the check button to see what is available
 - Select the library you want to install and click install now
 - The libraries will be automatically downloaded and unzipped

STM32CubeMX: Project settings

31

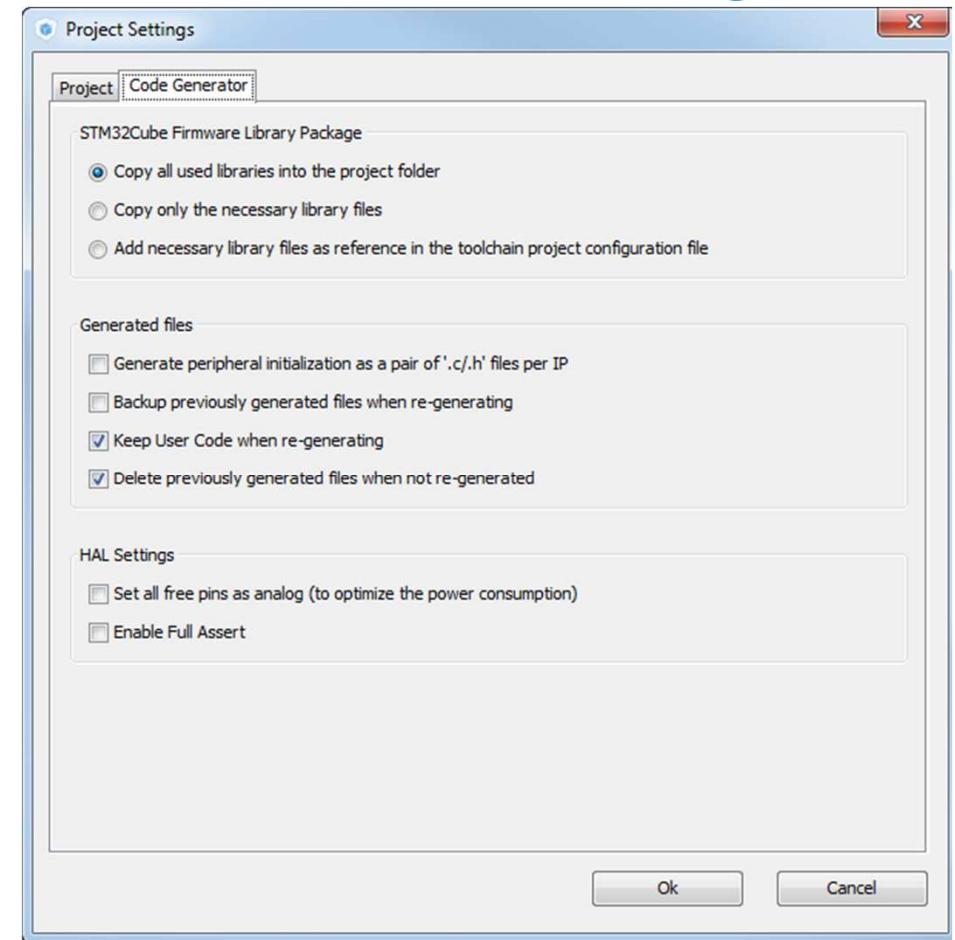
- Project -> Settings
 - Set project name and location
 - A full folder will be created named with the project name.
 - Inside this folder you'll find the saved configuration and all the generated code
 - Select toolchain (Keil, IAR, Atollic)
 - You can choose to use the latest version of the firmware library or a specific one



STM32CubeMX: Code Generator settings

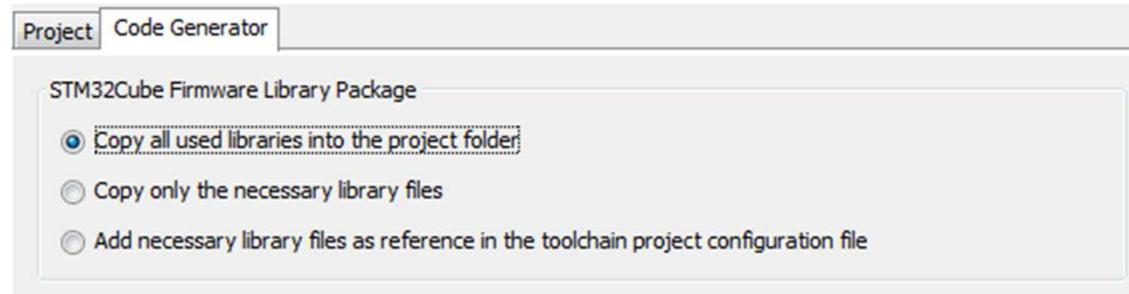
32

- Code generator options
 - Either copy the full library or only the necessary files or just reference the files from the common repository
 - Generate all peripherals initialization in the `stm32fYxx_hal_msp.c` file or one file per peripheral
 - Keep user code or overwrite it (code between User code comment sections)
 - Delete or keep files that are not useful anymore
 - Set free pins as analog, this settings helps keep low consumption (**if SWD/JTAG is not selected in pinout, this option will disable it**)
 - Enable full assert in project, this help discover incorrect HAL function parameter used in user code

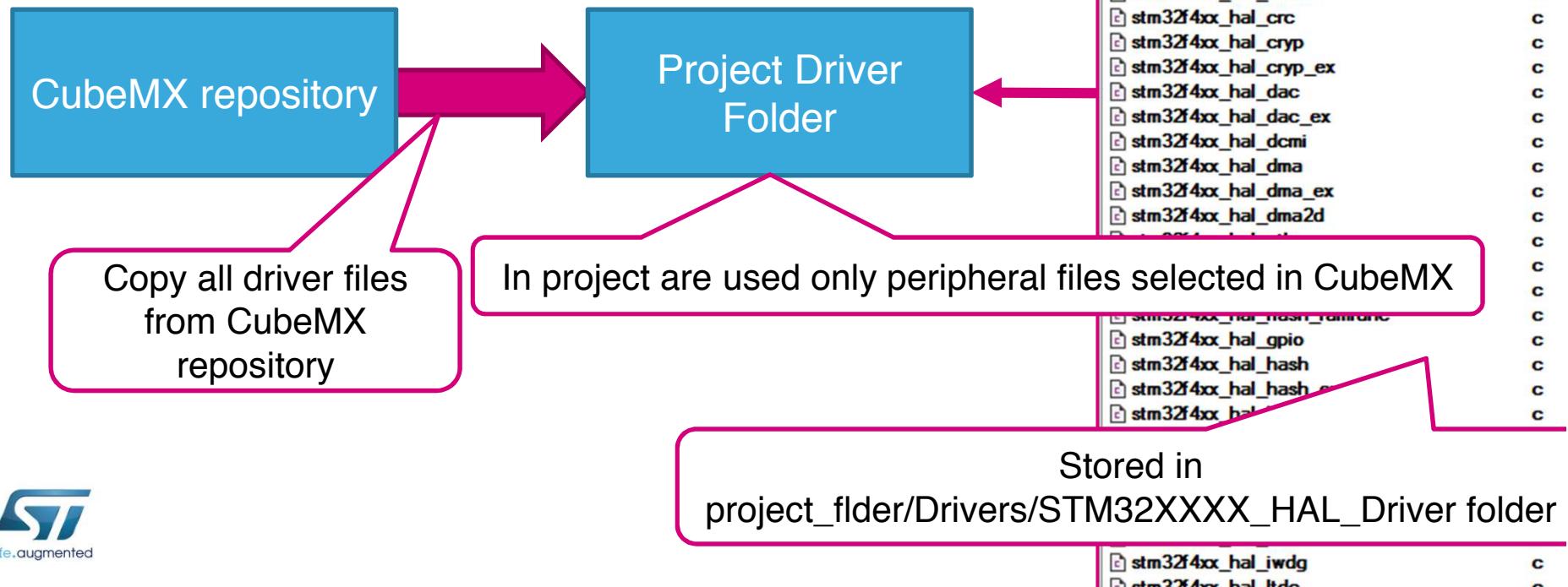


STM32Cube Firmware Library package

33

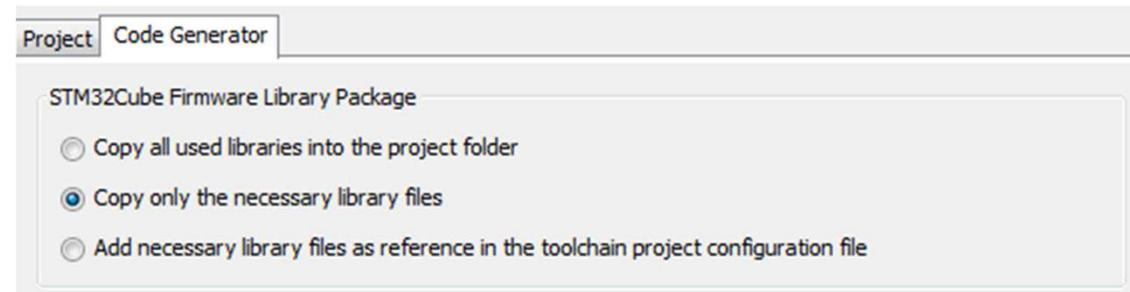


- Copy all used libraries into the project folder

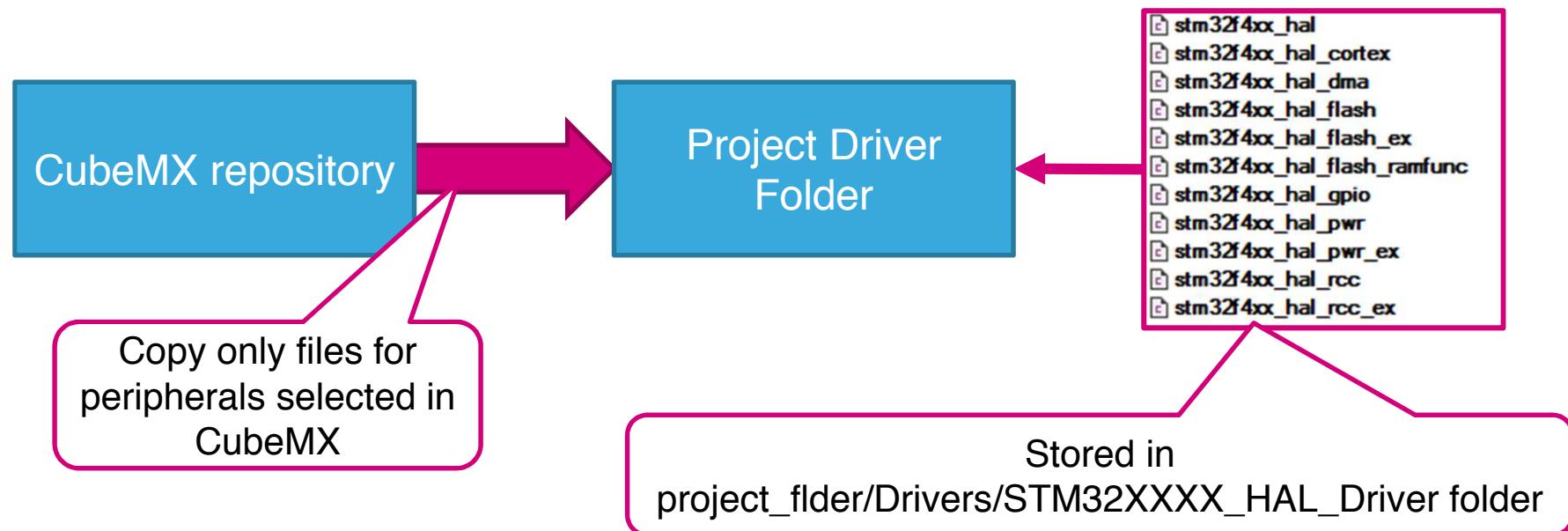


STM32Cube Firmware Library package

34

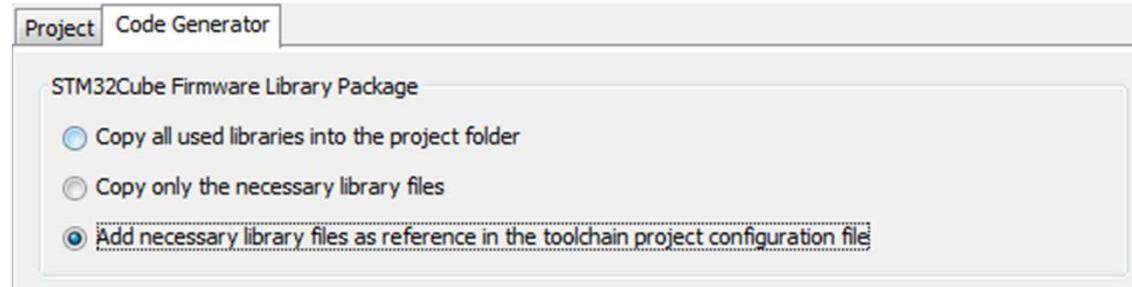


- Copy only the necessary library files

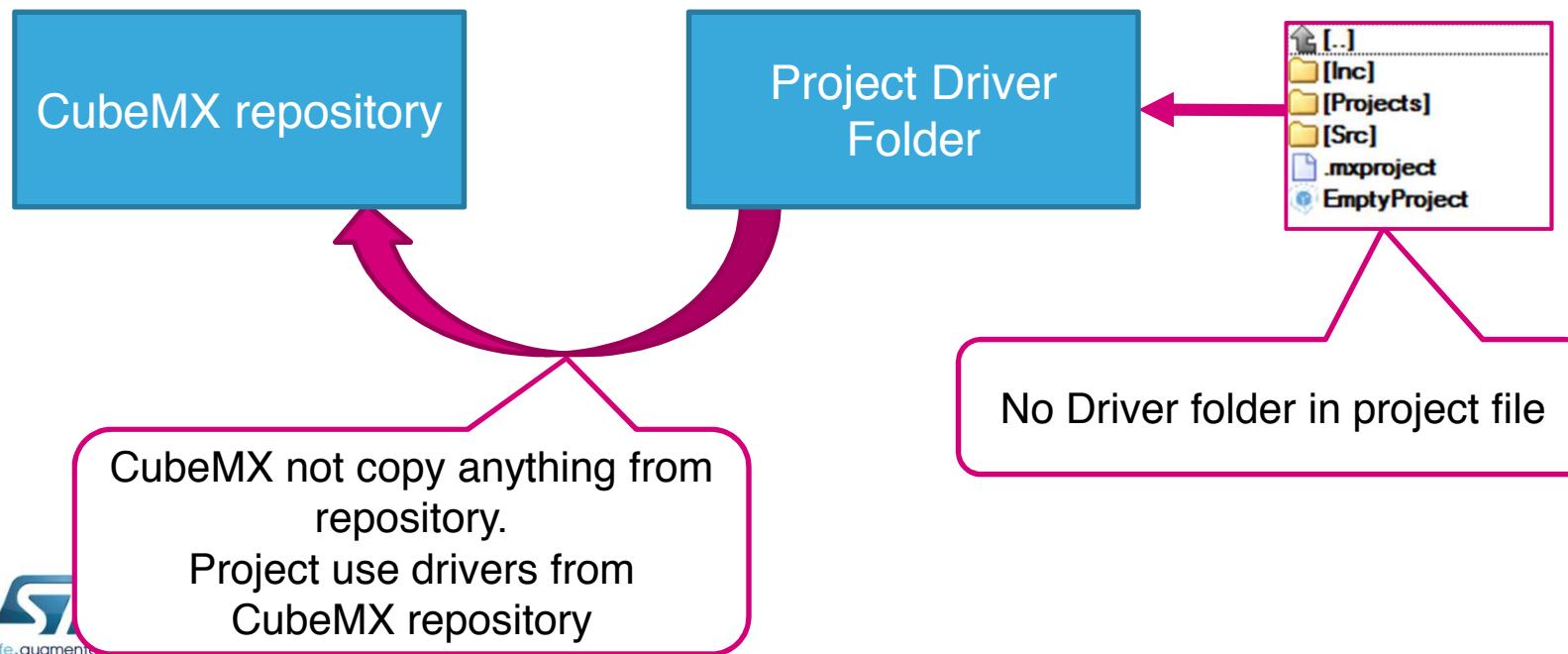


STM32Cube Firmware Library package

35

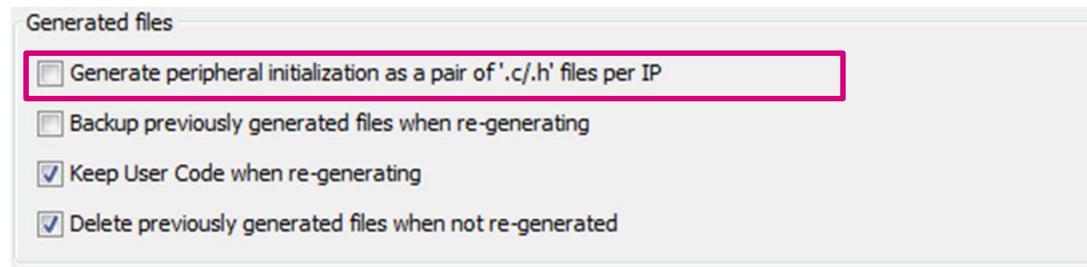


- Add necessary library files as reference in the toolchain project configuration file

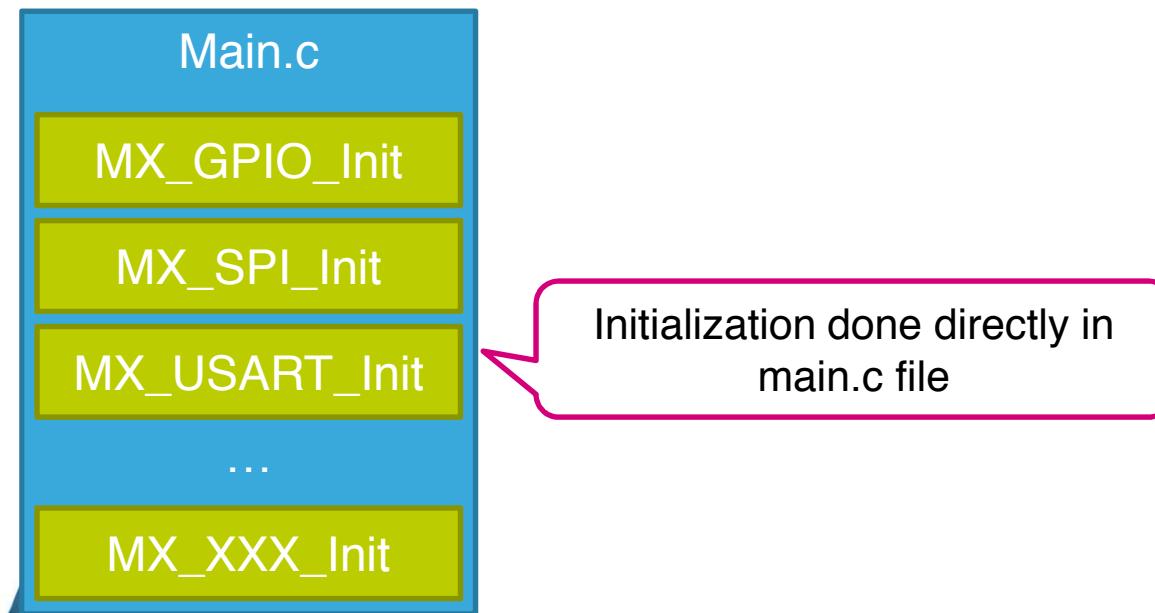


Generate peripheral initialization as a pair of '.c/.h' files per IP

36

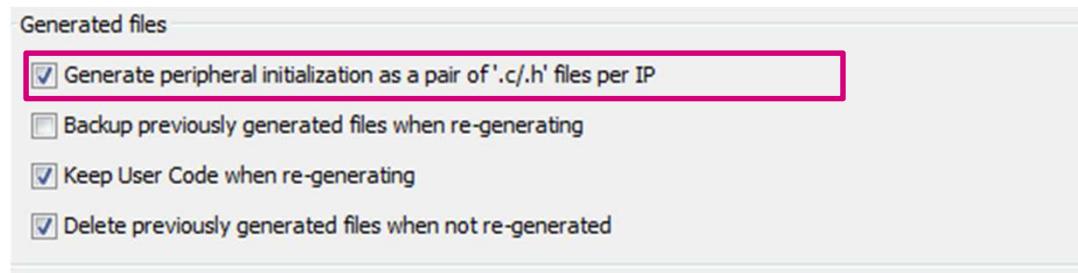


- In default not used
- Generate initialization of peripherals selected in CubeMX in main.c

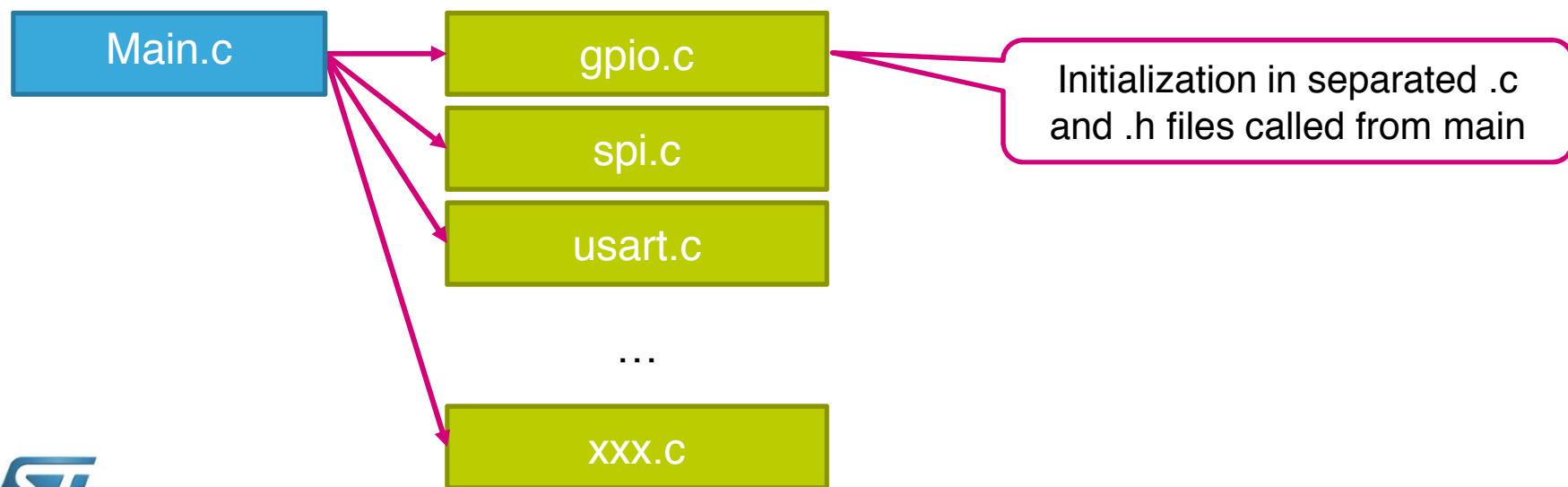


Generate peripheral initialization as a pair of '.c/.h' files per IP

37

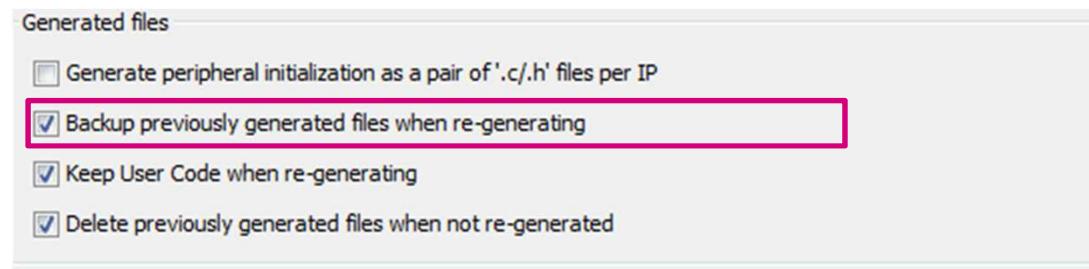


- Generate dedicated initialization .c and .h file for each periphery
- Advantage is that with .h file we can call MX_XXX init functions from every file in project not only from main.c

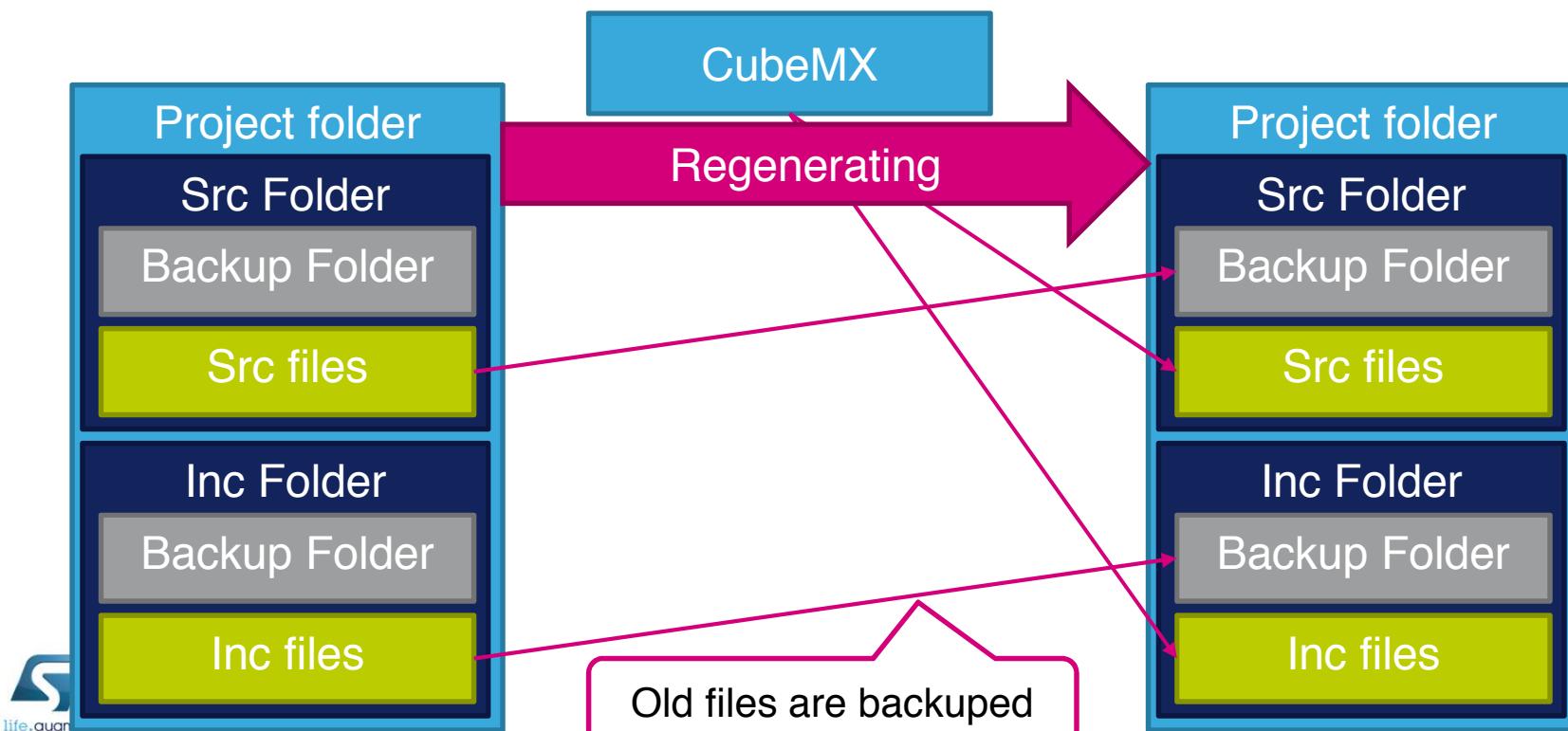


Backup previously generated files when re-generating

38



- Backup old files from **Src** and **Inc** folder into **Backup** folder



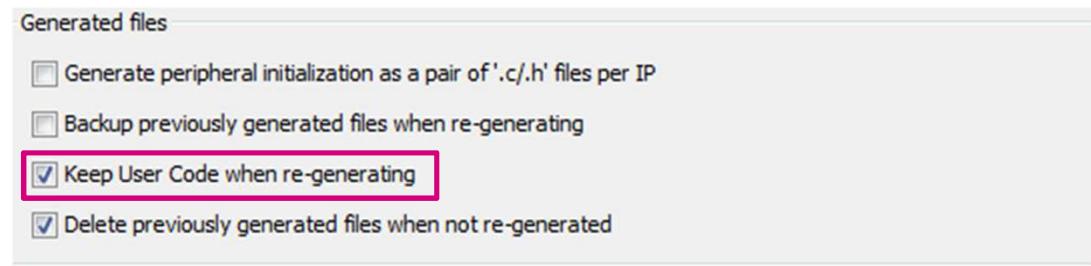
Keep User Code when re-generating

39

- Generated code contains USER CODE areas
- This areas are reserved in new code generation, if this option is selected

```
/* USER CODE BEGIN PFP */  
/* USER CODE END PFP */  
/* USER CODE BEGIN 0 */  
/* USER CODE END 0 */  
int main(void)  
{  
    /* USER CODE BEGIN 1 */  
    /* USER CODE END 1 */  
    /* MCU Configuration-----*/  
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */  
    HAL_Init();  
    /* Configure the system clock */  
    SystemClock_Config();  
    /* Initialize all configured peripherals */  
    /* USER CODE BEGIN 2 */  
    /* USER CODE END 2 */  
    /* USER CODE BEGIN 3 */  
    /* Infinite loop */  
    while (1)  
    {  
    }  
    /* USER CODE END 3 */  
}
```

Here user can put his code,
code will be preserved during
project generation



Keep User Code when re-generating

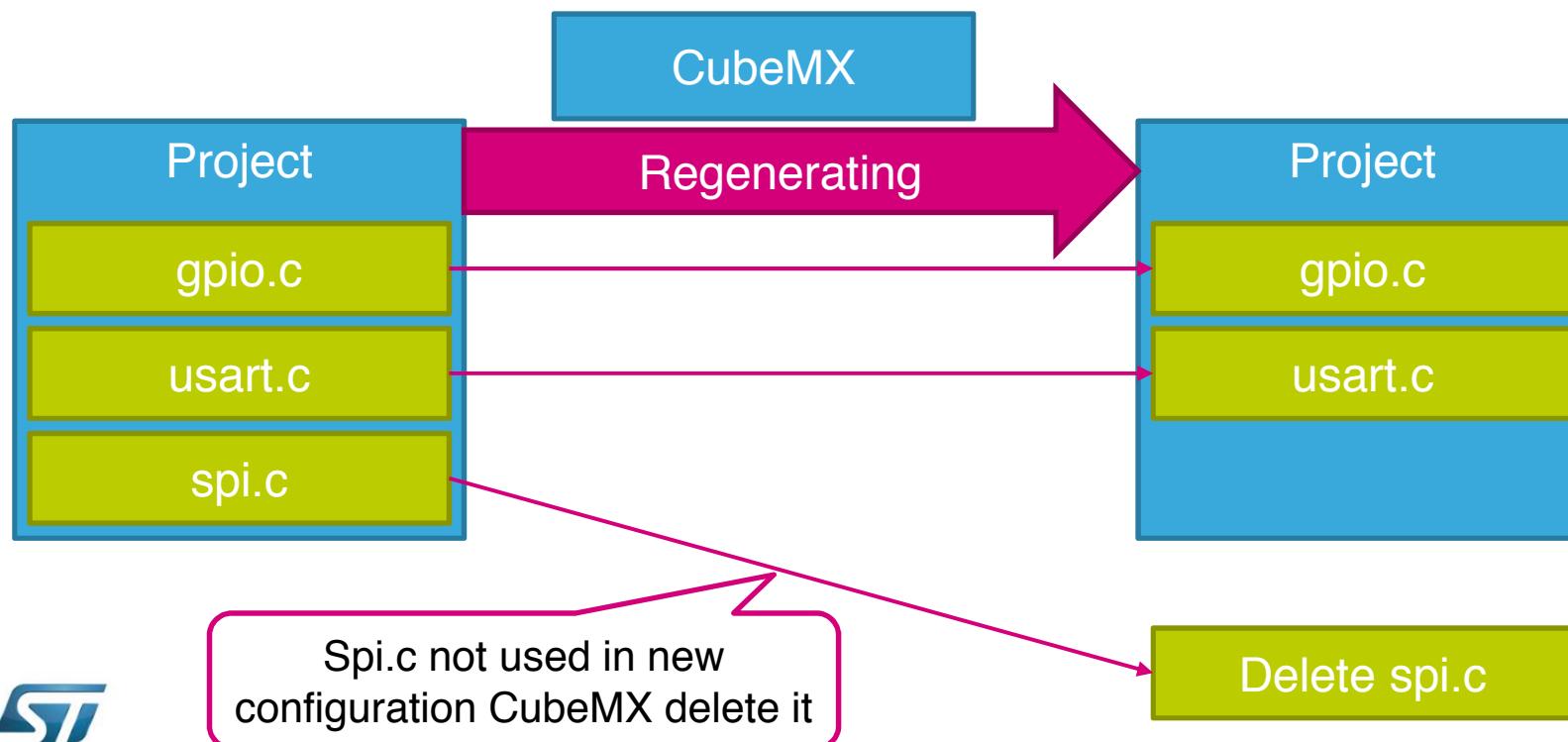
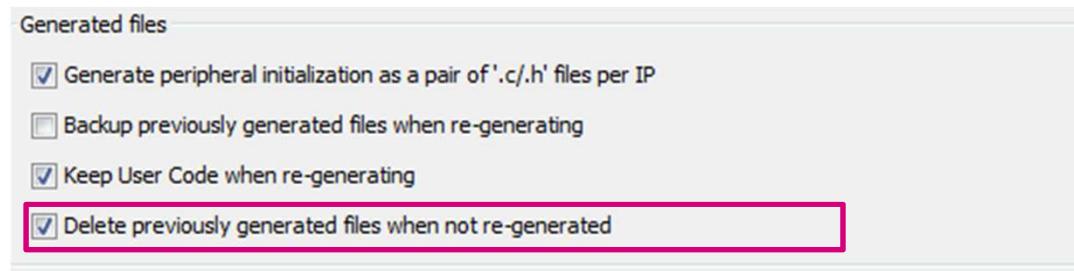
40

- Generated code contains USER CODE areas
- These areas are reserved in new code generation, if this option is selected
- Areas present in files generated by CubeMX
 - Main.c
 - Stm32f4xx_it.c
 - Stm32f4xx_hal_msp.c
- Areas cover important areas used for:
 - Includes
 - Variables
 - Function prototypes
 - Functions

```
/* USER CODE BEGIN PFP */  
/* USER CODE END PFP */  
/* USER CODE BEGIN 0 */  
/* USER CODE END 0 */
```

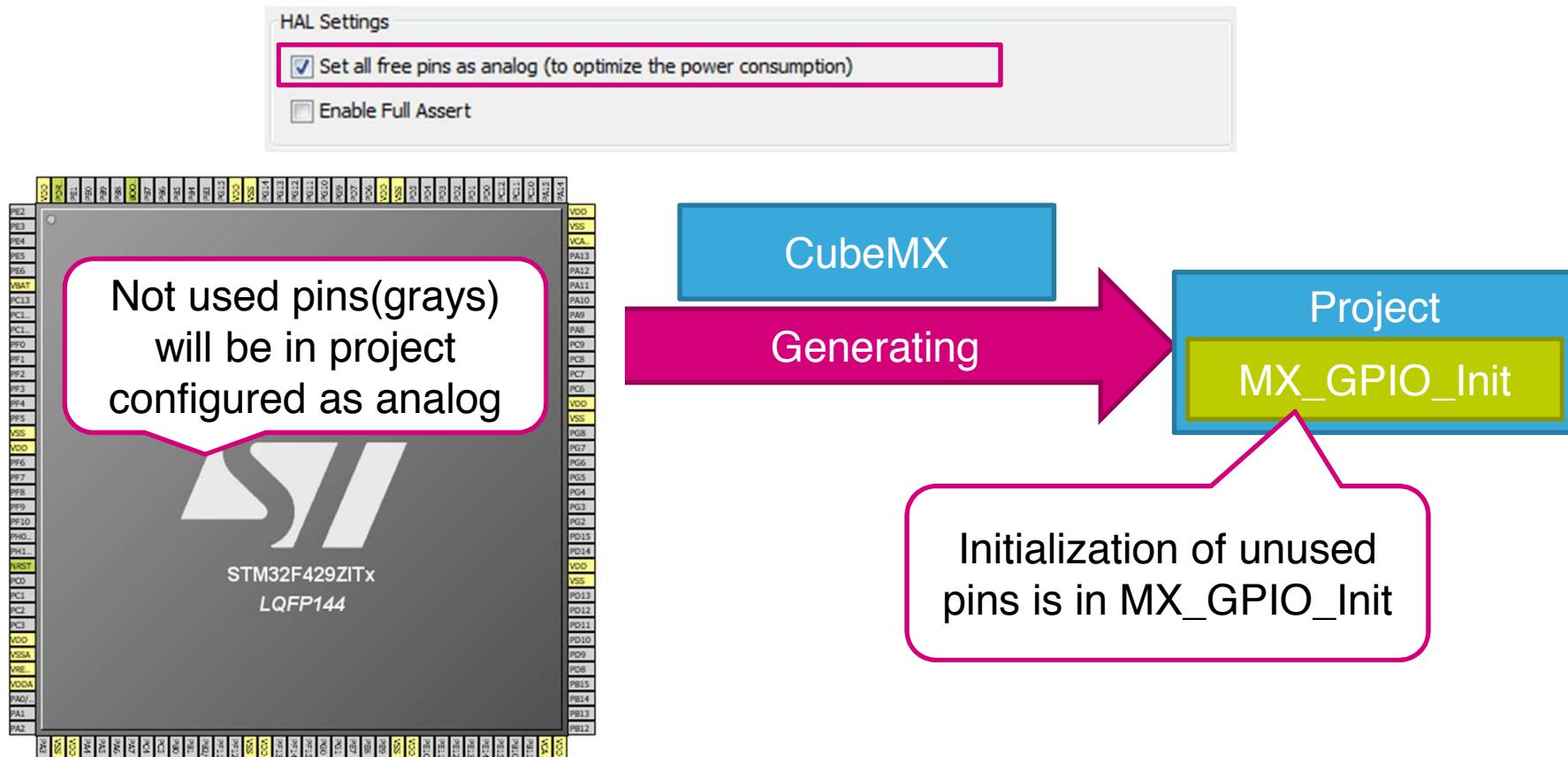
Delete previously generated files when re-generating

41



Set all free pins as analog

- This settings optimize power consumption of unused pins



- If the **JTAG/SWD is not selected** in CubeMX, MX_GPIO_Init reconfigure JTAG/SWD pins to analog and this **disable debug possibilities**

Enable Full Assert

43



- Feature very useful during debugging
- Function input parameters are checked if they are in correct range, if not application jump into assert_failed function in main.c

```
/* USER CODE BEGIN 2 */  
HAL_GPIO_TogglePin(GPIOA, (0x1<<17));  
/* USER CODE END 2 */
```

This function trying to
configure not existing
pin PA17

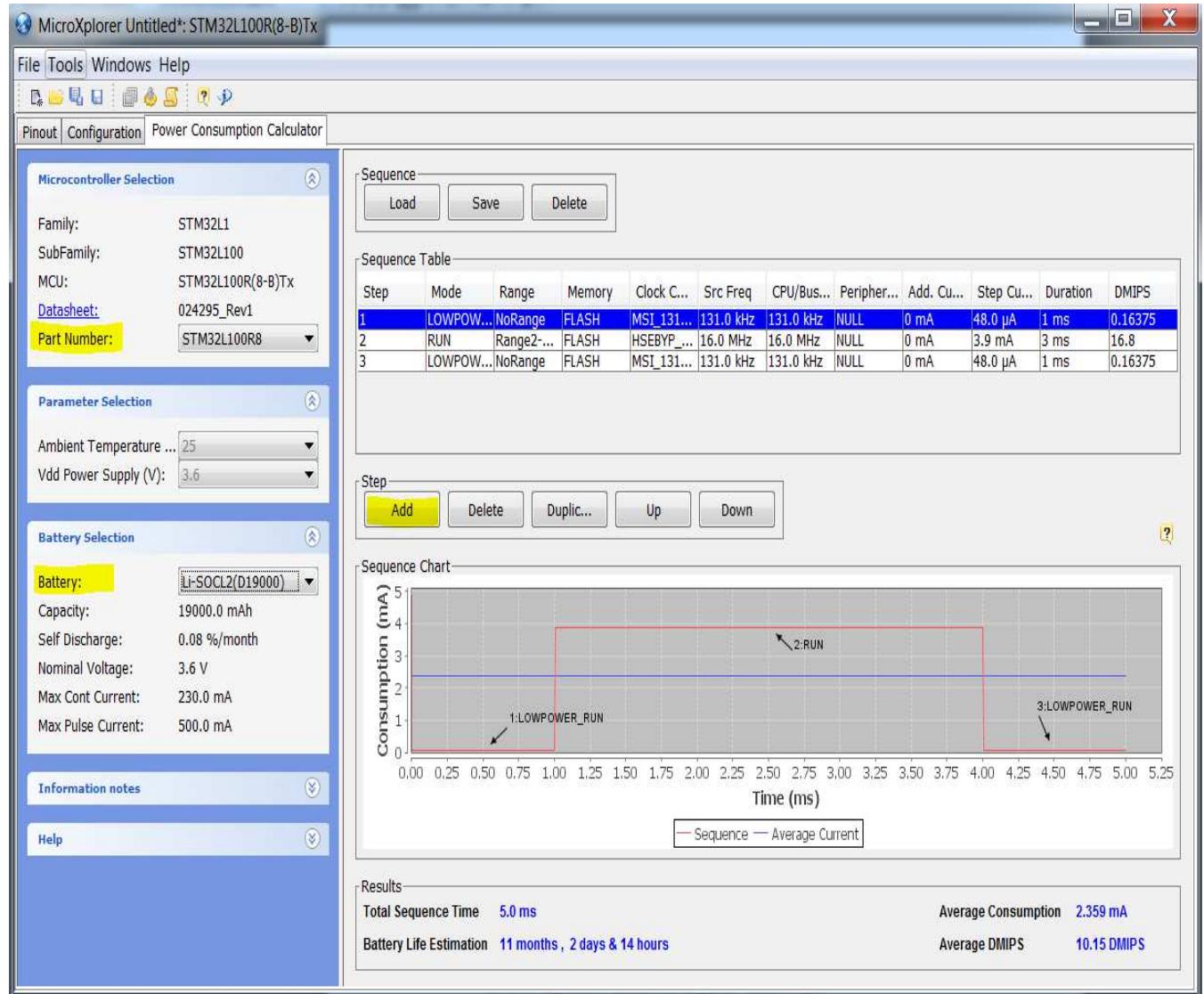
```
void assert_failed(uint8_t* file, uint32_t line)  
{  
    /* USER CODE BEGIN 6 */  
    /* User can add his own implementation to report the file name and line number,  
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */  
    /* USER CODE END 6 */  
}
```

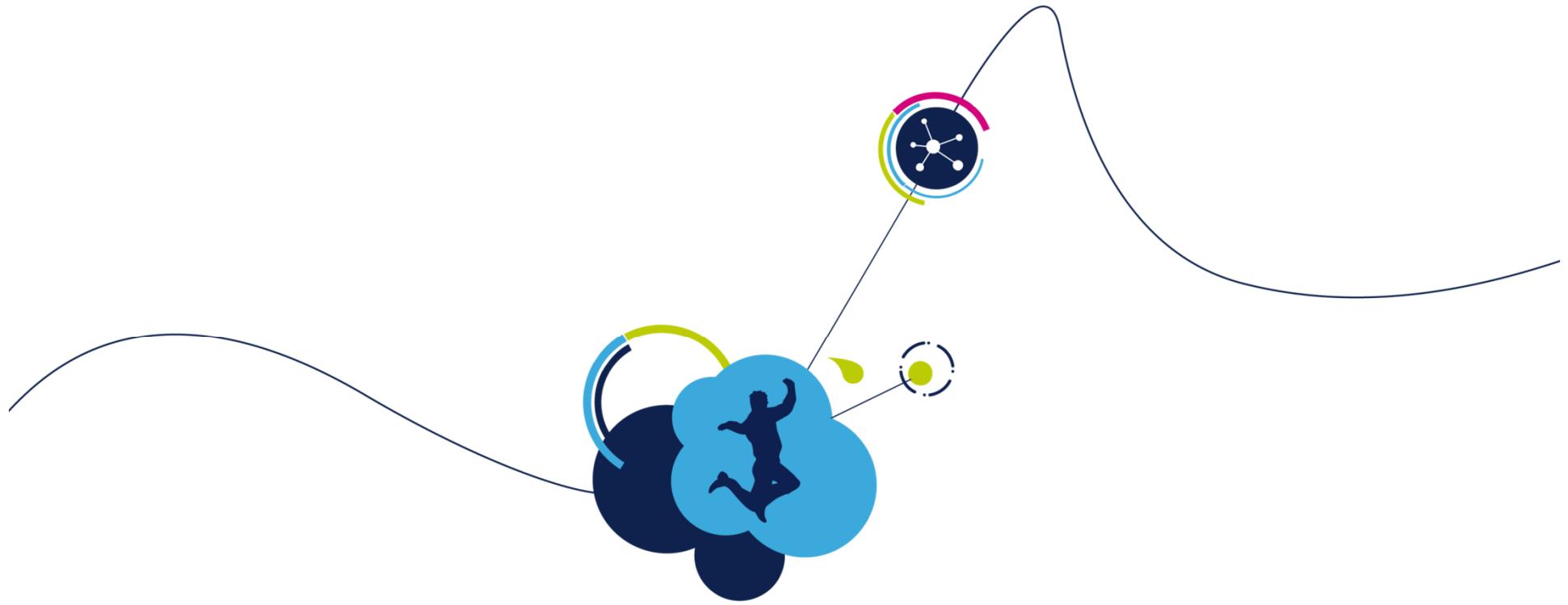
If parameter is not in valid
range program jump into
assert_failed function

STM32CubeMX: Power consumption calculator

44

- Power step definitions
- Battery selection
- Creation of consumption graph
- Display of
 - Average consumption
 - Average DMIPS
 - Battery lifetime
- Supported for L1 and F4





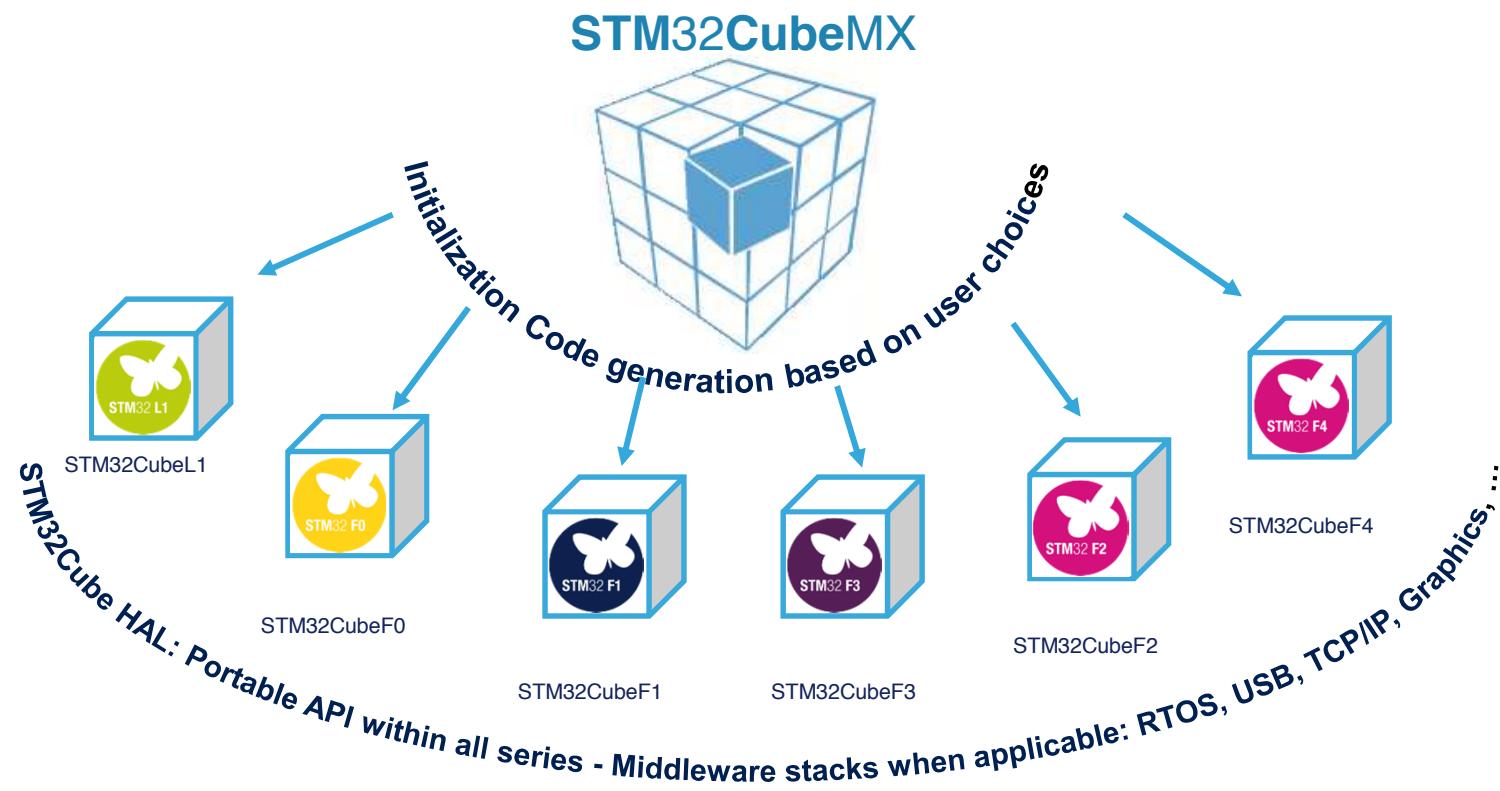
STM32Cube FW package

STM32Cube™ V1 Introduction

46

- STM32Cube™ Version 1:

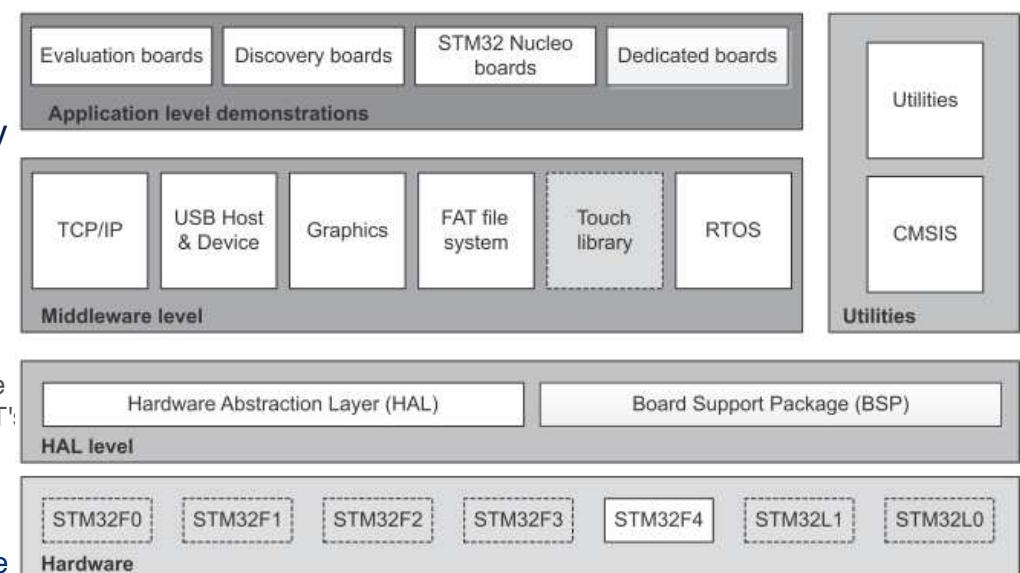
- A configuration tool, STM32CubeMX generating initialization code from user choices
- A full embedded software offer, delivered per series (like STM32CubeF4) with:
 - An STM32 Abstraction Layer embedded software: STM32Cube HAL
 - A consistent set of Middlewares: RTOS, USB, TCP/IP, Graphics, ...
- Available at st.com as free solution



STM32Cube Main Features

47

- STM32Cube gathers together, in a single package, all the generic and highly portable embedded software components required to develop an application on STM32 microcontrollers.
- The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards.
- It also contains a set of middleware components (*) with the corresponding examples. They come with very permissive license terms:
 - Full USB Host and Device stack supporting many classes.
 - Host Classes: HID, MSC, CDC, Audio, MTP
 - Device Classes: HID, MSC, CDC, Audio, DFU
 - Graphics
 - STemWin, a professional graphical stack solution available in binary format and based on the emWin solution from ST's partner SEGGER
 - LibJPEG, an open source implementation on STM32 for JPEG images encoding and decoding.
 - CMSIS-RTOS implementation with FreeRTOS open source solution
 - FAT File system based on open source FatFS solution
 - TCP/IP stack based on open source LwIP solution
 - SSL/TLS secure layer based on open source PolarSSL
- A demonstration implementing all these middleware components is also provided



(*) middleware components availability depends on STM32 Series

STM32Cube Documentation

48

Application Notes

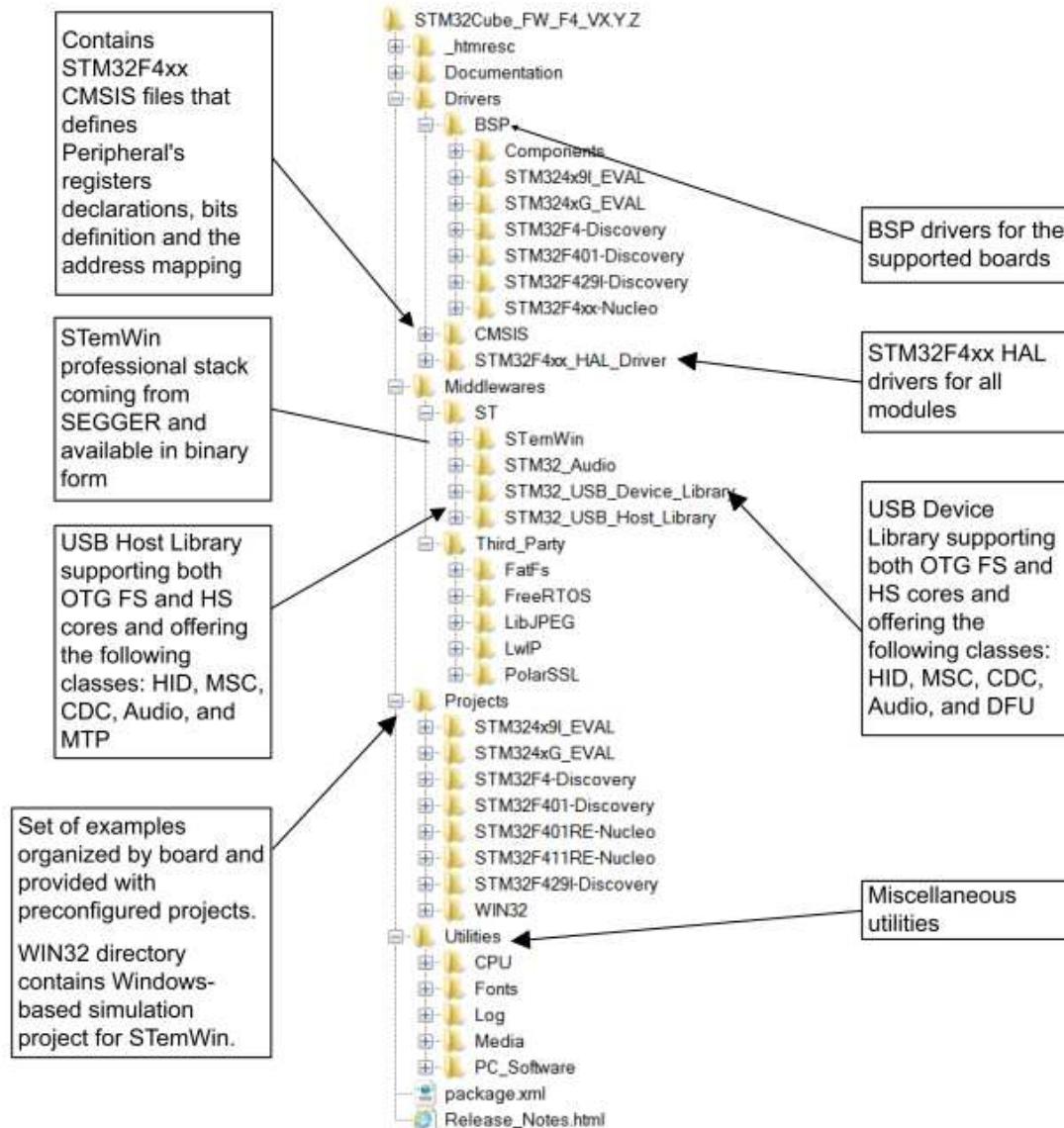
Description	Version	Size
 AN4323: Getting started with STemWin Library Generic doc	3.0	579 KB

User Manual

Description	Version	Size
 UM1709: STM32Cube Ethernet IAP example Generic doc	1.0	800 KB
 UM1713: Developing applications on STM32Cube™ with LwIP TCP/IP stack Generic doc	1.0	874 KB
 UM1720: STM32Cube™ USB host library Generic doc	1.0	610 KB
 UM1721: Developing Applications on STM32CubeF4 with FatFs Generic doc	1.0	514 KB
 UM1722: Developing Applications on STM32CubeF4 with RTOS Generic doc	1.0	707 KB
 UM1723: STM32CubeF4 PolarSSL example Generic doc	1.0	792 KB
 UM1725: Description of STM32F4xx HAL drivers Series specific doc	1.0	4 591 KB
 UM1730: Getting started with STM32CubeF4 firmware package for STM32F4xx series Series specific doc	1.0	404 KB
 UM1734: STM32Cube USB device library Generic doc	1.0	1 166 KB
 UM1743: STM32CubeF4 demonstration platform Series specific doc	1.0	4 110 KB

STM32Cube FW Package Organization

49



STM32Cube Examples Overview

50

For each board, a set of examples are provided with preconfigured projects for EWARM, MDK-ARM and TrueSTUDIO toolchains.

The right side figure shows the projects structure for STM324xG-EVAL board, which is identical for other boards.

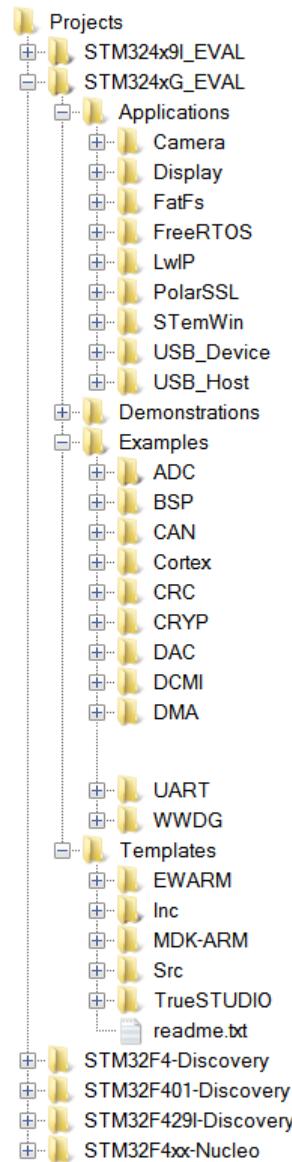
The examples are classified depending on the STM32Cube level they apply to, and are named as below:

- examples in level 0 are called **Examples**, that uses HAL drivers without any Middleware component
- examples in level 1 are called **Applications**, that provides typical use cases of each Middleware component
- examples in level 2 are called **Demonstration**, that implements all the HAL, BSP and Middleware components

Template project is provided to build quickly any firmware application on a given board.

All examples have the same structure,

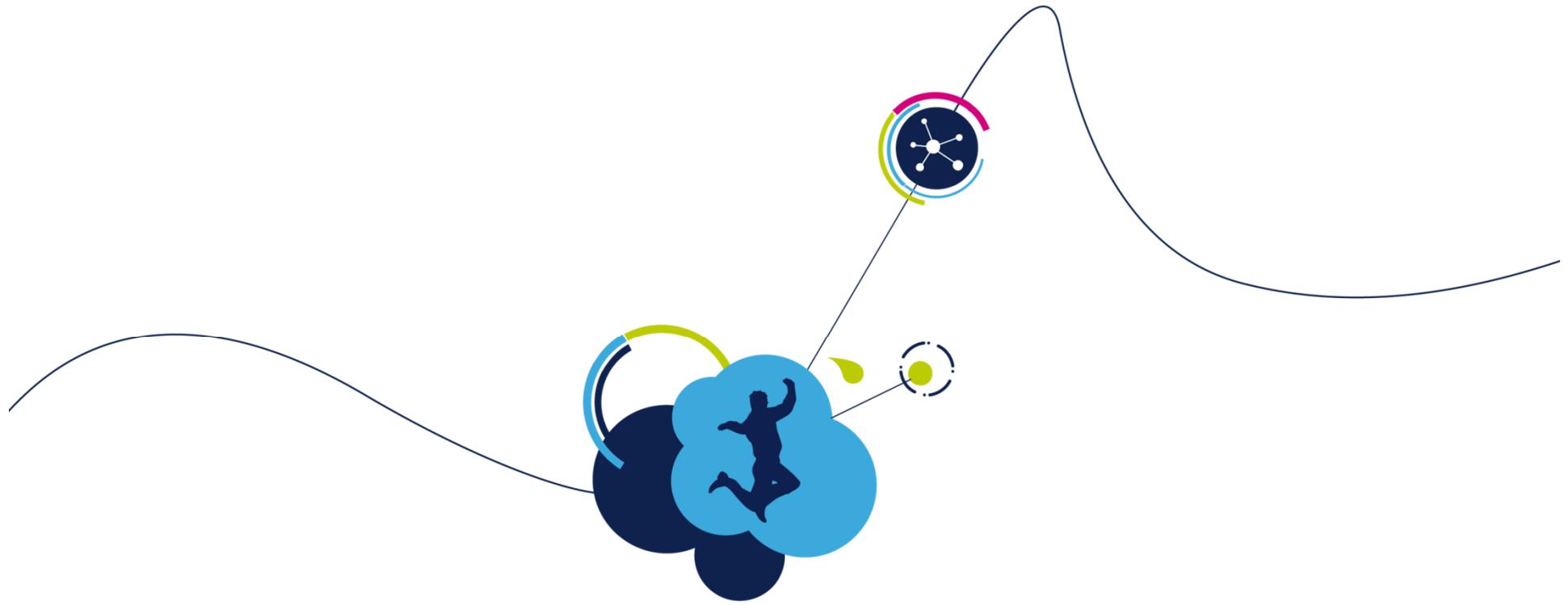
- \Inc folder that contains all header files
- \Src folder for the sources code
- \EWARM, \MDK-ARM and \TrueSTUDIO folders contain the preconfigured project for each toolchain.
- **readme.txt** describing the example behavior and needed environment to make it working



STM32Cube vs. StdLib

51

- Existing Standard Peripheral Libraries (StdLib) will be supported, but not recommended for new designs
- StdLib will be updated to support new F0/F3/F4 derivatives
- No StdLib for new Series, such as L0/L4/F7
- Migrating an application from StdLib to STM32Cube:
 - The 2 solutions are completely different (different architecture, APIs set...) and thus there is no direct migration path, i.e. no “find and replace” possibility.
 - Customers needs to rewrite their applications developed on StdLib to work with STM32Cube.
 - Combined with the fully portable HAL, user will do the migration to STM32Cube only once then can easily migrate to any other MCU with more performance, memory, and peripherals without rewriting the application. As a result, developers can leverage the same application and toolchain across an entire product line and a variety of MCUs.



STM32Cube Hardware Abstraction Layer (HAL)

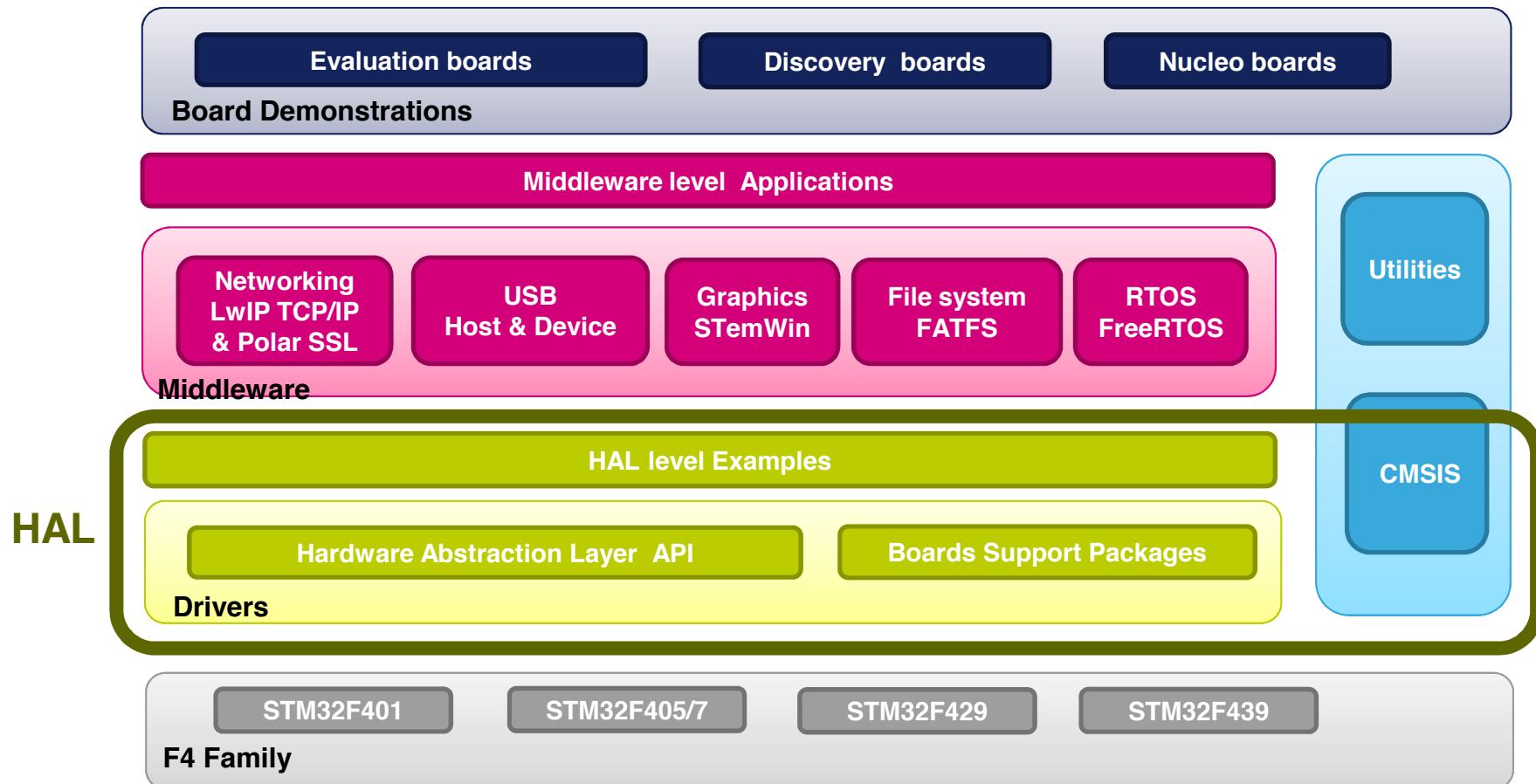


HAL general concepts

HAL general concepts

The HAL in STM32Cube FW package

54



HAL general concepts

Introduction to HAL

55

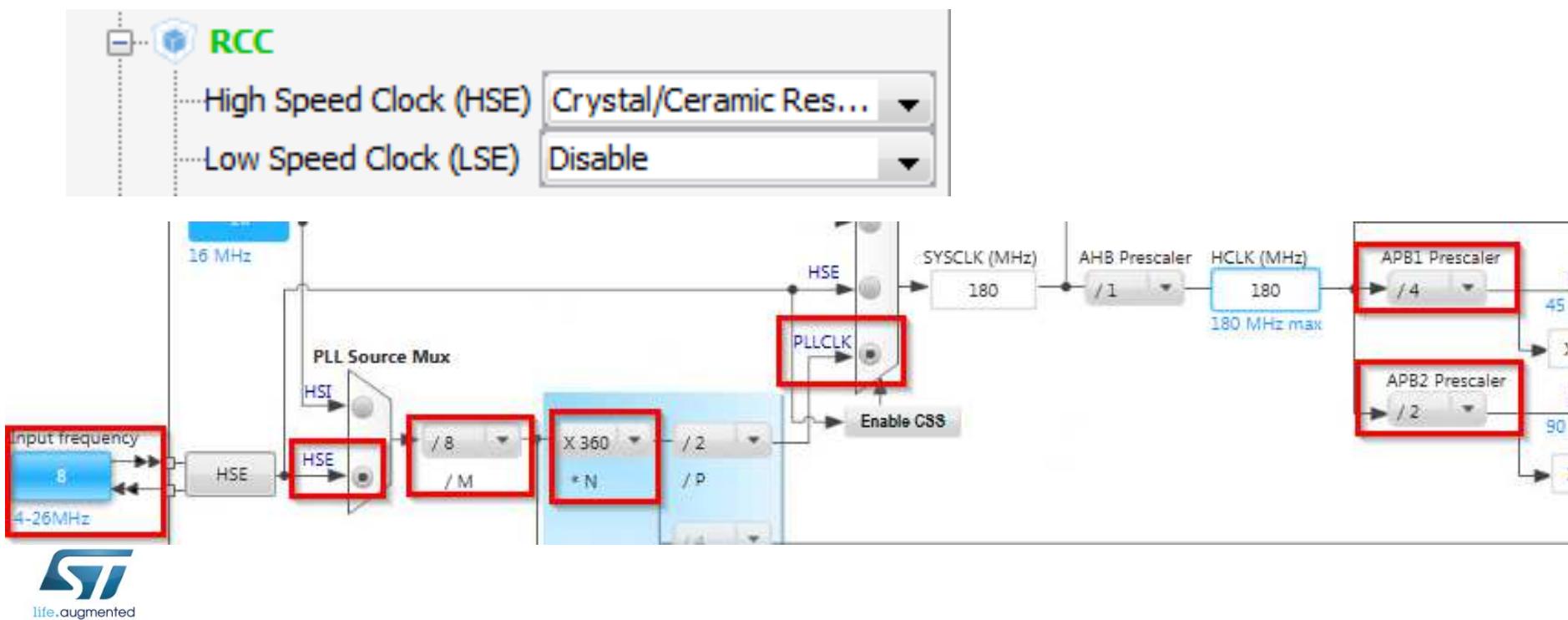
- The STM32Cube Hardware abstraction layer (HAL) is the replacement for the standard peripheral library
- The main objectives of the HAL is to offer
 - **User friendly APIs** that **hide the HW complexity** and focus on the **functionality**
 - **Portable APIs** that allowing **easy migration** of user application across different product families
- All HAL drivers follow a strict C coding rules and were tested using CodeSonar C code static analysis tool from GrammaTech
- HAL documentation is provided as a [PDF manual](#) based on Doxygen extracts

HAL general concepts

Introduction to HAL

56

- Create project in CubeMX
 - Menu > File > New Project
 - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- We need only blank project with clock initialization
 - We only set the RCC and configure the core to maximum speed

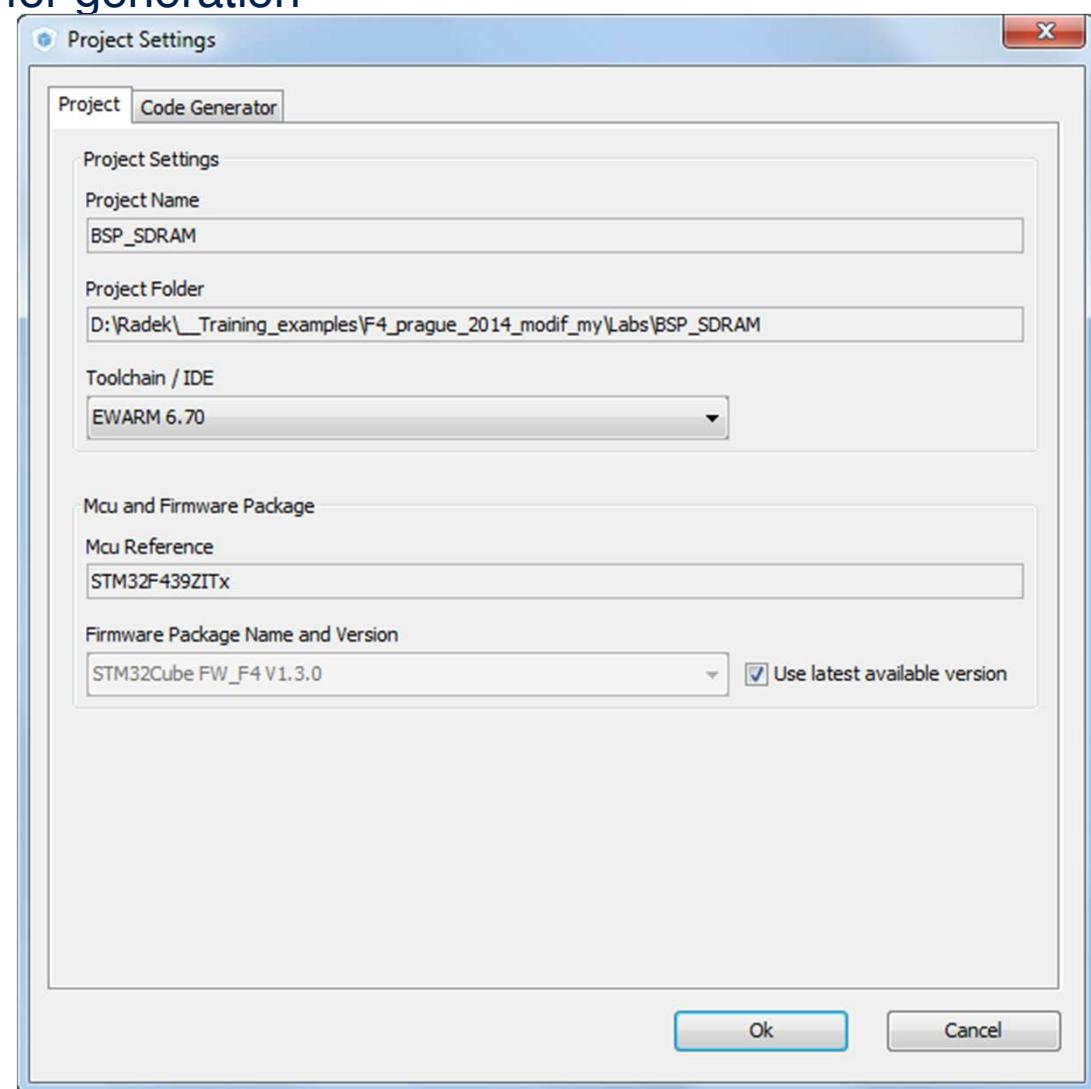


HAL general concepts

Introduction to HAL

57

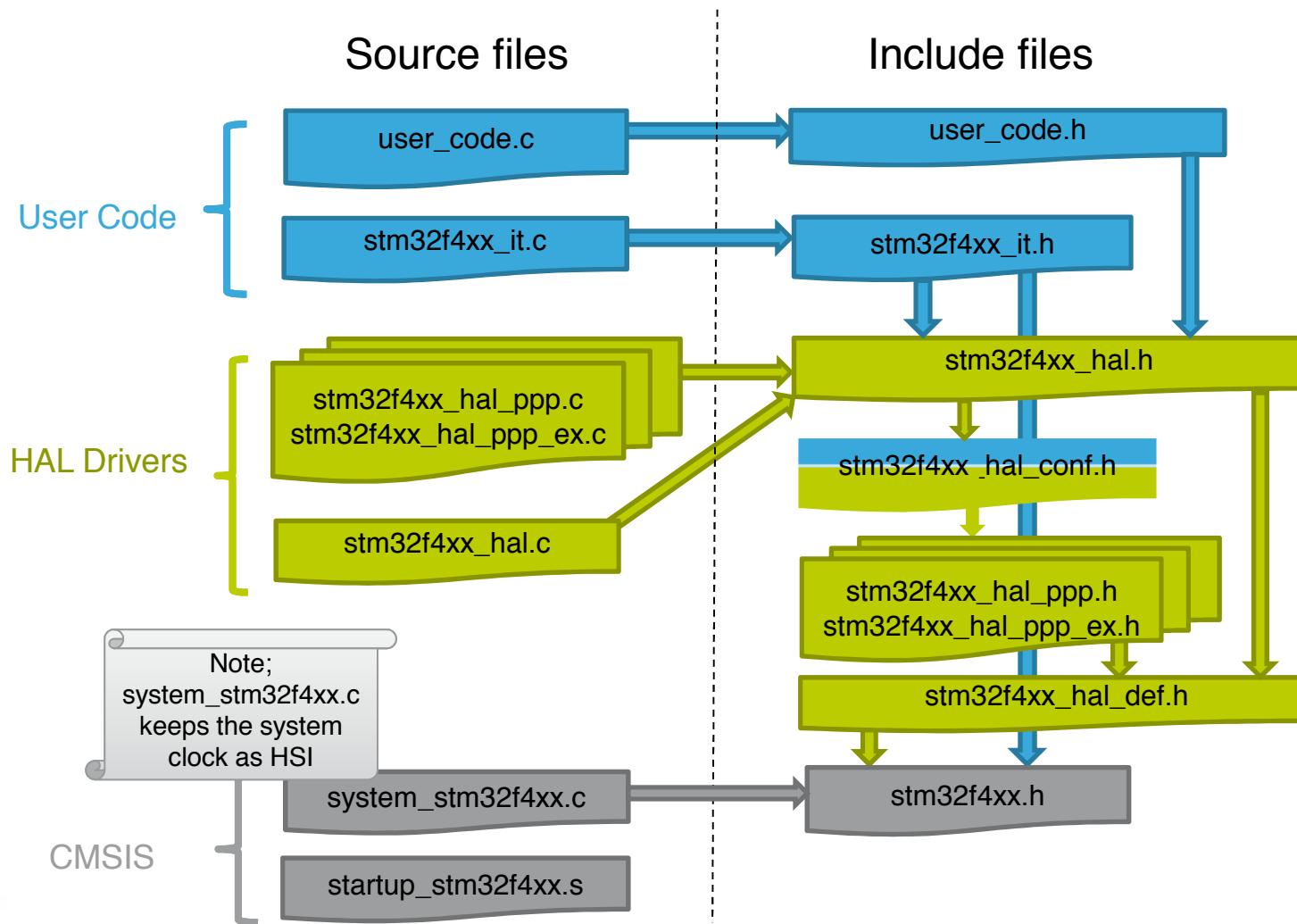
- Now we set the project details for generation
 - Menu > Project > Project Settings
 - Set the project name
 - Project location
 - Type of toolchain
- Now we can Generate Code
 - Menu > Project > Generate Code



HAL general concepts

HAL based project organization

58

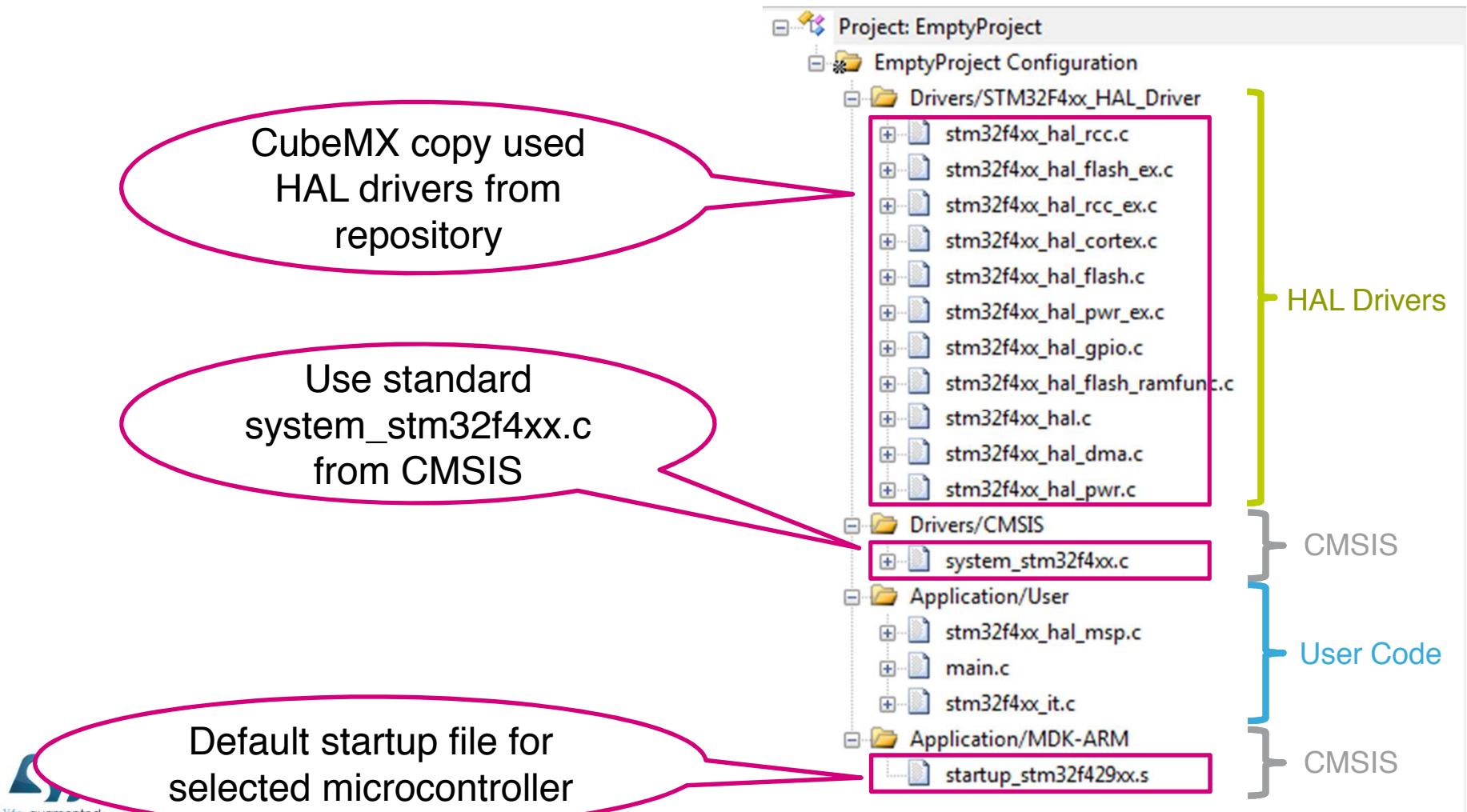


HAL general concepts

Introduction to HAL

59

- Structure of project generated with CubeMX with HAL

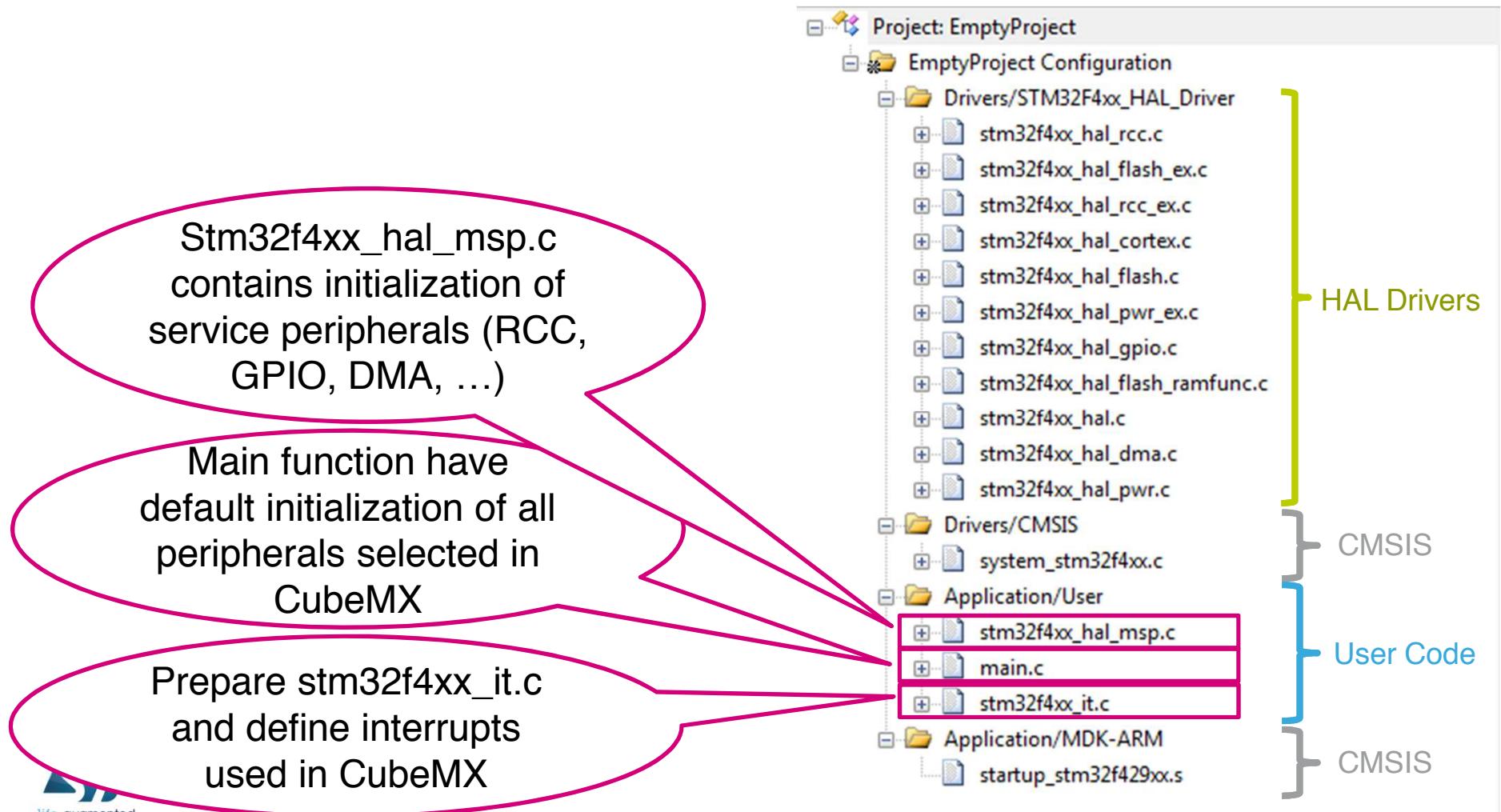


HAL general concepts

Introduction to HAL

60

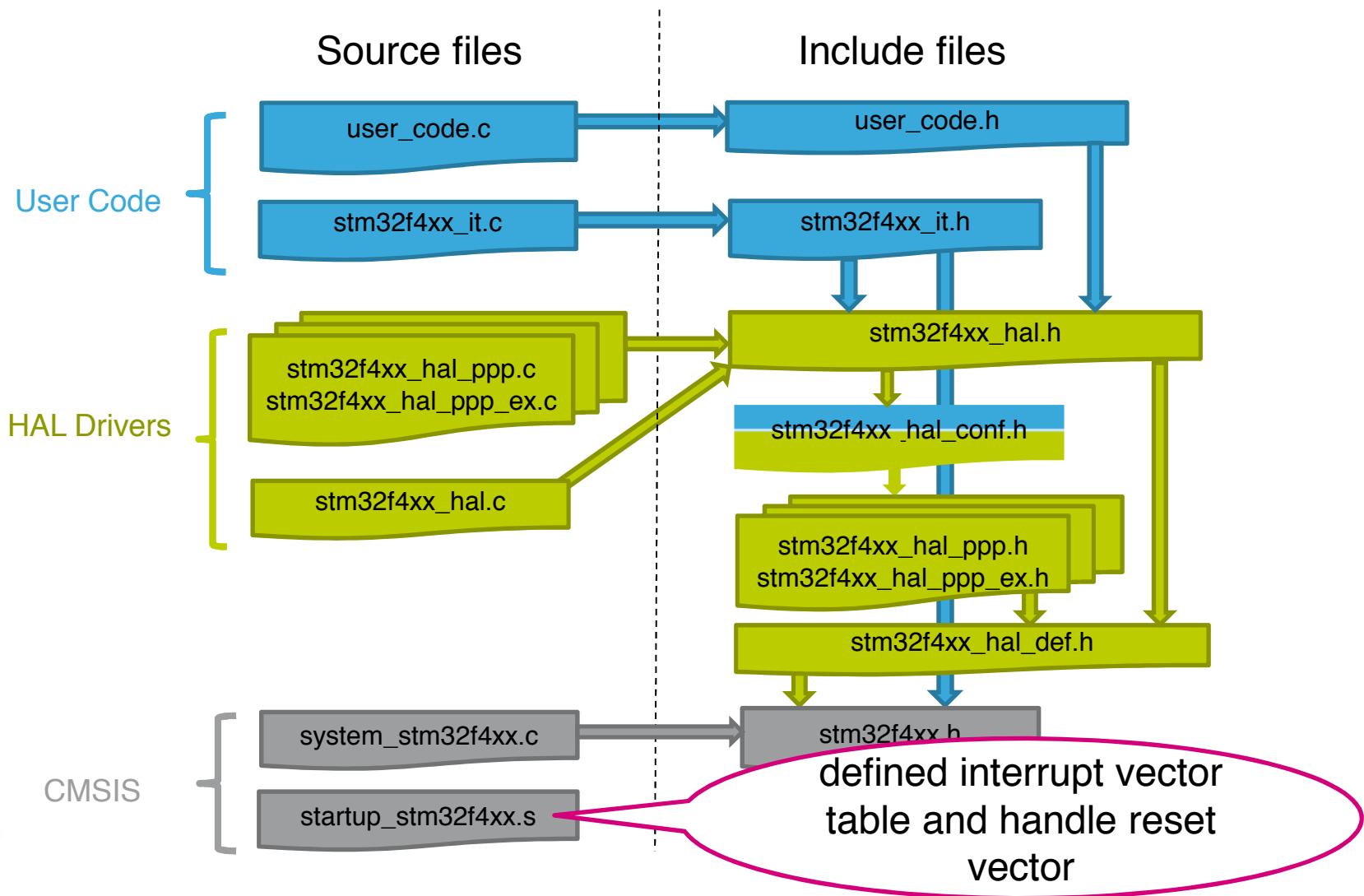
- Structure of project generated with CubeMX with HAL



HAL general concepts

HAL based project organization

61



HAL general concepts

HAL based project organization

62

```
; 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
```

Interrupt vector table definition

Name of interrupts
method which can be
used in user program
weak by default

CMSIS

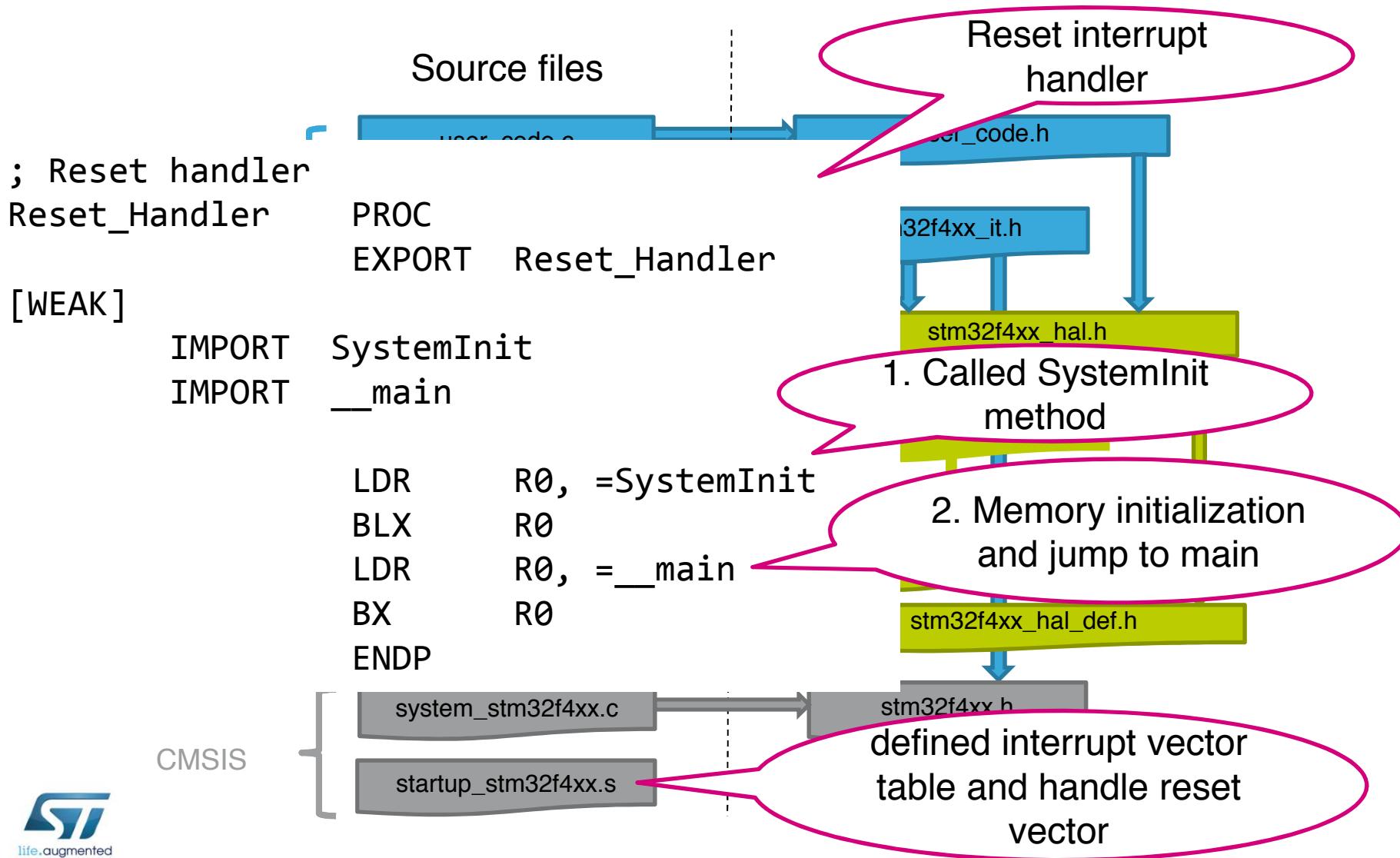
startup_stm32f4xx.s

defined interrupt vector table and handle reset vector

HAL general concepts

HAL based project organization

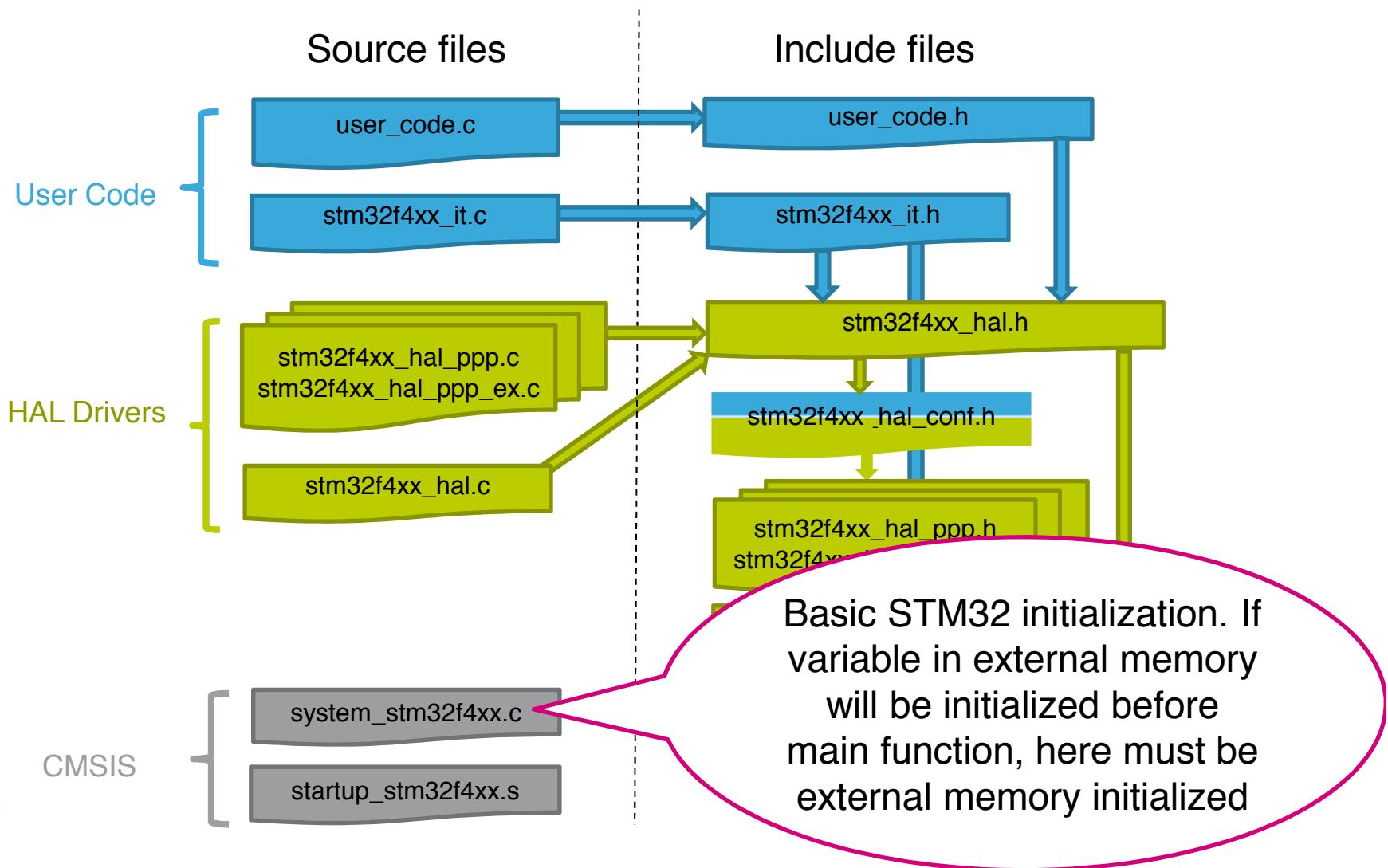
63



HAL general concepts

HAL based project organization

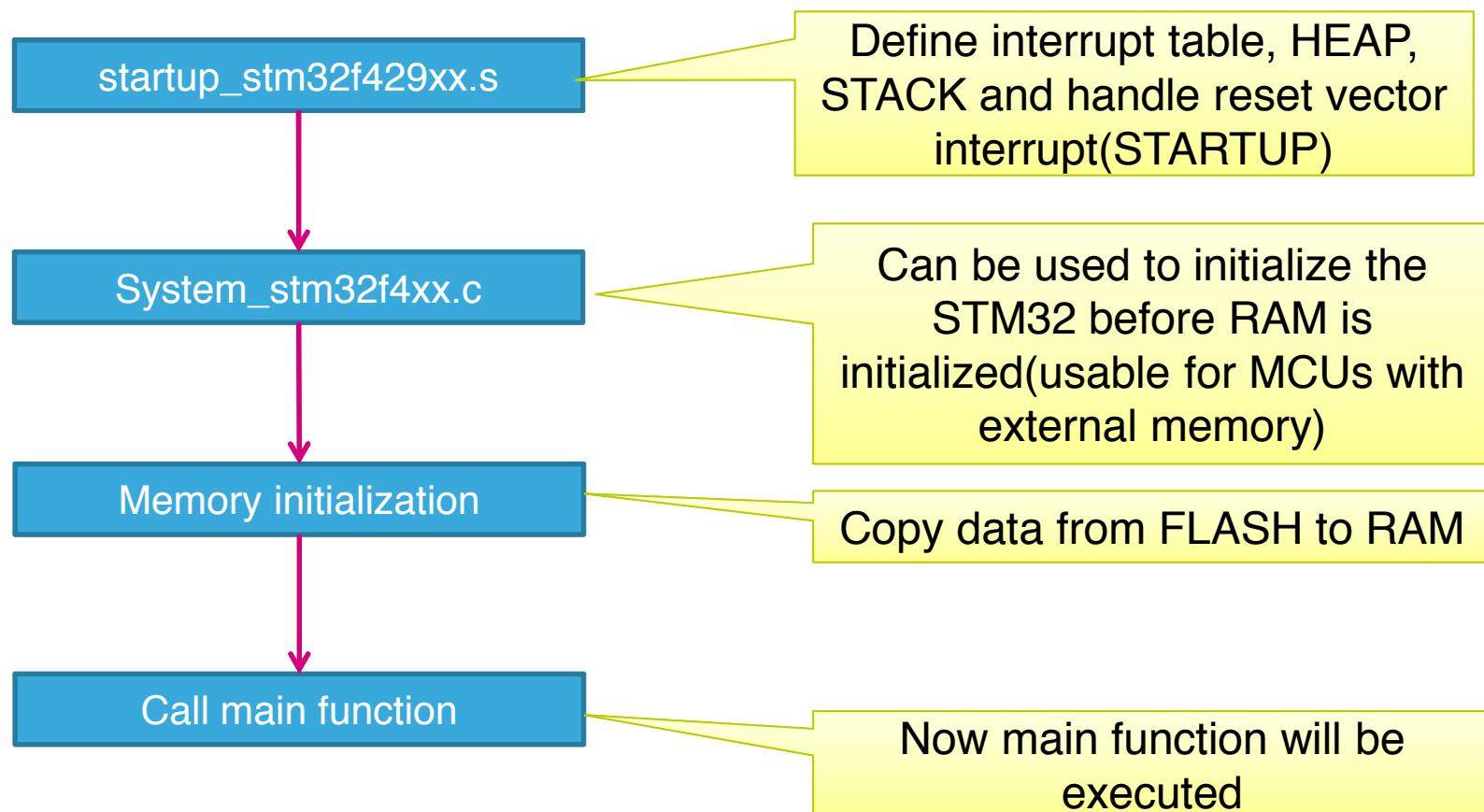
64



HAL general concepts

STM32 startup

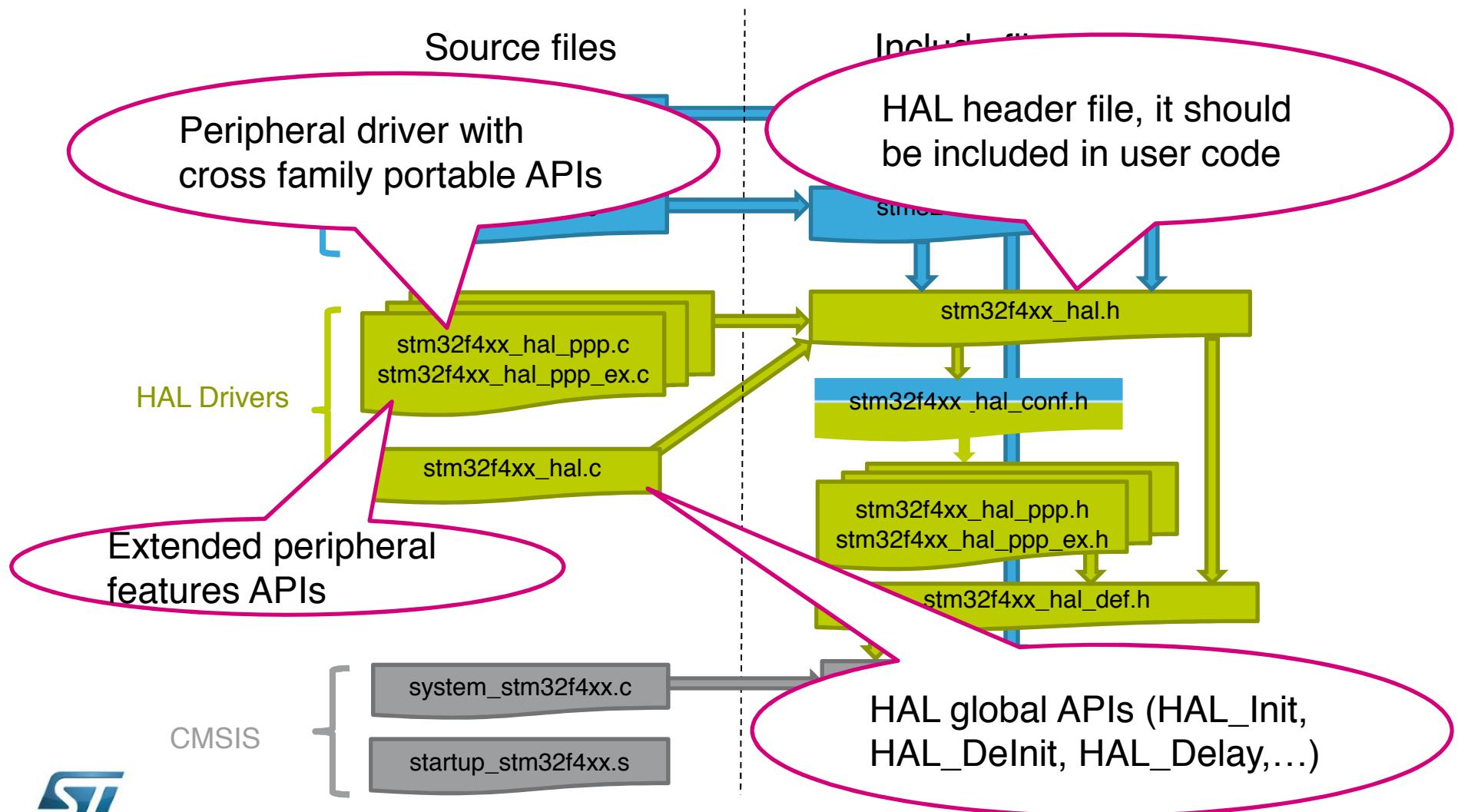
65



HAL general concepts

HAL based project organization

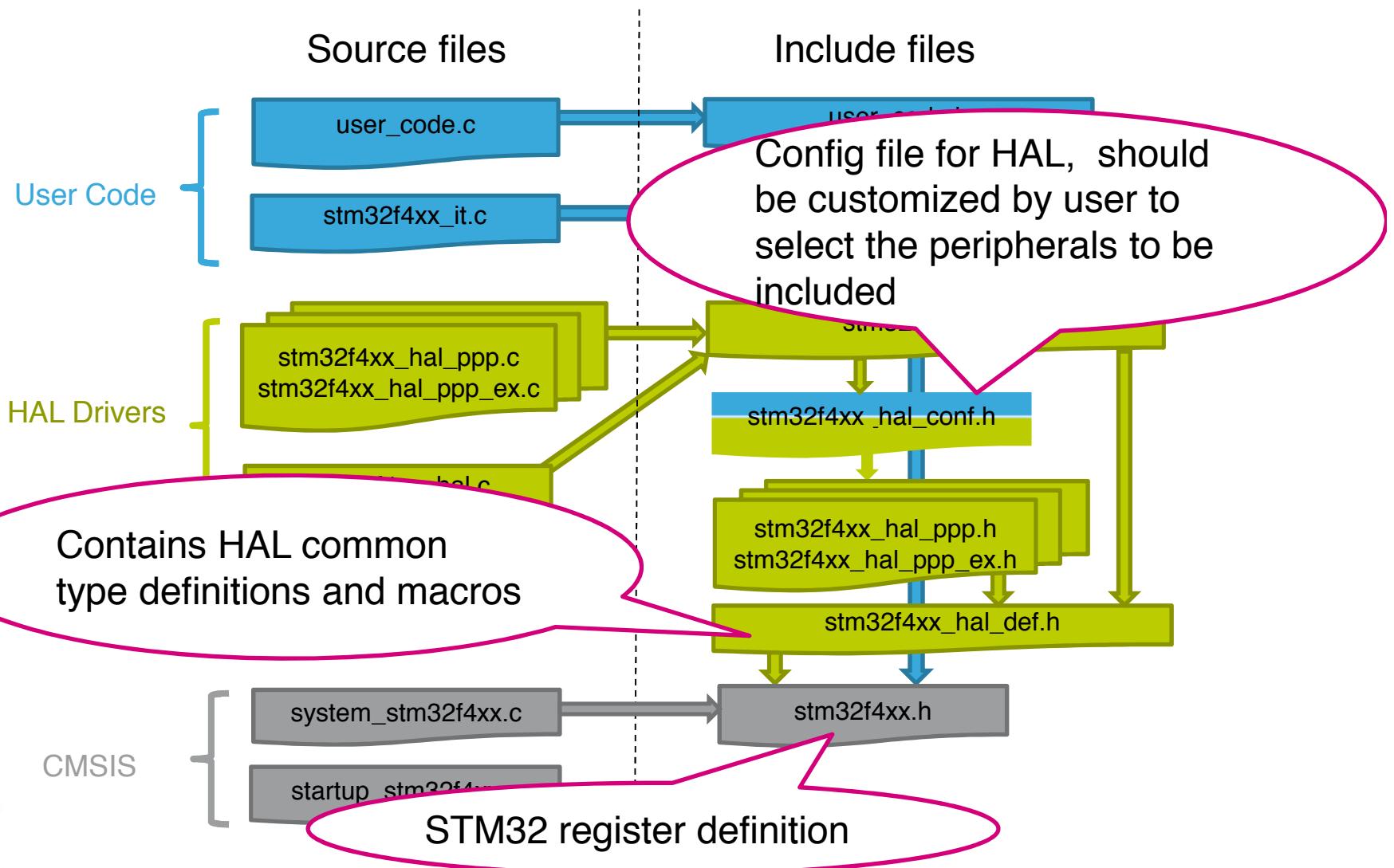
66



HAL general concepts

HAL based project organization

67



HAL general concepts

HAL drivers Vs. Refman peripherals

68

- In order to offer easy to use APIs, each HAL driver **handles a unique usage** of the peripheral
 - For example in F4x reference manual you have SPI peripheral which can be used as SPI or I2S. In this case the two **HAL module drivers** are defined:
 - stm32f4xx_hal_spi.c and stm32f4xx_i2s.c
 - In F4x family, this is also the case for
 - USART which can be: USART, UART, IRDA or Smartcard
 - FMC which can be: NOR, NAND, SRAM ,SDRAM or PCCARD
 - SDIO which can be: SD or SDMMC
- The system peripherals SYSCFG and EXTI do not have dedicated HAL drivers, they are intrinsically managed in the HAL

HAL general concepts

Compatibility Across STM32 Series

69

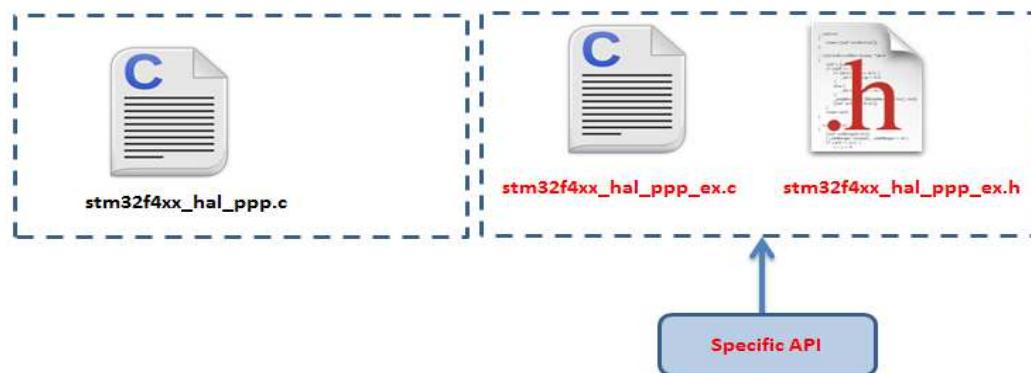
- The HAL offers Portable APIs across all the STM32 Series for what concerns the functional operations and routines of a given peripheral.
- However, considering the IPs specificities and implementation differences from one STM32 Serie to the other, the only discrepancies that may be found would be in the peripheral initialization (i.e. PPP_InitTypeDef) and in the RCC configuration (i.e. RCC_OscInitTypeDef and RCC_ClkInitTypeDef)

HAL general concepts

HAL extension APIs(1/3)

70

- **Case1:** APIs that are specific to **particular part numbers** within a product family



Example: case of Flash APIs which depend on part number in F4x family

```
/* Extension Program operation functions *****/
#ifndef STM32F427xx || defined(STM32F437xx) || defined(STM32F429xx) || defined(STM32F439xx) || \
    defined(STM32F401xC) || defined(STM32F401xE) || defined(STM32F411xE)
HAL_StatusTypeDef HAL_FLASHEx_AdvOBProgram (FLASH_AdvOBProgramInitTypeDef *pAdvOBInit);
void          HAL_FLASHEx_AdvOBGetConfig(FLASH_AdvOBProgramInitTypeDef *pAdvOBInit);
HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP(void);
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP(void);
#endif /* STM32F427xx || STM32F437xx || STM32F429xx || STM32F439xx || STM32F401xC || STM32F401xE || STM32F411xE */

#ifndef defined(STM32F427xx) || defined(STM32F437xx) || defined(STM32F429xx) || defined(STM32F439xx)
uint16_t      HAL_FLASHEx_OB_GetBank2WRP(void);
#endif /* STM32F427xx || STM32F437xx || STM32F429xx || STM32F439xx */
```

HAL general concepts

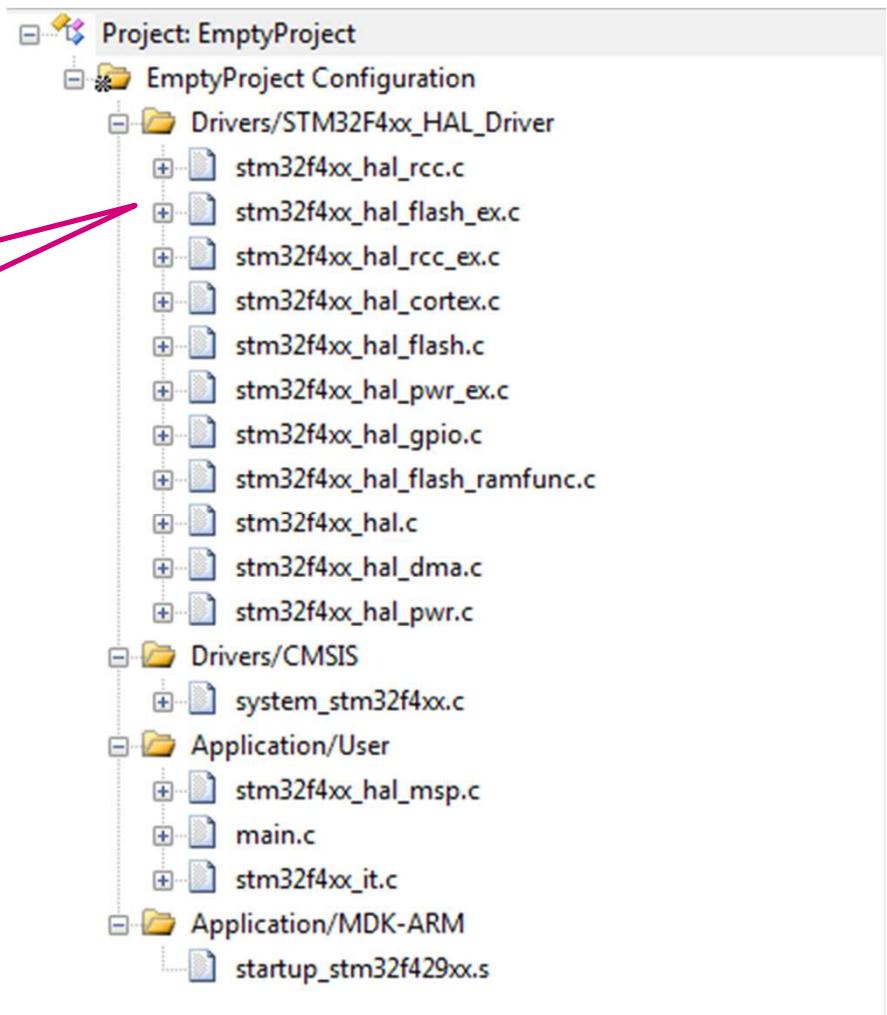
HAL extension APIs(1/3)

71

- **Case1:** APIs that are specific to **particular part numbers** within a product family

Example: case of Flash APIs which depend on part number in F4x family

Extension file with specific functions for F4xx parts

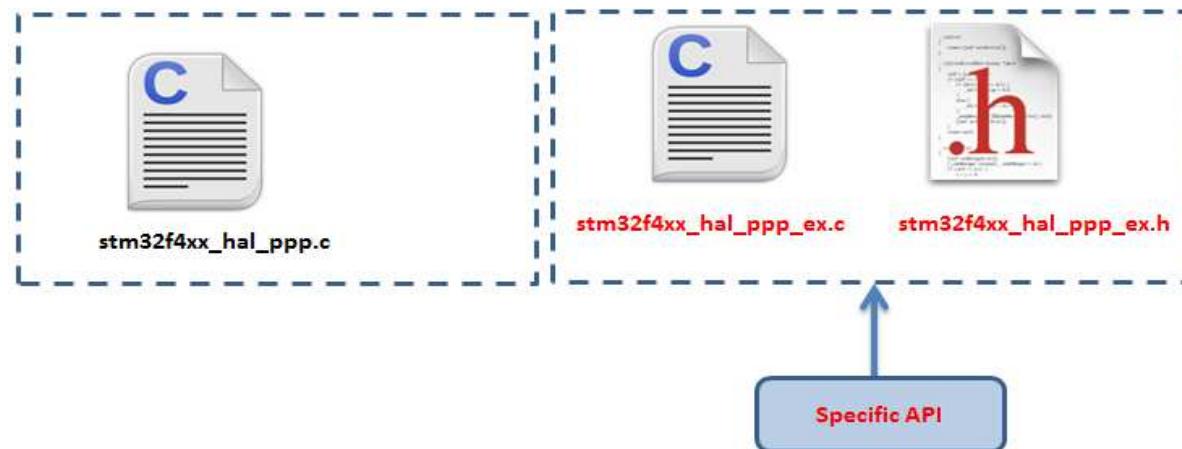


HAL general concepts

HAL extension APIs(2/3)

72

- **Case2:** APIs that are specific to a product family



Example: case of RCC extension APIs for F4x family

```
/* Exported functions -----*/
HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit);
void HAL_RCCEEx_GetPeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit);
```

HAL general concepts

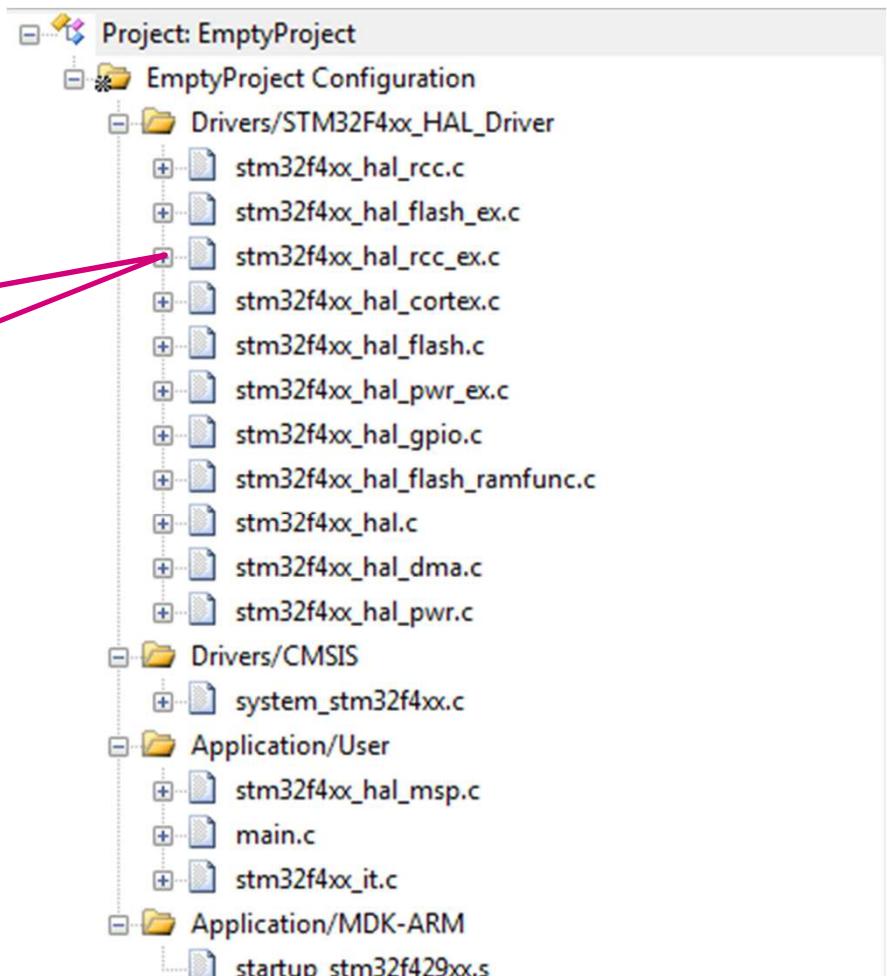
HAL extension APIs(2/3)

73

- **Case2:** APIs that are specific to **a product family**

Example: case of RCC extension
APIs for F4x family

Extension file with
specific functions for F4xx
family

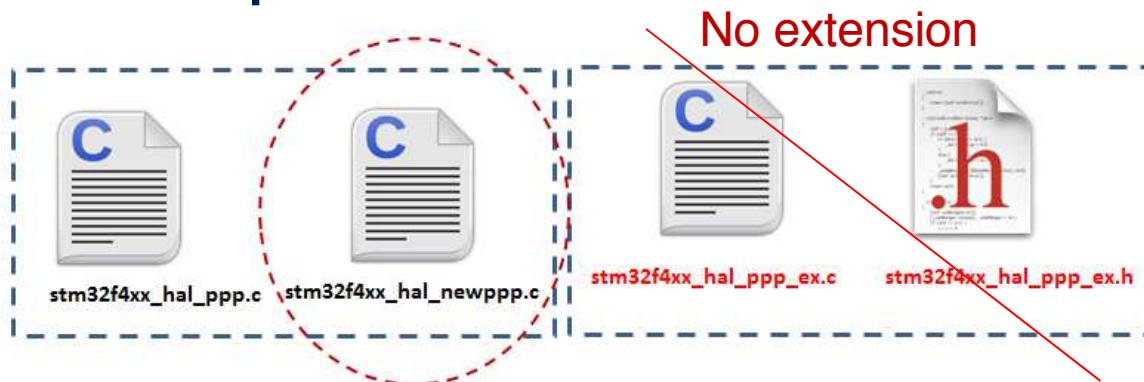


HAL general concepts

HAL extension APIs(3/3)

74

- **Case3:** In case a peripheral is present in particular **part numbers of one single family, no extension files** are used but simply a **new module driver is provided**



Example: LTDC is only present in F4x family in STM32F429 or STM32F439

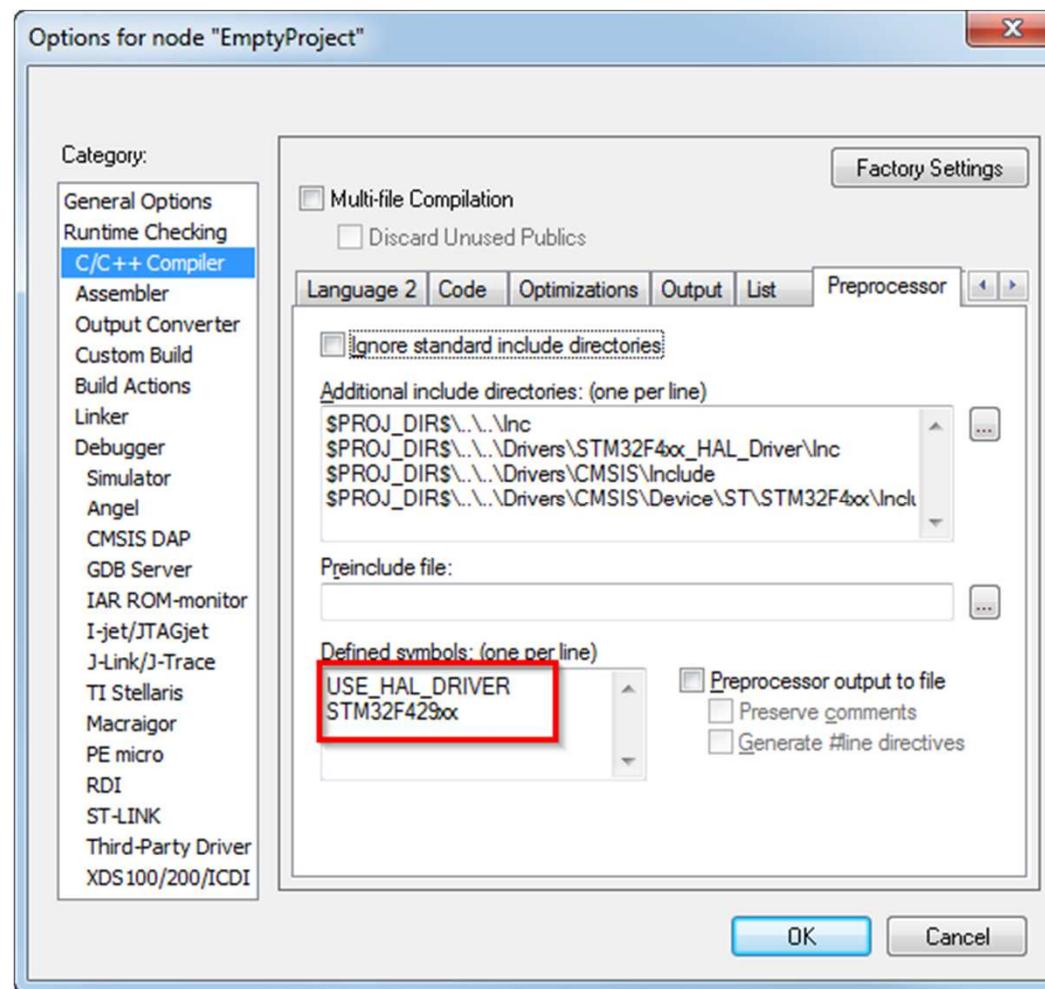
```
#ifdef HAL_LTDC_MODULE_ENABLED  
  
#if defined(STM32F429xx) || defined(STM32F439xx)  
  
/* Private typedef ----- */  
/* Private define ----- */  
/* Private macro ----- */  
/* Private variables ----- */  
/* Private function prototypes ----- */
```

HAL general concepts

Root part number selection

75

- The selection of root part number should be done through the project configuration by using specific P/N defines:



HAL general concepts

HAL configuration file

76

- The HAL config file `stm32f4xx_hal_conf.h` allows to select the modules to include:

```
/* ##### Module Selection ##### */
/** @brief This is the list of modules to be used in the HAL driver
 */
#define HAL_MODULE_ENABLED
##define HAL_ADC_MODULE_ENABLED
##define HAL_CAN_MODULE_ENABLED
##define HAL_CRC_MODULE_ENABLED
##define HAL_CRYP_MODULE_ENABLED
##define HAL_DAC_MODULE_ENABLED
##define HAL_DCMI_MODULE_ENABLED
##define HAL_DMA2D_MODULE_ENABLED
##define HAL_ETH_MODULE_ENABLED
##define HAL_NAND_MODULE_ENABLED
##define HAL_NOR_MODULE_ENABLED
```

- It defines also some system and HAL parameters including
 - HSE clock/HSI clock values
 - Instruction/data cache and prefetch queue setting
 - VDD voltage value



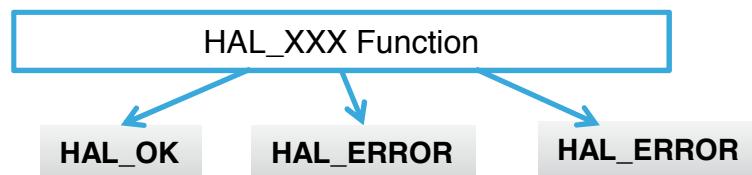
RTOS usage info (`#define USE_RTOS`)

HAL general concepts

HAL API returns

77

- A HAL API can return a value of enumerated type `HAL_StatusTypeDef`:
 - `HAL_OK` : API executed with success
 - `HAL_ERROR` : API call parameters error or operation execution error
 - `HAL_BUSY` : API was not executed because peripheral is busy with other operation
 - `HAL_TIMEOUT` : API timeout error

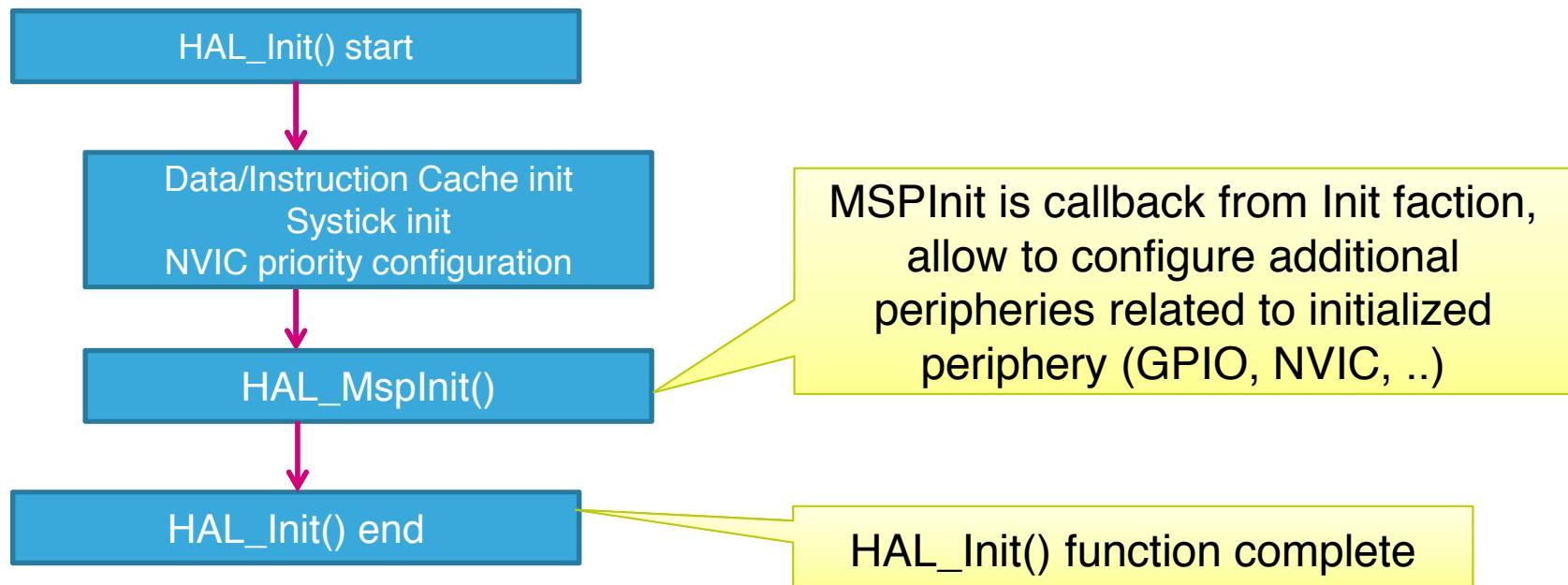


HAL general concepts

HAL global APIs HAL_Init()

78

- **HAL_Init() : need to be called as first function in main**
 - Initializes data/instruction cache and pre-fetch queue
 - Sets Time base to generate interrupt each 1ms with lowest priority , it can use systick (default) or other time source
 - Sets priority grouping to 4 preemption bits
 - Calls function HAL_MspInit() which a is user callback function to do MCU system level initializations (Clocks, GPIOs, DMA, Interrupts).

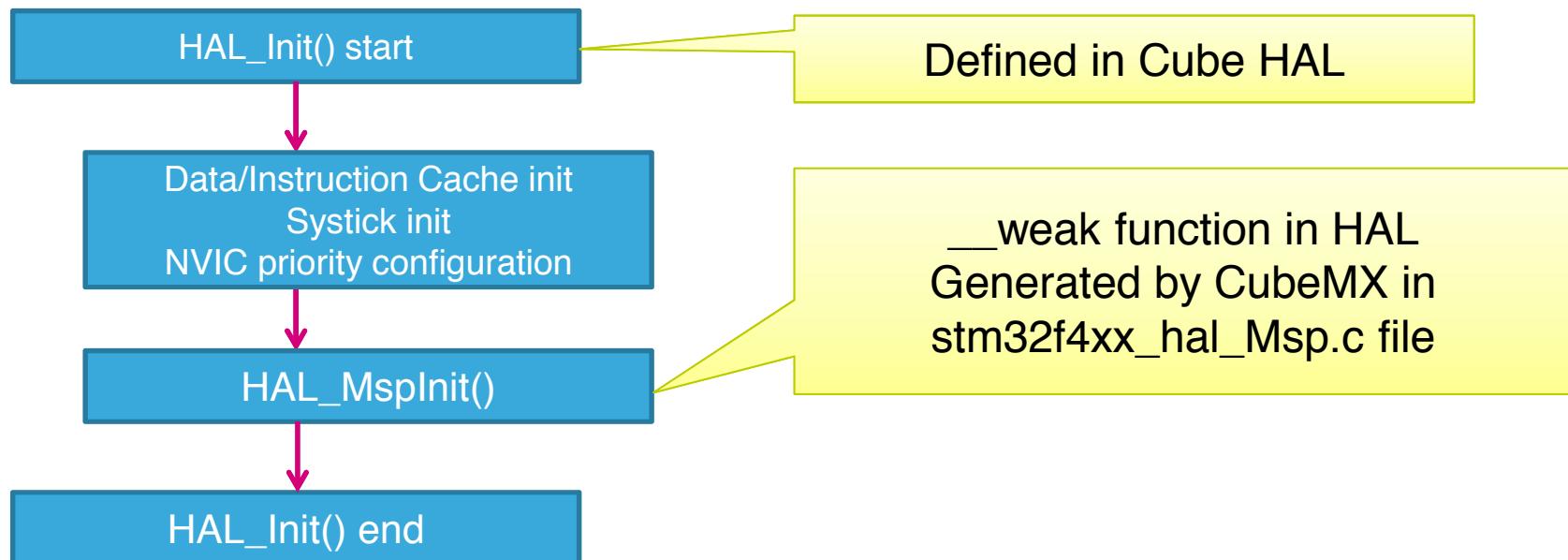


HAL general concepts

HAL global APIs HAL_Init()

79

- **HAL_Init() : need to be called as first function in main**
 - Initializes data/instruction cache and pre-fetch queue
 - Sets Time base to generate interrupt each 1ms with lowest priority , it can use systick (default) or other time source
 - Sets priority grouping to 4 preemption bits
 - Calls function HAL_MspInit() which is user callback function to do MCU system level initializations (Clocks, GPIOs, DMA, Interrupts).



HAL general concepts

HAL global APIs HAL_Init()

80

- HAL_Init() : need to be called as first function in main

```
HAL_StatusTypeDef HAL_Init(void)
{
    /* Configure Flash prefetch, Instruction cache, Data cache */
    #if (INSTRUCTION_CACHE_ENABLE != 0)
        __HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
    #endif /* INSTRUCTION_CACHE_ENABLE */
    #if (DATA_CACHE_ENABLE != 0)
        __HAL_FLASH_DATA_CACHE_ENABLE();
    #endif /* DATA_CACHE_ENABLE */
    #if (PREFETCH_ENABLE != 0)
        __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
    #endif /* PREFETCH_ENABLE */

    /* Set Interrupt Group Priority */
    HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
    /* Use systick as time base source and configure 1ms tick (default clock after Reset is HSI) */
    HAL_InitTick(TICK_INT_PRIORITY);
    /* Init the low level hardware */
    HAL_MspInit();
    /* Return function status */
    return HAL_OK;
}
```

Instruction/data cache and
prefetch initialization based
on stm32f4xx_hal_conf.h

HAL general concepts

HAL global APIs HAL_Init()

81

- HAL_Init() : need to be called as first function in main

```
HAL_StatusTypeDef HAL_Init(void)
{
    /* Configure Flash prefetch, Instruction cache, Data cache */
#if (INSTRUCTION_CACHE_ENABLE != 0)
    __HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
#endif /* INSTRUCTION_CACHE_ENABLE */
#if (DATA_CACHE_ENABLE != 0)
    __HAL_FLASH_DATA_CACHE_ENABLE();
#endif /* DATA_CACHE_ENABLE */
#if (PREFETCH_ENABLE != 0)
    __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
#endif /* PREFETCH_ENABLE */
    /* Set Interrupt Group Priority */
    HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
    /* Use systick as time base source and configure 1ms tick (default clock after Reset is HSI) */
    HAL_InitTick(TICK_INT_PRIORITY);
    /* Init the low level hardware */
    HAL_MspInit();
    /* Return function status */
    return HAL_OK;
}
```

NVIC priority grouping setup

Default systick init with default interrupt priority 0 (HIGH priority)

HAL general concepts

HAL global APIs HAL_Init()

82

- HAL_Init() : need to be called as first function in main

```
HAL_StatusTypeDef HAL_Init(void)
{
    /* Configure Flash prefetch, Instruction cache, Data cache */
#if (INSTRUCTION_CACHE_ENABLE != 0)
    __HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
#endif /* INSTRUCTION_CACHE_ENABLE */
#if (DATA_CACHE_ENABLE != 0)
    __HAL_FLASH_DATA_CACHE_ENABLE();
#endif /* DATA_CACHE_ENABLE */
#if (PREFETCH_ENABLE != 0)
    __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
#endif /* PREFETCH_ENABLE */
    /* Set Interrupt Group Priority */
    HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
    /* Use systick as time base source and configure it */
    HAL_InitTick(TICK_INT_PRIORITY);
    /* Init the low level hardware */
    HAL_MspInit();
    /* Return function status */
    return HAL_OK;
}
```

HAL_MspInit callback
function generated by
CubeMX in
stm32f4xx_hal_msp.c



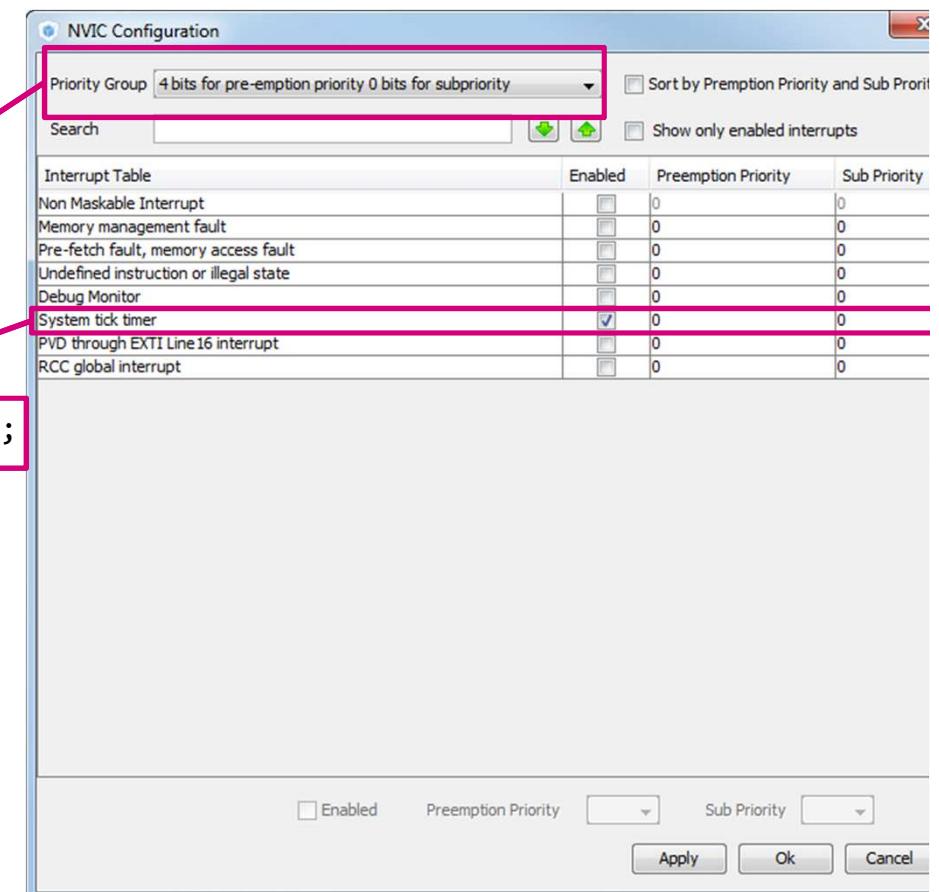
HAL general concepts

HAL global APIs HAL_Init()

83

- HAL_MspInit() generated by CubeMX
- contains setup selected in CubeMX

```
/**  
 * Initializes the Global MSP.  
 */  
void HAL_MspInit(void)  
{  
    /* USER CODE BEGIN MspInit 0 */  
  
    /* USER CODE END MspInit 0 */  
  
    HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);  
  
    /* System interrupt init */  
    /* Systick_IRQn interrupt configuration */  
    HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);  
  
    /* USER CODE BEGIN MspInit 1 */  
  
    /* USER CODE END MspInit 1 */  
}
```



HAL general concepts

HAL global APIs HAL_Init()

84

- HAL_MspInit() generated by CubeMX
- contains setup selected in CubeMX

```
/**  
 * Initializes the Global MSP.  
 */  
void HAL_MspInit(void)  
{  
    /* USER CODE BEGIN MspInit 0 */  
  
    /* USER CODE END MspInit 0 */  
  
    HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);  
  
    /* System interrupt init*/  
    /* SystTick_IRQn interrupt configuration */  
    HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);  
  
    /* USER CODE BEGIN MspInit 1 */  
    /* USER CODE END MspInit 1 */  
}
```

Space for additional user initialization

Space for additional user initialization

HAL general concepts

HAL global APIs (1/2)

85

- Implemented in file **stm32f4x_hal.c**, main function are:
 - HAL_DeInit()
 - Resets all peripherals
 - Calls function HAL_MspDeInit() which a is user callback function to do system level De-Initializations. HAL_MspDeInit() is defined as “weak” empty function in HAL
 - HAL_GetTick()
 - Get current tick counter (incremented in Time base interrupt)
 - **Used by peripherals drivers for timeout management**
 - HAL_Delay()
 - Implements a delay in ms
 - HAL_InitTick()
 - Weak function that configures the source of the time base, by default it is systick but can be redefined by user

HAL general concepts

HAL global APIs (2/2)

86

- HAL_IncTick()
 - Increment a global variable "uwTick" used as application time base . Should be called from Time base ISR (systick, timer, ...)
- HAL_SuspendTick() / HAL_ResumeTick()
 - Suspend Tick increment
 - Resume Tick increment.
- HAL_GetHalVersion()
 - Returns the HAL revision

HAL general concepts

HAL Tick functions

87

- Are defined as `__weak` by default used systick clocked from AHB
- In case you want change the clock source you need to change all Tick methods
 - `HAL_GetTick`
 - `HAL_InitTick`
 - `HAL_SuspendTick()`
 - `HAL_ResumeTick()`
- Use `HAL_SuspendTick` before MCU enter into low power mode, interrupt can wake up STM32 from LP mode
- After wakeup use again `HAL_ResumeTick` for enable time domain



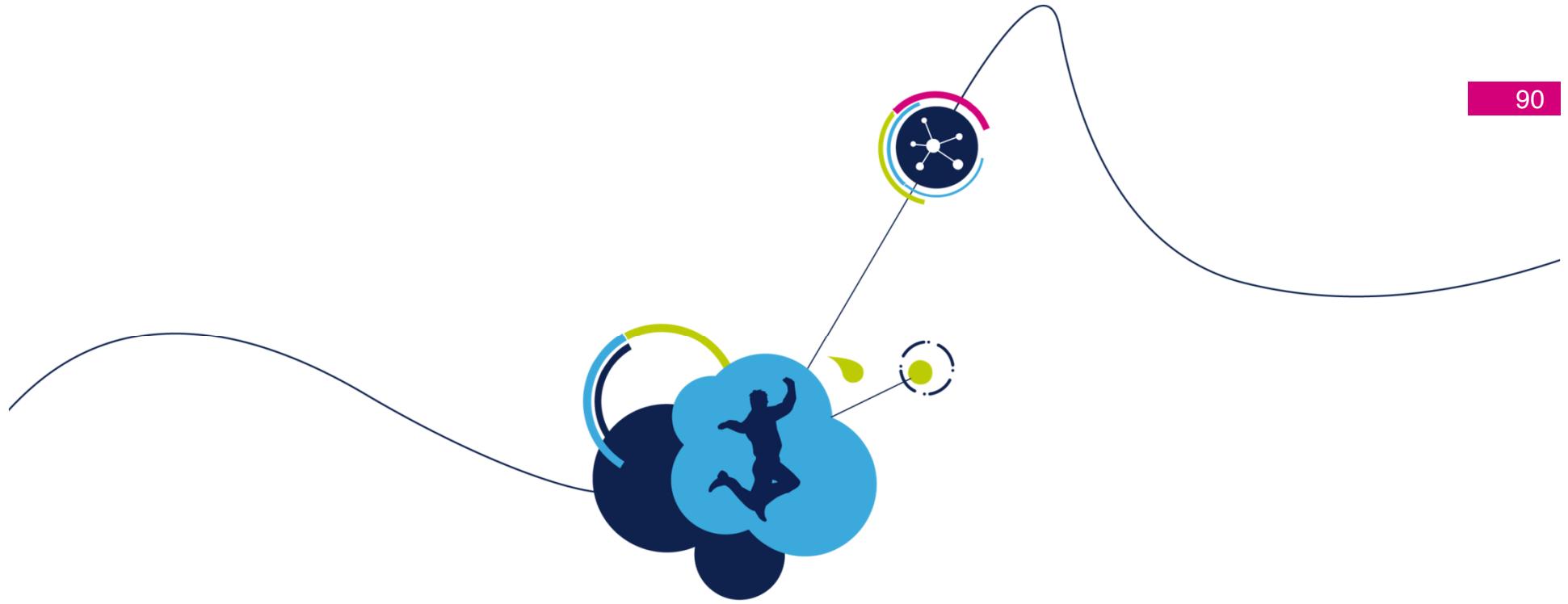
HAL service peripherals

HAL service peripherals

Introduction

89

- HAL service peripherals the main system IPs , including:
 - GPIO
 - RCC
 - DMA
 - Cortex (NVIC and Systick APIs)
 - PWR (power management APIs)
- The HAL offers simple to use, portable APIs for above system IPs with extensions provided also for non common features (ex: product specific low power mode, specific clock configurations ,....)



HAL service peripherals - RCC Reset and Clock Control

HAL service peripherals RCC macros

91

- Simple macros for RCC operations
 - Enabling peripheral clocks (_CLK_ENABLE)
 - Disabling peripheral clocks (_CLK_DISABLE)
 - Force periphery reset (_FORCE_RESET)
 - Release periphery reset (RELEASE_RESET)
- RCC macro example:

`_PERIPH_ACTION();`

Name of periphery:
GPIOA, USART1,
TIM1, ...

Enable peripheral clock

Ex.:

`_GPIOA_CLK_ENABLE();`

Disable peripheral clock

Ex.:

`_GPIOA_CLK_DISABLE();`

Force periphery reset

Ex.:

`_GPIOA_FORCE_RESET();`

Release periphery reset

Ex.:

`_GPIOA_RELEASE_RESET();`

HAL service peripherals

Enabling Peripherals

92

- All peripherals clocks can be enabled using the RCC macros
- A peripheral clock should be enabled before being used.

```
/* Exported macro -----*/  
  
/** @brief Enable or disable the AHB1 peripheral clock.  
 * @note After reset, the peripheral clock (used for registers read/write access)  
 *       is disabled and the application software has to enable this clock before  
 *       using it.  
 */  
#define __GPIOA_CLK_ENABLE()          (RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOAEN))  
#define __GPIOB_CLK_ENABLE()          (RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOBEN))  
#define __GPIOC_CLK_ENABLE()          (RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOCEN))  
#define __GPIOD_CLK_ENABLE()          (RCC->AHB1ENR |= (RCC_AHB1ENR_GPIODEN))  
#define __GPIOE_CLK_ENABLE()          (RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOEEN))  
#define __GPIOH_CLK_ENABLE()          (RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOHEN))  
#define __CRC_CLK_ENABLE()           (RCC->AHB1ENR |= (RCC_AHB1ENR_CRCEN))  
#define __BKPSRAM_CLK_ENABLE()        (RCC->AHB1ENR |= (RCC_AHB1ENR_BKPSRAMEN))  
#define __CCMDATARAMEN_CLK_ENABLE()   (RCC->AHB1ENR |= (RCC_AHB1ENR_CCMDATARAMEN))  
#define __DMA1_CLK_ENABLE()           (RCC->AHB1ENR |= (RCC_AHB1ENR_DMA1EN))  
#define __DMA2_CLK_ENABLE()           (RCC->AHB1ENR |= (RCC_AHB1ENR_DMA2EN))  
  
#define __GPIOA_CLK_DISABLE()         (RCC->AHB1ENR &= ~(RCC_AHB1ENR_GPIOAEN))  
#define __GPIOB_CLK_DISABLE()         (RCC->AHB1ENR &= ~(RCC_AHB1ENR_GPIOBEN))  
#define __GPIOC_CLK_DISABLE()         (RCC->AHB1ENR &= ~(RCC_AHB1ENR_GPIOCEN))  
#define __GPIOD_CLK_DISABLE()         (RCC->AHB1ENR &= ~(RCC_AHB1ENR_GPIODEN))  
#define __GPIOE_CLK_DISABLE()         (RCC->AHB1ENR &= ~(RCC_AHB1ENR_GPIOEEN))  
#define __GPIOH_CLK_DISABLE()         (RCC->AHB1ENR &= ~(RCC_AHB1ENR_GPIOHEN))  
#define __CRC_CLK_DISABLE()          (RCC->AHB1ENR &= ~(RCC_AHB1ENR_CRCEN))  
#define __BKPSRAM_CLK_DISABLE()       (RCC->AHB1ENR &= ~(RCC_AHB1ENR_BKPSRAMEN))  
#define __CCMDATARAMEN_CLK_DISABLE() (RCC->AHB1ENR &= ~(RCC_AHB1ENR_CCMDATARAMEN))  
#define __DMA1_CLK_DISABLE()          (RCC->AHB1ENR &= ~(RCC_AHB1ENR_DMA1EN))  
#define __DMA2_CLK_DISABLE()          (RCC->AHB1ENR &= ~(RCC_AHB1ENR_DMA2EN))
```

In file
stm32f4xx_hal_rcc.l

Enable Periphery
register access

Disable periphery
register access

HAL service peripherals

Reset Peripherals

93

- All peripherals can be reset to their default start using the RCC macros
- After Force periphery rese the reset must be release otherwise the periphery will not react on changes

```
/** @brief Force or release AHB1 peripheral reset.  
 */  
  
#define __AHB1_FORCE_RESET()      (RCC->AHB1RSTR = 0xFFFFFFFF)  
#define __GPIOA_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_GPIOARST))  
#define __GPIOB_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_GPIOBRST))  
#define __GPIOC_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_GPIOCRST))  
#define __GPIOD_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_GPIODRST))  
#define __GPIOE_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_GPIOERST))  
#define __GPIOH_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_GPIOHRST))  
#define __CRC_FORCE_RESET()     (RCC->AHB1RSTR |= (RCC_AHB1RSTR_CRCRST))  
#define __DMA1_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_DMA1RST))  
#define __DMA2_FORCE_RESET()    (RCC->AHB1RSTR |= (RCC_AHB1RSTR_DMA2RST))  
  
#define __AHB1_RELEASE_RESET()   (RCC->AHB1RSTR = 0x00)  
#define __GPIOA_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOARST))  
#define __GPIOB_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOBRST))  
#define __GPIOC_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOCRST))  
#define __GPIOD_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIODRST))  
#define __GPIOE_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOERST))  
#define __GPIOF_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOFRST))  
#define __GPIOG_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOGRST))  
#define __GPIOH_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOHRST))  
#define __GPIOI_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIOIRST))  
#define __CRC_RELEASE_RESET()   (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_CRCRST))  
#define __DMA1_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_DMA1RST))  
#define __DMA2_RELEASE_RESET()  (RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_DMA2RST))
```

Held periphery in
RESET, periphery
registers are in
default state

In file
stm32f4xx_hal_rcc.h

Release RESET,
periphery can be
now enabled

HAL service peripherals

RCC – macros

94

- RCC macros are defined in `stm32f4xx_hal_rcc.h` file, **they are needed** for peripherals clock gating, reset control and sleep mode clock config
 - Peripheral clock enable/disable: `__TIM1_CLK_ENABLE()` / `__TIM1_CLK_DISABLE()`
 - Peripheral reset enable/disable: `__GPIOA_FORCE_RESET()`/ `__GPIOA_RELEASE_RESET()`
 - Peripheral Sleep mode clock enable/disable: `__SPI3_CLK_SLEEP_ENABLE()`/ `SPI3_CLK_SLEEP_DISABLE()`
- Other RCC macros are available for direct register access, they can be used in the application instead of the HAL clock config APIs, for example, when restoring clock after Stop mode, user can use directly below macros
 - `__HAL_RCC_HSE_CONFIG`
 - `__HAL_RCC_PLL_ENABLE`
 - `__HAL_RCC_GET_FLAG`

HAL service peripherals

RCC - clock configuration (1/2)

95

- Two functions needed for clock configuration
 - HAL_RCC_OscConfig
 - HAL_RCC_ClockConfig
- **HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct)**
 - Configures/Enables multiple clock sources : HSE, HSI, LSE, LSI, PLL

```
/**  
 * @brief  RCC Internal/External Oscillator (HSE, HSI, LSE and LSI) configuration structure definition  
 */  
typedef struct  
{  
    uint32_t OscillatorType;      /*!< The oscillators to be configured.  
                                This parameter can be a value of @ref RCC_Oscillator_Type */  
    uint32_t HSEState;           /*!< The new state of the HSE.  
                                This parameter can be a value of @ref RCC_HSE_Config */  
    uint32_t LSEState;           /*!< The new state of the LSE.  
                                This parameter can be a value of @ref RCC_LSE_Config */  
    uint32_t HSISState;          /*!< The new state of the HSI.  
                                This parameter can be a value of @ref RCC_HSI_Config */  
    uint32_t HSICalibrationValue; /*!< The calibration trimming value.  
                                This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F */  
    uint32_t LSISState;          /*!< The new state of the LSI.  
                                This parameter can be a value of @ref RCC_LSI_Config */  
    RCC_PLLInitTypeDef PLL;     /*!< PLL structure parameters */  
}RCC_OscInitTypeDef;
```

HAL service peripherals

RCC - clock configuration (1/2)

96

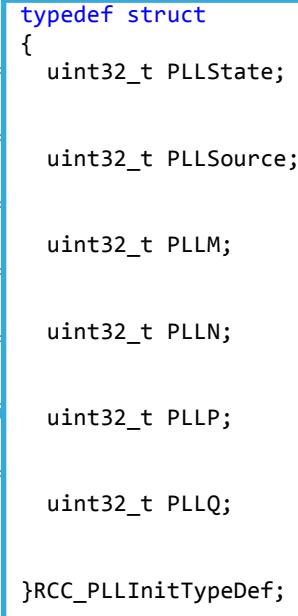
- Two functions needed for clock configuration

- HAL_RCC_OscConfig
- HAL_RCC_ClockConfig

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct)

- Configures/Enables multiple clock sources : HSE, HSI, LSE, LSI, PLL

```
/**  
 * @brief  RCC Internal/External  /** @brief  RCC PLL configuration structure definition  
 */  
typedef struct  
{  
    uint32_t OscillatorType;          /*!< The new state of the PLL.  
                                         This parameter can be a value of @ref RCC_PLL_Config */  
    uint32_t HSEState;               /*!< RCC_PLLSource: PLL entry clock source.  
                                         This parameter must be a value of @ref RCC_PLL_Clock_Source */  
    uint32_t LSEState;               /*!< PLLM: Division factor for PLL VCO input clock.  
                                         This parameter must be a number between Min_Data = 0 and Max_Data = 63 */  
    uint32_t HSISState;              /*!< PLLN: Multiplication factor for PLL VCO output clock.  
                                         This parameter must be a number between Min_Data = 192 and Max_Data = 192 */  
    uint32_t HSICalibrationValue;    /*!< PLLP: Division factor for main system clock (SYSCLK).  
                                         This parameter must be a value of @ref RCC_PLLP_Clock_Divider */  
    uint32_t LSISState;              /*!< PLLQ: Division factor for OTG FS, SDIO and RNG clocks.  
                                         This parameter must be a number between Min_Data = 0 and Max_Data = 63 */  
    RCC_PLLInitTypeDef PLL;         /*!< RCC_OscInitTypeDef */  
}RCC_OscInitTypeDef;
```



HAL service peripherals

RCC - clock configuration (2/2)

97

- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency)
 - Selects system clock source
 - Configures AHB, APB1 and APB2 clock dividers
 - Configures Flash Wait States
 - Updates systick config following HCLK clock changes to generate 1ms timebase

```
/**  
 * @brief  RCC System, AHB and APB busses clock configuration structure definition  
 */  
typedef struct  
{  
    uint32_t ClockType;           /*!< The clock to be configured.  
                                 This parameter can be a value of @ref RCC_System_Clock_Type      */  
  
    uint32_t SYSCLKSource;        /*!< The clock source (SYSCLKS) used as system clock.  
                                 This parameter can be a value of @ref RCC_System_Clock_Source    */  
  
    uint32_t AHBCLKDivider;       /*!< The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK).  
                                 This parameter can be a value of @ref RCC_AHB_Clock_Source      */  
  
    uint32_t APB1CLKDivider;       /*!< The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK).  
                                 This parameter can be a value of @ref RCC_APB1_APB2_Clock_Source */  
  
    uint32_t APB2CLKDivider;       /*!< The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK).  
                                 This parameter can be a value of @ref RCC_APB1_APB2_Clock_Source */  
}  
RCC_ClkInitTypeDef;
```

HAL service peripherals

RCC - clock configuration (2/2)

98

- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency)
 - Selects system clock source
 - Configures AHB, APB1 and APB2 clock dividers
 - Configures Flash Wait States
 - Updates systick config following HCLK clock changes to generate 1ms timebase

```
/**  
 * @brief  RCC System, AHB and APB busses clock configuration structure definition  
 */  
typedef struct  
{  
    uint32_t ClockType;           /*!< The clock to be configured.  
                                 This parameter can be a value of @ref RCC_System_Clock_Type      */  
  
    uint32_t SYSCLKSource;        /*!< The clock source (SYSCLKS) used as system clock.  
                                 This parameter can be a value of @ref RCC_System_Clock_Source    */  
  
    uint32_t AHBCLKDivider;       /*!< The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK).  
                                 This parameter can be a value of @ref RCC_AHB_Clock_Source      */  
  
    uint32_t APB1CLKDivider;       /*!< The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK).  
                                 This parameter can be a value of @ref RCC_APB1_APB2_Clock_Source */  
  
    uint32_t APB2CLKDivider;       /*!< The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK).  
                                 This parameter can be a value of @ref RCC_APB1_APB2_Clock_Source */  
}  
RCC_ClkInitTypeDef;
```

HAL service peripherals RCC— APIs

99

- Main function structure, generated by CubeMX

```
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */
    /* MCU Configuration----*/
    /* Reset of all peri
    HAL_Init();           Initialize Systick,  
NVIC, and call  
HAL_Init_MSP
    /* Configure the system clock */
    SystemClock_Config(); Initialize STM32  
clock tree, based on  
Clock Configuration  
in CubeMX
    /* Initialize all configu
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */

    /* USER CODE END 2 */
    /* USER CODE BEGIN 3 */
    /* Infinite loop */
    while (1)
    {

    }
    /* USER CODE END 3 */
}
```

HAL service peripherals

RCC – APIs

100

- MX_GPIO_Init function structure, generated by CubeMX

```
/** System Clock Configuration
*/
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;

    __PWR_CLK_ENABLE();

    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    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 = 8;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK|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;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);

}
```

Enable clocks for
PWR periphery

Configure voltage
scale (refer RM)

HAL service peripherals

RCC – APIs

101

- SystemClock_Config function structure, generated by CubeMX

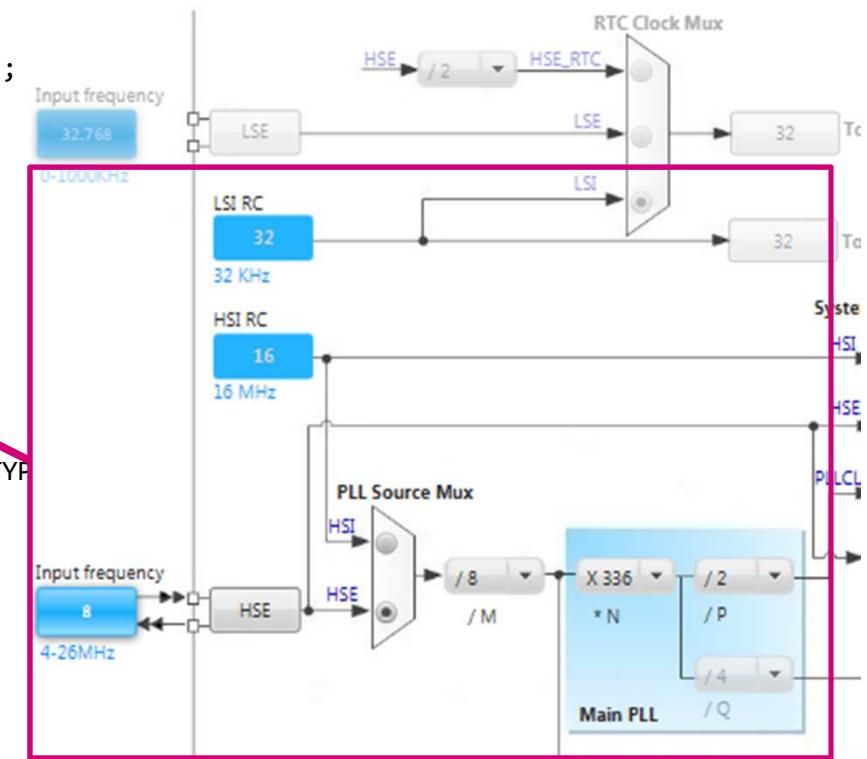
```
/** System Clock Configuration
*/
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;

    __PWR_CLK_ENABLE();

    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    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 = 8;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK|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;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
}
```



HAL service peripherals

RCC – APIs

102

- SystemClock_Config function structure, generated by CubeMX

```
/** System Clock Configuration */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;

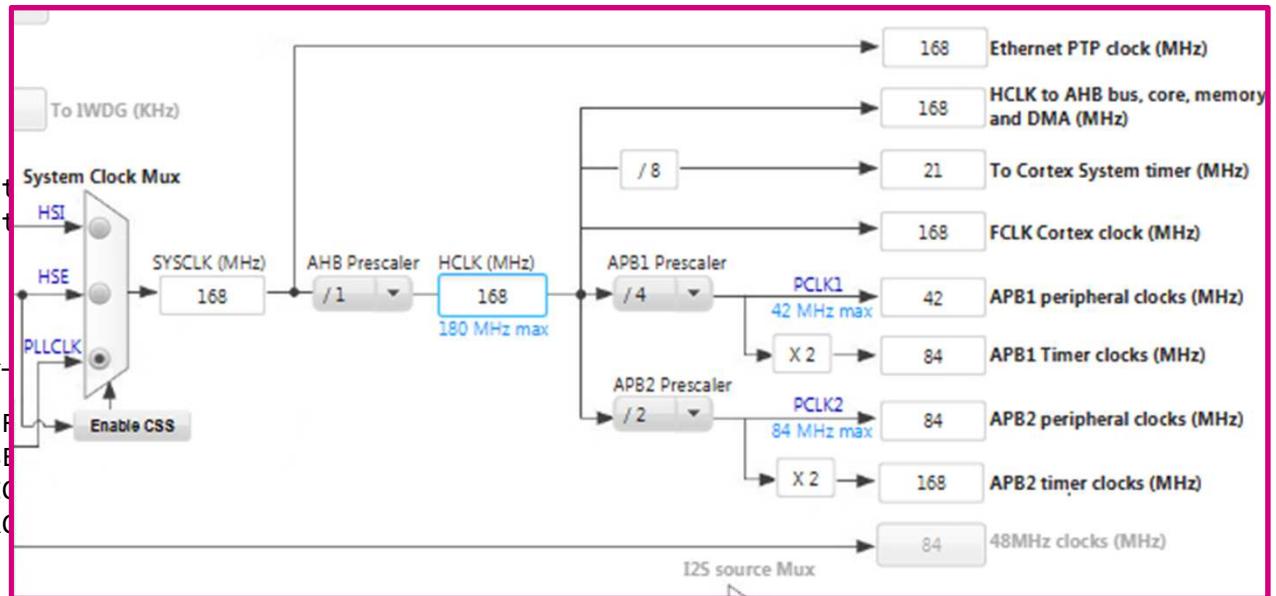
    __PWR_CLK_ENABLE();

    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_VOS_0);

    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 = 8;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK | 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;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);

}
```



HAL service peripherals

RCC – other APIs

103

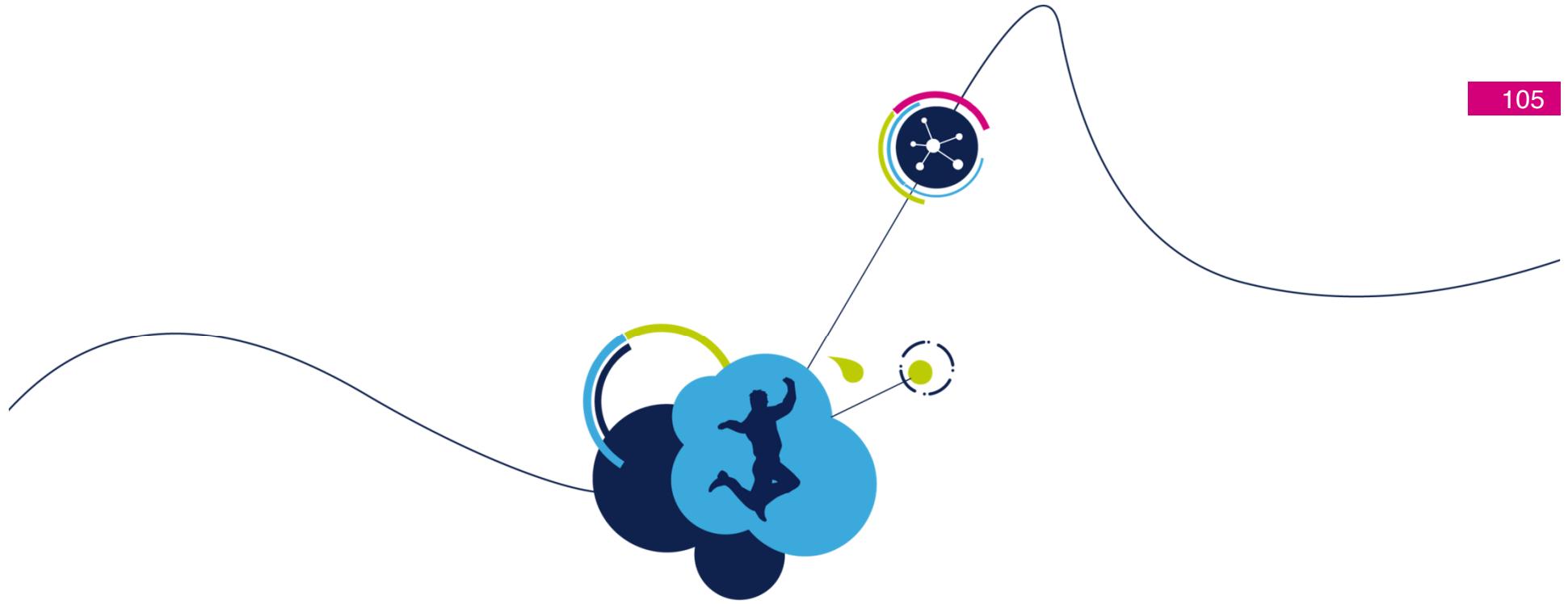
- Clock DelInit: returns to reset clock config
 - HAL_RCC_DelInit()
- Get Clock functions with config read from RCC registers
 - HAL_RCC_GetSysClockFreq()
 - HAL_RCC_GetHCLKFreq()
 - HAL_RCC_GetPCLK1Freq()
 - HAL_RCC_GetPCLK2Freq()
 - HAL_RCC_GetOscConfig()
 - HAL_RCC_GetClockConfig()
- MCO clock selection (it includes the GPIO AF config as MCO)
 - HAL_RCC_MCOConfig()
- CSS enable/disable and interrupt handling
 - HAL_RCC_EnableCSS() / HAL_RCC_DisableCSS()
 - HAL_RCC_NMI_IRQHandler()
 - HAL_RCC_CSSCallback()

HAL service peripherals

RCC – extension APIs

104

- For RCC block, the extension APIs cover specific product or family clock tree feature, for example
 - In F4x family, extension APIs (defined in file `stm32f4xx_hal_rcc_ex.c`) include:
 - Clock configuration of dedicated PLLs for SAI and I2S peripherals (PLLI2S, PLLSAI”)
 - Clock source selection for RTC block
 - In L0x family, extension APIs (defined in `stm32l0xx_hal_rcc_ex.c`) include:
 - Clock source selection for some peripherals: USART1, USART2, LPUART, I2C, RTC, USB,...
 - LSE clock security system enable/disable
 - Clock Recovery system (CRS) config



HAL service peripherals – Core Core peripherals handling

HAL service peripherals

Cortex

106

- Cortex HAL driver provides APIs for handling NVIC and Systick, supported APIs include
 - HAL_NVIC_SetPriorityGrouping
 - HAL_NVIC_SetPriority
 - HAL_NVIC_EnableIRQ /HAL_NVIC_DisableIRQ
 - HAL_SYSTICK_Config
 - HAL_SYSTICK_CLKSourceConfig

HAL service peripherals

NVIC Priority Grouping

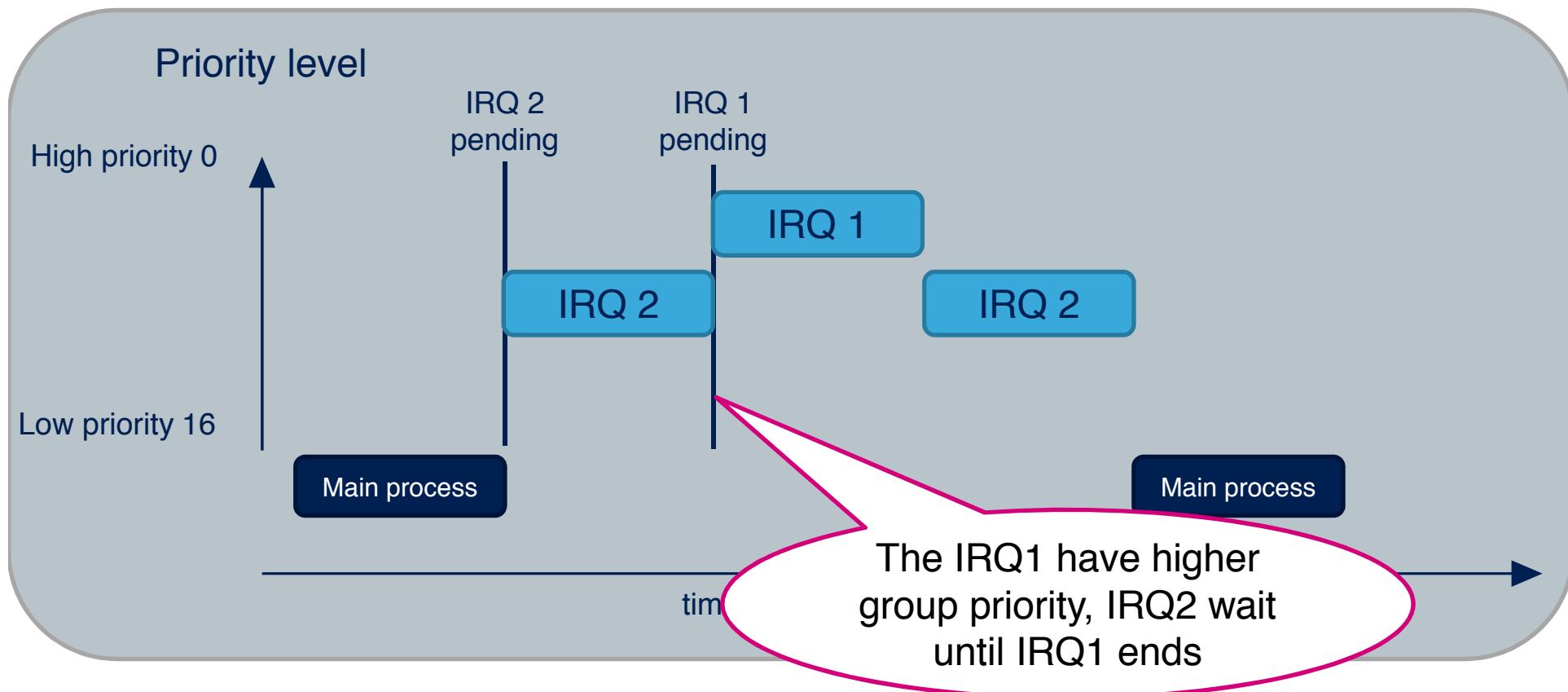
107

- STM32 allow to use up to 4bits for Interrupt priorities
- User can decide how many bits will be used for Pre-empt (Group) priorities and for Subpriorities
 - By default set in HAL_Init() to 4bit for Group priorities
- Possible settings:

NVIC_PRIORITYGROUP_0	Subpriority bit 3	Subpriority bit 2	Subpriority bit 1	Subpriority bit 0
NVIC_PRIORITYGROUP_1	Pre-emption bit 0	Subpriority bit 2	Subpriority bit 1	Subpriority bit 0
NVIC_PRIORITYGROUP_2	Pre-emption bit 1	Pre-emption bit 0	Subpriority bit 1	Subpriority bit 0
NVIC_PRIORITYGROUP_3	Pre-emption bit 2	Pre-emption bit 1	Pre-emption bit 0	Subpriority bit 0
NVIC_PRIORITYGROUP_4	Pre-emption bit 3	Pre-emption bit 2	Pre-emption bit 1	Pre-emption bit 0

Group priority

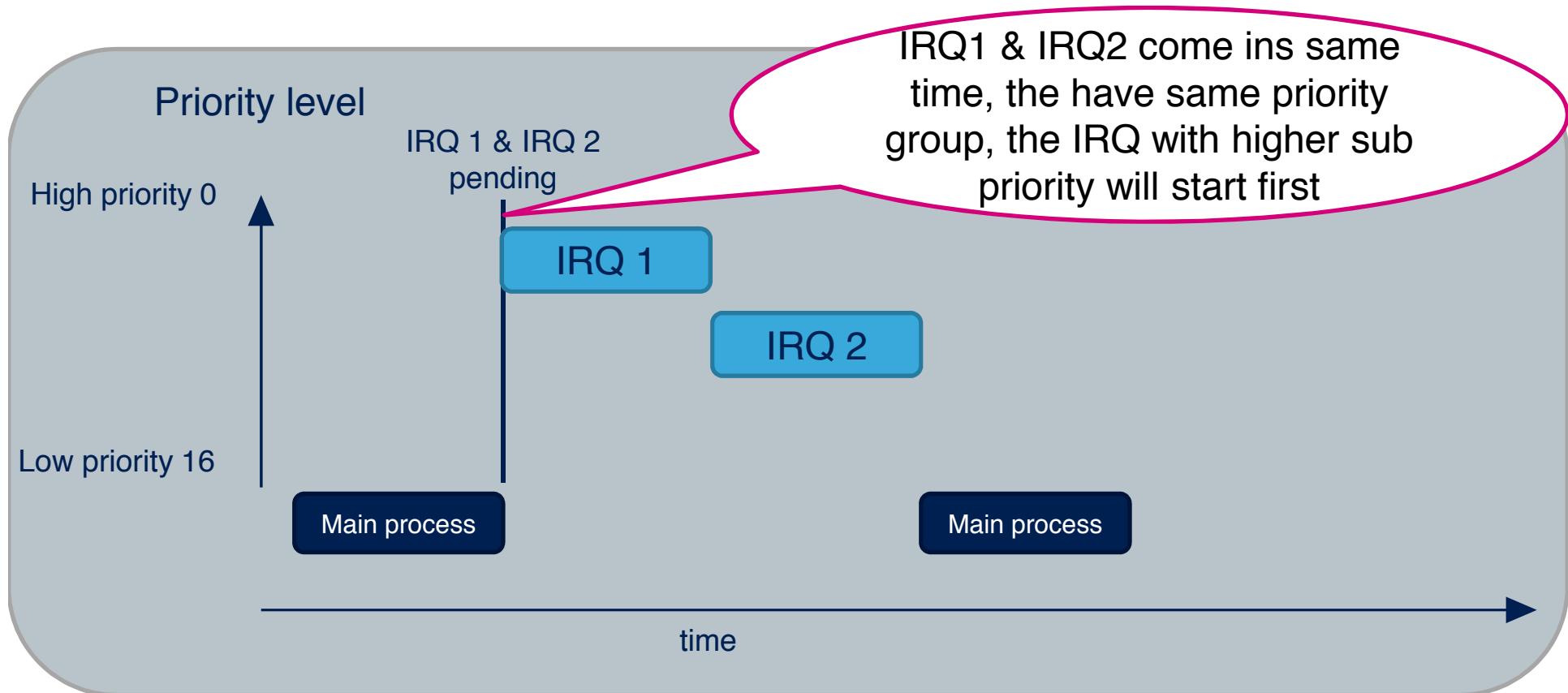
108



- IRQ1 - Group priority 0
- IRQ2 - Group priority 1

Sub priority

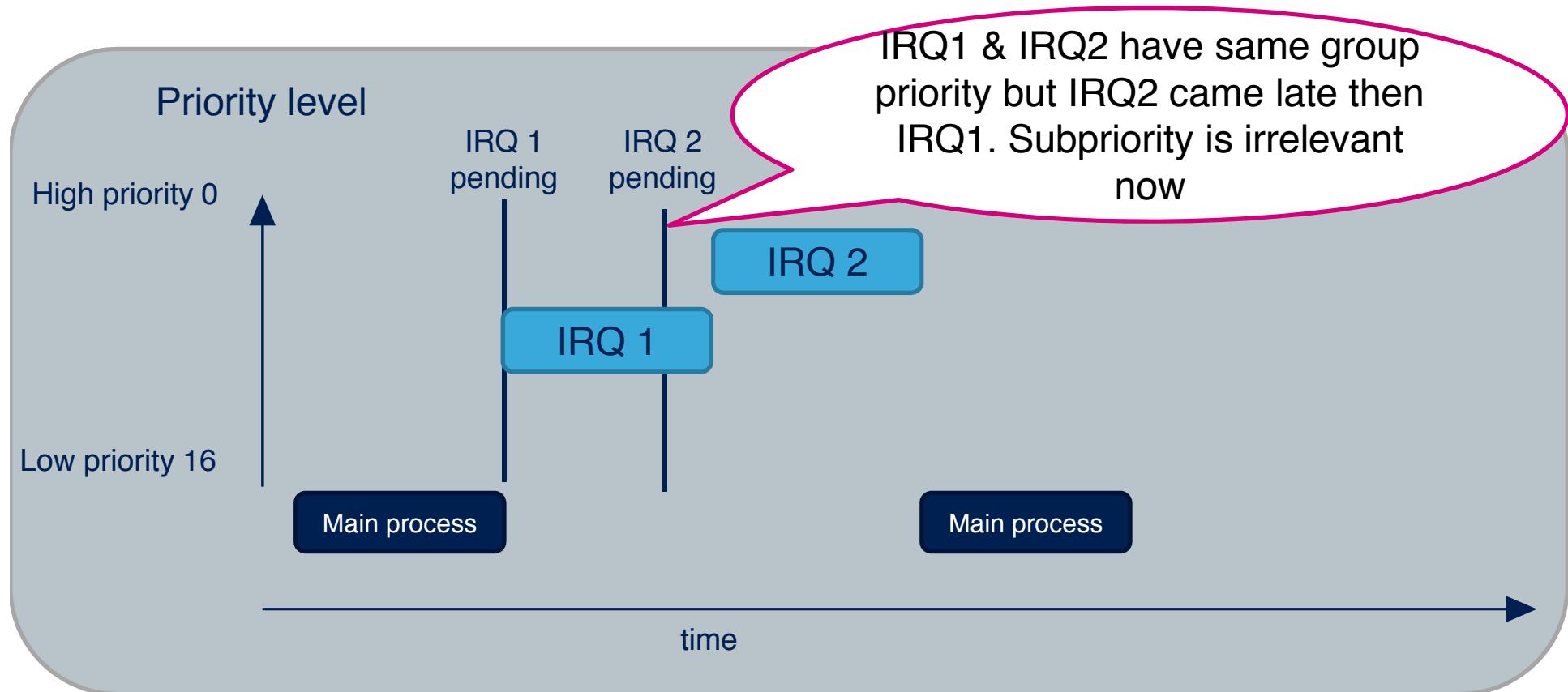
109



- IRQ1 - Group priority 0, Sub priority 0
- IRQ2 - Group priority 0, Sub priority 1

Sub priority

110



- IRQ1 - Group priority 0, Sub priority 1
- IRQ2 - Group priority 0, Sub priority 0

HAL service peripherals

NVIC interrupt settings

111

- For Interrupt Priority settings and for enable and disable are used:
 - HAL_NVIC_SetPriority
 - HAL_NVIC_EnableIRQ /HAL_NVIC_DisableIRQ
- Most important is the IRQn_Type which describe which interrupt will be set/enabled/disabled
- This parameter is defined in stm32f4xx.h

```
/**  
 * @brief STM32F4XX Interrupt Number Definition, according to the selected device  
 *        in @ref Library_configuration_section  
 */
```

```
typedef enum
```

```
{
```

```
***** Cortex-M4 Processor Exceptions N
```

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 Debug Monitor Interrupt	*/
PendSV_IRQn	= -2,	/*!< 13 PendSV Interrupt	*/
SysTick_IRQn	= -1.	/*!< 14 SysTick Interrupt	*/

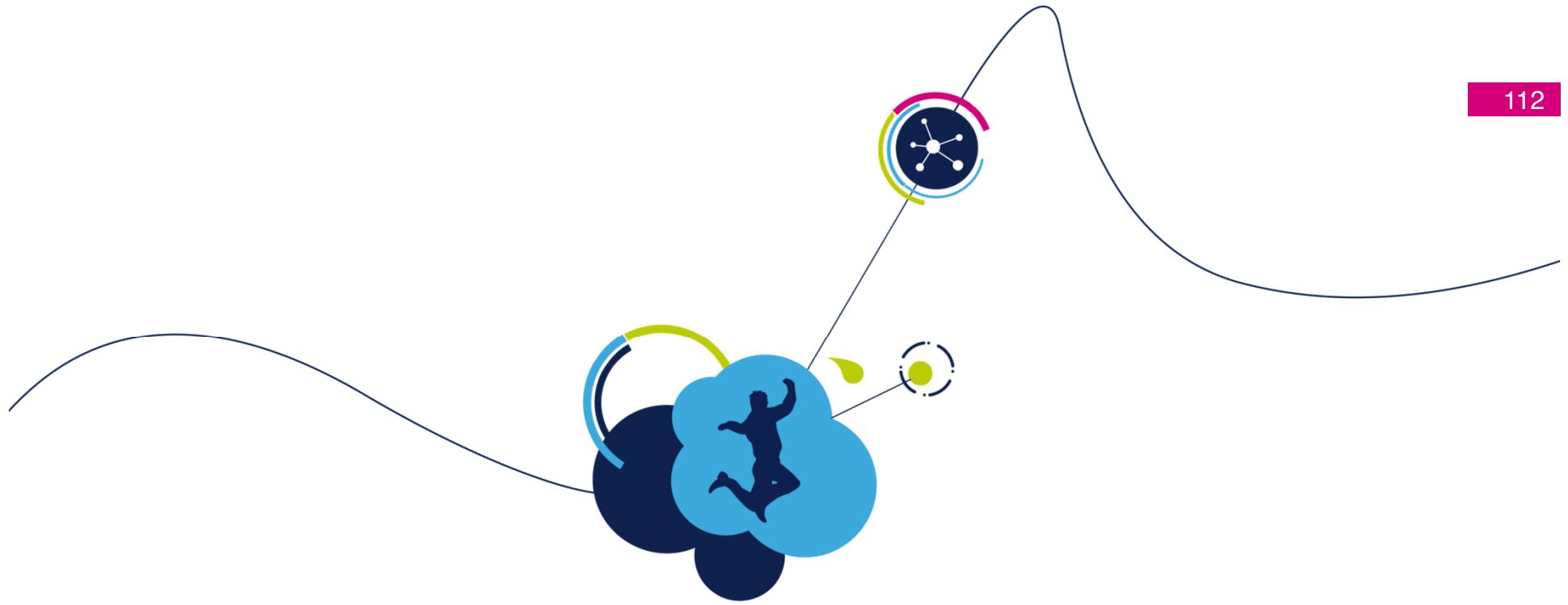
```
***** STM32 specific Interrupt Numbers
```

WWDG_IRQn	= 0,	/*!< Window WatchDog Interrupt	*/
PVD_IRQn	= 1,	/*!< PVD through EXTI Line detection Interrupt	*/
TAMP_STAMP_IRQn	= 2,	/*!< Tamper andTimeStamp interrupts through the EXTI line	*/
DAC_IRQn	= 3,	/*!< DAC Update interrupt through the EXTI line	*/

For STM32F429 in
file stm32f429.h

Defined by
ARM

Specific for
STM32F429

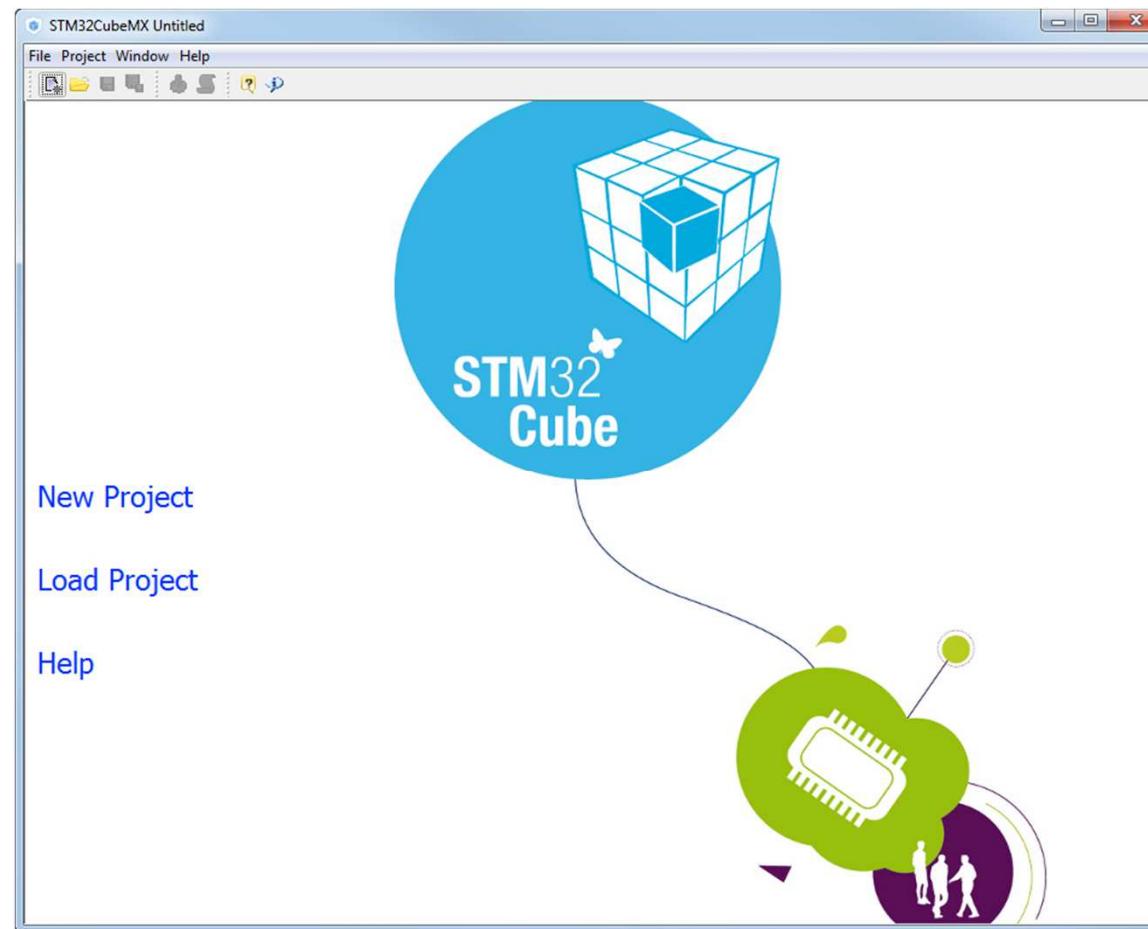


HAL service peripherals - GPIO General purpose I/Os

Use GPIO lab

113

- Use GPIO lab to create project with GPIO
- We use this project for better demonstration how the HAL library works



HAL service peripherals

GPIO - initialization structure

114

- Two new features compared to std library GPIO driver
 - A pin can be configured as EXTI with interrupt or event generation
 - Alternate field allows selection of the Alternate function for a pin (AFxx)

```
/**  
 * @brief  GPIO Init structure definition  
 */  
typedef struct  
{  
    uint32_t Pin;          /*!< Specifies the GPIO pins to be configured.  
                           This parameter can be any value of @ref GPIO_pins_define */  
  
    uint32_t Mode;         /*!< Specifies the operating mode for the selected pins.  
                           This parameter can be a value of @ref GPIO_mode_define */  
  
    uint32_t Pull;         /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.  
                           This parameter can be a value of @ref GPIO_pull_define */  
  
    uint32_t Speed;        /*!< Specifies the speed for the selected pins.  
                           This parameter can be a value of @ref GPIO_speed_define */  
  
    uint32_t Alternate;   /*!< Peripheral to be connected to the selected pins.  
                           This parameter can be a value of @ref GPIO_Alternat_function_selection */  
}  
}GPIO_InitTypeDef;
```

HAL service peripherals

GPIO – initialization – GPIO Mode

115

GPIO_MODE_OUTPUT_PP



GPIO_MODE_OUTPUT_OD



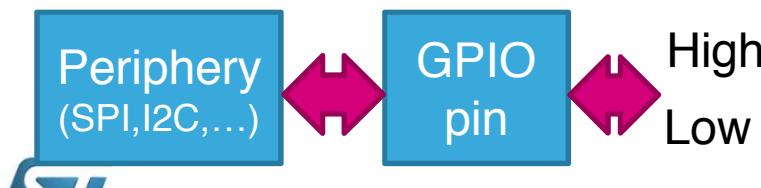
GPIO_MODE_INPUT



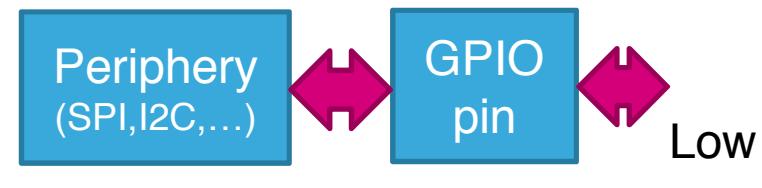
GPIO_MODE_ANALOG



GPIO_MODE_AF_PP



GPIO_MODE_AF_OD



HAL service peripherals

GPIO – initialization – GPIO Mode

116

GPIO_MODE_IT_RISING



GPIO_MODE_IT_FALLING



GPIO_MODE_IT_RISING_FALLING



GPIO_MODE_EVT_RISING



GPIO_MODE_EVT_FALLING



GPIO_MODE_EVT_RISING_FALLING

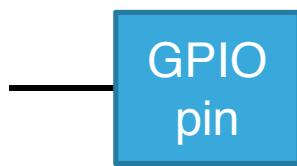


HAL service peripherals

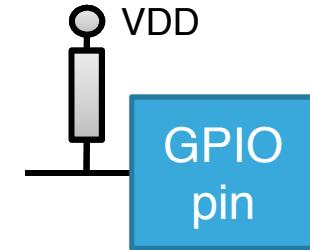
GPIO – initialization – GPIO Pull

117

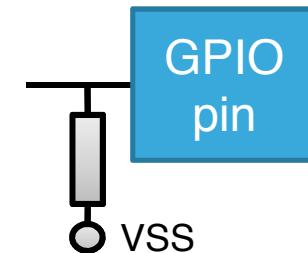
GPIO_NOPULL



GPIO_PULLUP



GPIO_PULLDOWN

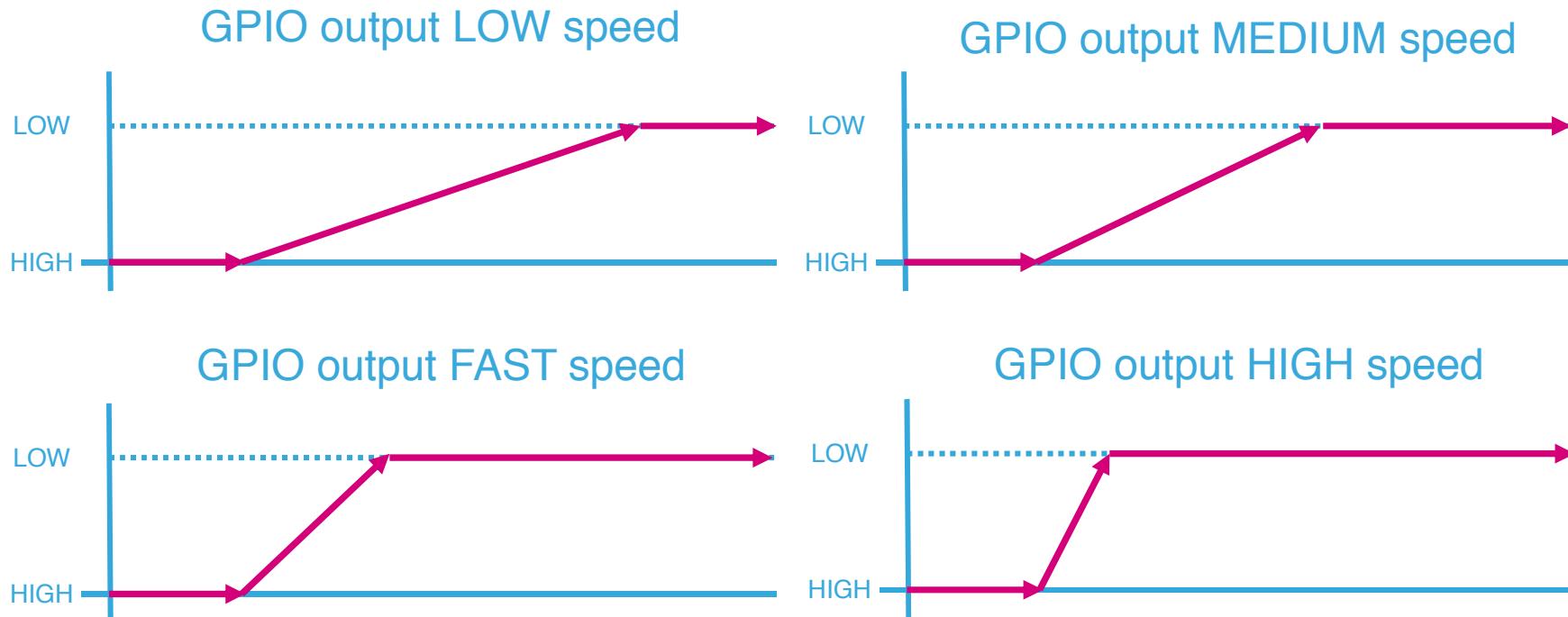


HAL service peripherals

GPIO – initialization – GPIO Speed

118

- GPIO(Pin) output speed configuration
 - Change the rising and falling edge when pin change state from high to low or low to high
 - **Higher** GPIO speed increase **EMI noise** from STM32 and increase STM32 **consumption**
 - It is good to adapt GPIO speed with periphery speed. Ex.: Toggling GPIO on 1Hz is LOW optimal settings, but SPI on 45MHz the HIGH must be set

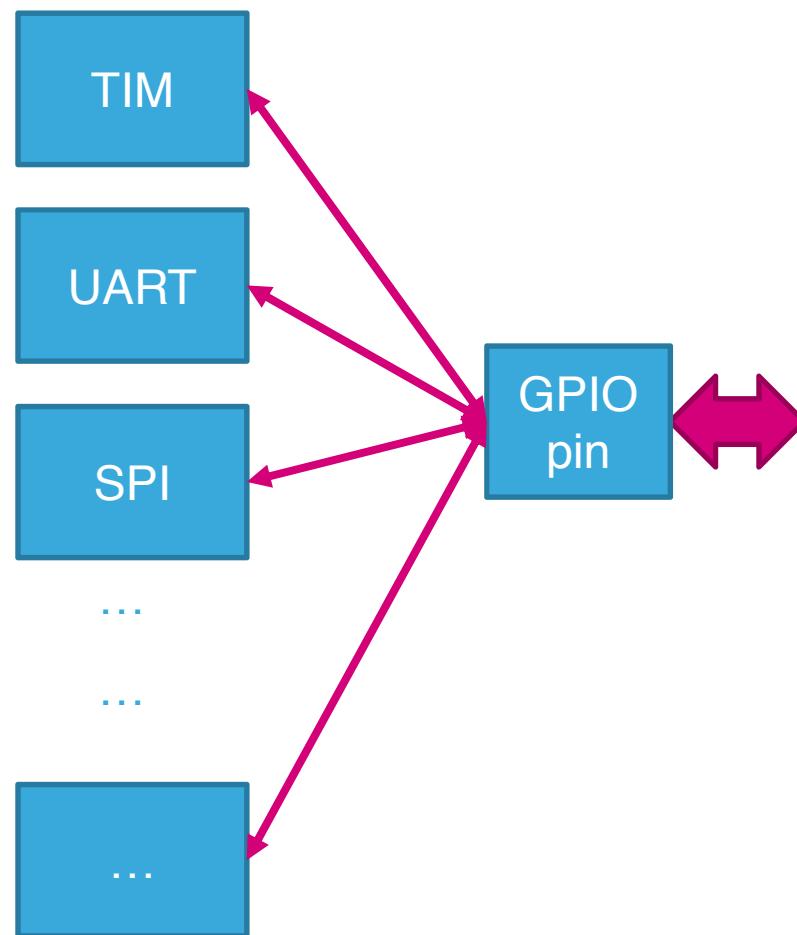


HAL service peripherals

GPIO – initialization – GPIO Alternate

119

Select alternate function for GPIO



HAL service peripherals

GPIO – APIs

120

- The following APIs are provided
 - HAL_GPIO_Init()
 - HAL_GPIO_DeInit()
 - HAL_GPIO_ReadPin()
 - HAL_GPIO_WritePin()
 - HAL_GPIO_TogglePin()

HAL service peripherals

GPIO – APIs

121

- Main function structure, generated by CubeMX

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration----*/
    /* Reset of all peri---*/
    HAL_Init();           Initialize Systick,  
NVIC, and call  
HAL_Init_MSP
    /* Configure the system clock */
    SystemClock_Config(); Initialize STM32  
clock tree, based on  
Clock Configuration  
in CubeMX
    /* Initialize all configu---*/
    MX_GPIO_Init();       Initialize GPIO which  
are not used with any  
other periphery
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* USER CODE BEGIN 3 */
    /* Infinite loop */
    while (1)
    {
    }
    /* USER CODE END 3 */
}
```

HAL service peripherals

GPIO – APIs

122

- MX_GPIO_Init function structure, generated by CubeMX

```
/** Configure pins as
 * Analog
 * Input
 * Output
 * EVENT_OUT
 * EXTI
*/
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct,
    /* GPIO Ports Clock Enable
     * _GPIOG_CLK_ENABLE(); */
    /*Configure GPIO pin : PG14 */
    GPIO_InitStruct.Pin = GPIO_PIN_14;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
    HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
}
```

Enable clock
for GPIO port

Initialize the
GPIO pin
configured in
CubeMX

Initialize the
GPIO registers

HAL service peripherals

GPIO – APIs

123

- MX_GPIO_Init function structure, generated by CubeMX

```
/** Configure pins as
 * Analog
 * Input
 * Output
 * EVENT_OUT
 * EXTI
*/
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;

    /* GPIO Ports Clock Enable */
    __GPIOG_CLK_ENABLE();

    /*Configure GPIO pin : PG14 */
    GPIO_InitStruct.Pin = GPIO_PIN_14;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
    HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
}

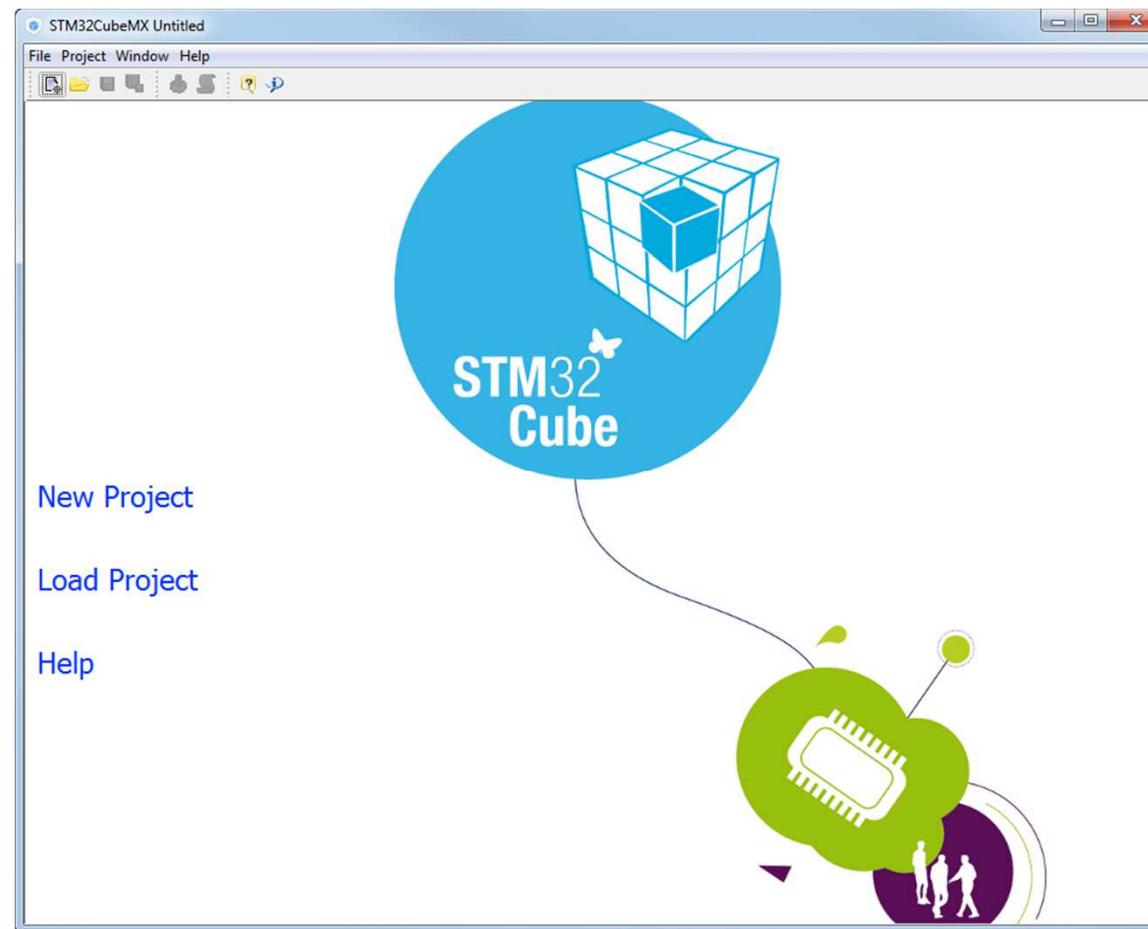
}
```

Like other Service init
functions
**HAL_GPIO_Init have
no MSP function**

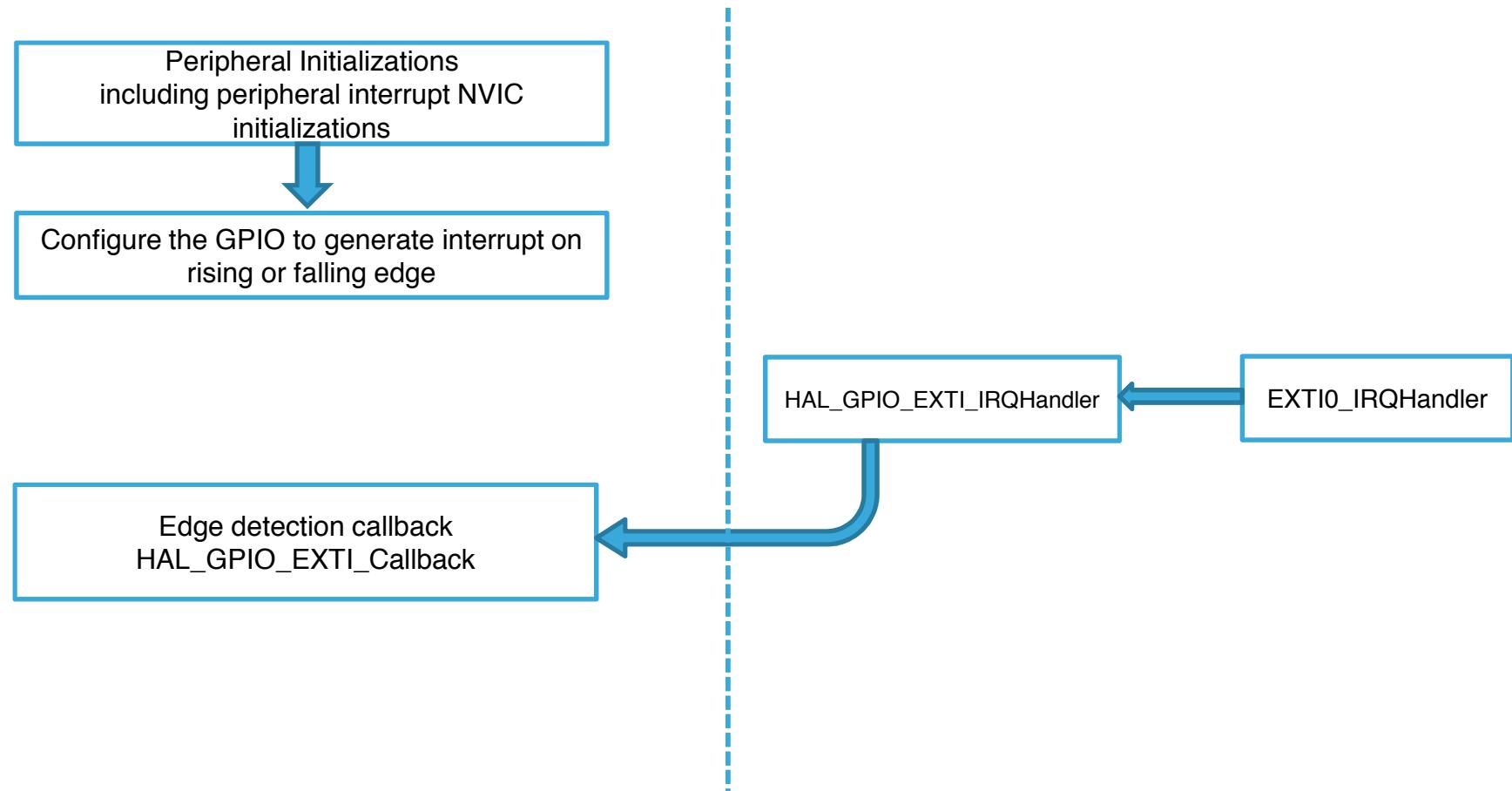
Use EXTI lab

124

- Use EXTI lab to create project with GPIO
- We use this project for better demonstration how the HAL library works

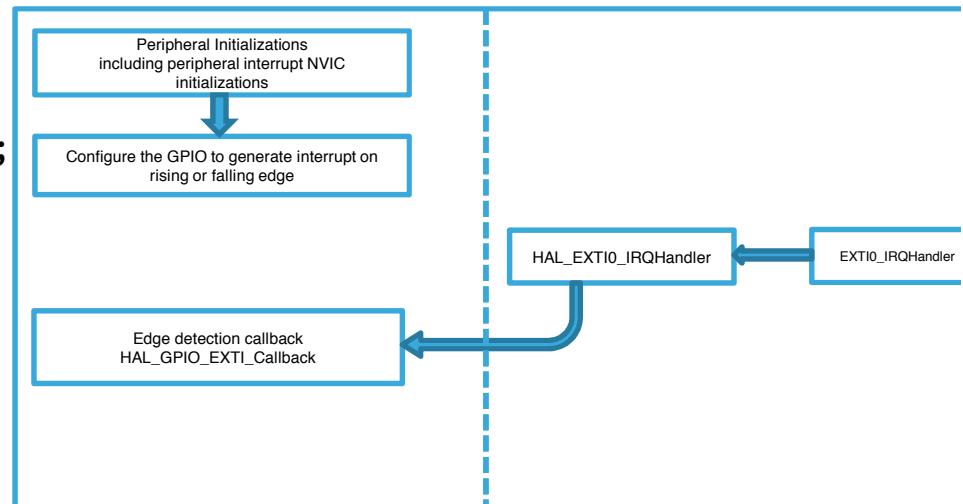


HAL EXTI workflow



- GPIO with EXTI initialization

```
/** Configure pins as
 * Analog
 * Input
 * Output
 * EVENT_OUT
 * EXTI
*/
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    /* GPIO Ports Clock Enable */
    __GPIOA_CLK_ENABLE();
    /*Configure GPIO pin : PA0 */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    /* EXTI interrupt init*/
    HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}
```



- GPIO with EXTI initialization in main.c

```
/** Configure pins as
 * Analog
 * Input
 * Output
 * EVENT_OUT
 * EXTI
*/
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    /* GPIO Ports Clock Enable */
    __GPIOA_CLK_ENABLE();
    /*Configure GPIO pin : PA0 */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    /* EXTI interrupt init*/
    HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}
```

Rising edge with interrupt mode select

Peripheral Initializations
including peripheral interrupt NVIC initializations

Configure the GPIO to generate interrupt on rising or falling edge

Edge detection callback
HAL_GPIO_EXTI_Callback

HAL_GPIO_EXTI_IRQHandler

EXTI0_IRQHandler

NVIC configuration

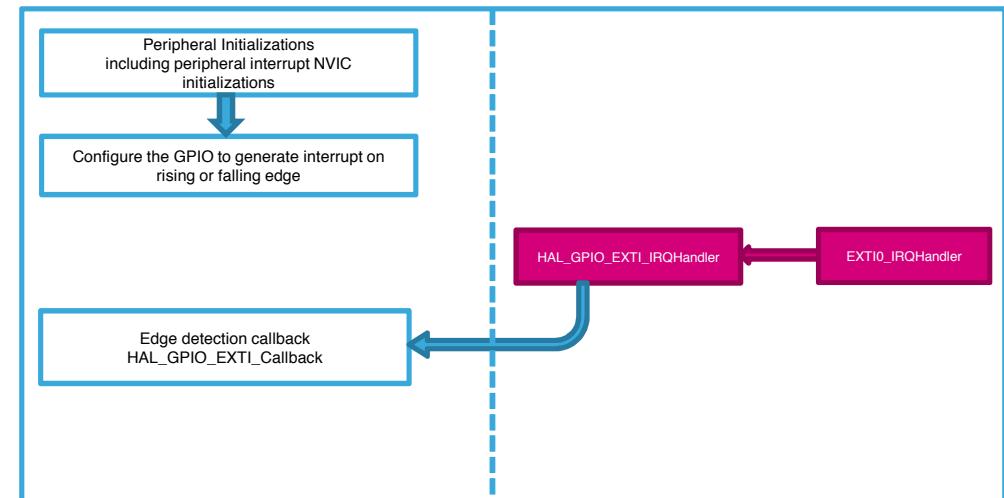
- Interrupt handling in stm32f4xx_it.c

```
/*
 * @brief This function handles EXTI Line0 interrupt.
 */
void EXTI0_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI0_IRQn_0 */

    /* USER CODE END EXTI0_IRQn_0 */
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
    /* USER CODE BEGIN EXTI0_IRQn_1 */

    /* USER CODE END EXTI0_IRQn_1 */
}
```

Call HAL_GPIO_EXTI_IRQHandler parameter is peripheral pin for which we want check the interrupt state
GPIO/EXTI don't need any handler



- EXTI interrupt callback
- Need to be defined by user, in default defined as `__weak`
- Can be found in `hal_stm32f4xx_gpio.c`

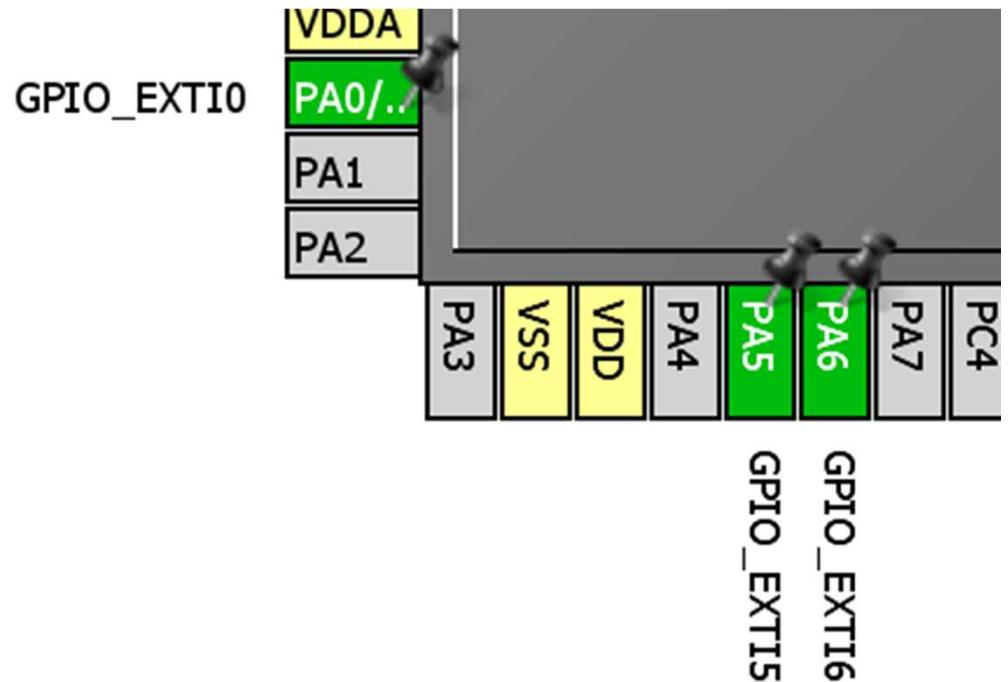
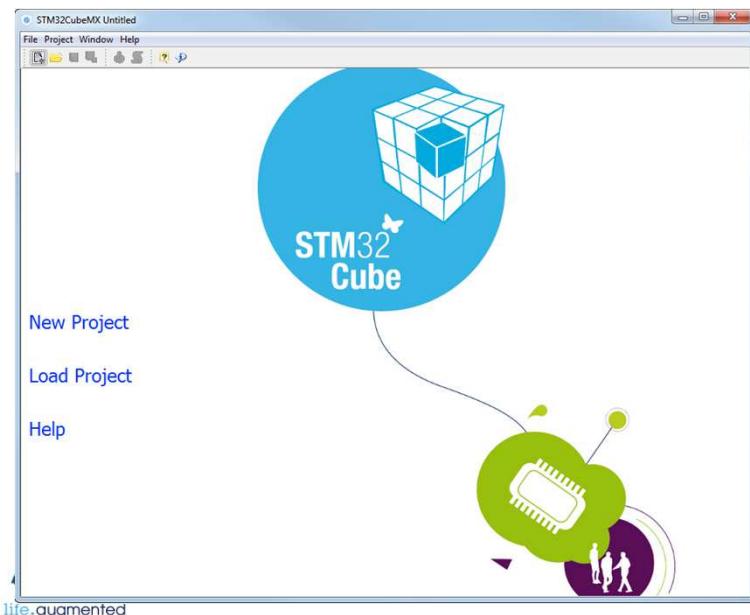
```
/**  
 * @brief  EXTI line detection callbacks.  
 * @param  GPIO_Pin: Specifies the pins connected EXTI line  
 * @retval None  
 */  
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)  
{  
    /* NOTE: This function Should not be modified, when the callback is needed,  
           the HAL_GPIO_EXTI_Callback could be implemented in the user file  
    */  
}  
  
/* USER CODE BEGIN 4 */  
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)  
{  
}  
/* USER CODE END 4 */
```

User defined in main.c

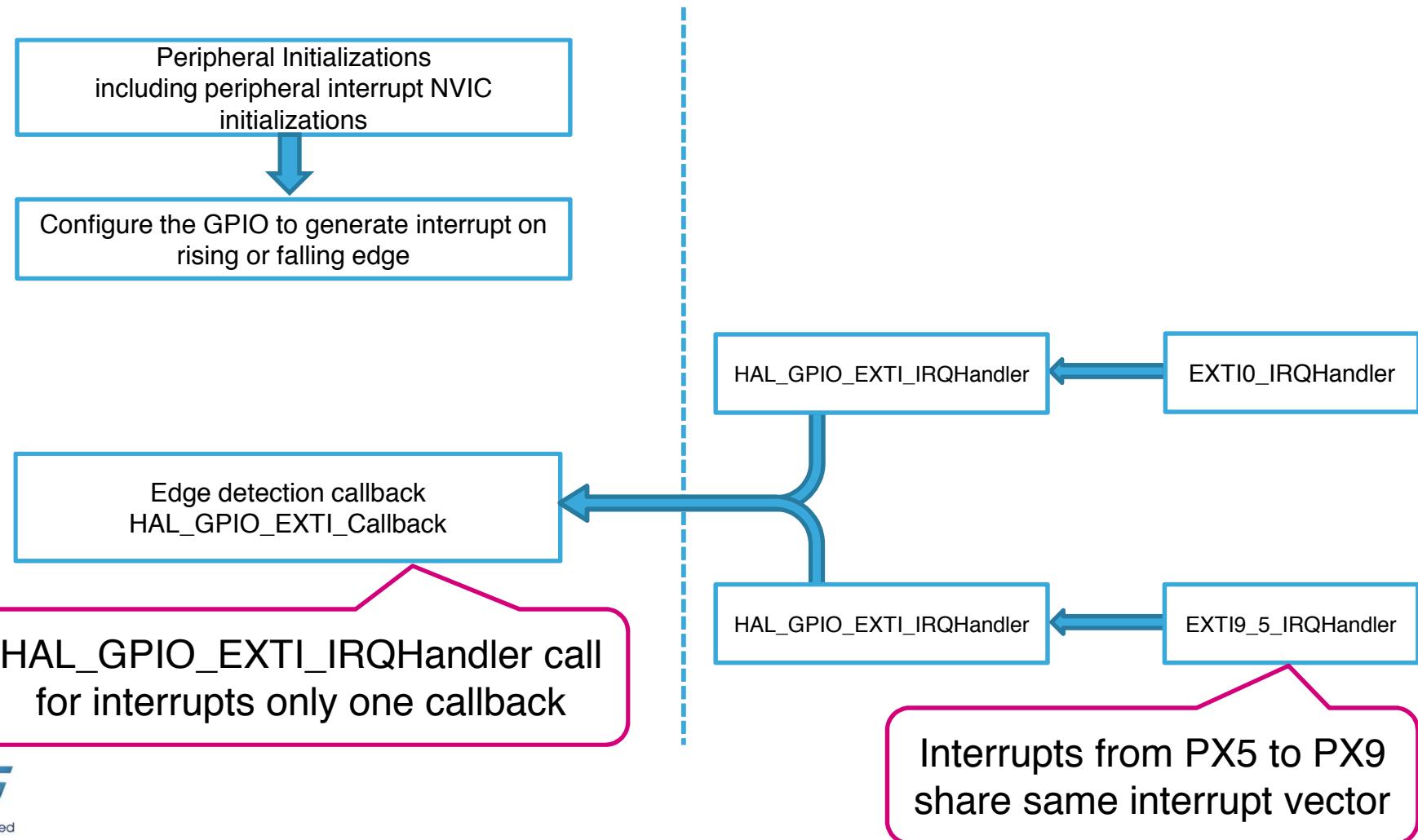
Use EXTI lab

130

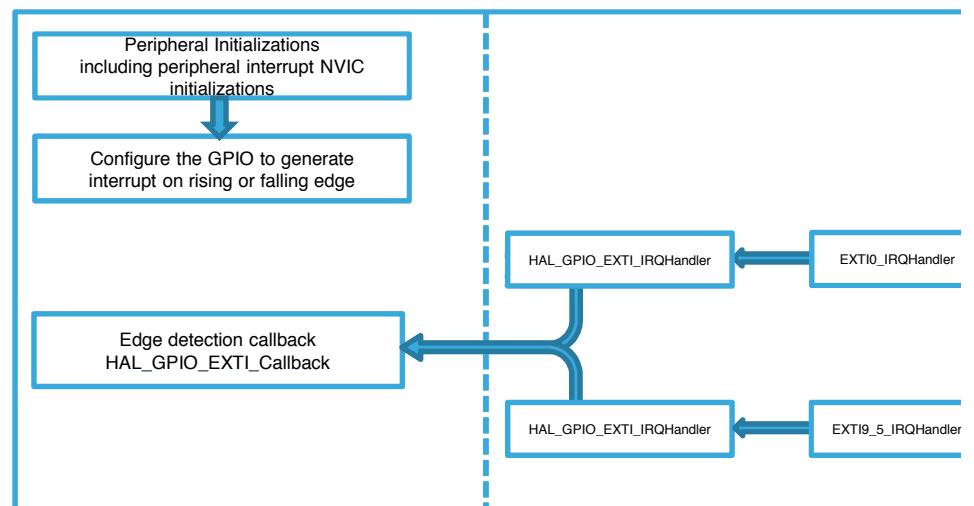
- EXTI lab extension
- Use now the same example but now use two or more EXTI pins
- You can use only EXTI with different number



HAL multiple EXTI workflow



HAL multiple EXTI workflow



- GPIO with EXTI initialization in main.c

```
/** Configure pins as
 * Analog
 * Input
 * Output
 * EVENT_OUT
 * EXTI
*/
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    /* GPIO Ports Clock Enable */
    __GPIOA_CLK_ENABLE();
    /*Configure GPIO pins : PA0 PA5 PA6 ,*/
    GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_5|GPIO_PIN_6;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    /* EXTI interrupt init*/
    HAL_NVIC_SetPriority(EXTI0 IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(EXTI0 IRQn);

    HAL_NVIC_SetPriority(EXTI9_5 IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(EXTI9_5 IRQn);
}
```

Interrupt enable for PA5 and PA6

Interrupt enable for PA0

Peripheral Initializations
including peripheral interrupt NVIC
initializations

Configure the GPIO to generate
interrupt on rising or falling edge

Edge detection callback
HAL_GPIO_EXTI_Callback

HAL_GPIO_EXTI_IRQHandler

EXTI0_IRQHandler

HAL_GPIO_EXTI_IRQHandler

EXTI9_5_IRQHandler

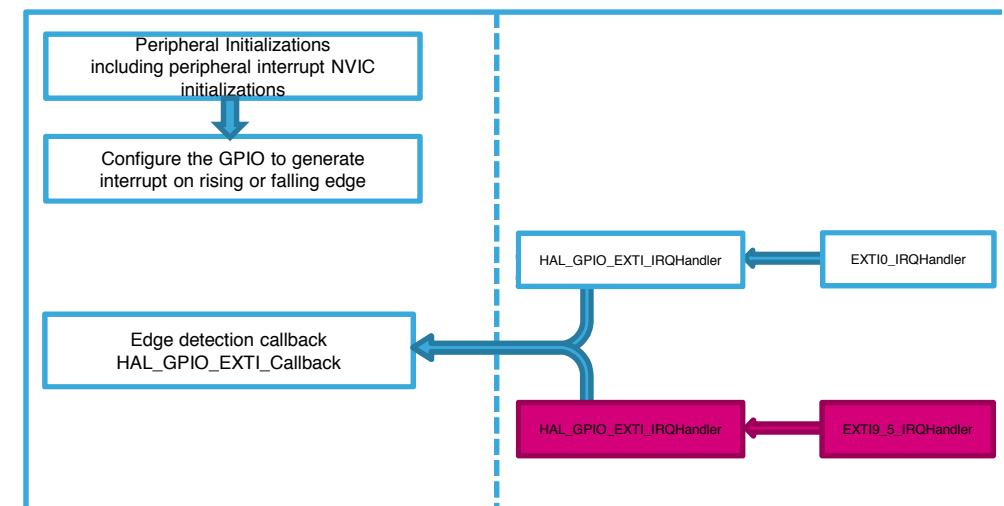
- Interrupt handling in `stm32f4xx_it.c`

```
/** @brief This function handles EXTI Line[9:5] interrupts.
*/
void EXTI9_5_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI9_5_IRQn_0 */

    /* USER CODE END EXTI9_5_IRQn_0 */
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
    /* USER CODE BEGIN EXTI9_5_IRQn_1 */

    /* USER CODE END EXTI9_5_IRQn_1 */
}
```

If EXTI interrupt handle multiple pins we must call `HAL_GPIO_EXTI_IRQHandler` for each of them



Use EXTI lab

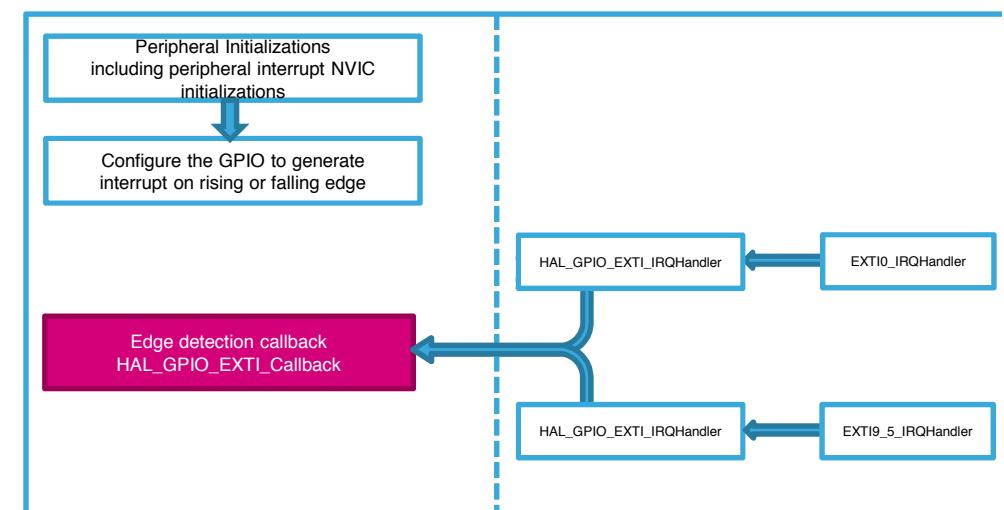
135

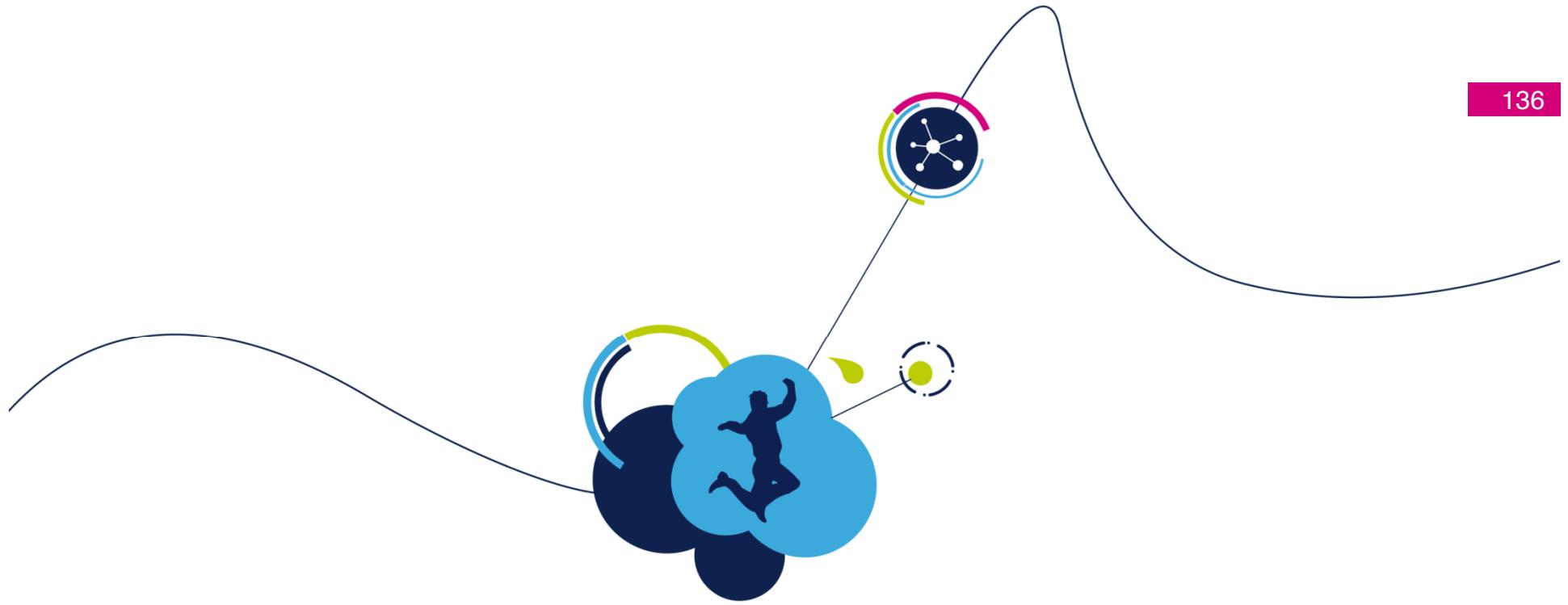
- Need to be defined by user, in default defined as `__weak`

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    switch(GPIO_Pin){
        case GPIO_PIN_0:
            /*GPIO_PIN_0 EXTI handling*/
            break;
        case GPIO_PIN_5:
            /*GPIO_PIN_5 EXTI handling*/
            break;
        case GPIO_PIN_6:
            /*GPIO_PIN_6 EXTI handling*/
            break;
        default:
            break;
    }
}
/* USER CODE END 4 */
```

HAL_GPIO_EXTI_IRQHandler use only one callback for all interrupts

Here we can use SWITCH to handle correct action for specific pin



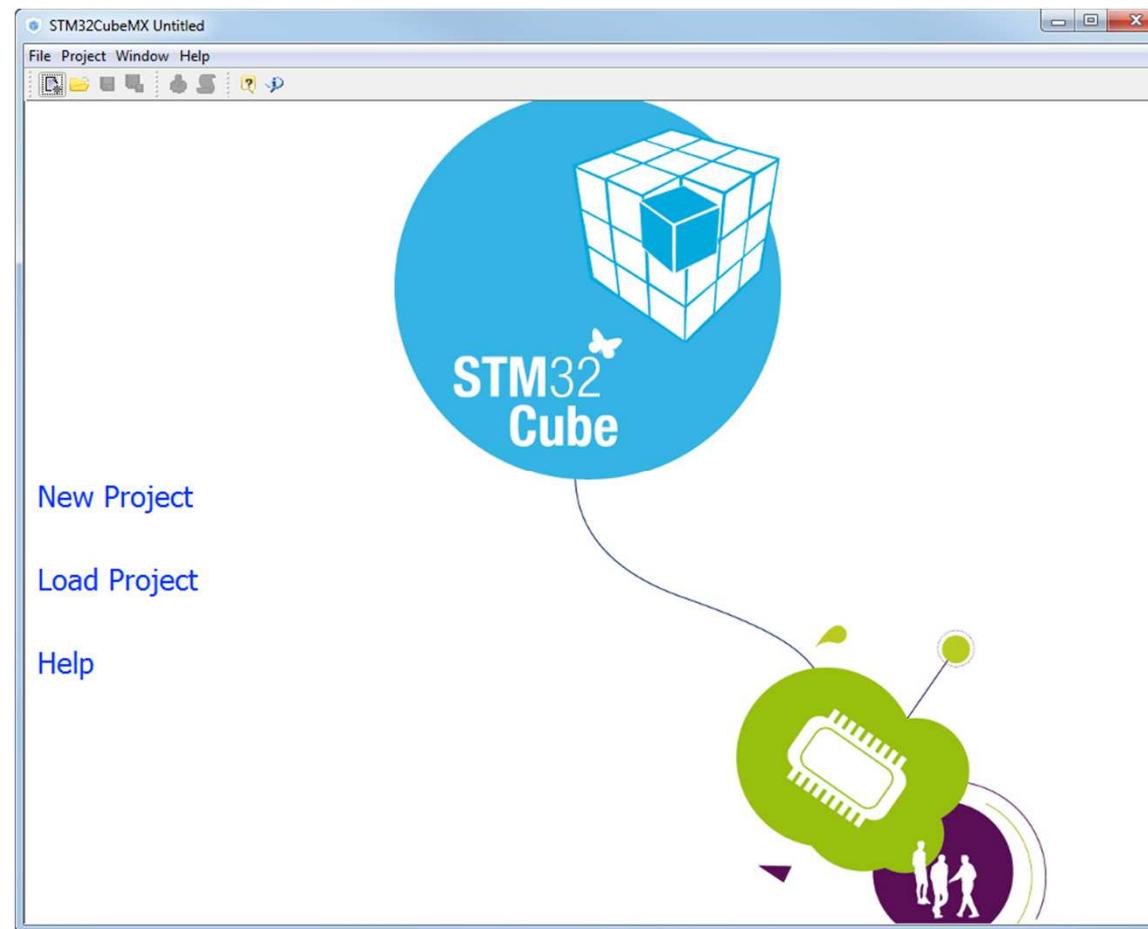


HAL service peripherals - DMA DIRECT MEMORY ACCESS

Use DMA Poll lab

137

- Use **DMA Poll** lab to create project with DMA
- We use this project for better demonstration how the HAL library works



Handler structures

138

- Many of Cube HAL function require handlers for correct function(DMA, TIM, UART, ...)
- Each periphery must have own handler
- Handler for periphery must be defined only once. CubeMX define it in main.c
- Handler is used when:
 - Calling HAL periphery function
 - In interrupt handler

Handler structures

139

- Handler is basic structure contains information about periphery
 - Instance
 - Define which periphery is it(ex. TIM1, or TIM5)
 - Only one instance per handler
 - Init
 - Here are stored initialization parameters for periphery(ex.: DMA: destination address)
 - Lock
 - Lock to periphery status, cannot be used from different function or thread
 - Status
 - Status of periphery(Ready, Busy or Error)
 - Error
 - Indicate which type of error was indicated during operation
 - Parent
 - If DMA work with other periphery(ex. SPI) this points to periphery handler
 - Callbacks
 - Callback to other functions (Complete callback, HalfTransferComplete callback, Error callback, ...)
 - Depends on type of periphery
 - Periphery dependent parts
 - Unique for each periphery

HAL service peripherals

DMA – HAL APIs Handle structure

140

- DMA HAL APIs use a DMA Handle structure as parameter, it is defined as following:

```
/**  
 * @brief DMA handle Structure definition  
 */  
typedef struct __DMA_HandleTypeDef  
{  
    DMA_Stream_TypeDef           *Instance;      Pointer to DMA stream instance (ex: DMA1_Stream0)  
    DMA_InitTypeDef              Init;          DMA channel init structure  
    HAL_LockTypeDef              Lock;  
    __IO HAL_DMA_StateTypeDef   State;          DMA channel state  
    void                         *Parent;  
    void             (*XferCpltCallback) ( struct __DMA_HandleTypeDef * hdma );  
    void             (*XferHalfCpltCallback) ( struct __DMA_HandleTypeDef * hdma );  
    void             (*XferM1CpltCallback) ( struct __DMA_HandleTypeDef * hdma );  
    void             (*XferErrorCallback) ( struct __DMA_HandleTypeDef * hdma );  
    __IO uint32_t          ErrorCode;        DMA channel callbacks  
} DMA_HandleTypeDef;
```

- DMA channel callbacks need to be initialized by user only in case of memory to memory transfer.
- For peripheral - memory transfers, the HAL peripheral driver offer APIs that handles DMA transfer for peripheral ((ex: HAL_USART_Receive_DMA()) these APIs will assign callback functions.

HAL service peripherals

DMA – HAL APIs Handle structure

141

- DMA HAL APIs use a DMA Handle structure as parameter, it is defined as following:

```
/**  
 * @brief DMA handle Structure definition  
 */  
typedef struct __DMA_HandleTypeDef  
{  
    DMA_StreamTypeDef     *Instance;           /* Pointer to DMA stream instance (ex: DMA1_Stream0) base address */  
    DMA_InitTypeDef       Init;                /* DMA channel init structure */  
    HAL_LockTypeDef       Lock;                /* DMA locking object */  
    __IO HAL_DMA_StateTypeDef State;            /* DMA transfer state */  
    void                 *Parent;              /* DMA parent */  
    void                 (*XferCpltCallback)( struct __DMA_HandleTypeDef * hdma); /* DMA transfer complete callback */  
    void                 (*XferHalfCpltCallback)( struct __DMA_HandleTypeDef * hdma); /* DMA Half transfer complete callback */  
    void                 (*XferM1CpltCallback)( struct __DMA_HandleTypeDef * hdma); /* DMA transfer complete Memory1 callback */  
    void                 (*XferErrorCallback)( struct __DMA_HandleTypeDef * hdma); /* DMA transfer error callback */  
    __IO uint32_t          ErrorCode;           /* DMA Error code */  
}DMA_HandleTypeDef;
```

Defined in
stm32f4xx_hal_dma.h

- **DMA channel callbacks need to be initialized by user only in case of memory to memory transfer.**

- For peripheral - memory transfers, the HAL peripheral driver offer APIs that handles DMA transfer for peripheral ((ex: HAL_USART_Receive_DMA()) these APIs will assign callback functions.

HAL service peripherals

DMA – HAL APIs Init structure

142

```
/*
 * @brief DMA Configuration Structure definition
 */
typedef struct
{
    uint32_t Channel;           /*!< Specifies the channel used for the specified stream.  
This parameter can be a value of @ref DMA_Channel_selection */

    uint32_t Direction;        /*!< Specifies if the data will be transferred from memory to peripheral  
from memory to memory or from peripheral to memory.  
This parameter can be a value of @ref DMA_Data_transfer_direction */

    uint32_t PeriphInc;        /*!< Specifies whether the Peripheral address register should be incremented or not.  
This parameter can be a value of @ref DMA_Peripheral_incremented_mode */

    uint32_t MemInc;           /*!< Specifies whether the memory address register should be incremented or not.  
This parameter can be a value of @ref DMA_Memory_incremented_mode */

    uint32_t PeriphDataAlignment; /*!< Specifies the Peripheral data width.  
This parameter can be a value of @ref DMA_Peripheral_data_size */

    uint32_t MemDataAlignment;  /*!< Specifies the Memory data width.  
This parameter can be a value of @ref DMA_Memory_data_size */

    uint32_t Mode;              /*!< Specifies the operation mode of the DMAy Streamx.  
This parameter can be a value of @ref DMA_mode  
@note The circular buffer mode cannot be used if the memory-to-memory  
data transfer is configured on the selected Stream */

    uint32_t Priority;          /*!< Specifies the software priority for the DMAy Streamx.  
This parameter can be a value of @ref DMA_Priority_level */

    uint32_t FIFOMode;          /*!< Specifies if the FIFO mode or Direct mode will be used for the specified stream.  
This parameter can be a value of @ref DMA_FIFO_direct_mode  
@note The Direct mode (FIFO mode disabled) cannot be used if the  
memory-to-memory data transfer is configured on the selected stream */

    uint32_t FIFOThreshold;     /*!< Specifies the FIFO threshold level.  
This parameter can be a value of @ref DMA_FIFO_threshold_level */

    uint32_t MemBurst;          /*!< Specifies the Burst transfer configuration for the memory transfers.  
It specifies the amount of data to be transferred in a single non interruptable  
transaction.  
This parameter can be a value of @ref DMA_Memory_burst  
@note The burst mode is possible only if the address Increment mode is enabled. */

    uint32_t PeriphBurst;        /*!< Specifies the Burst transfer configuration for the peripheral transfers.  
It specifies the amount of data to be transferred in a single non interruptable  
transaction.  
This parameter can be a value of @ref DMA_Peripheral_burst  
@note The burst mode is possible only if the address Increment mode is enabled. */

}DMA_InitTypeDef;
```

IN this structure are stored information about DMA.
Define in **stm32f4xx_hal_dma.h**

HAL service peripherals

DMA M2M code from CubeMX

143

- CubeMX create DMA handler in main.c

```
/* Includes ----- */  
#include "stm32f4xx_hal.h"  
  
/* USER CODE BEGIN Includes */  
  
/* USER CODE END Includes */  
  
/* Private variables ----- */  
DMA_HandleTypeDef hdma_memtomem_dma2_stream0;  
  
/* USER CODE BEGIN PV */  
DMA handler type  
/*****/  
DMA handler name
```

HAL service peripherals

DMA M2M code from CubeMX

144

- If M2M transfer is selected, CubeMX create and call MX_DMA_Init

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_DMA_Init(); Initialize DMA
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* USER CODE BEGIN 3 */
    /* Infinite loop */
    while (1)
    {
    }
    /* USER CODE END 3 */
}
```

HAL service peripherals

DMA M2M code from CubeMX

145

- Initialize DMA handler structure

```
/*
 * Enable DMA controller clock
 * Configure DMA for memory to memory transfers
 *   hdma_memtomem_dma2_stream0
 */
void MX_DMA_Init(void)
{
    /* DMA controller clock enable */
    __DMA2_CLK_ENABLE();
    /* Configure DMA request hdma_memtomem_dma2_stream0 on DMA2_Stream0 */
    hdma_memtomem_dma2_stream0.Instance = DMA2_Stream0;
    hdma_memtomem_dma2_stream0.Init.Channel = DMA_CHANNEL_0;
    hdma_memtomem_dma2_stream0.Init.Direction = DMA_MEMORY_TO_MEMORY;
    hdma_memtomem_dma2_stream0.Init.PeriphInc = DMA_PINC_ENABLE;
    hdma_memtomem_dma2_stream0.Init.MemInc = DMA_MINC_ENABLE;
    hdma_memtomem_dma2_stream0.InitPeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_memtomem_dma2_stream0.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_memtomem_dma2_stream0.Init.Mode = DMA_NORMAL;
    hdma_memtomem_dma2_stream0.Init.Priority = DMA_PRIORITY_LOW;
    hdma_memtomem_dma2_stream0.Init.FIFOMode = DMA_FIFOMODE_ENABLE;
    hdma_memtomem_dma2_stream0.Init.FIFOThreshold = DMA_FIFO_THRESHOLD_HALFFULL;
    hdma_memtomem_dma2_stream0.Init.MemBurst = DMA_MBURST_SINGLE;
    hdma_memtomem_dma2_stream0.Init.PeriphBurst = DMA_PBURST_SINGLE;
    HAL_DMA_Init(&hdma_memtomem_dma2_stream0);
    /* DMA interrupt init */
}
```

DMA clock enable

Store parameters which
was selected in CubeMX
into init structure

Write content of Init structure
into DMA registers
Now is DMA ready to use

HAL service peripherals

DMA – APIs

146

- The following table lists the DMA APIs

DMA HAL APIs	Description
HAL_DMA_Init	Initializes a DMA channel
HAL_DMA_DeInit	De-initializes a DMA channel
HAL_DMA_Start	Starts DMA transfer
HAL_DMA_Start_IT	Starts DMA channel with interrupt generation at end of transfer or half transfer or on DMA error
HAL_DMA_Abort	Aborts a DMA transfer
HAL_DMA_PollForTransfer	Blocking function that polls for transfer complete or half complete, this function can also return a Timeout or a DMA error
HAL_DMA_IRQHandler	Interrupt handler for DMA
HAL_DMA_GetState	Gets DMA channel state
HAL_DMA_GetError	Gets DMA error code

HAL service peripherals

DMA – extension APIs

147

- Extension APIs for DMA are present in F4x family for handling double buffer feature, API include:

DMA HAL EX APIs	Description
HAL_DMAEx_MultiBufferStart	double buffer DMA transfer in polling mode
HAL_DMAEx_MultiBufferStart_IT	double buffer DMA transfer with Interrupt generation
HAL_DMAEx_ChangeMemory	allows changing non used buffer address on the fly

DMA Poll lab result

148

- Lab shows how to start the DMA transfer

```
/* USER CODE BEGIN 0 */
uint8_t Buffer_Src[]={0,1,2,3,4,5,6,7,8,9};
uint8_t Buffer_Dest[10];
/* USER CODE END 0 */

/* USER CODE BEGIN 2 */
HAL_DMA_Start(&hdma_memtomem_dma2_stream0, (uint32_t) (Buffer_Src), (uint32_t) (Buffer_Dest), 10);
while(HAL_DMA_PollForTransfer(&hdma_memtomem_dma2_stream0, HAL_DMA_FULL_TRANSFER, 100) != HAL_OK)
{
    __NOP();
}
/* USER CODE END 2 */
```

Handling of HAL return statuses

149

- Correct return handling of HAL function

```
status=HAL_DMA_PollForTransfer(&hdma_memtomem_dma2_stream0,HAL_DMA_FULL_TRANSFER,100);
/*ERROR*/
switch(status)
{
    case HAL_ERROR
        error=HAL_DMA_GetError(&hdma_memtomem_dma2_stream0);
        /*Handle ERROR*/
        switch(error){
            default:
                break;
        }
        break;
    case HAL_TIMEOUT:
        /*Handle TIMEOUT*/
        break;
    case HAL_BUSY:
        /*Handle BUSY*/
        break;
    case HAL_OK:
        /*Handle BUSY*/
        break;
    default:
        break;
}
```

Store function return status

If error detected

Discover what type of error was detected

Function timedouted

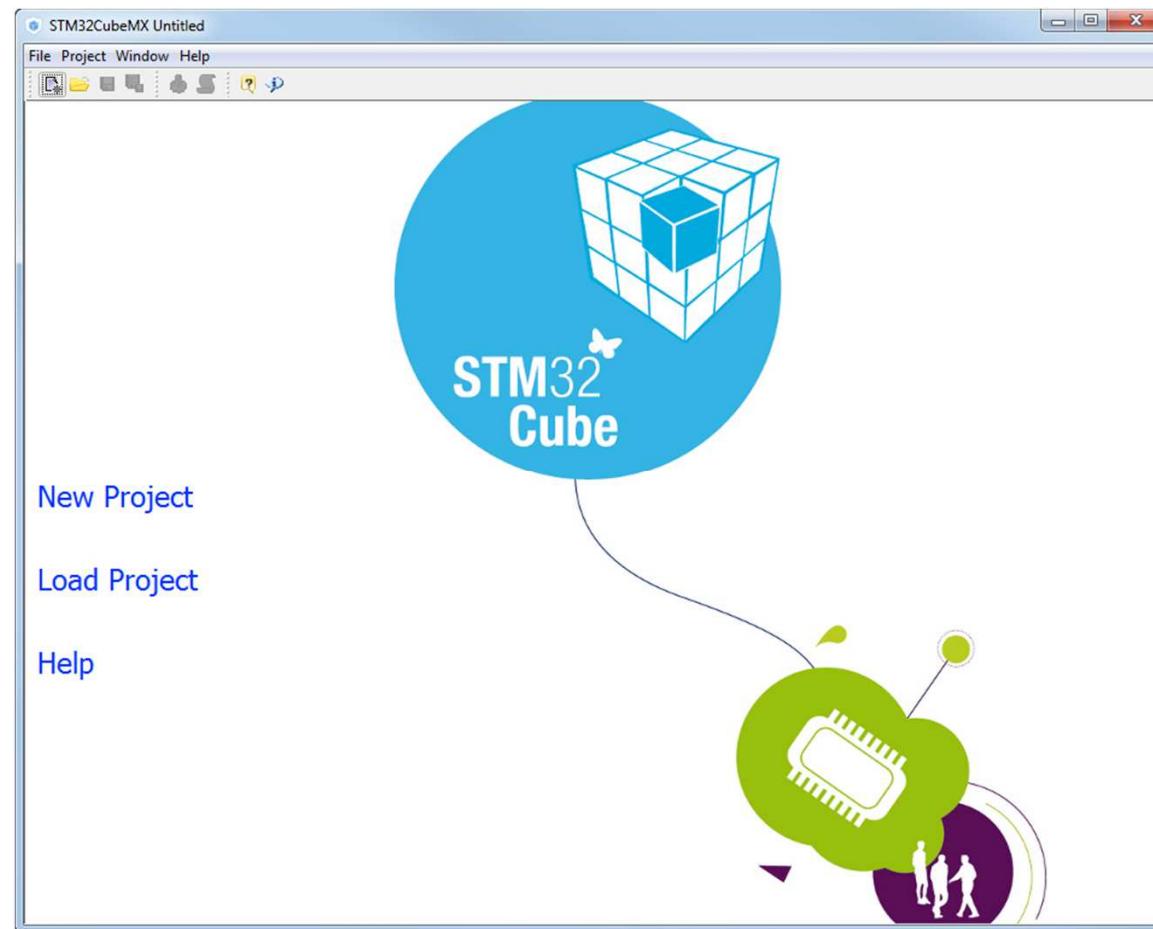
Periphery is in use

Function ends successfully

Use DMA with Interrupt lab

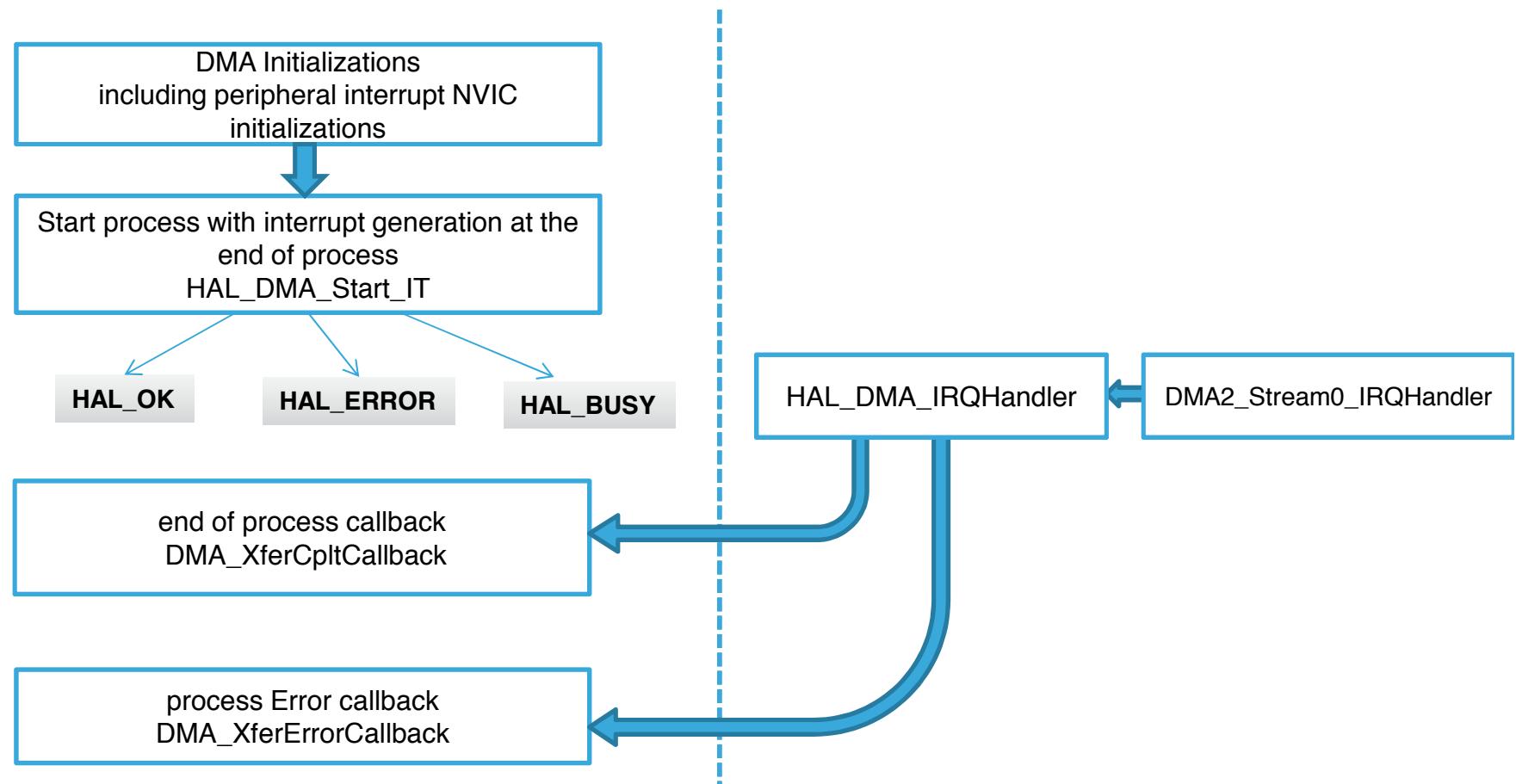
150

- Use **DMA with Interrupt** lab to create project with DMA
- We use this project for better demonstration how the HAL library works



HAL Library DMA with IT flow

151



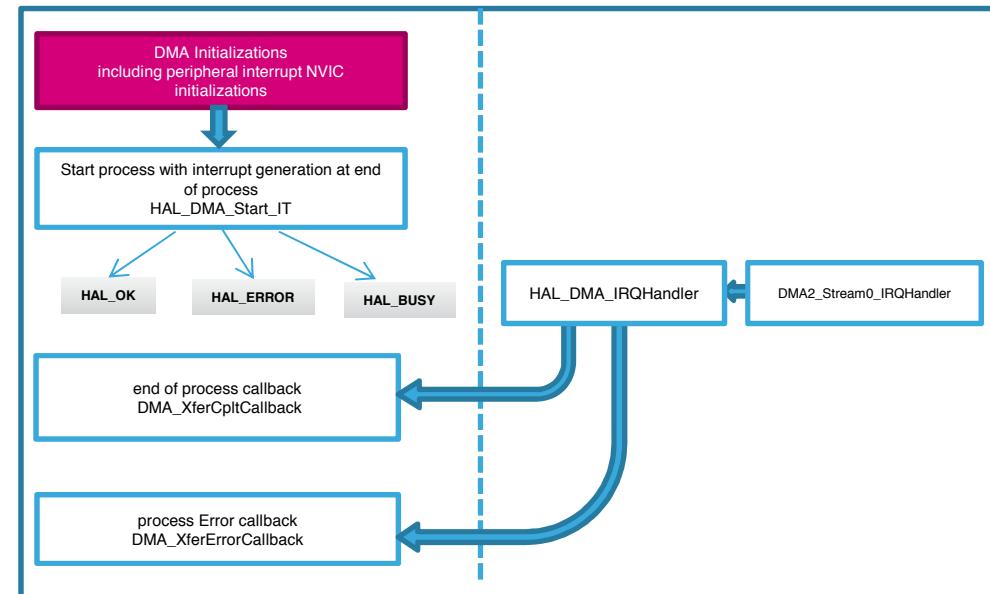
DMA with Interrupt lab

152

- Generated main.c is same as in DMA Poll lab

```
/* Private variables -----*/
DMA_HandleTypeDef hdma_memtomem_dma2_stream0;

int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_DMA_Init();
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* USER CODE BEGIN 3 */
    /* Infinite loop */
    while (1)
    {
    }
    /* USER CODE END 3 */
}
```



DMA with Interrupt lab

153

- New is content of stm32f4xx_it.c

```
/* External variables -----*/
```

```
extern DMA_HandleTypeDef hdma_memtomem_dma2_stream0;
```

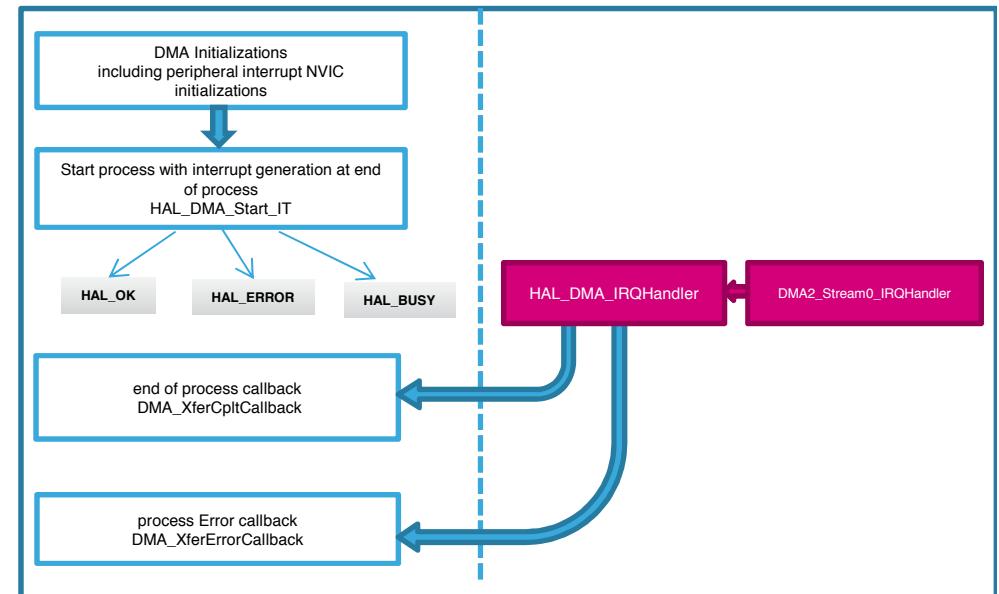
DMA handler is exported
stm32f4xx_if.c HAL functions need it

```
/**  
 * @brief This function handles DMA2 Stream0 global interrupt.  
 */  
void DMA2_Stream0_IRQHandler(void)  
{  
    /* USER CODE BEGIN DMA2_Stream0_IRQn 0 */  
}
```

Standard DMA interrupt handler
defined in startup_stm32f439xx.s

```
/* USER CODE END DMA2_Stream0_IRQn 0 */  
HAL_DMA_IRQHandler(&hdma_memtomem_dma2_stream0);  
/* USER CODE BEGIN DMA2_Stream0_IRQn 1 */  
/* USER CODE END DMA2_Stream0_IRQn 1 */
```

HAL DMA interrupt handling routine,
requires **handler**



DMA with Interrupt lab

154

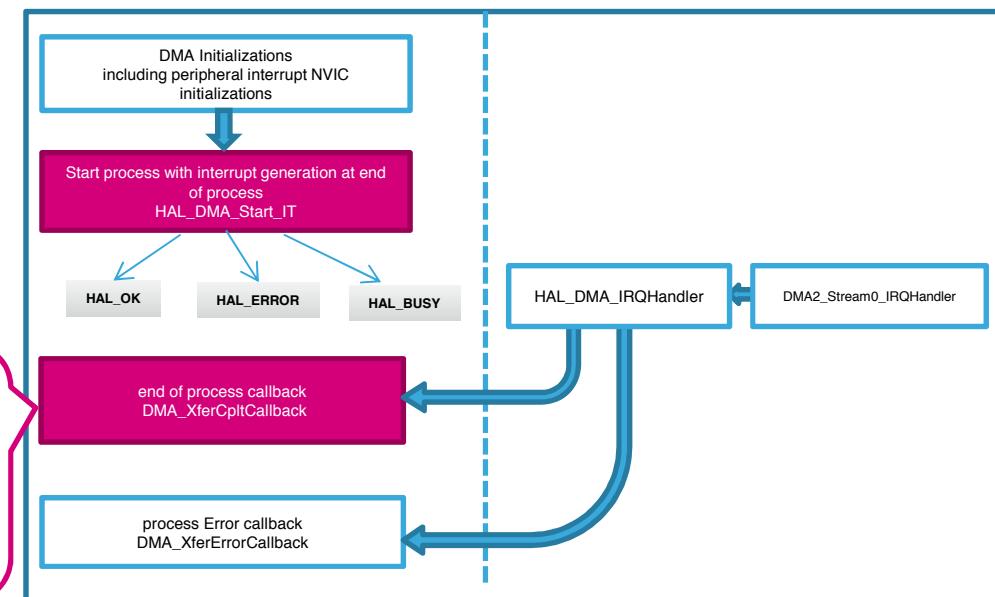
- If is DMA used with second periphery, their function assign callbacks to interrupt handlers
- We need to assign and create callback manually

Assign name of callback function directly to DMA handler

```
/* USER CODE BEGIN 2 */  
hdma_memtomem_dma2_stream0.XferCpltCallback=&XferCpltCallback;  
/* USER CODE END 2 */  
  
HAL_DMA_Start_IT(&hdma_memtomem_dma2_stream0,(uint32_t)Buffer_Src,(uint32_t)Buffer_Dest,10);
```

Start DMA transfer
End of transfer is indicated interrupt callback

After callback assign HAL_DMA_IRQHandler can call correct callback function



DMA with Interrupt lab

155

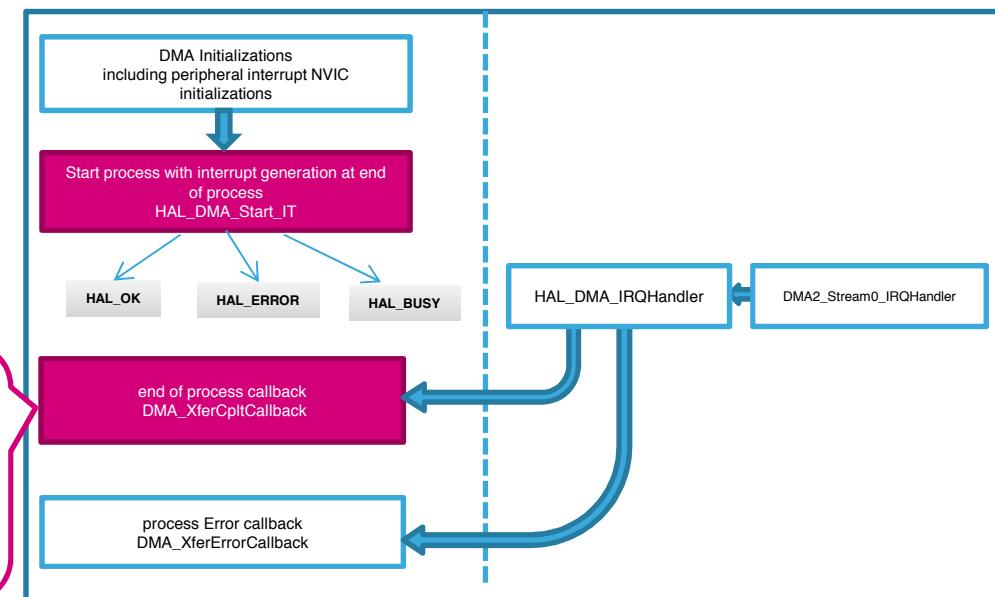
- If is DMA used with second periphery, their function assign callbacks to interrupt handlers
- We need to assign and create callback manually

Assign name of callback function directly to DMA handler

```
/* USER CODE BEGIN 2 */  
hdma_memtomem_dma2_stream0.XferCpltCallback=&XferCpltCallback;  
/* USER CODE END 2 */  
  
HAL_DMA_Start_IT(&hdma_memtomem_dma2_stream0,(uint32_t)Buffer_Src,(uint32_t)Buffer_Dest,10);
```

Start DMA transfer
End of transfer is indicated interrupt callback

After callback assing
HAL_DMA_IRQHandler
can call correct
callback function



DMA with Interrupt lab

156

- DMA callback creation function prototype

```
/* USER CODE BEGIN 0 */  
uint8_t Buffer_Src[]={0,1,2,3,4,5,6,7,8,9};  
uint8_t Buffer_Dest[10];  
  
void XferCpltCallback(DMA_HandleTypeDef *hdma);  
/* USER CODE END 0 */
```

- DMA complete callback with nop where we can put breakpoint

```
/* USER CODE BEGIN 4 */  
void XferCpltCallback(DMA_HandleTypeDef *hdma)  
{  
    __NOP(); //we reach this only if DMA transfer was correct  
}  
/* USER CODE END 4 */
```



HAL service peripherals –PWR Power Controller

HAL service peripherals

PWR –main APIs

158

- PWR HAL driver handles power management features
 - PVD configuration, enabling/disabling and interrupt handling
 - HAL_PWR_PVDConfig()
 - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
 - HAL_PWR_PVD_IRQHandler()
 - HAL_PWR_PVDCallback()
 - Low power mode entry
 - HAL_PWR_EnterSLEEPMode()
 - HAL_PWR_EnterSTOPMode()
 - HAL_PWR_EnterSTANDBYMode()

HAL service peripherals

PWR –extension APIs

159

- For F4x family the following extension function are available
 - Flash overdrive control and flash power-down (for F429/F439 only)
 - HAL_PWREx_ActivateOverDrive()
 - HAL_PWREx_EnableFlashPowerDown()
 - Backup domain registers enable/disable
 - HAL_PWREx_EnableBkUpReg() / HAL_PWREx_DisableBkUpReg

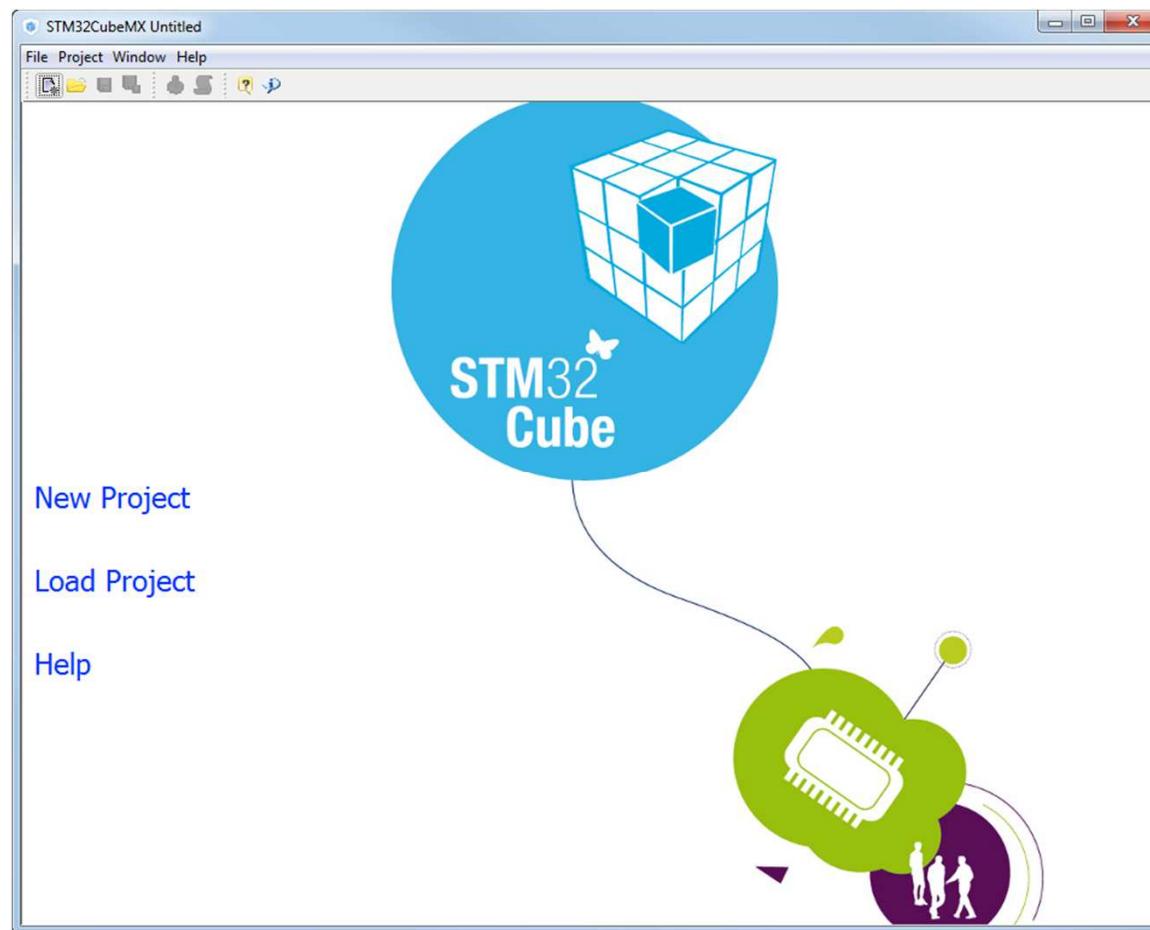


Communication Peripherals HAL driver model

Use UART Poll lab

161

- Use **UART Poll lab** to create project with UART
- We use this project for better demonstration how the HAL library works



Peripheral HAL driver model

HAL peripheral Handle

162

```
/**  
 * @brief  UART handle Structure definition  
 */  
typedef struct  
{  
    USART_TypeDef  
    USART_InitTypeDef  
    uint8_t  
    uint16_t  
    uint16_t  
    uint8_t  
    uint16_t  
    uint16_t  
    DMA_HandleTypeDef  
    DMA_HandleTypeDef  
    HAL_LockTypeDef  
    __IO HAL_UART_StateTypeDef  
    __IO HAL_UART_ErrorTypeDef  
}  
UART_HandleTypeDef;
```

* Instance; USART instance (ex. USART1, USART2, ...)
Init; Initialization structure(baudrate, parity, ...)
TxXferSize; /* UART Tx Transfer size */
TxXferCount; /* UART Tx Transfer Counter */
pRxBuffPtr; /* Pointer to UART Rx transfer Buffer */
RxXferSize; /* UART Rx Transfer size */
RxXferCount; /* UART Rx Transfer Counter */
hdmatx; /* UART Tx DMA Handle parameters */
hdmarx; /* UART Rx DMA Handle parameters */
Lock; Lock object /*
State; State (BUSY, TIMEOUT, ERROR)
ErrorCode; Error code /*



life.augment

Peripheral HAL driver model

HAL peripheral Handle

163

```
/**  
 * @brief  UART handle Structure definition  
 */  
typedef struct  
{  
    USART_TypeDef                Instance;          /* UART registers base address */  
    UART_InitTypeDef             Init;              /* UART communication parameters */  
    uint8_t                      *pTxBuffPtr;        /* Pointer on data to send */  
    uint16_t                     TxXferSize;         /* Number of bytes to send */  
    uint16_t                     TxXferCount;        /* Count of bytes sent by UART */  
    uint8_t                      *pRxBuffPtr;        /* Pointer on received data */  
    uint16_t                     RxXferSize;         /* Number of bytes to receive */  
    uint16_t                     RxXferCount;        /* Count of bytes deceived by UART */  
    DMA_HandleTypeDef            *hdmatx;           /* Pointer on DMA transmit handler */  
    DMA_HandleTypeDef            *hdmarx;           /* Pointer on DMA receive handler */  
    HAL_LockTypeDef               Lock;              /* Locking object */  
    __IO HAL_UART_StateTypeDef   State;              /* UART communication state */  
    __IO HAL_UART_ErrorTypeDef   ErrorCode;         /* UART Error code */  
}UART_HandleTypeDef;
```



life.augment

Peripheral HAL driver model

HAL peripheral Handle

164

- HAL peripheral APIs use a **peripheral handle structure** as parameter
- The handle structure allows to save particular parameters for a peripheral instance (peripherals initialization parameters, global variables, peripheral state, DMA channel handles)
- The handle structure is peripheral dependent, the following is an example for the USART handle (parameters in red are present in all peripherals handlers)

Field	Type	Description
Instance	USART_TypeDef*	Pointer to the peripheral instance register base address
Init	USART_InitTypeDef	Peripheral initialization structure
pTxBuffPtr	uint8_t*	Pointer to Transmit buffer
pRxBuffPtr	uint8_t*	Pointer to Receive buffer
TxXferSize	uint16_t	Usart Tx Transfer size
RxXferSize	uint16_t	Usart Rx Transfer size
TxXferCount	uint16_t	counter of the transmitted data
RxXferCount	uint16_t	counter of the received data
Lock	HAL_LockTypeDef	Lock object : used internally to lock access to shared HW resources
State	USART_StateTypeDef	USART peripheral state
ErrorCode	unit8_t	Error code
hdmatx	DMA_HandleTypeDef*	DMA handle for Transmit: should be initialized in case DMA will be used for USART transmit operation
hdmarx	DMA_HandleTypeDef*	DMA handle for Receive : should be initialized in case DMA will be used for USART receive operation

Peripheral HAL driver model

Driver API groups

165

- Peripheral driver APIs are organized in four groups
 - Initialization and de-initialization APIs
 - Process APIs : can be polling, interrupt or DMA based
 - Peripheral subsystem configuration and feature control APIs
 - Peripheral GetState and Get Errors APIs

API group	examples
Initialization and de-initialization (HAL_PPP_Init()/_DeInit)	HAL_USART_Init() HAL_USART_DeInit()
Process operation	HAL_SPI_Receive() HAL_SPI_Receive_IT() HAL_USART_Transmit_DMA()
Peripheral subsystem configuration Peripheral feature control	HAL_ADC_ConfigChannel() HAL_RTC_SetAlarm()
Peripheral GetState and GetError	HAL_I2C_GetState() HAL_I2C_GetError()

Peripheral HAL driver model

Interrupt handler & callback functions

166

- Besides the APIs, HAL peripheral drivers implement
 - The peripheral interrupt handler(ex: HAL_SPI_IRQHandler): should be called from stm32f4xx_it.c
 - User callback functions
- User callback functions are defined as empty functions with “weak” attribute , they need to be redefined in application code when used
- The following user callbacks functions are defined
 - Peripheral HAL_PPP_Init()/_DeInit() APIs call
 - **HAL_PPP_MspInit()/_DeInit** : these callback functions can be used by the user to do system level initialization(de-initialization) of the peripheral (clock, GPIO, DMA, NVIC)
 - When using interrupt and DMA process APIs, the following callbacks are called
 - Process complete callback functions : **HAL_PPP_ProcessCpltCallback**
 - Error callback in case of peripheral or DMA error: **HAL_PPP_ErrorCallback**
 - Peripheral interrupt handler may signal events to the user through callback functions
 - **HAL_PPP_PeripheralEvent_Callback**

- CubeMX generate UART handler structure in main.c

```
/* Private variables -----*/
UART_HandleTypeDef huart1;
```

- In main we can find MX_USART1_UART_Init function

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* USER CODE BEGIN 3 */
    /* Infinite loop */
    while (1)
    {
    }
    /* USER CODE END 3 */
}
```

Setup parameters which we selected in CubeMX

- CubeMX generate MX_USART1_UART_Init function in main.c

```
/* USART1 init function */
void MX_USART1_UART_Init(void)
{
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 9600;
    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;
    HAL_UART_Init(&huart1);
}
```

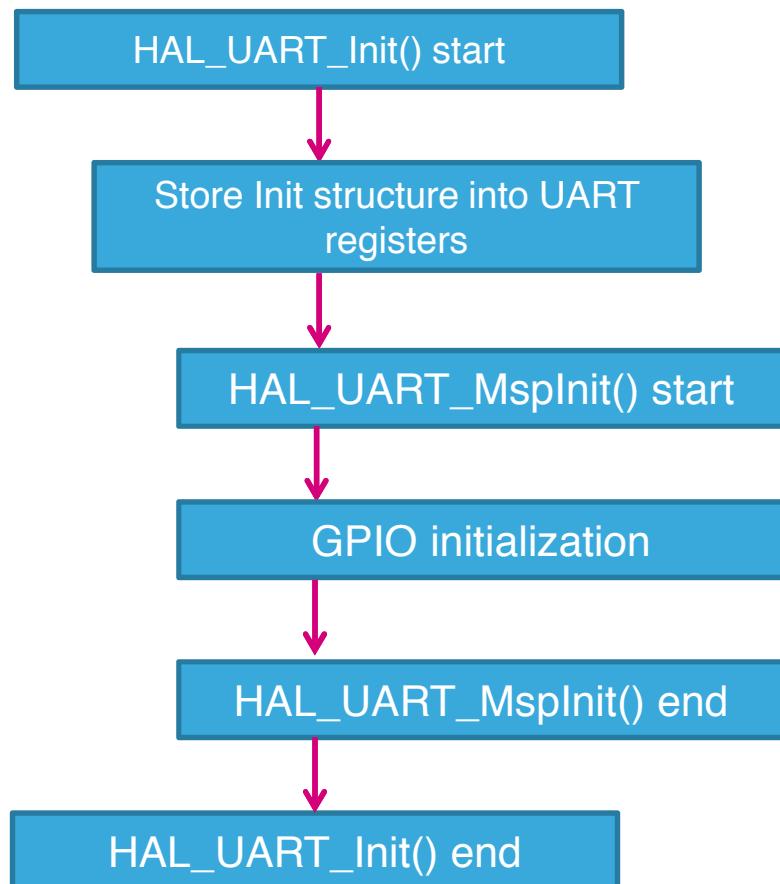
Store UART parameters into Init structure in UART handler

HAL UART initialization procedure

HAL Init structure

169

- HAL_UART_Init function details



```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    if(huart->Instance==USART1)
    {
        /* USER CODE BEGIN USART1_MspInit 0 */

        /* USER CODE END USART1_MspInit 0 */
        /* Peripheral clock enable */
        __USART1_CLK_ENABLE();

        /**USART1 GPIO Configuration
        PA9      -----> USART1_TX
        PA10     -----> USART1_RX
        */
        GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
        GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

        /* USER CODE BEGIN USART1_MspInit 1 */

        /* USER CODE END USART1_MspInit 1 */
    }
}
```

HAL HAL_UART_MspInit structure

170

All HAL_UART_Init functions
calling same function
HAL_UART_MspInit

From handler instance
parameter is discovered
which UART need to be
initialized

Initialize GPIO selected
in CubeMX

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    if(huart->Instance==USART1)
    {
        /* USER CODE BEGIN USART1_MspInit 0 */

        /* USER CODE END USART1_MspInit 0 */
        /* Peripheral clock enable */
        __USART1_CLK_ENABLE();

        /**USART1 GPIO Configuration
        PA9      -----> USART1_TX
        PA10     -----> USART1_RX
        */
        GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
        GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

        /* USER CODE BEGIN USART1_MspInit 1 */

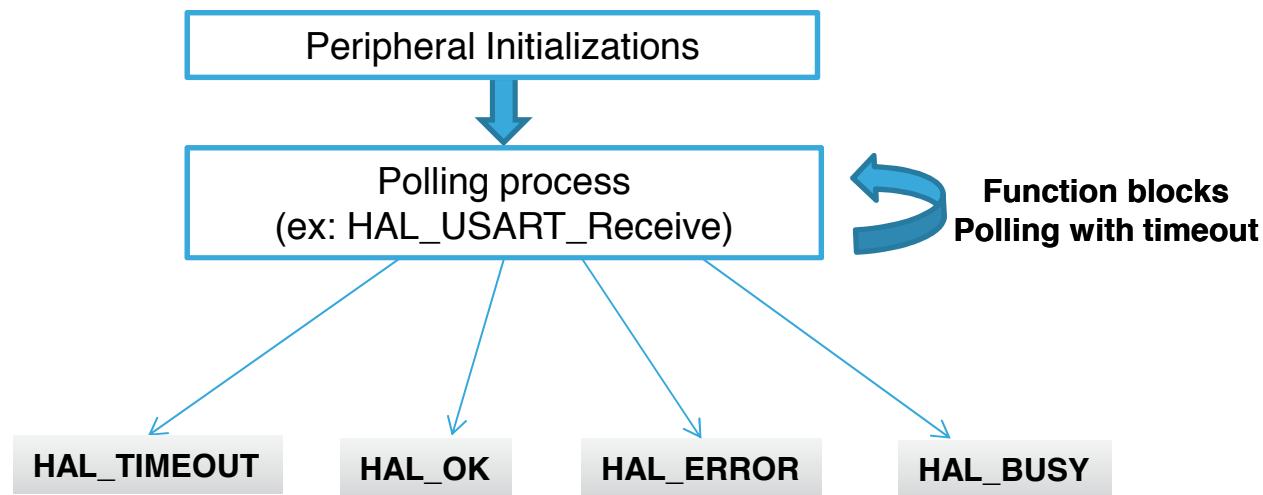
        /* USER CODE END USART1_MspInit 1 */
    }
}
```

Peripheral HAL driver model

Blocking polling process

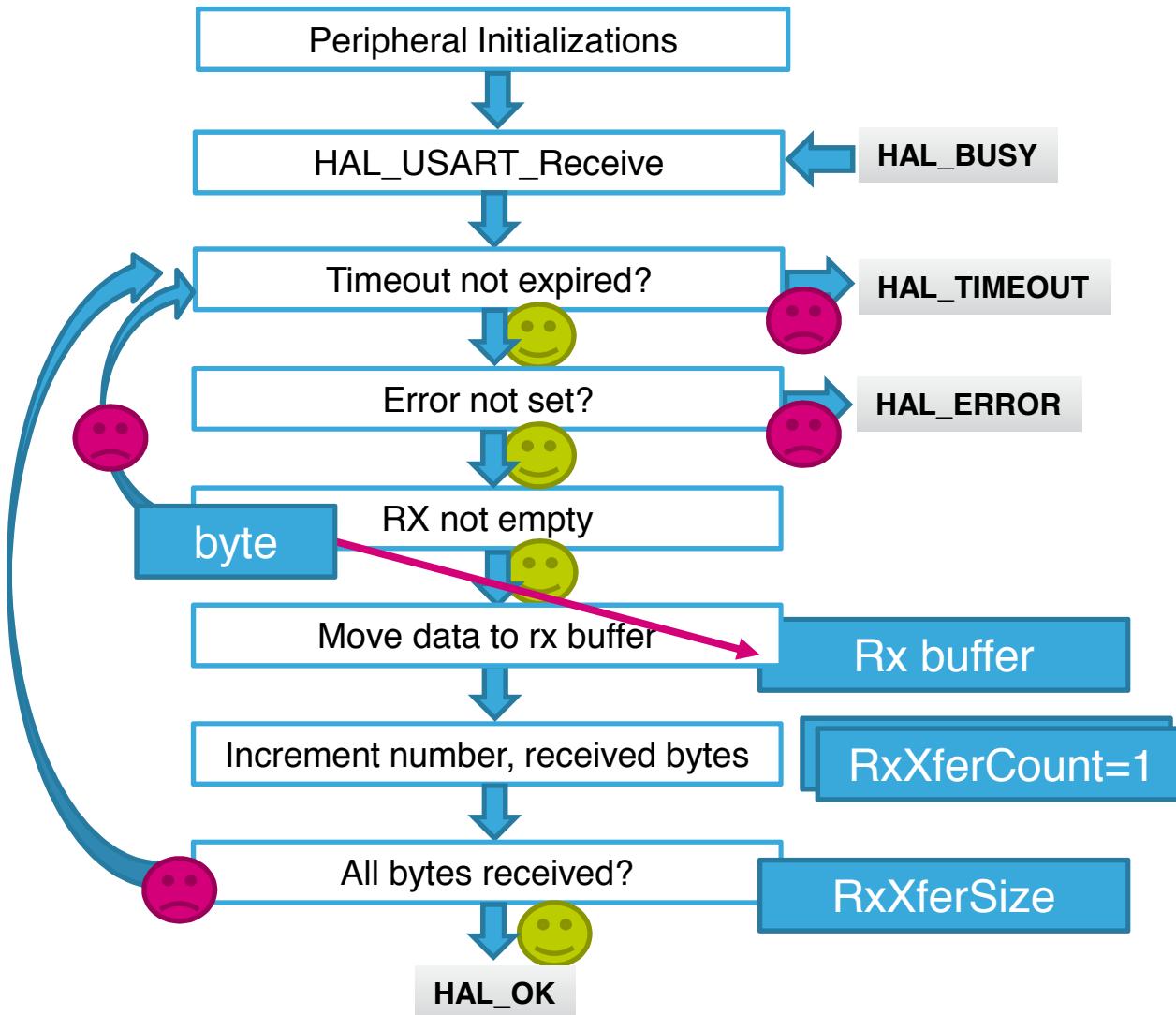
171

- Blocking polling process APIs
 - Block until end of the process, or exit with timeout , error or busy
 - Ex: HAL_USART_Receive()



Pooling receive functionality

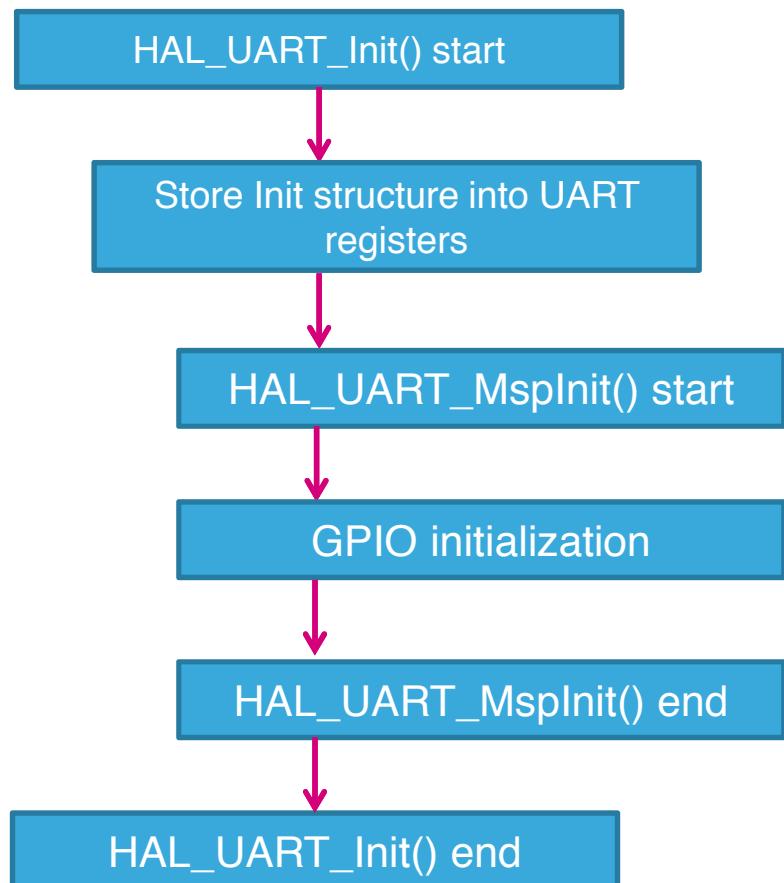
172



HAL general concepts

HAL global APIs HAL_Init()

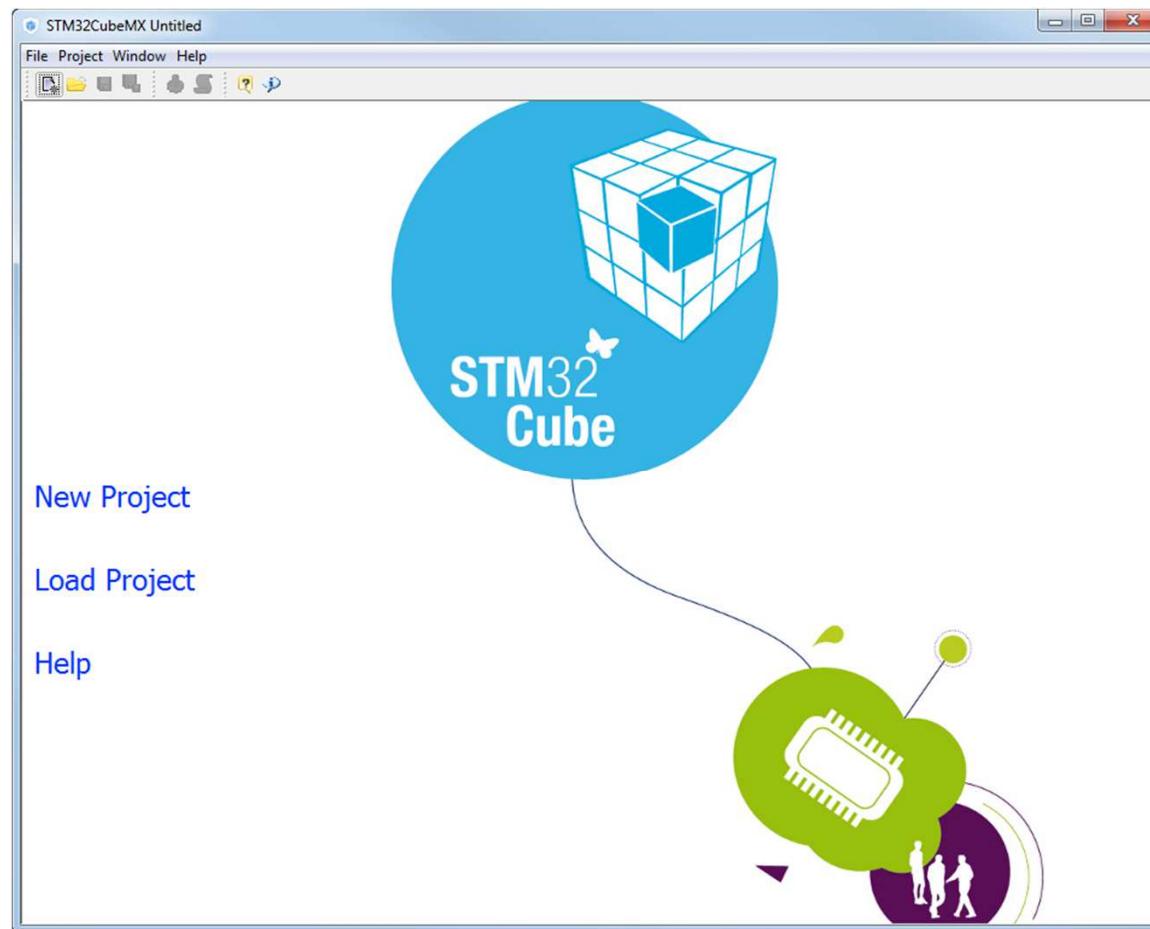
173



Use UART with Interrupt lab

174

- Use **UART with Interrupt lab** to create project with UART
- We use this project for better demonstration how the HAL library works



- Main.c will be same as in USART Poll example
- MSP initialization is now different in stm32f4xx_hal_uart.c

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    if(huart->Instance==USART1)
    {
        /* USER CODE BEGIN USART1_MspInit 0 */
        /* USER CODE END USART1_MspInit 0 */
        /* Peripheral clock enable */
        __USART1_CLK_ENABLE();
        /**USART1 GPIO Configuration
        PA9      -----> USART1_TX
        PA10     -----> USART1_RX      */
        GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
        GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
        /* System interrupt init*/
        HAL_NVIC_SetPriority(USART1_IRQn, 0, 0);
        HAL_NVIC_EnableIRQ(USART1_IRQn);
        /* USER CODE BEGIN USART1_MspInit 1 */
        /* USER CODE END USART1_MspInit 1 */
    }
}
```

USART1 interrupt is now enabled in NVIC.

Now function HAL_UART_Receive_IT or HAL_UART_Receive_IT will trigger interrupt

Interrupt handling

176

- Interrupt handling in `stm32f4xx_it.c`
 - UART Handler is imported from `main.c`

```
/* External variables -----*/
extern UART_HandleTypeDef huart1;
```

- `USART1_IRQHandler` is called if interrupt is triggered
- `HAL_UART_IRQHandler` is hal function which manage UART interrupts

```
/**
 * @brief This function handles USART1 global interrupt.
 */
void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQHandler_0 */
    /* USER CODE END USART1_IRQHandler_0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQHandler_1 */
    /* USER CODE END USART1_IRQHandler_1 */
}
```

As parameter is used
USART1 handle

Callback creation

177

- Callbacks are defined as `_weak` in `stm32f4xx_hal_uart.c`

```
/**  
 * @brief Rx Transfer completed callbacks.  
 * @param huart: pointer to a UART_HandleTypeDef structure that contains  
 *                 the configuration information for the specified UART module.  
 * @retval None  
 */  
_weak void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  
{  
    /* NOTE: This function Should not be modified, when the callback is needed,  
           the HAL_UART_TxCpltCallback could be implemented in the user file  
    */  
}
```

- If we defined callback, for example in `main.c`, after successful reception HAL will jump into it

```
/* USER CODE BEGIN 4 */  
void HAL_UART_RxCpltCallback(UART_HandleTypeDef  
*huart)  
{  
}  
/* USER CODE END 4 */
```

UART callback types

178

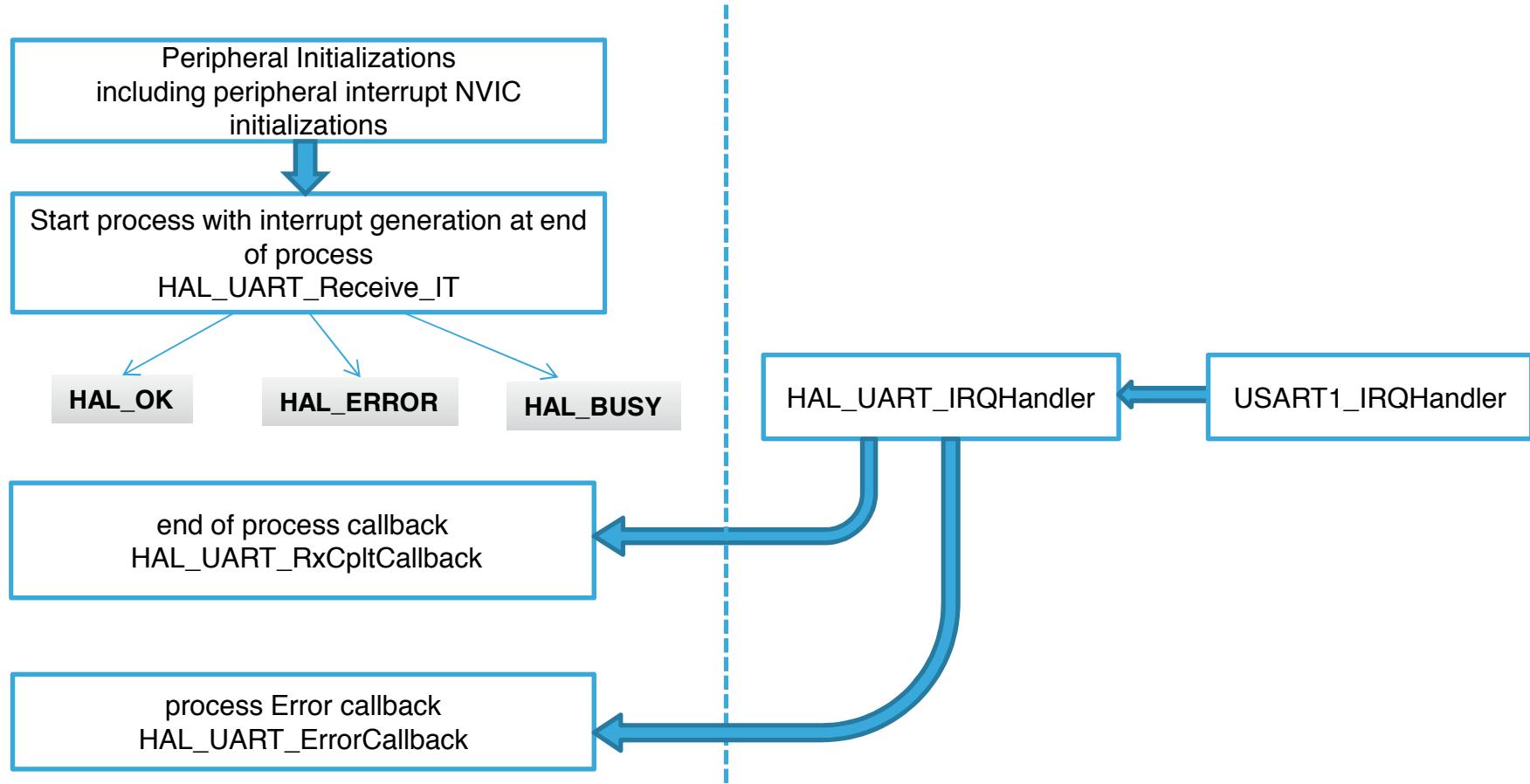
- Can be found in `stm32f4xx_hal_uart.c` (search for “`__weak`”)

DMA HAL APIs	Description	Poll	IT	DMA
<code>HAL_UART_MspInit</code>	Initialize related system peripherals	X	X	X
<code>HAL_UART_MspDeInit</code>	Deinitialize related system peripherals	X	X	X
<code>HAL_UART_TxCpltCallback</code>	Called if complete buffer was transmitted		X	X
<code>HAL_UART_TxHalfCpltCallback</code>	Called if half of buffer was transmitted			X
<code>HAL_UART_RxCpltCallback</code>	Called if complete buffer was received		X	X
<code>HAL_UART_RxHalfCpltCallback</code>	Called if half of buffer was received			X
<code>HAL_UART_ErrorCallback</code>	Called if error occurred		X	X

Use UART with Interrupt lab

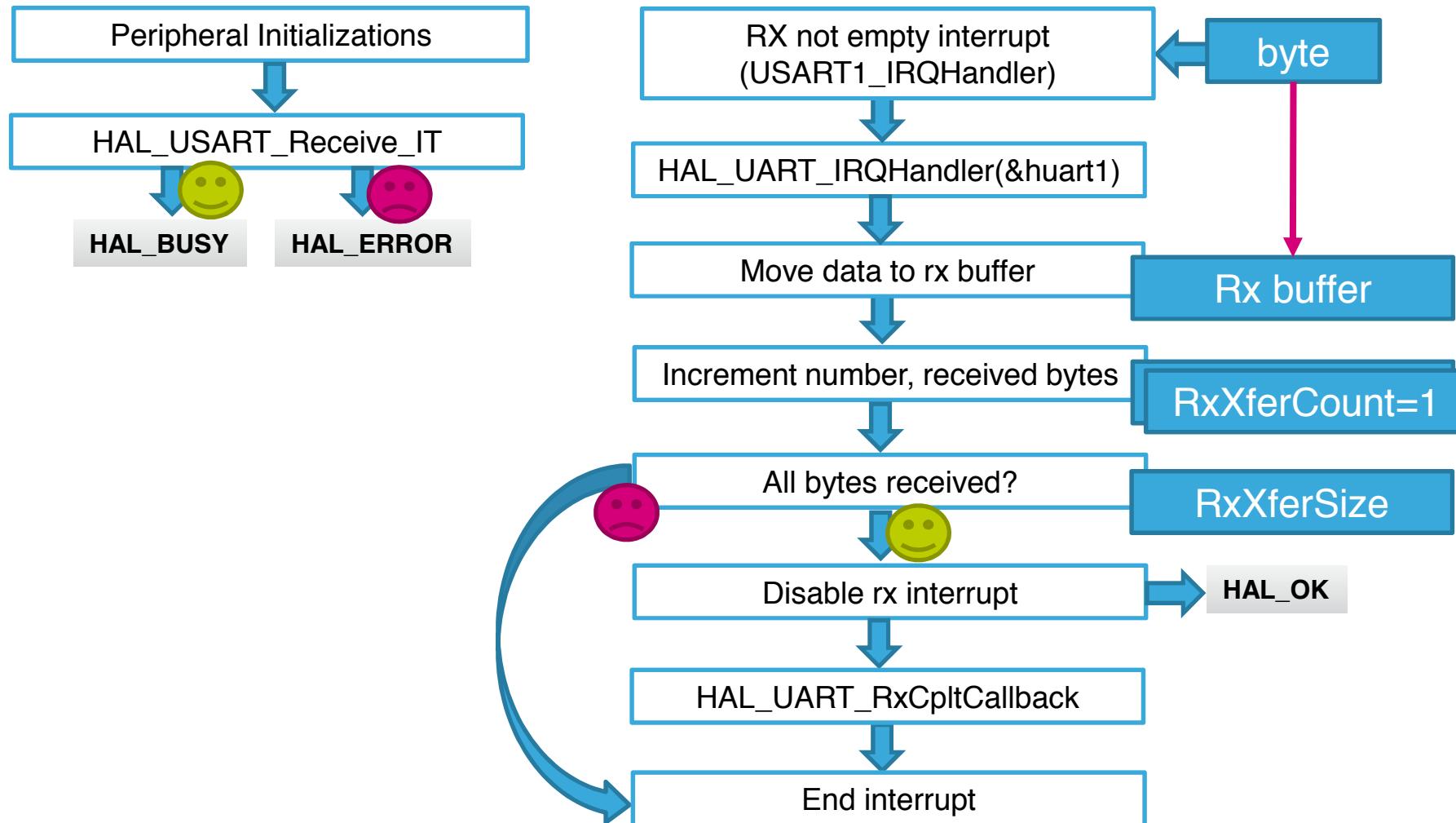
179

HAL Library UART with IT receive flow



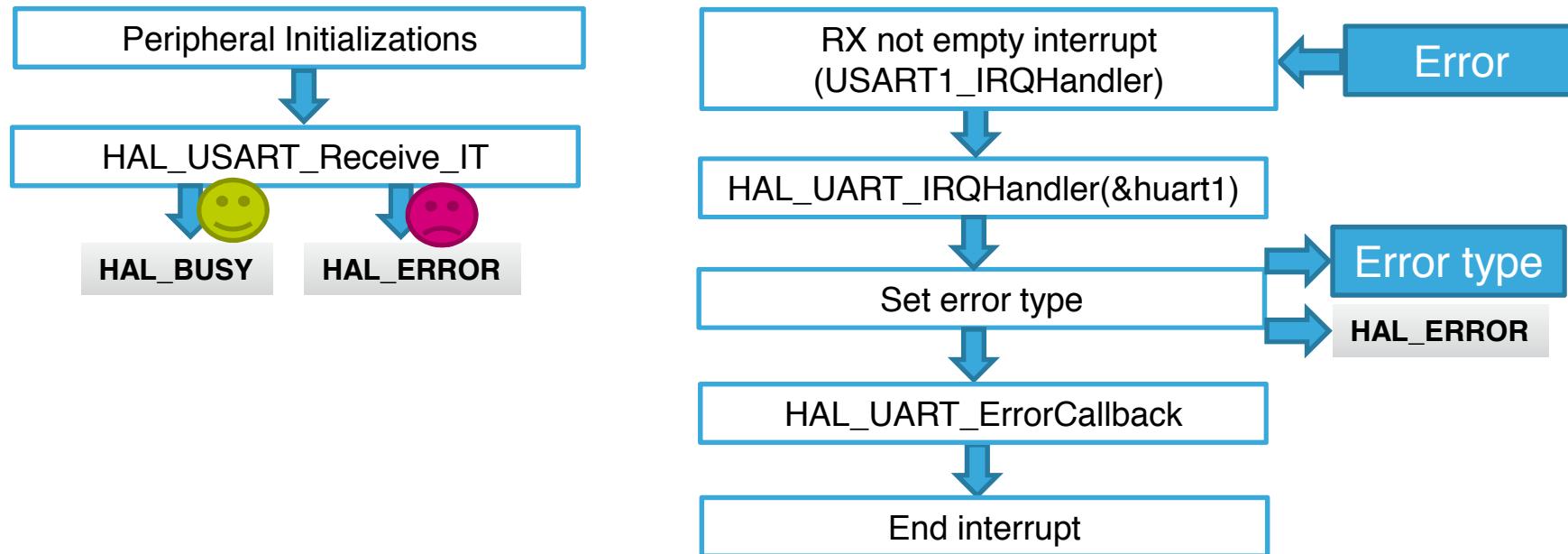
UART receive with interrupt

180



UART receive with interrupt

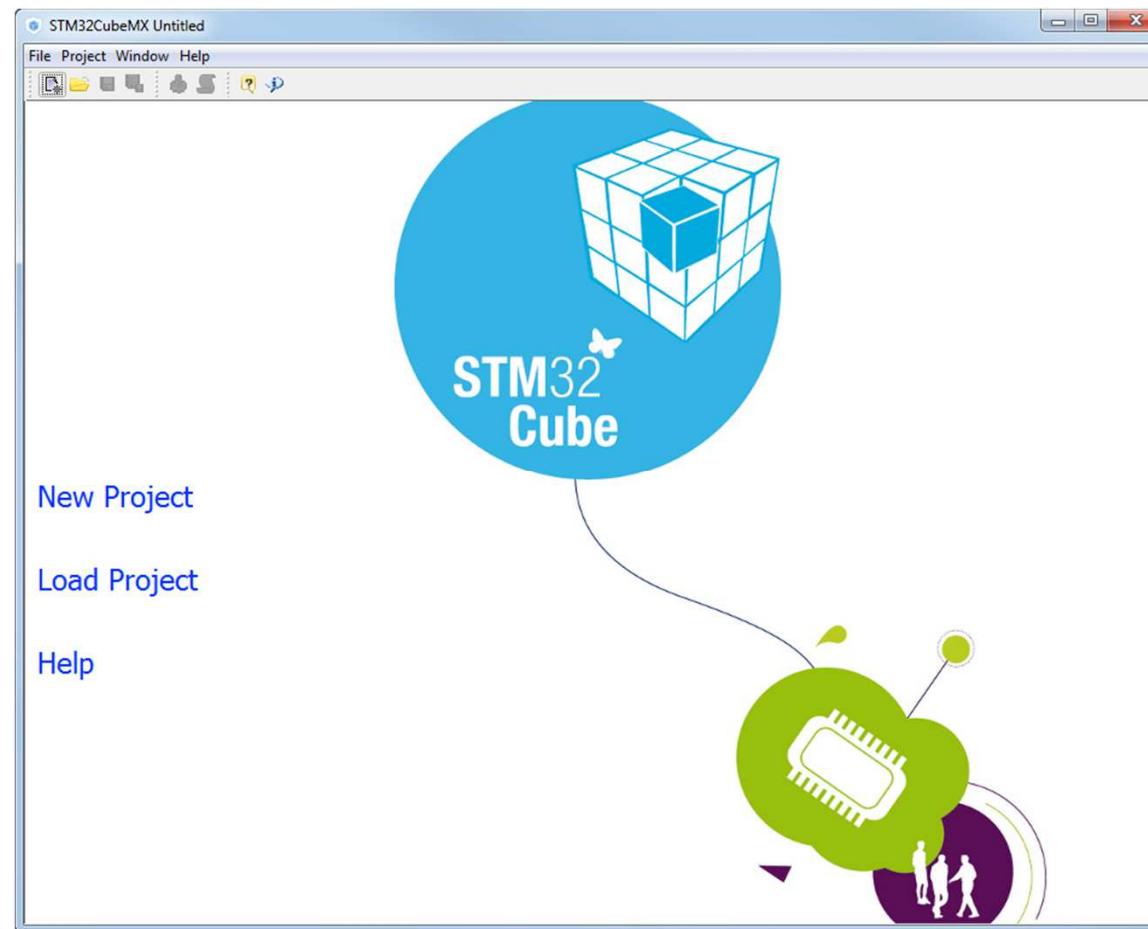
181



Use UART with DMA lab

182

- Use **UART with DMA lab** to create project with UART
- We use this project for better demonstration how the HAL library works



UART with DMA initialization

183

- In main.c is now defined UART handler and also DMA handlers

```
/* Private variables -----*/
UART_HandleTypeDef huart1;
DMA_HandleTypeDef hdma_usart1_rx;
DMA_HandleTypeDef hdma_usart1_tx;
```

- In main function we can find additional initialization for DMA

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init(),
    MX_DMA_Init();
    MX_USART1_UART_Init(),
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* USER CODE BEGIN 3 */
    /* Infinite loop */
    while (1)
    {
    }
    /* USER CODE END 3 */}
```

MX_DMA initialization



life.augr

UART with DMA initialization

184

- In function MX_USART1_UART_Init is nothing new

```
/* USART1 init function */
void MX_USART1_UART_Init(void)
{
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 9600;
    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;
    HAL_UART_Init(&huart1);

}
```

UART with DMA initialization

185

- Function MX_DMA_Init enable DMA2 clock and enable DMA interrupt vectors
- If periphery use DMA to transfer data, the **NVIC interrupt** vectors must be **ENABLED** (DMA transfer use interrupts by default)

```
/**  
 * Enable DMA controller clock  
 */  
void MX_DMA_Init(void)  
{
```

```
    /* DMA controller clock enable */
```

```
    __DMA2_CLK_ENABLE();
```

DMA clock enable

```
    /* DMA interrupt init */
```

```
    HAL_NVIC_SetPriority(DMA2_Stream7_IRQn, 0, 0);
```

```
    HAL_NVIC_EnableIRQ(DMA2_Stream7_IRQn);
```

```
    HAL_NVIC_SetPriority(DMA2_Stream2_IRQn, 0, 0);
```

```
    HAL_NVIC_EnableIRQ(DMA2_Stream2_IRQn);
```

DMA NVIC vector
initialization
and enabling



UART with DMA - MSP file

186

- In `stm32f4xx_hal_msp.c` are now imported DMA handlers

```
extern DMA_HandleTypeDef hdma_usart1_rx;  
extern DMA_HandleTypeDef hdma_usart1_tx;
```

- It is because we need to put information about DMA into UART handler

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
```

- Without this UART cannot cooperate with DMA on HAL base library

UART with DMA - HAL_UART_MspInit

187

- Top of UART MSP function is still same only GPIO initilization

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    if(huart->Instance==USART1)
    {
        /* USER CODE BEGIN USART1_MspInit 0 */
        /* USER CODE END USART1_MspInit 0 */
        /* Peripheral clock enable */
        __USART1_CLK_ENABLE();
        /**USART1 GPIO Configuration
        PA9      -----> USART1_TX
        PA10     -----> USART1_RX
        */
        GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
        GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```



life.augmented

UART with DMA - HAL_UART_MspInit

188

- On the middle start DMA initialization based on CubeMX settings
- Most important part is the **_HAL_LINKDMA** macro which connects UART and DMA structures together

```
/* Peripheral DMA init*/
hdma_usart1_rx.Instance = DMA2_Stream2;
hdma_usart1_rx.Init.Channel = DMA_CHANNEL_4;
hdma_usart1_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
hdma_usart1_rx.Init.PeriphInc = DMA_PINC_DISABLE;
hdma_usart1_rx.Init.MemInc = DMA_MINC_ENABLE;
hdma_usart1_rx.InitPeriphDataAlignment = DMA_PDATAALIGN_BYTE;
hdma_usart1_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
hdma_usart1_rx.Init.Mode = DMA_NORMAL;
hdma_usart1_rx.Init.Priority = DMA_PRIORITY_LOW;
hdma_usart1_rx.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
HAL_DMA_Init(&hdma_usart1_rx);

__HAL_LINKDMA(huart,hdmarx,hdma_usart1_rx);
```

Stream selection,
DMA direction,
data width, ...

__HAL_LINKDMA macro description

189

- In previous slides we work with DMA handler and UART handle
- There are three parameters in their structures which we need to know

```
/** @brief UART handle Structure definition */
typedef struct
{
    USART_TypeDef                *Instance;
    UART_InitTypeDef             Init;
    uint8_t                      *pTxBuffPtr;
    uint16_t                     TxXferSize;
    uint16_t                     TxXferCount;
    uint8_t                      *pRxBuffPtr;
    uint16_t                     RxXferSize;
    uint16_t                     RxXferCount;
    DMA_HandleTypeDef           *hdmatx;
    DMA_HandleTypeDef           *hdmarx;
    HAL_LockTypeDef              Lock;
    __IO HAL_UART_StateTypeDef   State;
    __IO HAL_UART_ErrorTypeDef   ErrorCode;
}UART_HandleTypeDef;
```

UART handle


```
/** @brief DMA handle Structure definition */
typedef struct __DMA_HandleTypeDef
{
    DMA_Stream_TypeDef          *Instance;
    DMA_InitTypeDef             Init;
    HAL_LockTypeDef              Lock;
    __IO HAL_DMA_StateTypeDef   State;
    void                        *Parent;
    XferCpltCallback( struct __DMA_HandleTypeDef * hdma );
    HalfCpltCallback( struct __DMA_HandleTypeDef * hdma );
    __IO uint32_t                ErrorCode;
}DMA_HandleTypeDef;
```

DMA handle

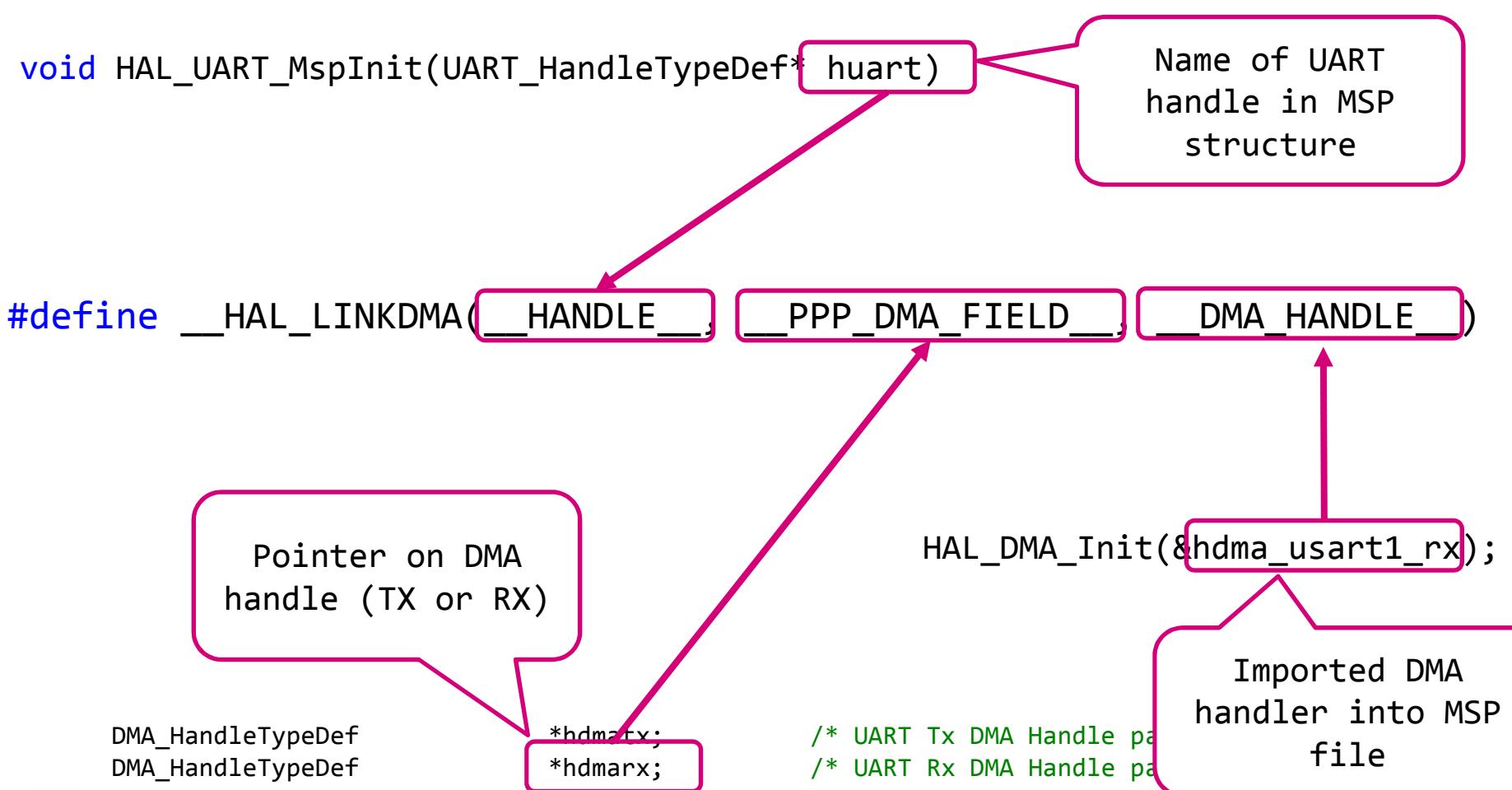
Pointers to DMA tx and RX handles

Pointer to handle which cooperate with DMA, now it will be UART

__HAL_LINKDMA macro description

190

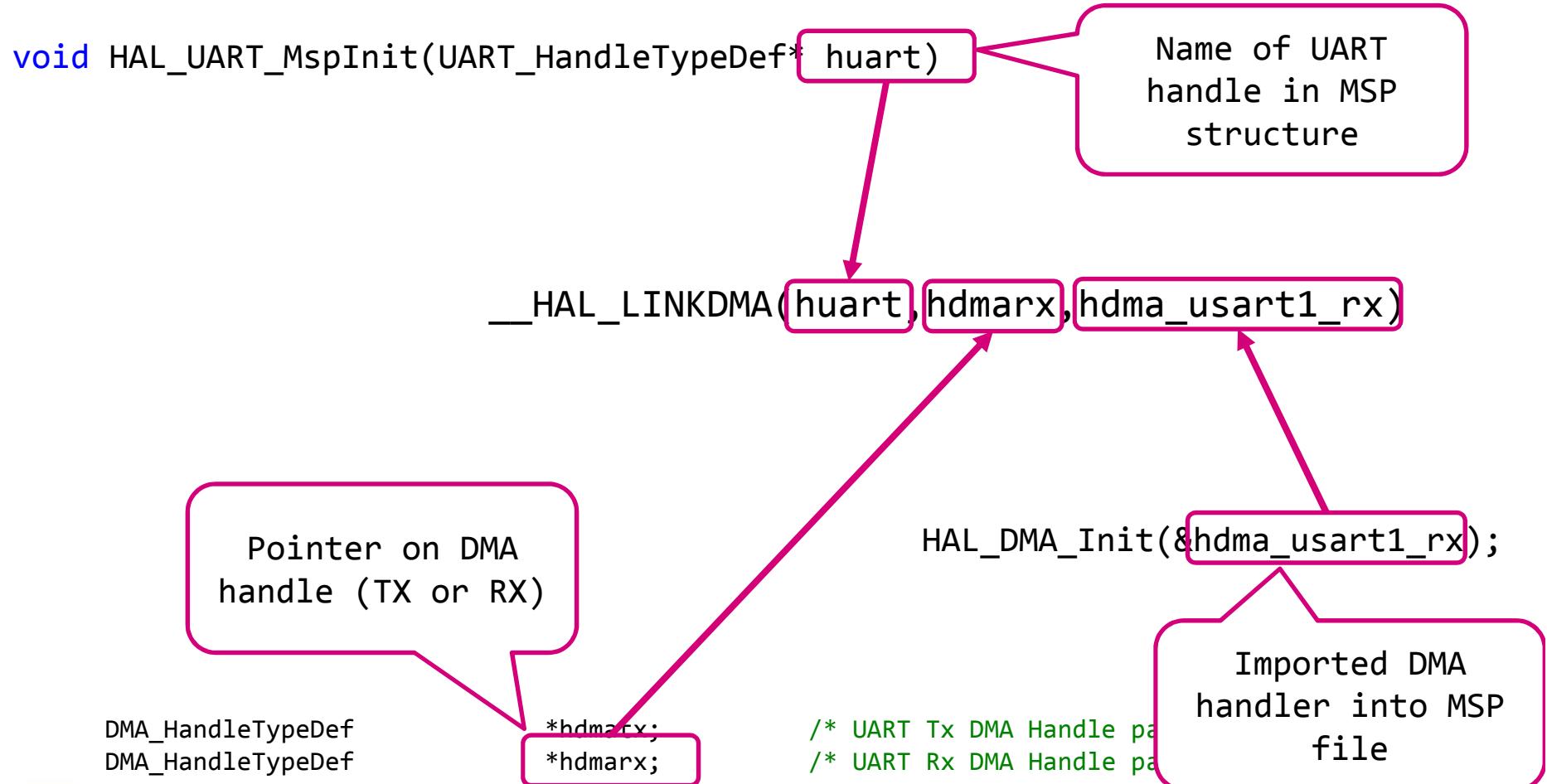
- Definition of macro is in `stm32f4xx_hal_def.h`



__HAL_LINKDMA macro description

191

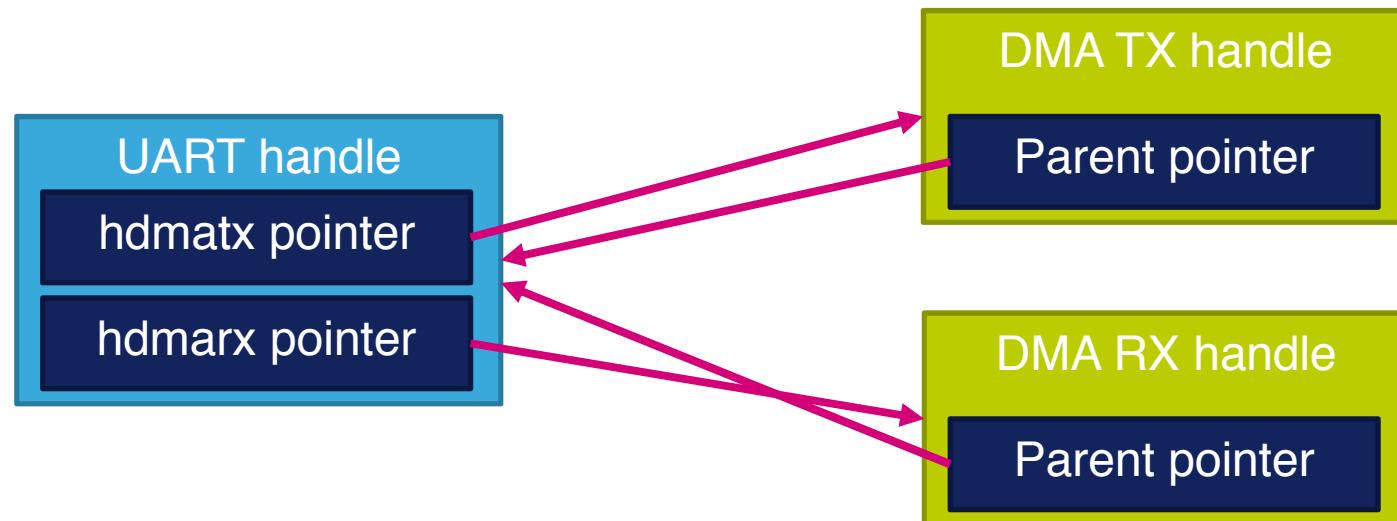
- Definition of macro is in `stm32f4xx_hal_def.h`



__HAL_LINKDMA function description

192

- LINKDMA macro connects DMA and UART handle together
- Is possible get information about DMA from UART handle
- And UART handle from DMA



UART with DMA - HAL_UART_MspInit

193

- Transmit part of DMA initialization is similar to receiving part

```
hdma_usart1_tx.Instance = DMA2_Stream7;
hdma_usart1_tx.Init.Channel = DMA_CHANNEL_4;
hdma_usart1_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
hdma_usart1_tx.Init.PeriphInc = DMA_PINC_DISABLE;
hdma_usart1_tx.Init.MemInc = DMA_MINC_ENABLE;
hdma_usart1_tx.InitPeriphDataAlignment = DMA_PDATAALIGN_BYTE;
hdma_usart1_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
hdma_usart1_tx.Init.Mode = DMA_NORMAL;
hdma_usart1_tx.Init.Priority = DMA_PRIORITY_LOW;
hdma_usart1_tx.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
HAL_DMA_Init(&hdma_usart1_tx);

__HAL_LINKDMA(huart,hdmatx,hdma_usart1_tx);
/* USER CODE BEGIN USART1_MspInit 1 */
/* USER CODE END USART1_MspInit 1 */
}
```

UART with DMA Interrupt

194

- DMA handles are imported into stm32f4xx_it.c

```
/* External variables -----*/
extern DMA_HandleTypeDef hdma_usart1_rx;
extern DMA_HandleTypeDef hdma_usart1_tx;
```

- DMA interrupt call HAL_DMA_IRQHandler not UART Handler

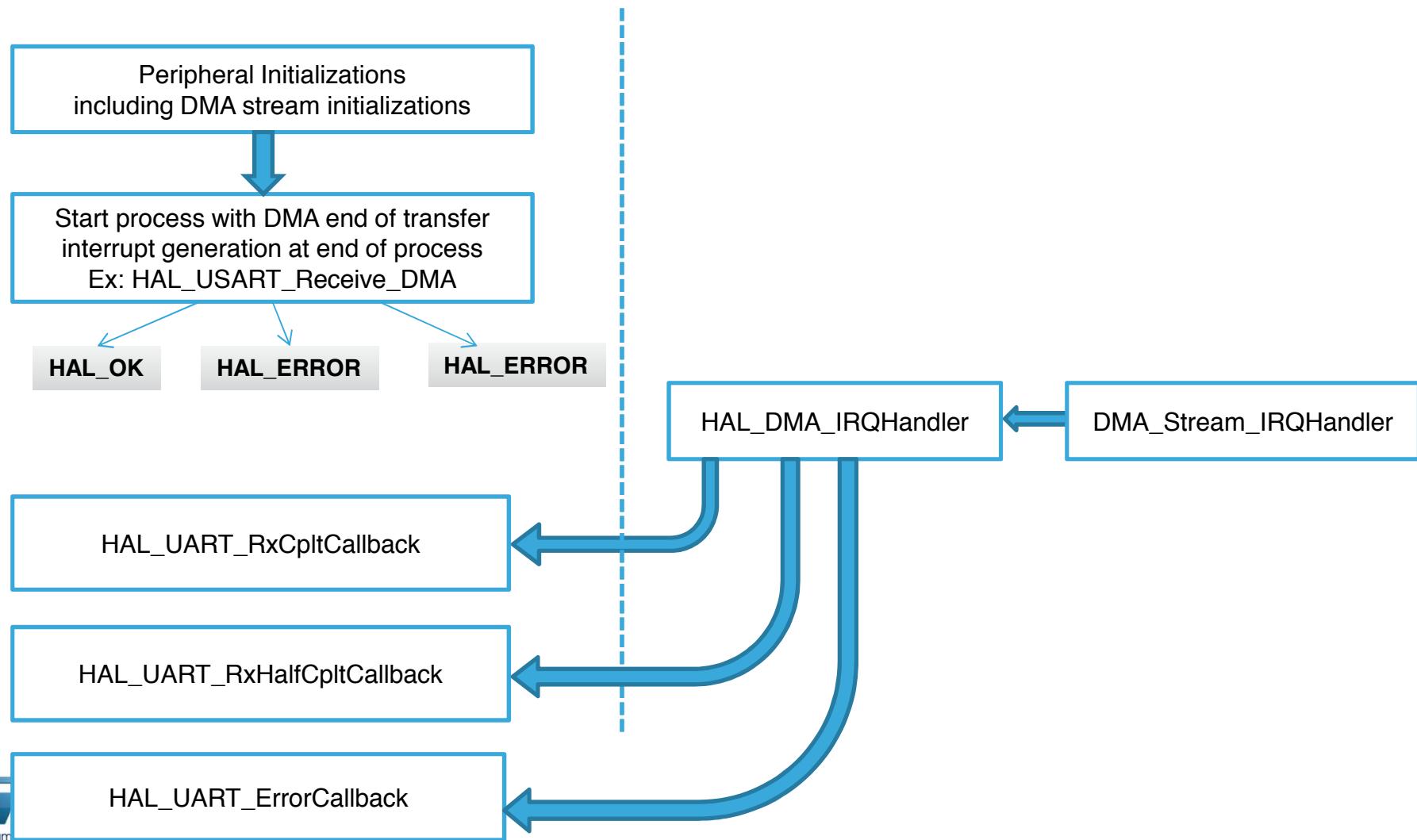
```
/**
 * @brief This function handles DMA2 Stream7 global interrupt.
 */
void DMA2_Stream7_IRQHandler(void)
{
    /* USER CODE BEGIN DMA2_Stream7_IRQHandler_0 */
    /* USER CODE END DMA2_Stream7_IRQHandler_0 */
    HAL_DMA_IRQHandler(&hdma_usart1_tx);
    /* USER CODE BEGIN DMA2_Stream7_IRQHandler_1 */
    /* USER CODE END DMA2_Stream7_IRQHandler_1 */
}
/**
 * @brief This function handles DMA2 Stream2 global interrupt.
 */
void DMA2_Stream2_IRQHandler(void)
{
    /* USER CODE BEGIN DMA2_Stream2_IRQHandler_0 */
    /* USER CODE END DMA2_Stream2_IRQHandler_0 */
    HAL_DMA_IRQHandler(&hdma_usart1_rx);
    /* USER CODE BEGIN DMA2_Stream2_IRQHandler_1 */
    /* USER CODE END DMA2_Stream2_IRQHandler_1 */
}
```

DMA interrupt handled by
HAL_DMA_IRQHandler
Internally is connected to UART
callbacks

Peripheral HAL driver model

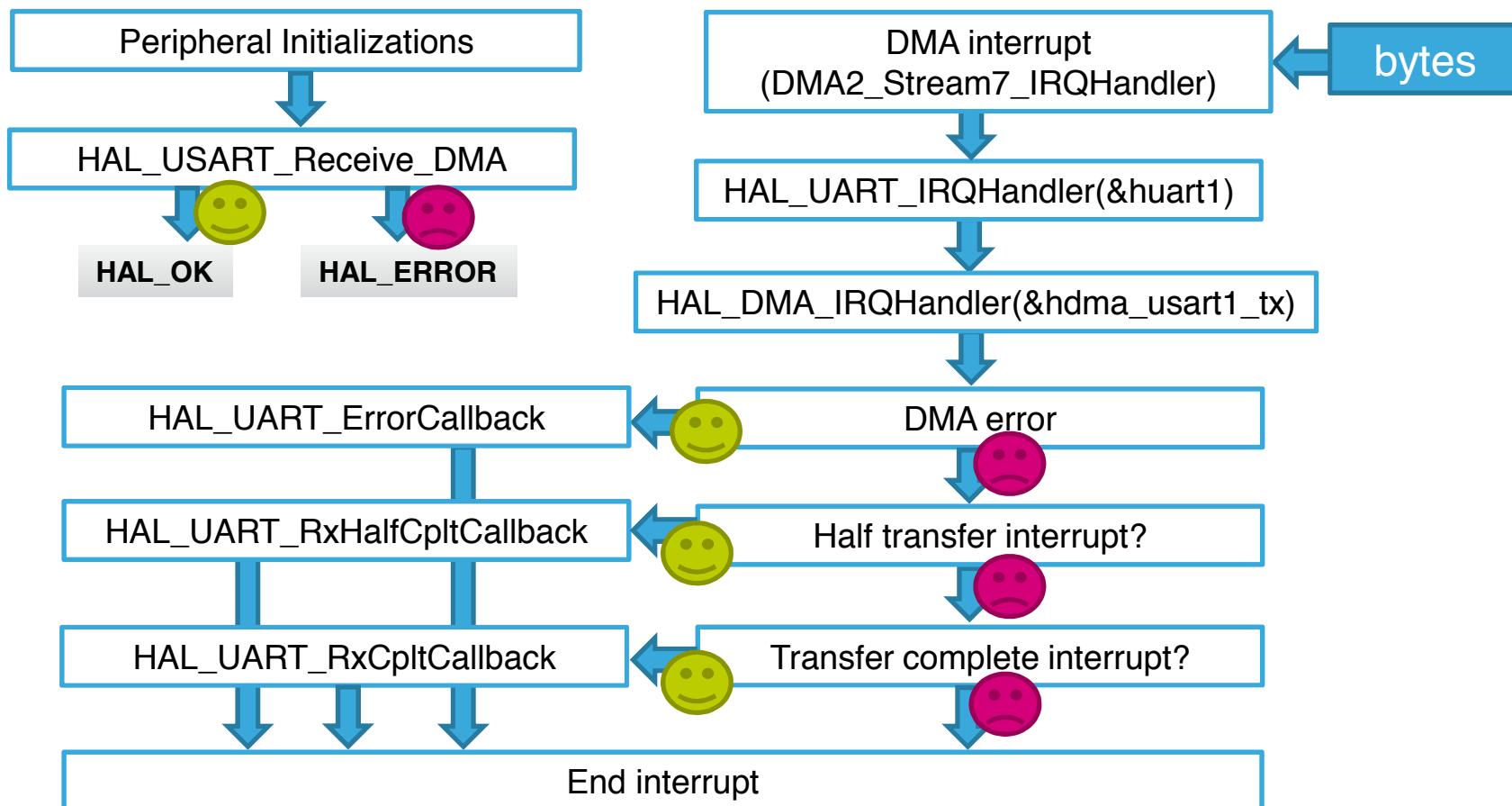
Non blocking process with DMA transfer

195



UART receive with DMA

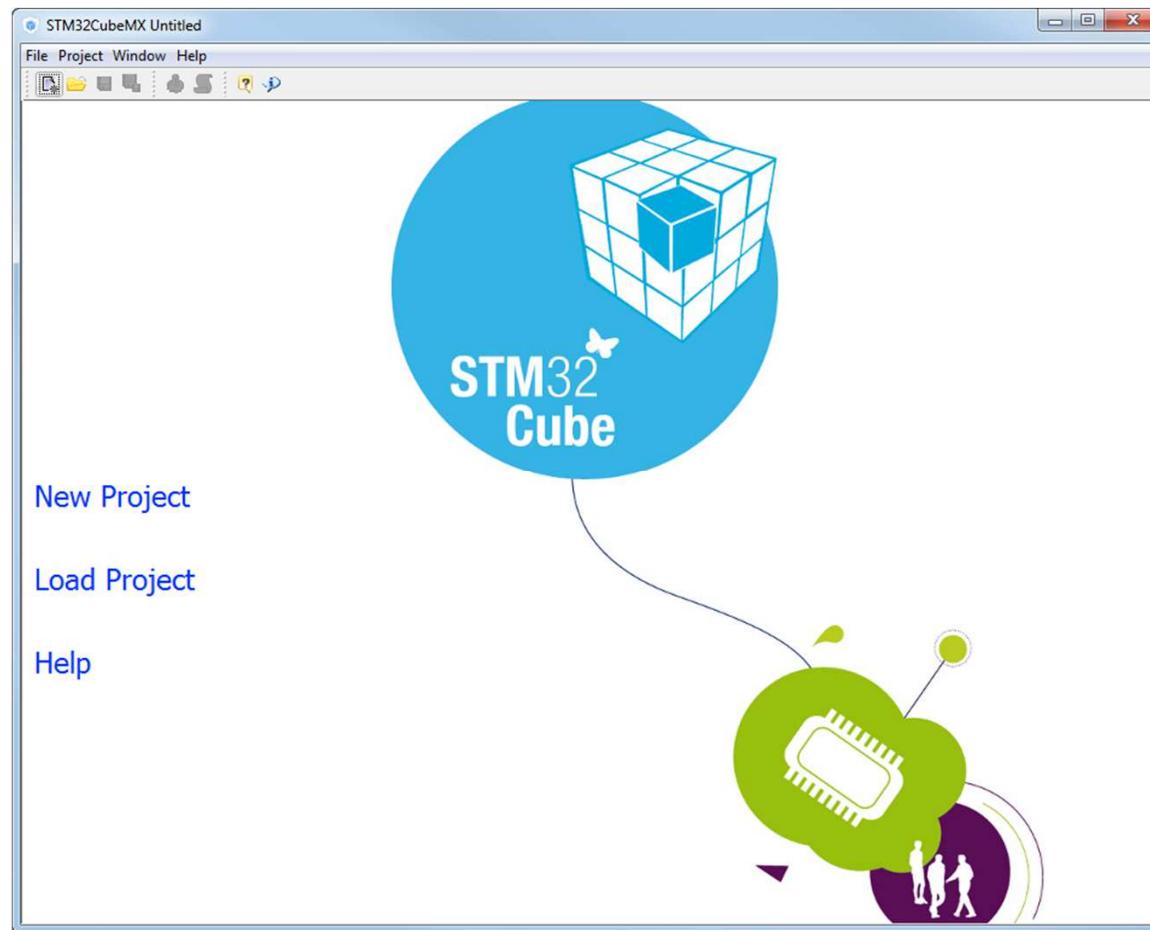
196



Examples for other peripherals

197

- SPI labs very similar like UART

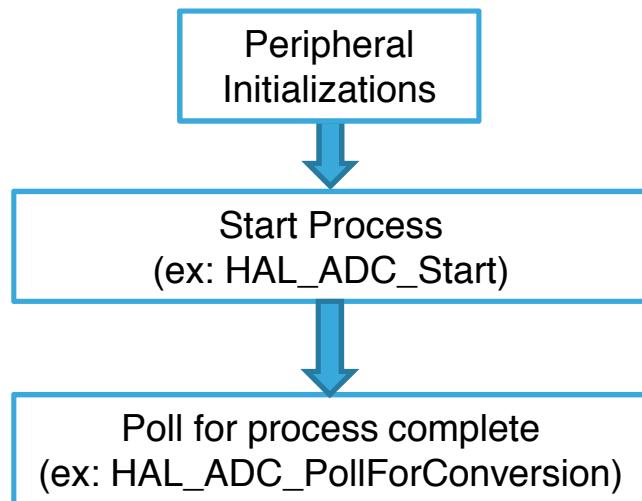


Peripheral HAL driver model

Non Blocking Start process

198

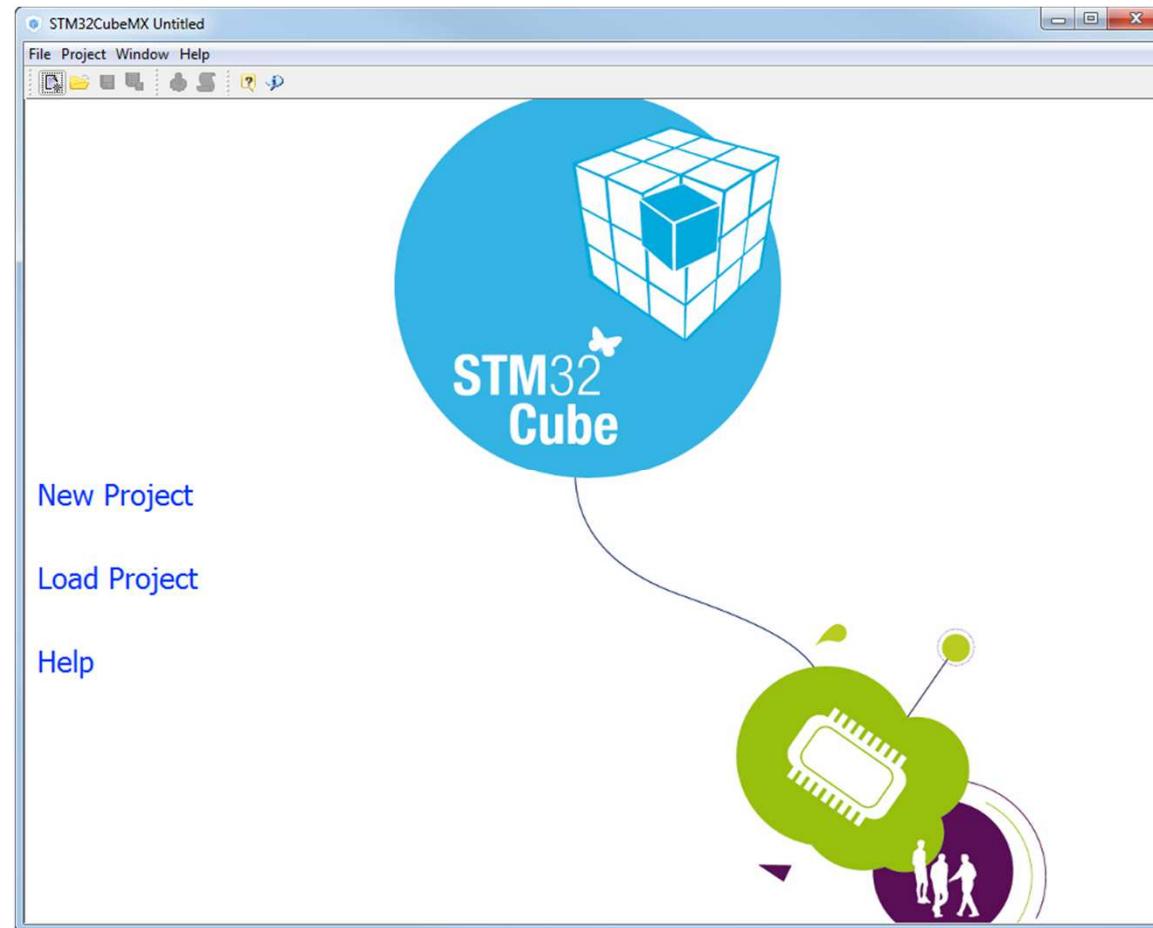
- Non blocking start process
 - Exits directly after starting the process
 - Used mainly for ADC,DAC, Timer
 - Ex: HAL_ADC_Start()



Examples for other peripherals

199

- For example TIM labs, DAC labs, ADC labs
- Still same principle as UART



Guidelines for writing a HAL code

200

- You can start from the template project
- The template project includes all HAL files and implements a main.c files with startup code to initialize system clock (@ max speed)
- Start by writing the peripheral HAL_PPP_MspInit function, in this function you need to:
 - Enable the peripheral clock
 - Configure the peripheral GPIOs
 - configure DMA channel and enable DMA interrupt (if needed)
 - Enable peripheral interrupt (if needed)
- Edit the stm32f4xx_it.c to call required interrupt handlers (periph and DMA)
- Write process complete callback functions if you plan to use peripheral interrupt or DMA
- In your main.c file, initialize the peripheral handle structure then call function HAL_PPP_Init() to initialize your peripheral



At this step you can call any peripheral process or control function

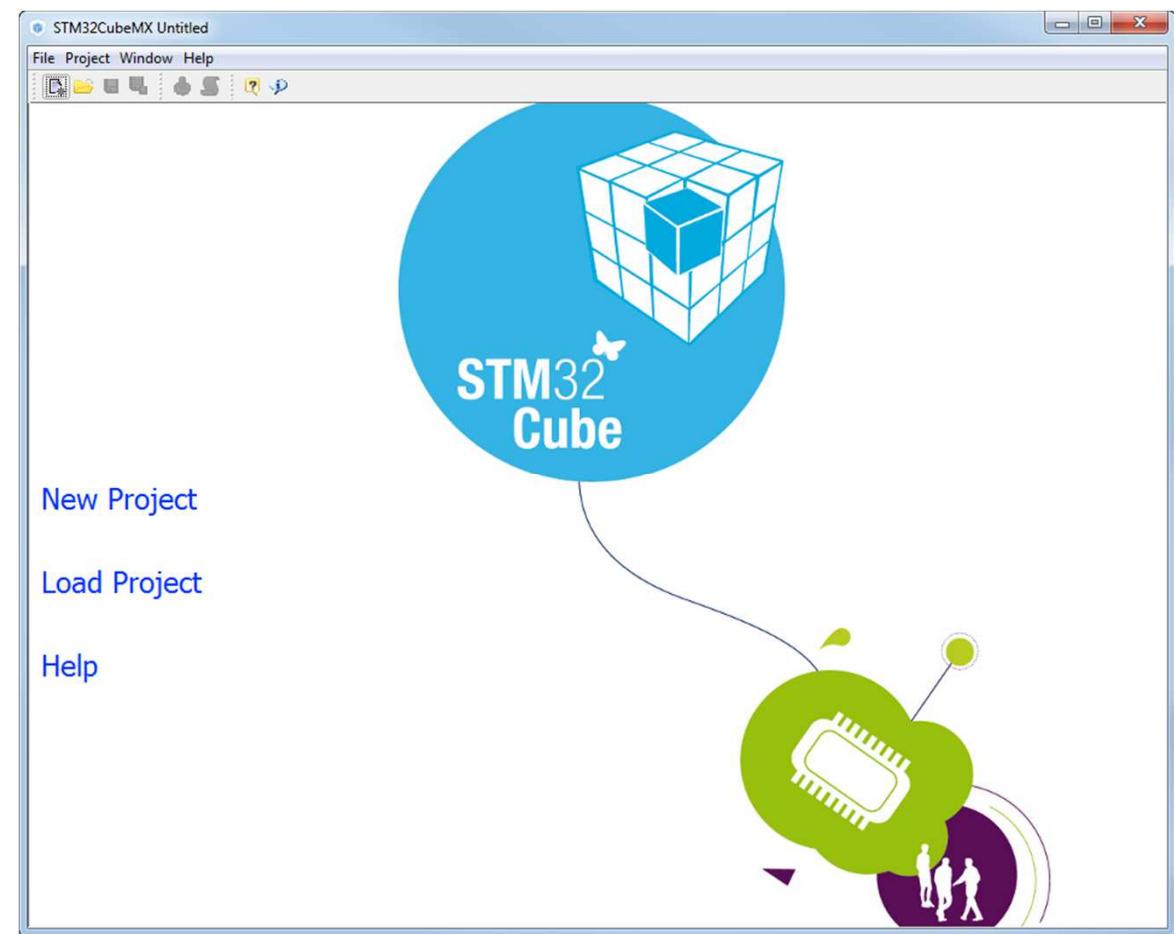


Recapitulation

What is CubeMX

202

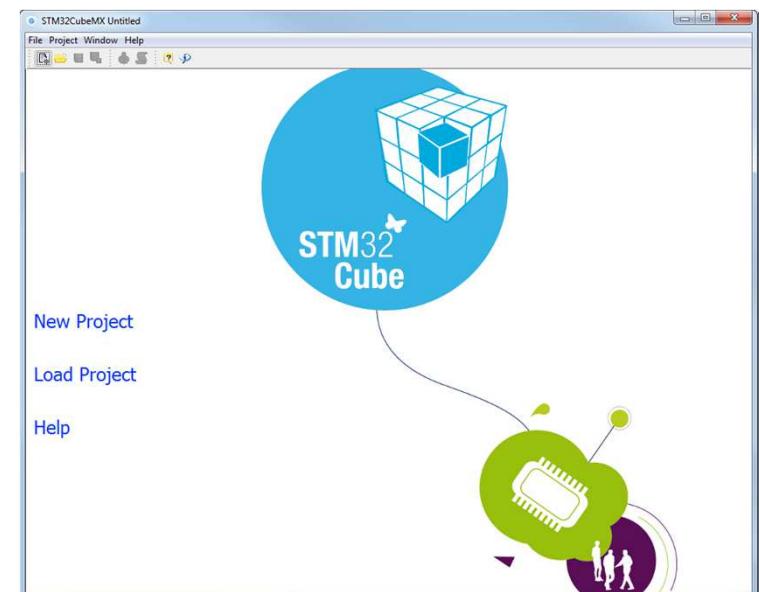
- MCU selector
- Pin out configurator
- Clock configurator
- Periphery configurator
- Code generator



Code generation

203

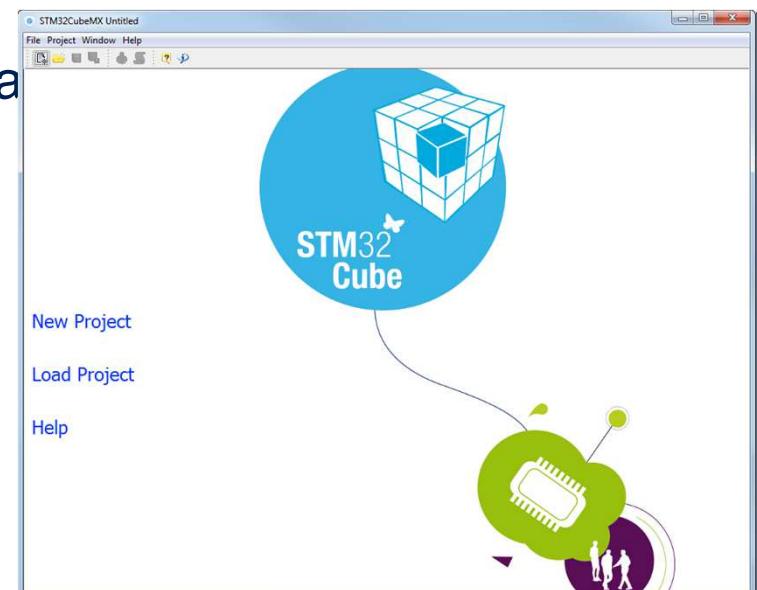
- Generate initialization for peripherals selected in CubeMX
- Create Interrupt handlers and use HAL handlers
- Create not fully functional program
- User must still start peripheries or define handlers
- Create only initialization for options which are possible in CubeMX



Code generation

204

- Generate initialization for peripherals selected in CubeMX
 - Create Interrupt handlers and use HAL handlers
 - Help us faster create program draft then with STD
-
- Create not fully functional program
 - User must still start peripheries or define handles
 - Create only initialization for options which are possible in CubeMX



- Handle based library
- HAL function are not low layer
- If function which you want not exists in HAL or work different way how you want, is very difficult to do it with HAL library
- HAL have no functions to access all functions
Is possible access SR and some functional registers (TIM->CNT reg)



Example of HAL library limitation

206

- TIM trigger DMA transfer, which move data from RAM to GPIO (TIM DMA lab)
- Function `HAL_TIM_Base_Start_DMA` exists, but transfer data from RAM to `TIM->ARR` register
- If we want to transfer data from places which we want we need to use more functions

```
/* USER CODE BEGIN 2 */
__HAL_TIM_ENABLE_DMA(&htim1, TIM_DMA_UPDATE);
HAL_DMA_Start(&hdma_tim1_up,(uint32_t)data,(uint32_t)&GPIOG>ODR,2);
HAL_TIM_Base_Start(&htim1);
/* USER CODE END 2 */
```

- We must use three HAL function to make it working
- They are also some situation where HAL functions not exists and we must use direct access into registers



Board Support Package (BSP)

Board support package Overview

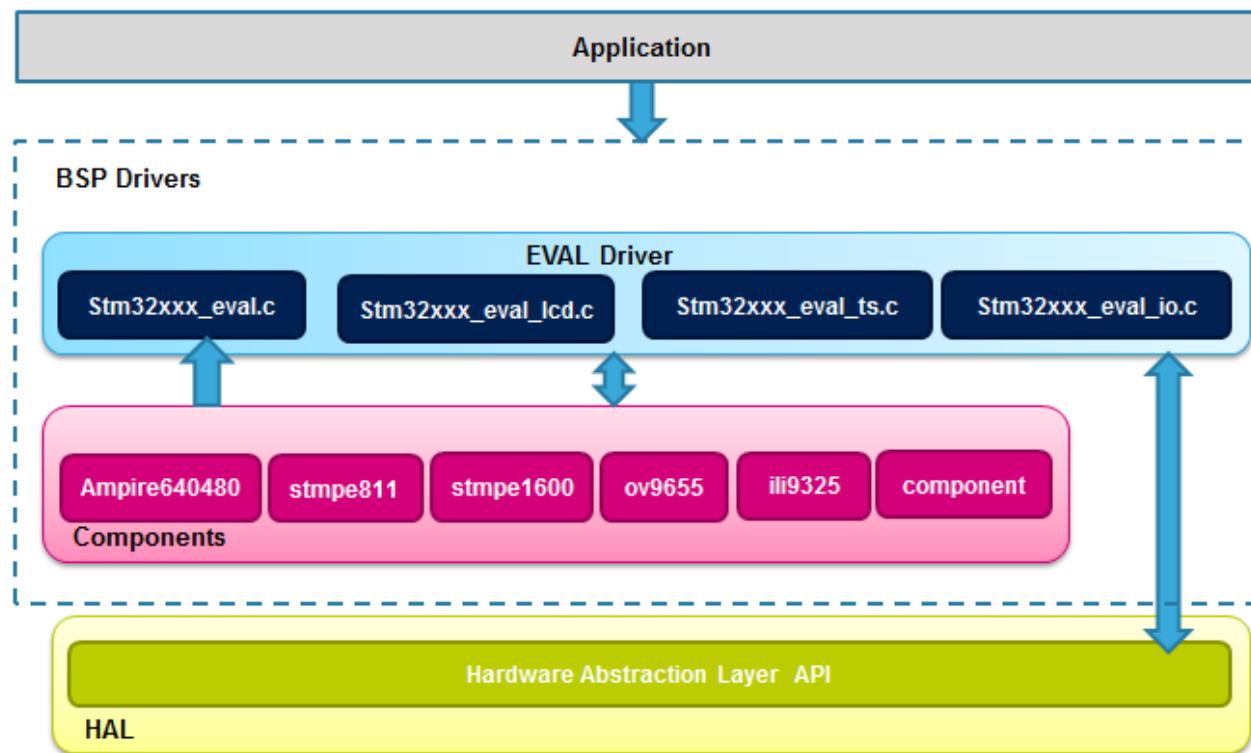
208

- A board support package is available per each product board (EVAL, Discovery, Nucleo)
- The board support package architecture was defined in order to allow easy portability and reuse of the same components (LCD, IO expander,...) on different boards
- This portability is achieved by defining component classes
 - LCD
 - Accelerometer
 - Gyroscope
 - Audio speaker codec
 - IO expander
 - Touch screen controller
- Drivers for the components present on the board are written according to the component class definition (see .h files in \Drivers\BSP\Components\Common) which is **board and MCU independent**
- A board support package is implemented by doing the link with a particular component driver and by associating to it a particular MCU peripheral and needed I/Os

Board support package Overview

209

- The BSP drivers are generally composed of two parts:
 - Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provide specific APIs to the external components and could be portable on any other board.
 - Board Driver: the Board driver permits to link the component driver to a specific board and provides a set of friendly used APIs.



Examples with BSP

210

- How to import BSP into your project
- How use BSP examples:
 - FMC SDRAM BSP lab
 - LCD BSP Print text lab
 - I2C BSP EEPROM lab
 - SPI BSP GYROSCOPE lab

