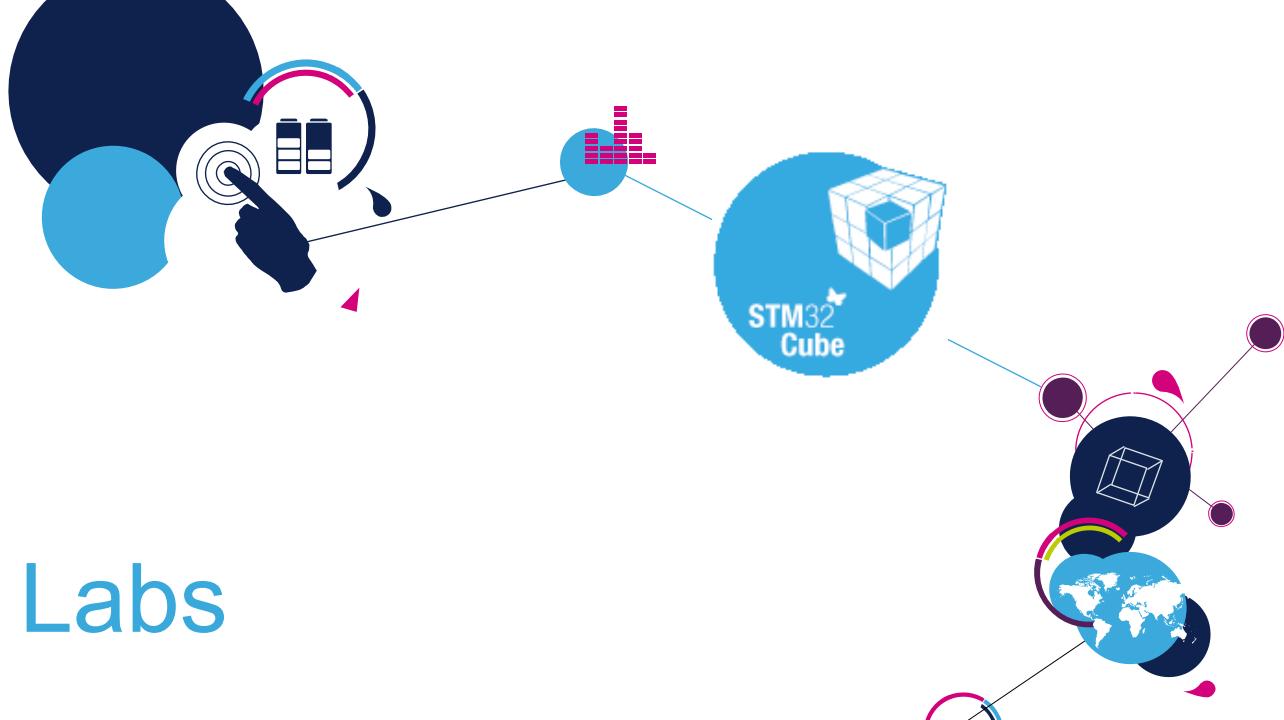
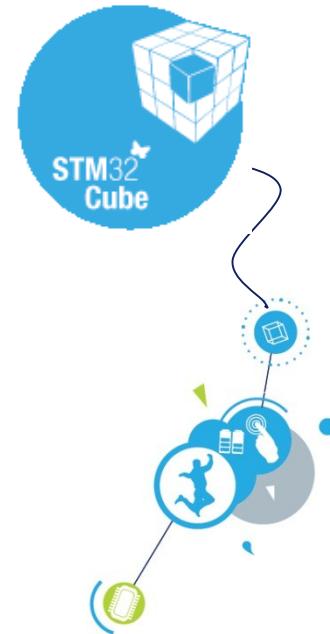


# STM32F4 Labs

T.O.M.A.S – Technically Oriented Microcontroller Application Services  
V1.07



1. GPIO lab
2. EXTI lab
3. SLEEP lab
4. STOP lab
5. STANDBY lab
6. DMA Poll lab
7. DMA Interrupt lab
8. RTC Alarm lab
9. UART Poll lab
10. UART Interrupt lab
11. UART DMA lab

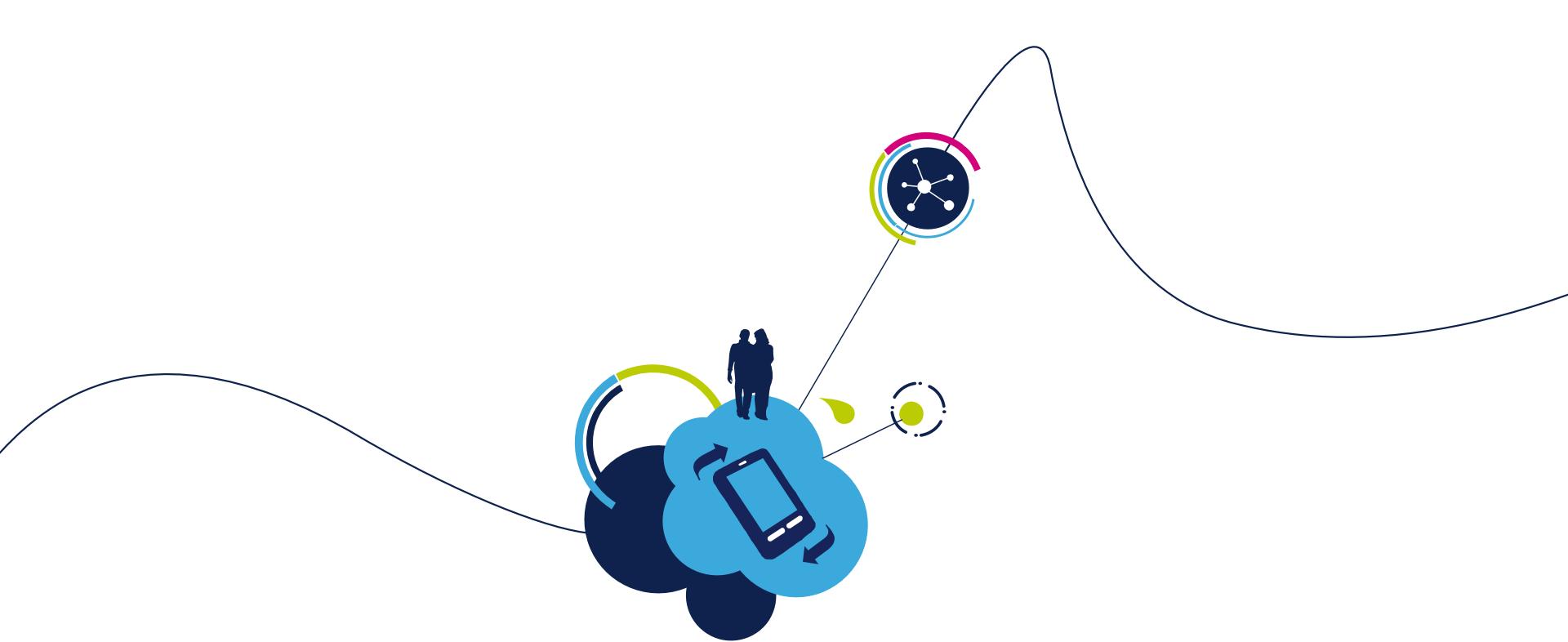


- 12. SPI Poll lab
- 13. SPI Interrupt lab
- 14. SPI DMA lab
- 15. TIM Interrupt lab
- 16. TIM PWM out lab
- 17. TIM DMA lab
- 18. TIM Counter lab
- 19. DAC wave generation lab
- 20. ADC Poll lab
- 21. ADC Interrupt lab
- 22. ADC DMA lab



- 23. WWDG lab
- 24. IWDG lab
- 25. FMC SDRAM BSP lab
- 26. LCD BSP Print text lab
- 27. I2C BSP EEPROM lab
- 28. SPI BSP GYROSCOPE lab





# GPIO Lab 1

- Objective

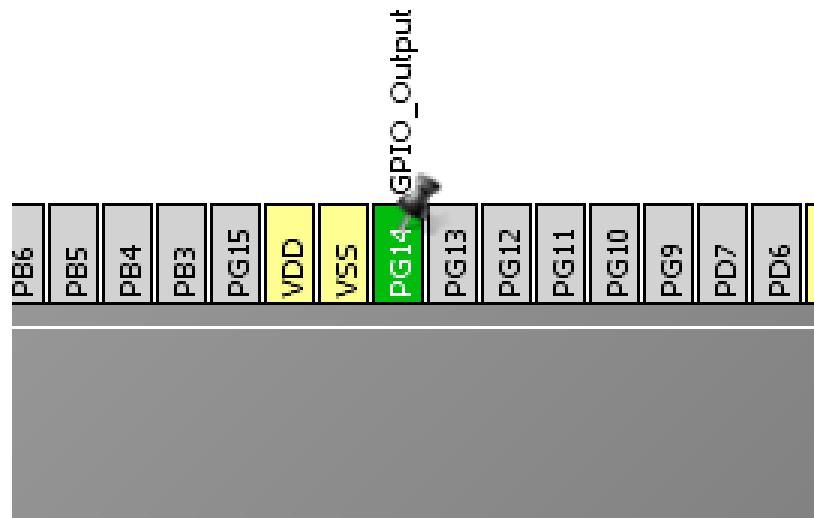
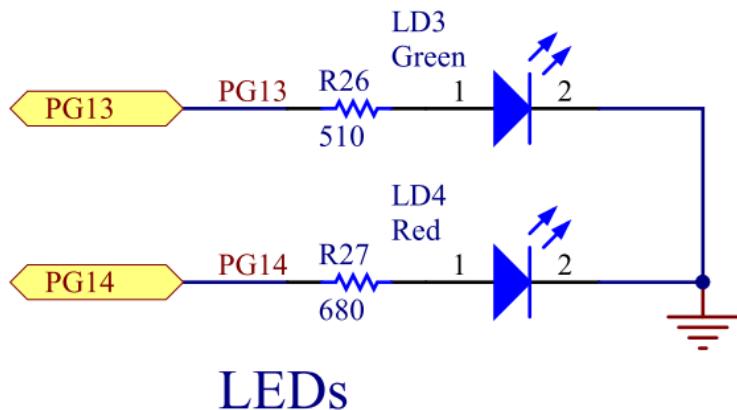
- Learn how to setup pin and GPIO port in CubeMX
  - How to Generate Code in CubeMX and use HAL functions

- Goal

- Configure GPIO pin in CubeMX and Generate Code
  - Add in to project HAL\_Delay function and HAL\_GPIO\_Toggle function
  - Verify the correct functionality on toggling LED

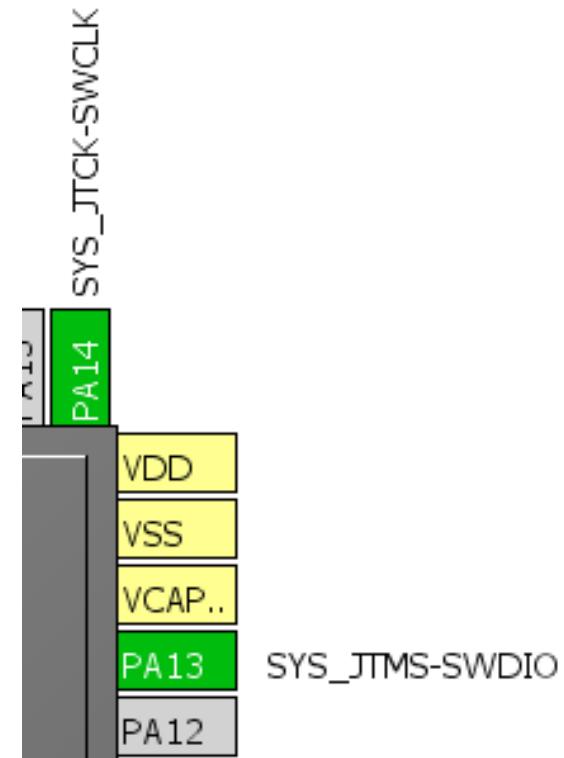
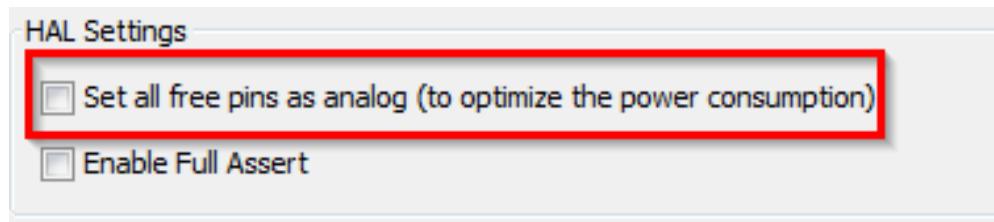
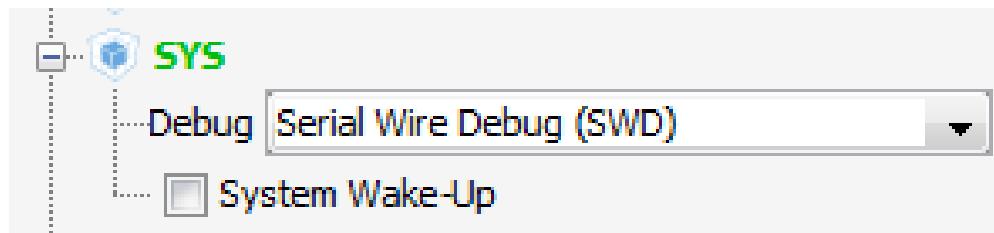
# Configure GPIO for LED toggling

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Configure LED pin as GPIO\_Output



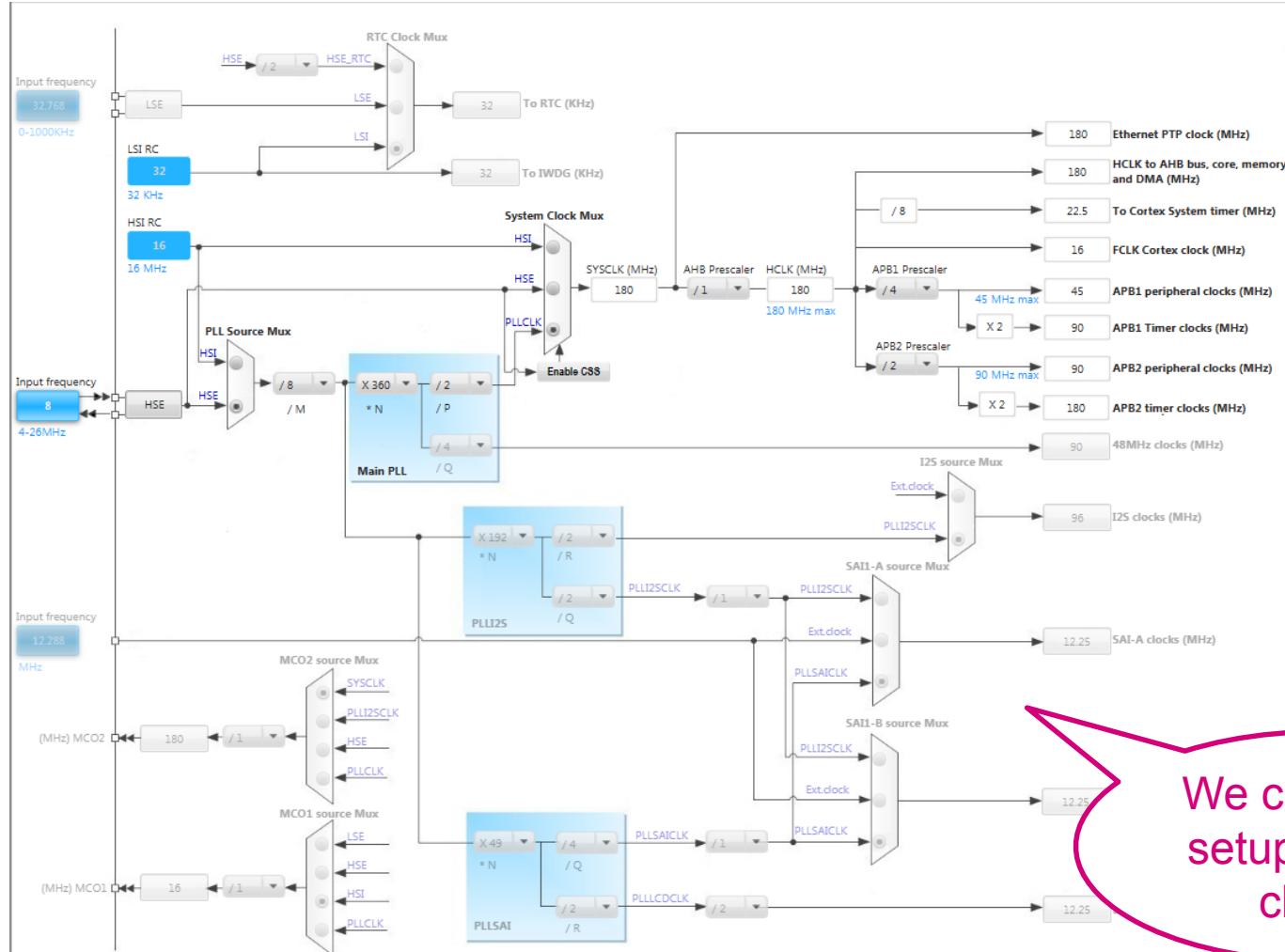
# Configure GPIO for LED toggling

- For debug purpose is recommended to select debug pins SWD or JTAG
  - Select can be done in TAB>Pinout>SYS
  - On discovery is available only SWD option
  - If SWD/JTAG is not selected and the Set all free pins as analog (MENU>Project>Settings>TAB>Code Generator) is selected, debug is not possible



# Configure GPIO for LED toggling

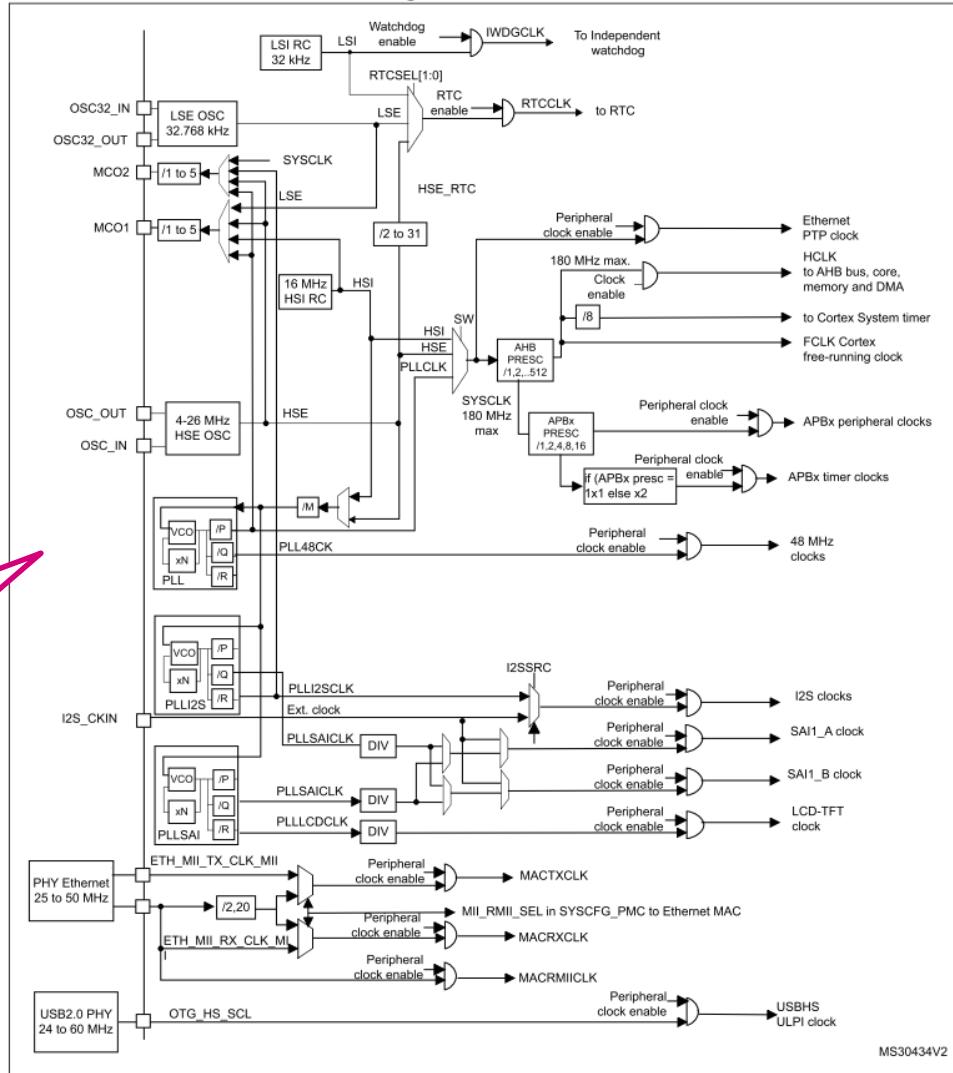
- Clock Configuration
  - TAB>Clock Configuration



# Configure GPIO for LED toggling

- The Clock configuration tree is interactive version of tree from RM

Figure 16. Clock tree



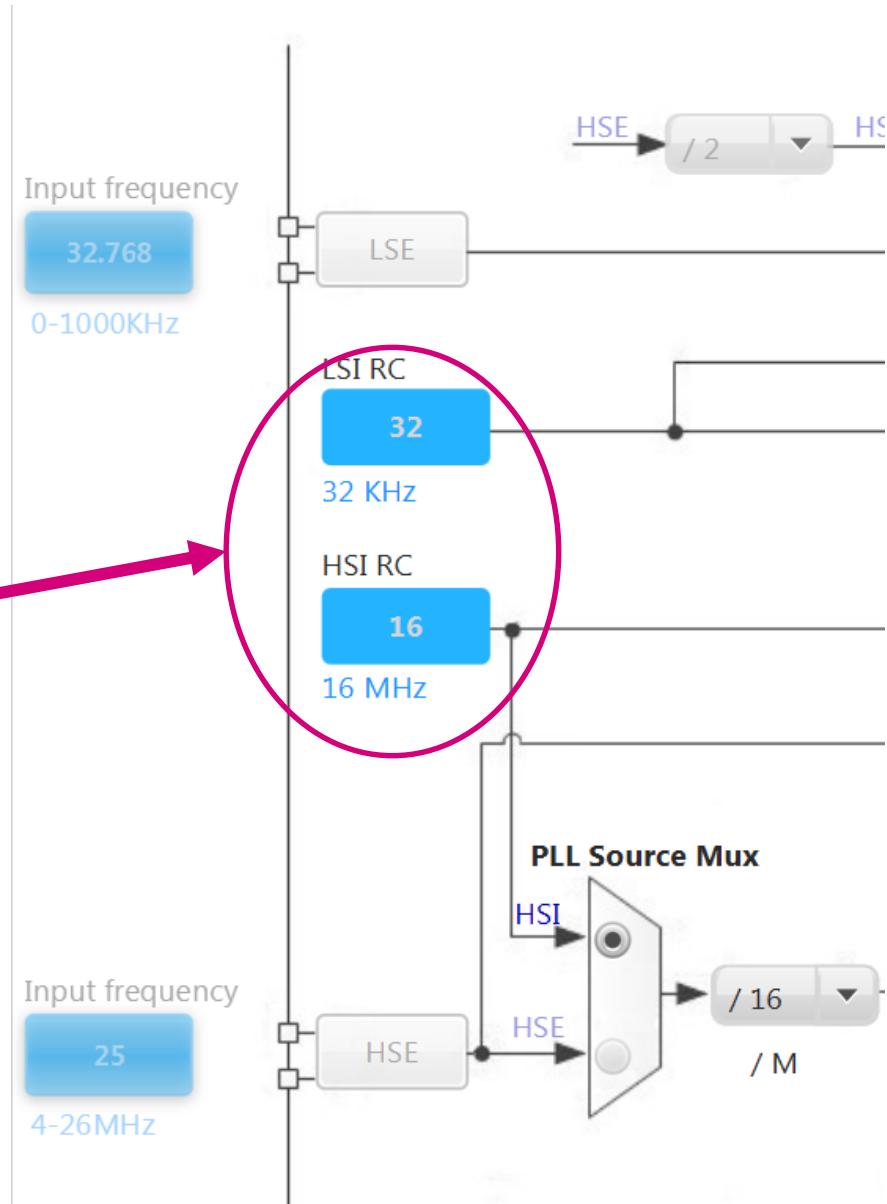
RM0090 Chapter 6  
Reset and clock  
control  
Page 151

# Configure GPIO for LED toggling

## Clock Configuration overview 1

- Clock sources
  - Internal oscillators

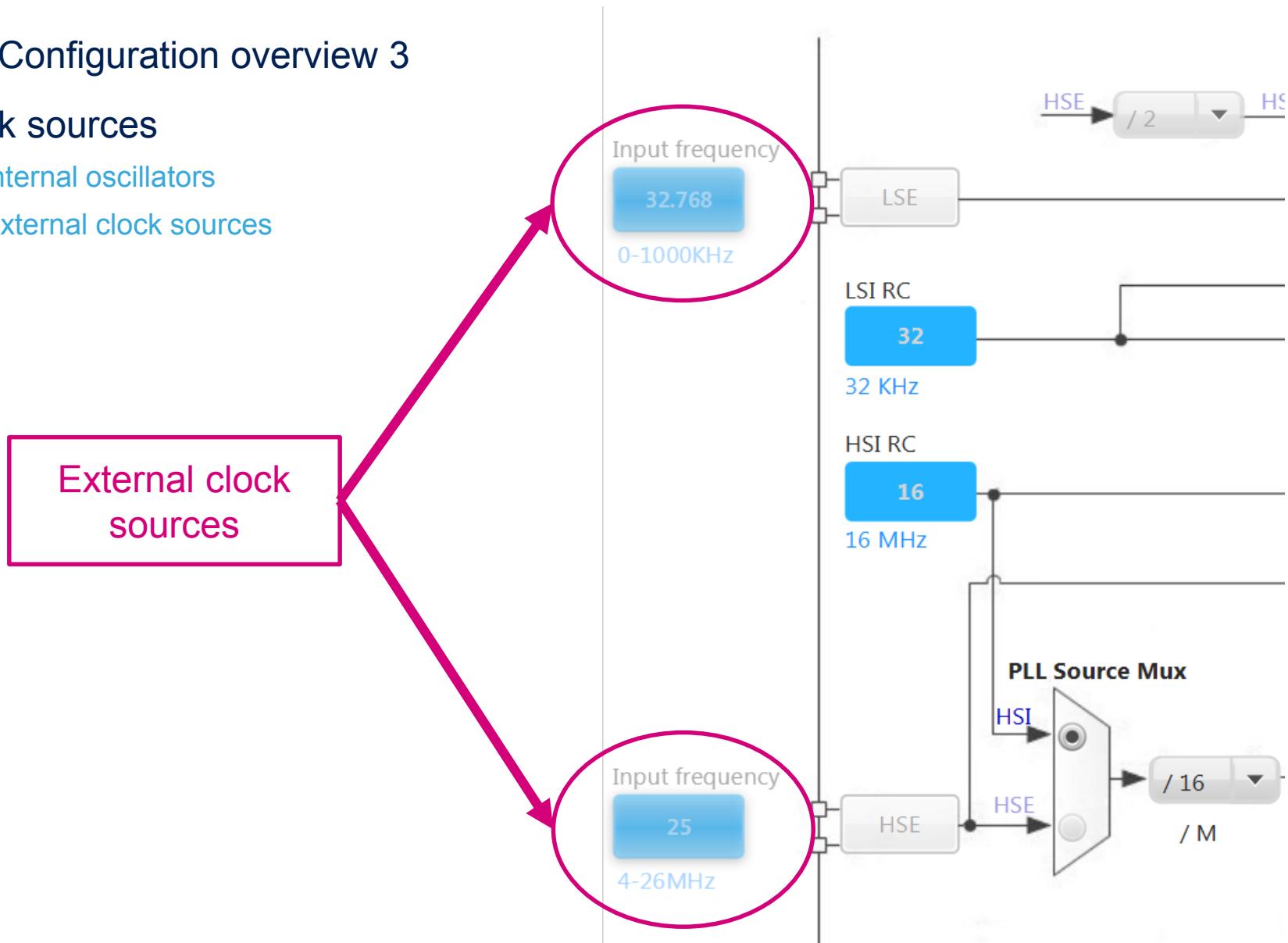
Internal  
oscillators



# Configure GPIO for LED toggling

## Clock Configuration overview 3

- Clock sources
  - Internal oscillators
  - External clock sources



## Clock Configuration overview 4

- Clock sources
  - Internal oscillators
    - LSI
  - External clock sources

## Low-speed internal (LSI) RC oscillator

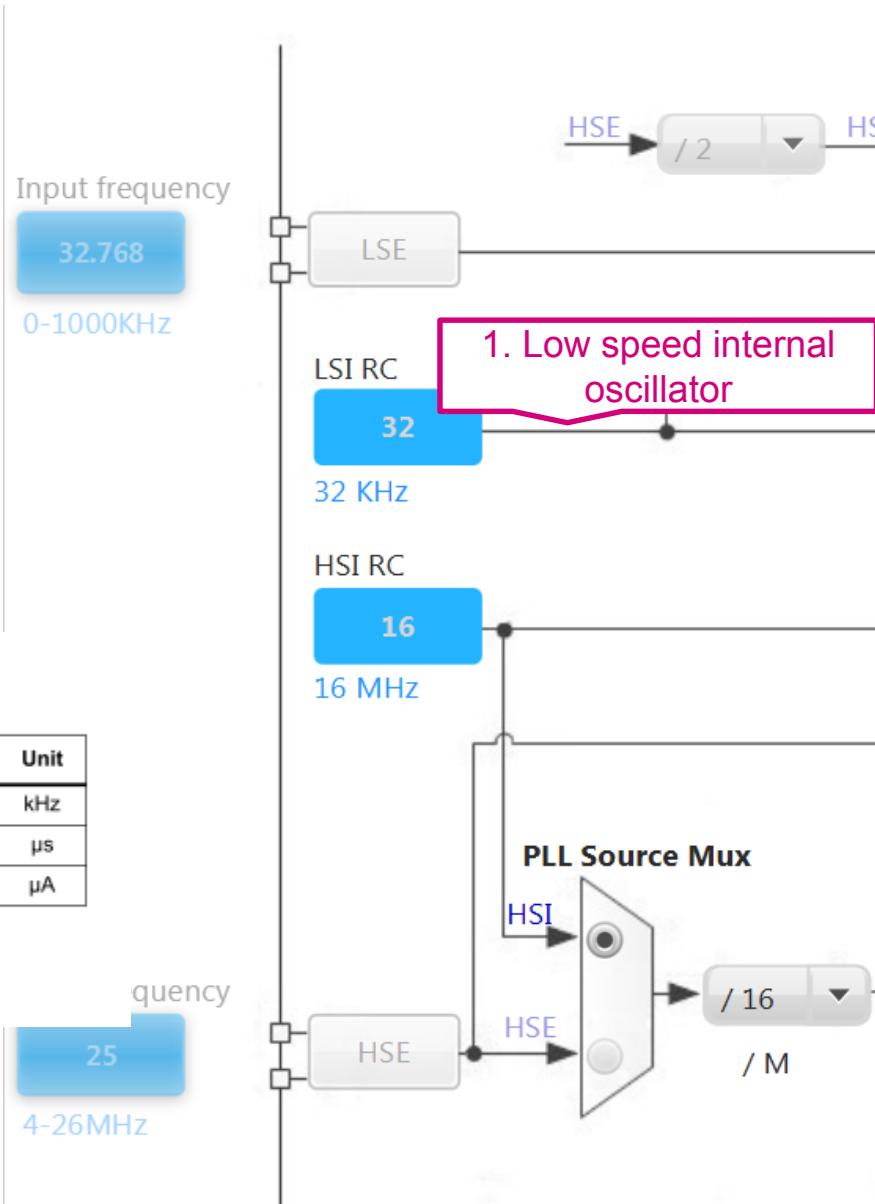
Table 42. LSI oscillator characteristics <sup>(1)</sup>

Symbol	Parameter	Min	Typ	Max	Unit
$f_{LSI}^{(2)}$	Frequency	17	32	47	kHz
$t_{su(LSI)}^{(3)}$	LSI oscillator startup time	-	15	40	$\mu s$
$I_{DD(LSI)}^{(3)}$	LSI oscillator power consumption	-	0.4	0.6	$\mu A$

1.  $V_{DD} = 3 V$ ,  $T_A = -40$  to  $105^\circ C$  unless otherwise specified.

2. Based on characterization, not tested in production.

3. Guaranteed by design, not tested in production.



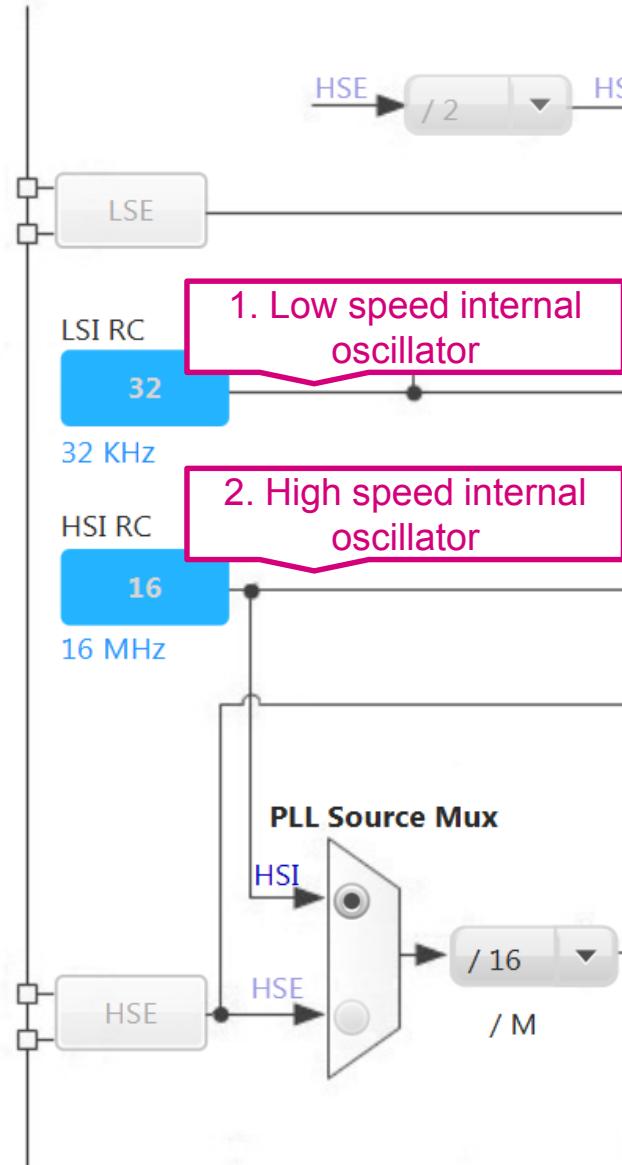
## Clock Configuration overview 5

## • Clock sources

- Internal oscillators
  - LSI
  - HSI
- External clock sources

Input frequency  
32.768  
0-1000KHz

frequency  
25  
4-26MHz

Table 41. HSI oscillator characteristics <sup>(1)</sup>

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
$f_{HSI}$	Frequency			-	16	-	MHz
ACC <sub>HSI</sub>	Accuracy of the HSI oscillator	User-trimmed with the RCC_CR register <sup>(2)</sup>		-	-	1	%
		Factory-calibrated	$T_A = -40 \text{ to } 105^\circ\text{C}$ <sup>(3)</sup>	-8	-	4.5	%
			$T_A = -10 \text{ to } 85^\circ\text{C}$ <sup>(3)</sup>	-4	-	4	%
			$T_A = 25^\circ\text{C}$	-1	-	1	%
$t_{su(HSI)}$ <sup>(2)</sup>	HSI oscillator startup time			-	2.2	4	$\mu\text{s}$
$I_{DD(HSI)}$ <sup>(2)</sup>	HSI oscillator power consumption			-	60	80	$\mu\text{A}$

1.  $V_{DD} = 3.3 \text{ V}$ ,  $T_A = -40 \text{ to } 105^\circ\text{C}$  unless otherwise specified.

2. Guaranteed by design, not tested in production

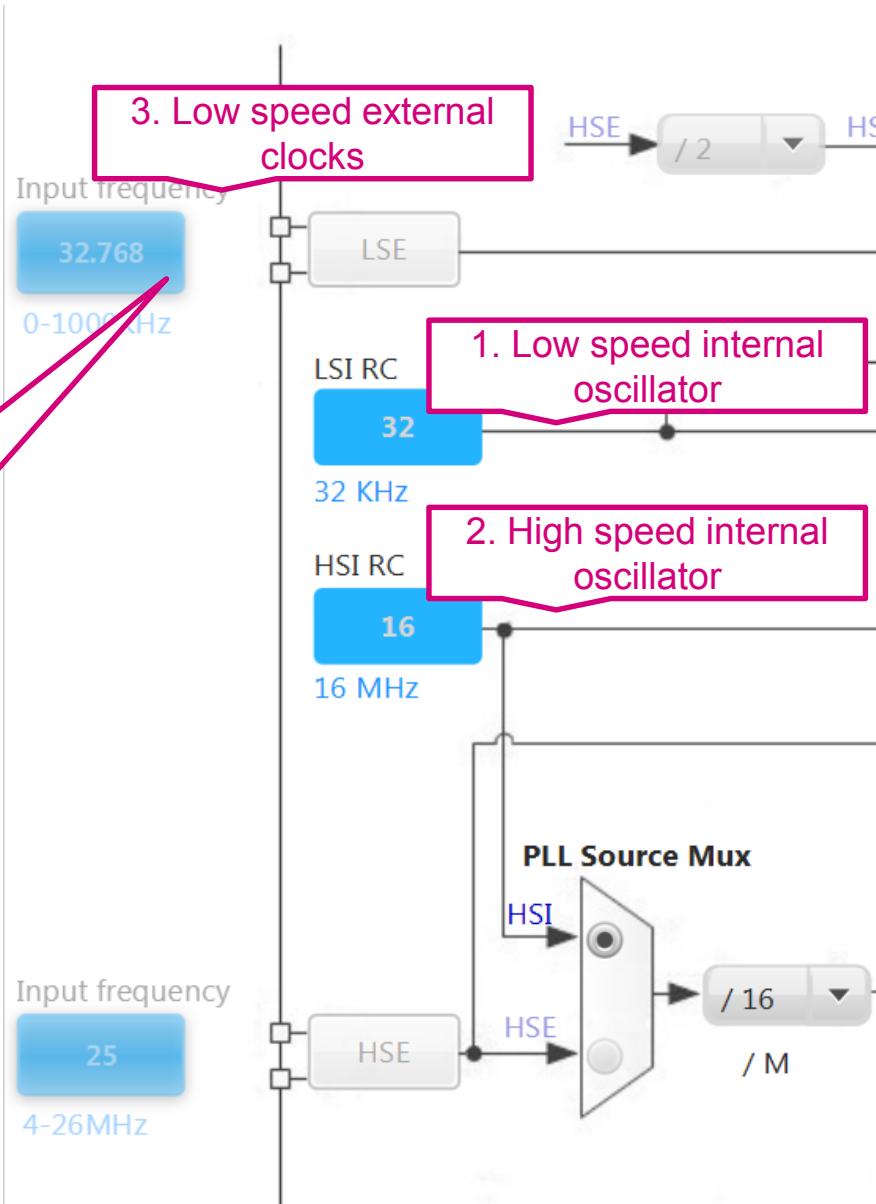
3. Based on characterization, not tested in production.

# Configure GPIO for LED toggling

## Clock Configuration overview 6

- Clock sources
  - Internal oscillators
    - LSI
    - HSI
  - External clock sources
    - LSE

External 32,768Hz crystal or external signal in bypass mode

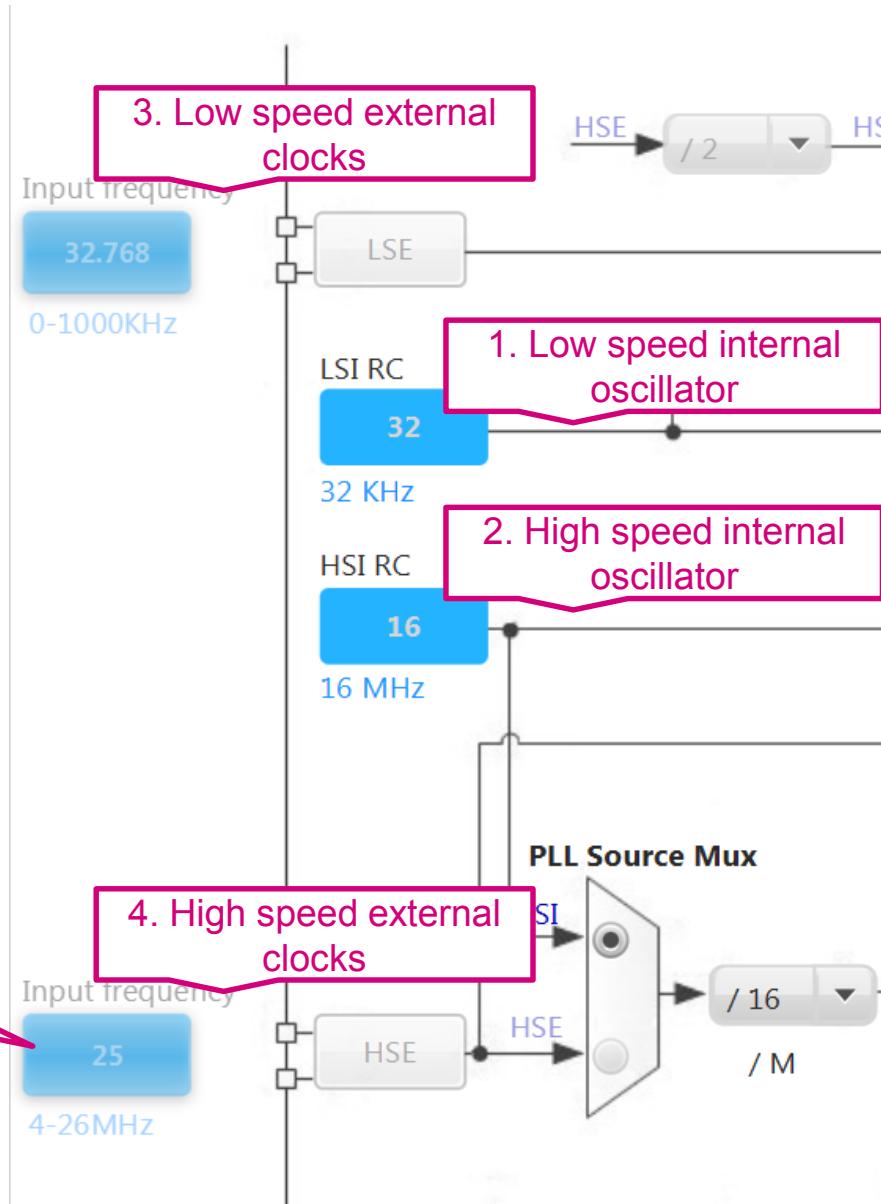


# Configure GPIO for LED toggling

## Clock Configuration overview 7

- Clock sources
  - Internal oscillators
    - LSI
    - HSI
  - External clock sources
    - LSE – crystal or external signal in bypass mode
    - HSE

External crystal  
4-26MHz or  
external signal 1-50Mhz  
in bypass mode



# Configure GPIO for LED toggling

## Clock Configuration overview 8

- Clock sources

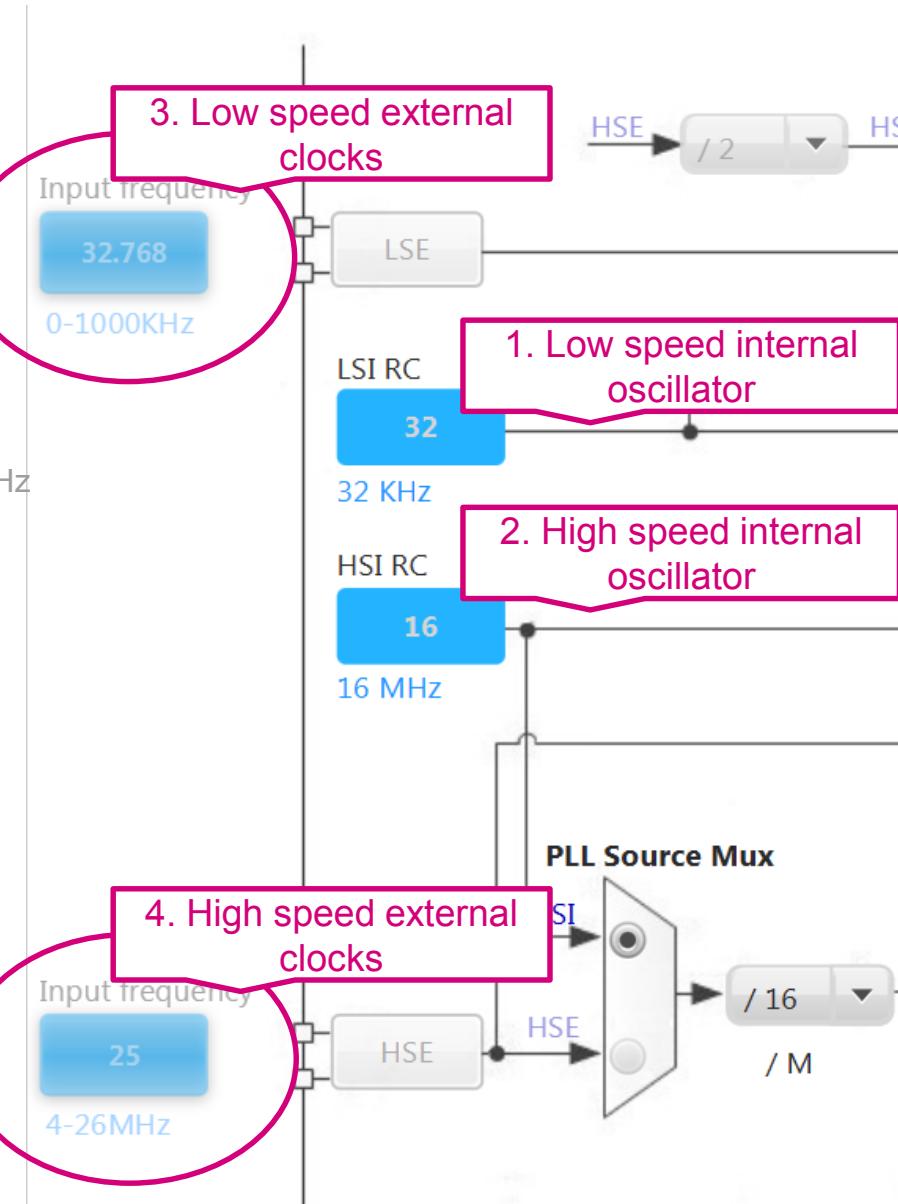
- Internal oscillators

- LSI
    - HSI

- External clock sources

- LSE – crystal or external signal in bypass mode
    - HSE – crystal 4-26MHz or external signal 1-50MHz in bypass mode

Disabled clocks  
How to enable?



# Configure GPIO for LED toggling

## Clock Configuration overview 9

- Clock sources

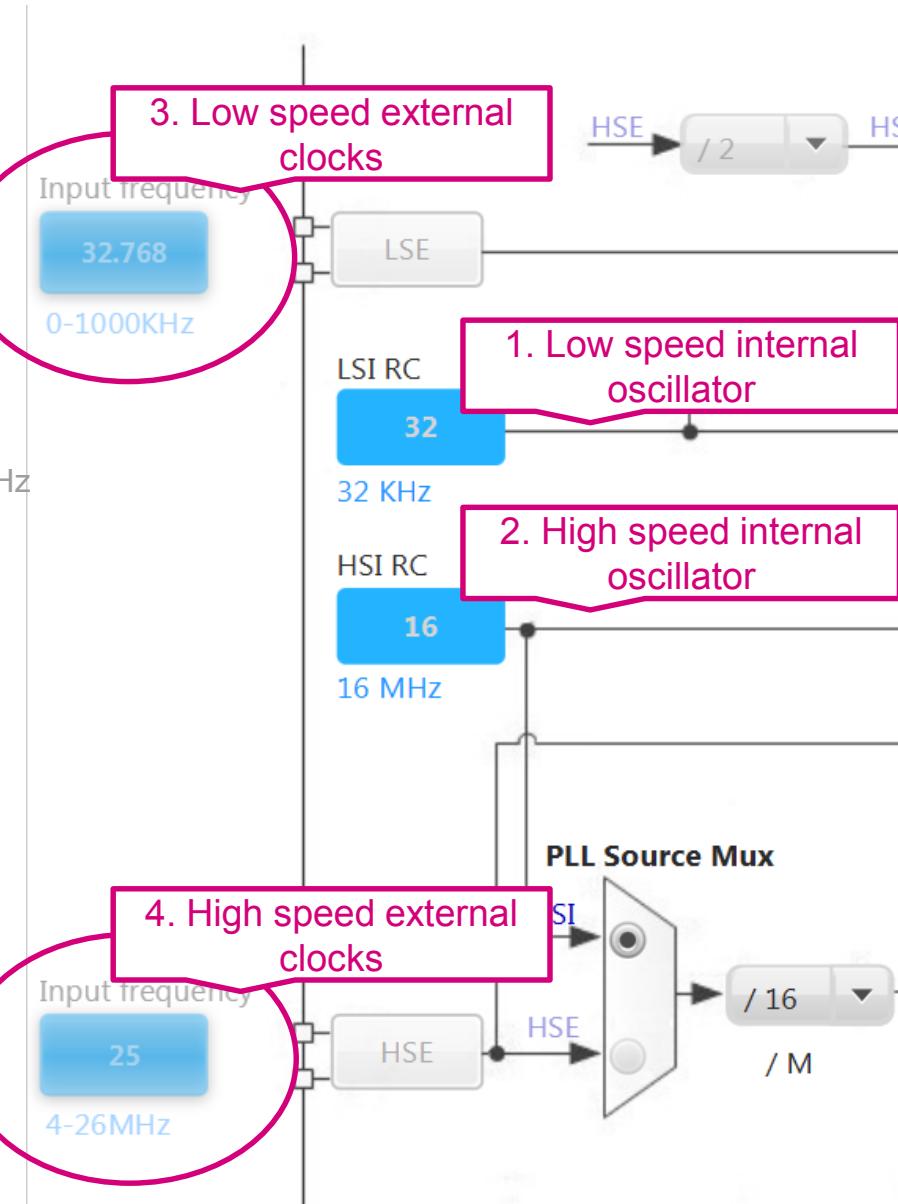
- Internal oscillators

- LSI
    - HSI

- External clock sources

- LSE – crystal or external signal in bypass mode
    - HSE – crystal 4-26MHz or external signal 1-50MHz in bypass mode

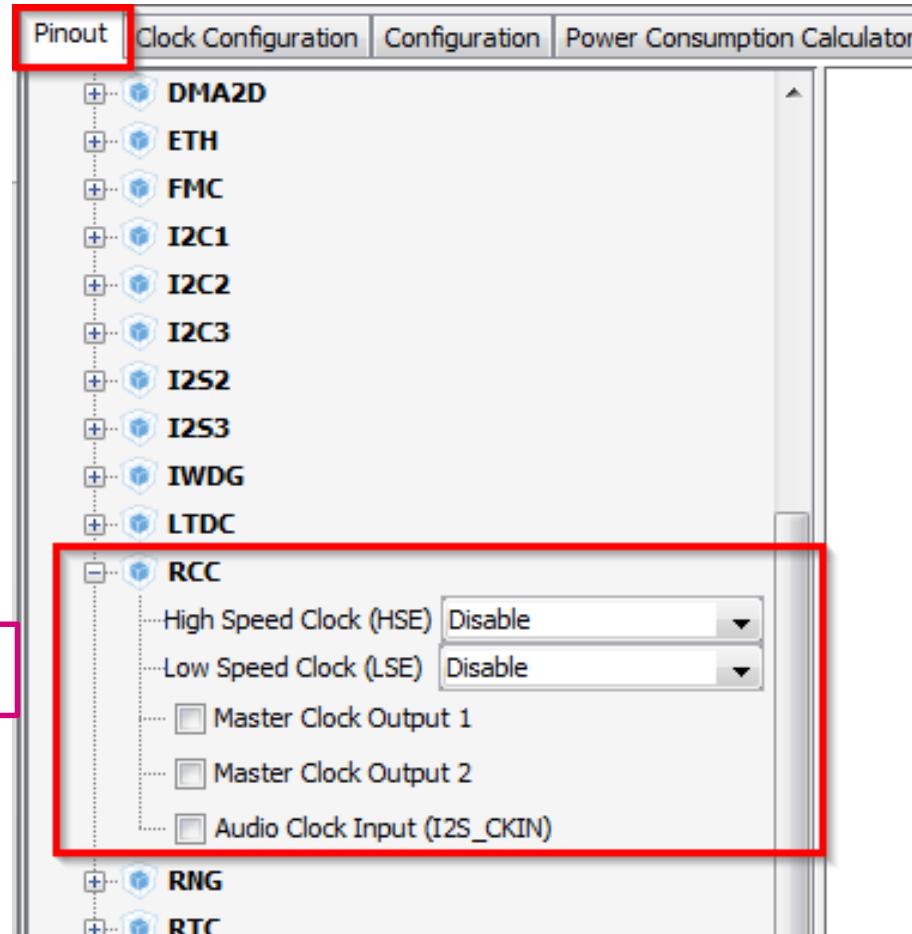
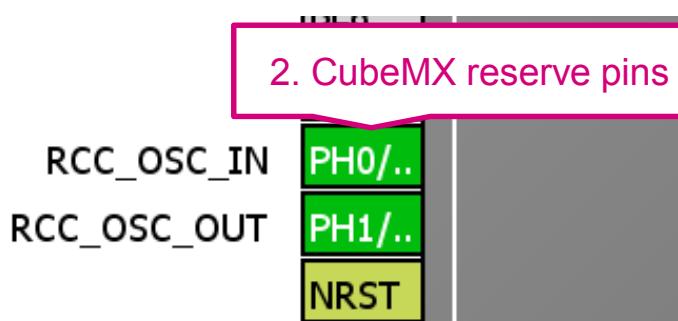
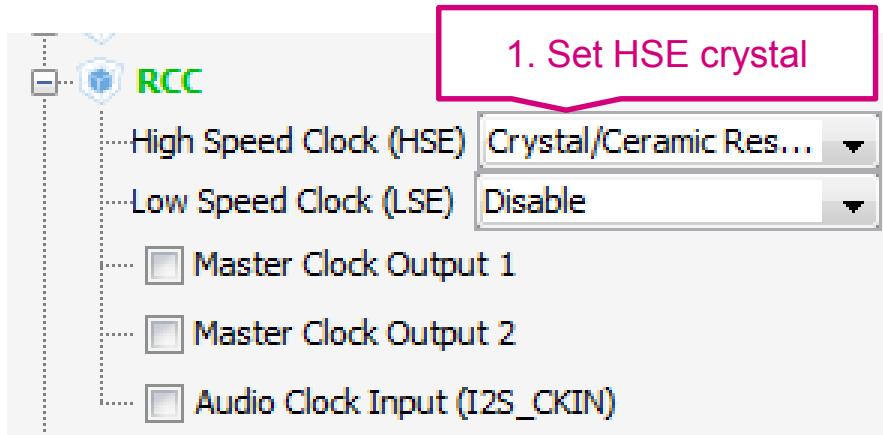
Disabled clocks  
How to enable?



# Configure GPIO for LED toggling

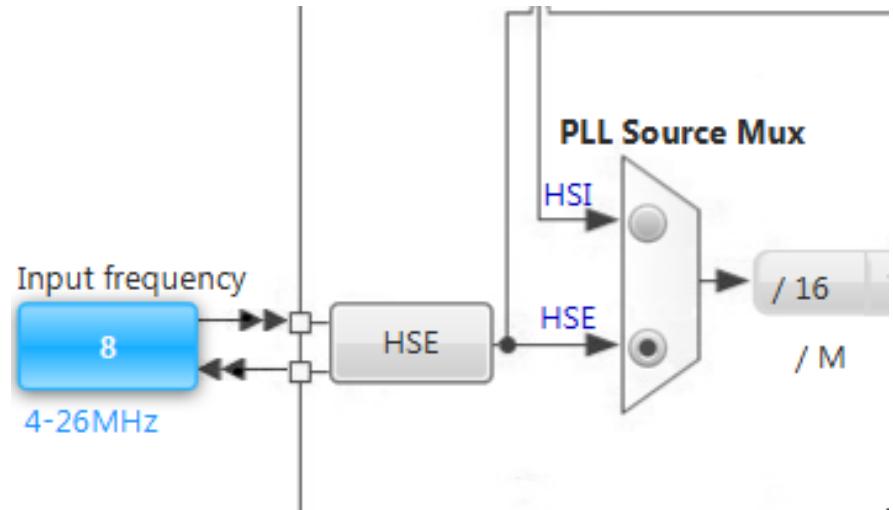
## Clock Configuration overview 10

- External clock enabling
  - TAB>Pinout
  - Select HSE and LSE clocks
    - Bypass or crystal



## Clock Configuration overview 11

- The HSE is now available
  - TAB>Clock configuration
  - Click on Blue square and change frequency on 8MHz

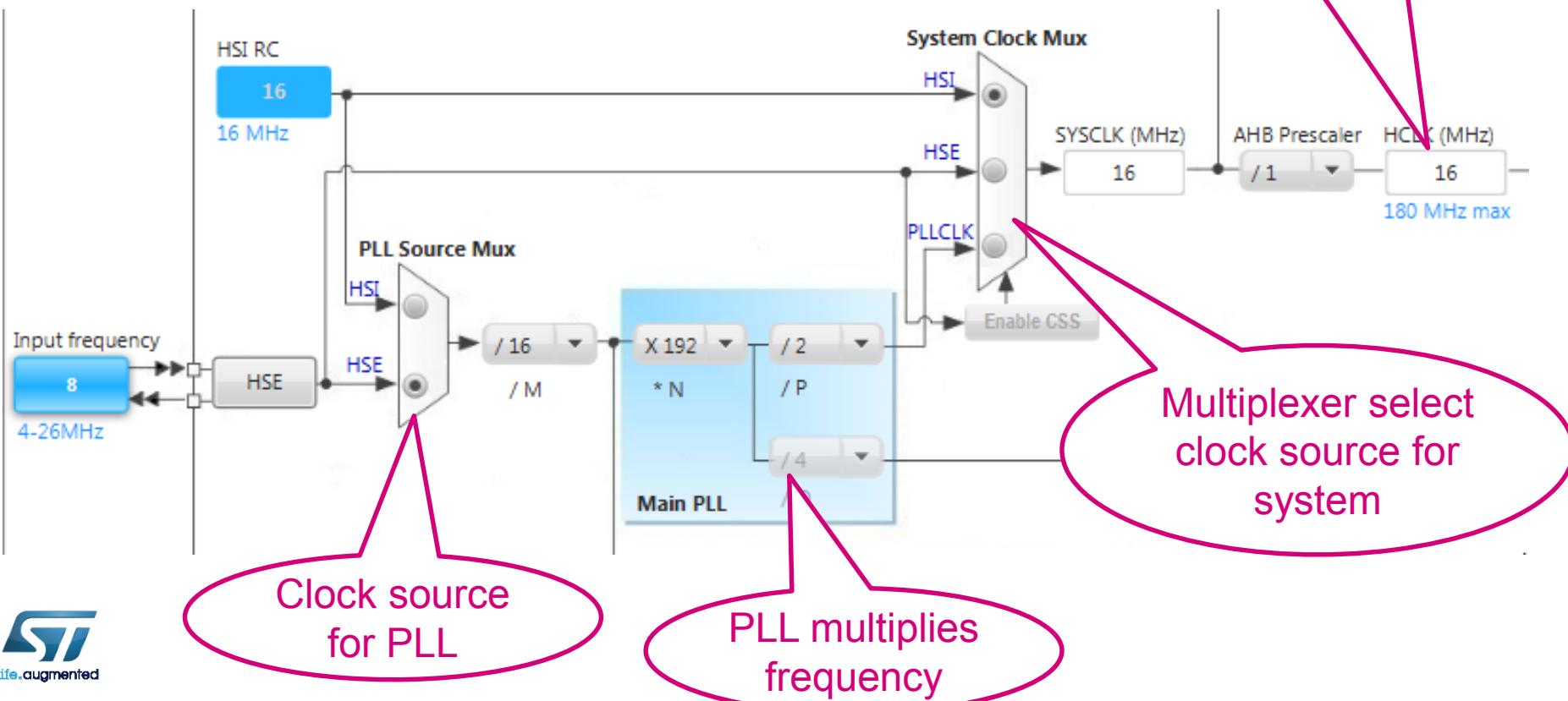


# Configure GPIO for LED toggling

Clock Configuration overview 12

Clock tree for core description

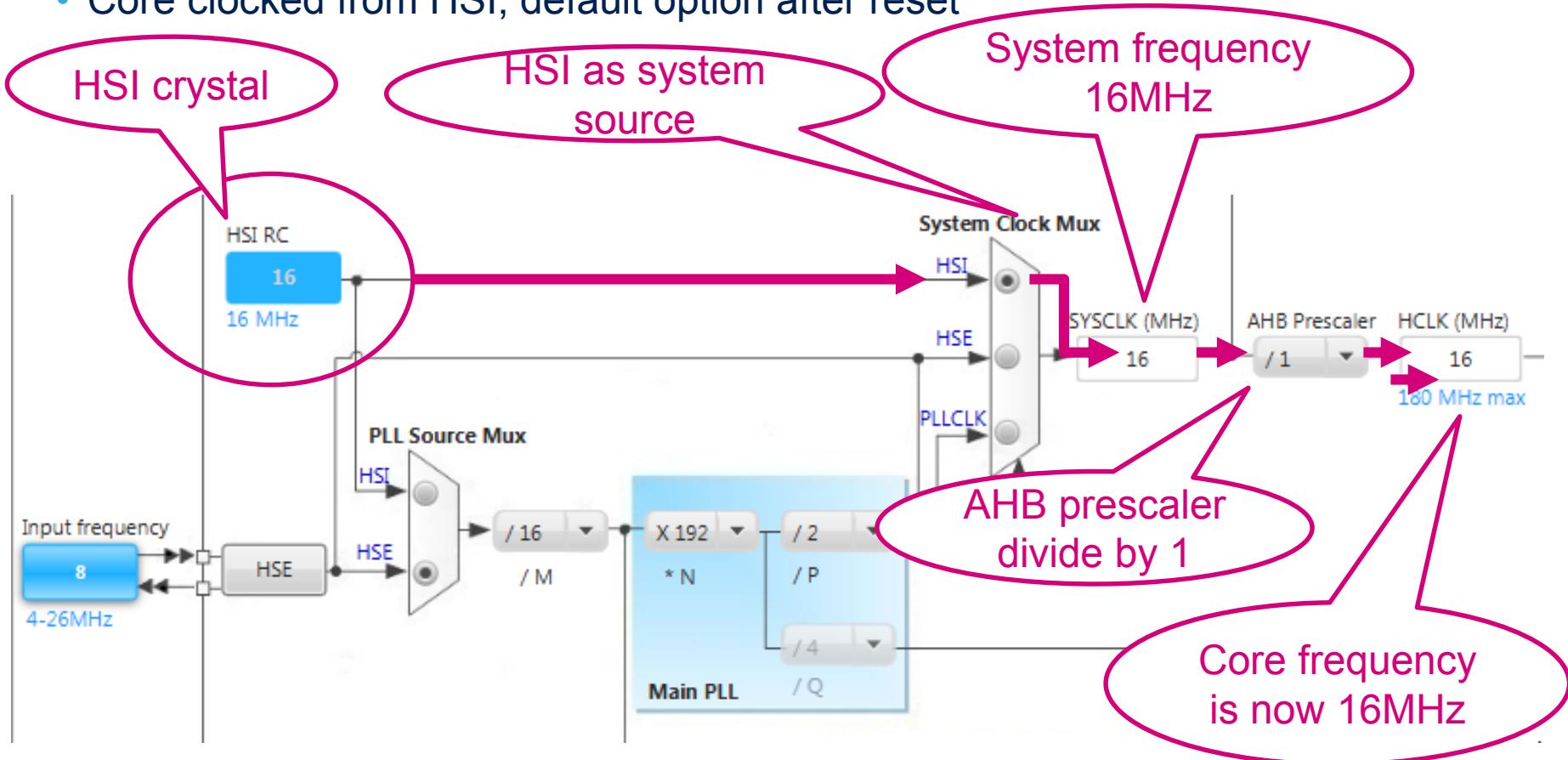
- System multiplexer
- PLL
- PLL source multiplexer



# Configure GPIO for LED toggling

## Clock Configuration overview 13

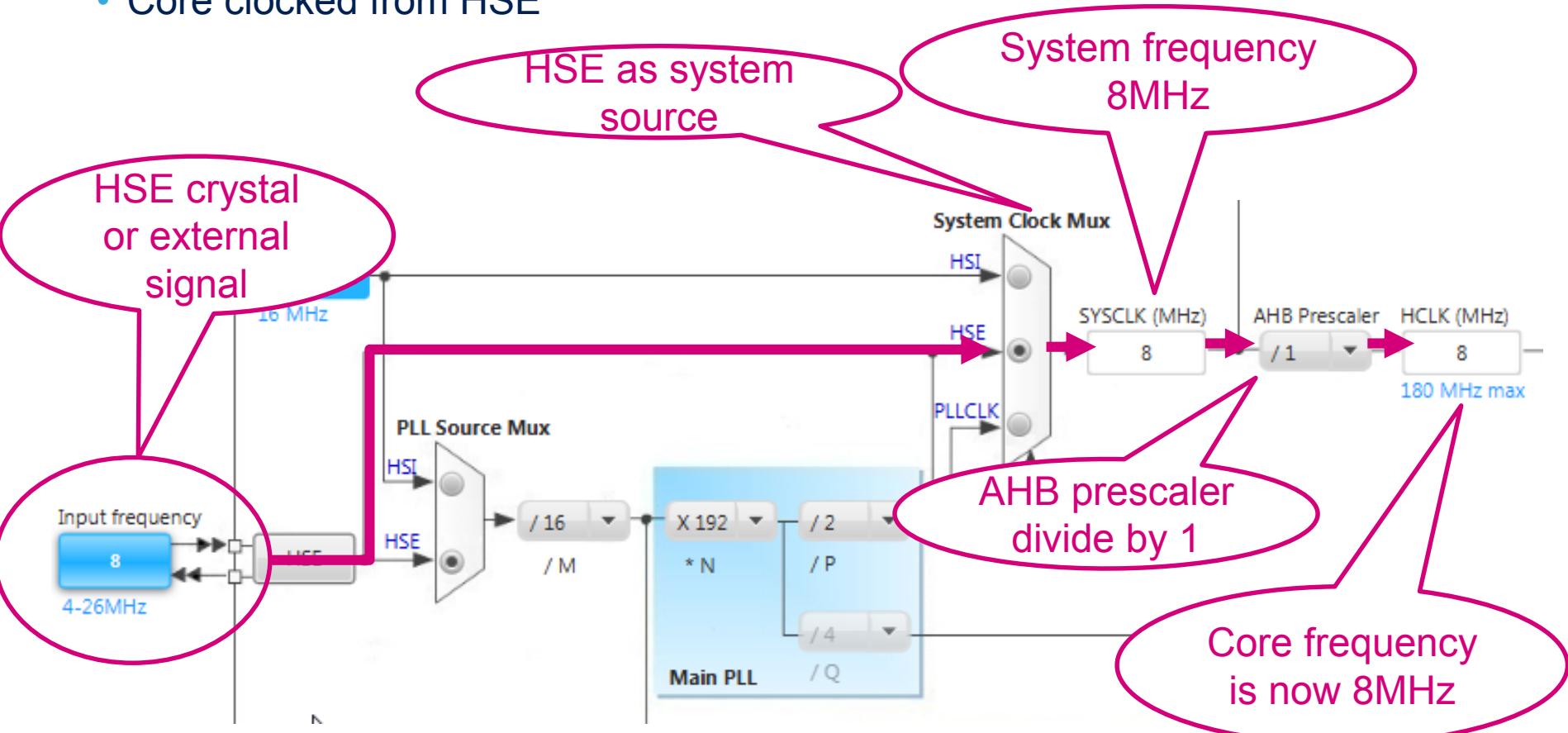
- Core clocked from HSI, default option after reset



# Configure GPIO for LED toggling

## Clock Configuration overview 14

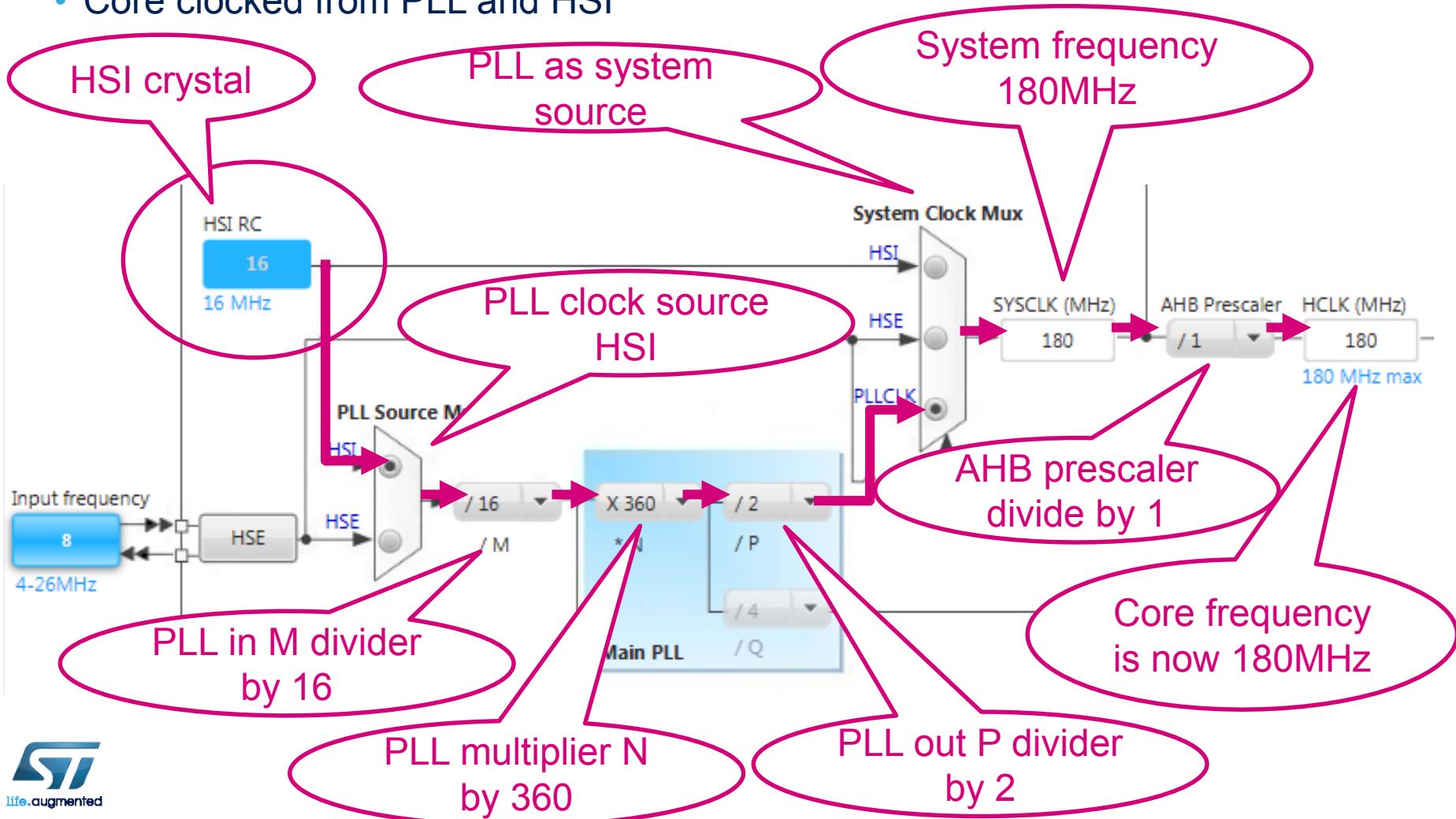
- Core clocked from HSE



# Configure GPIO for LED toggling

## Clock Configuration overview 15

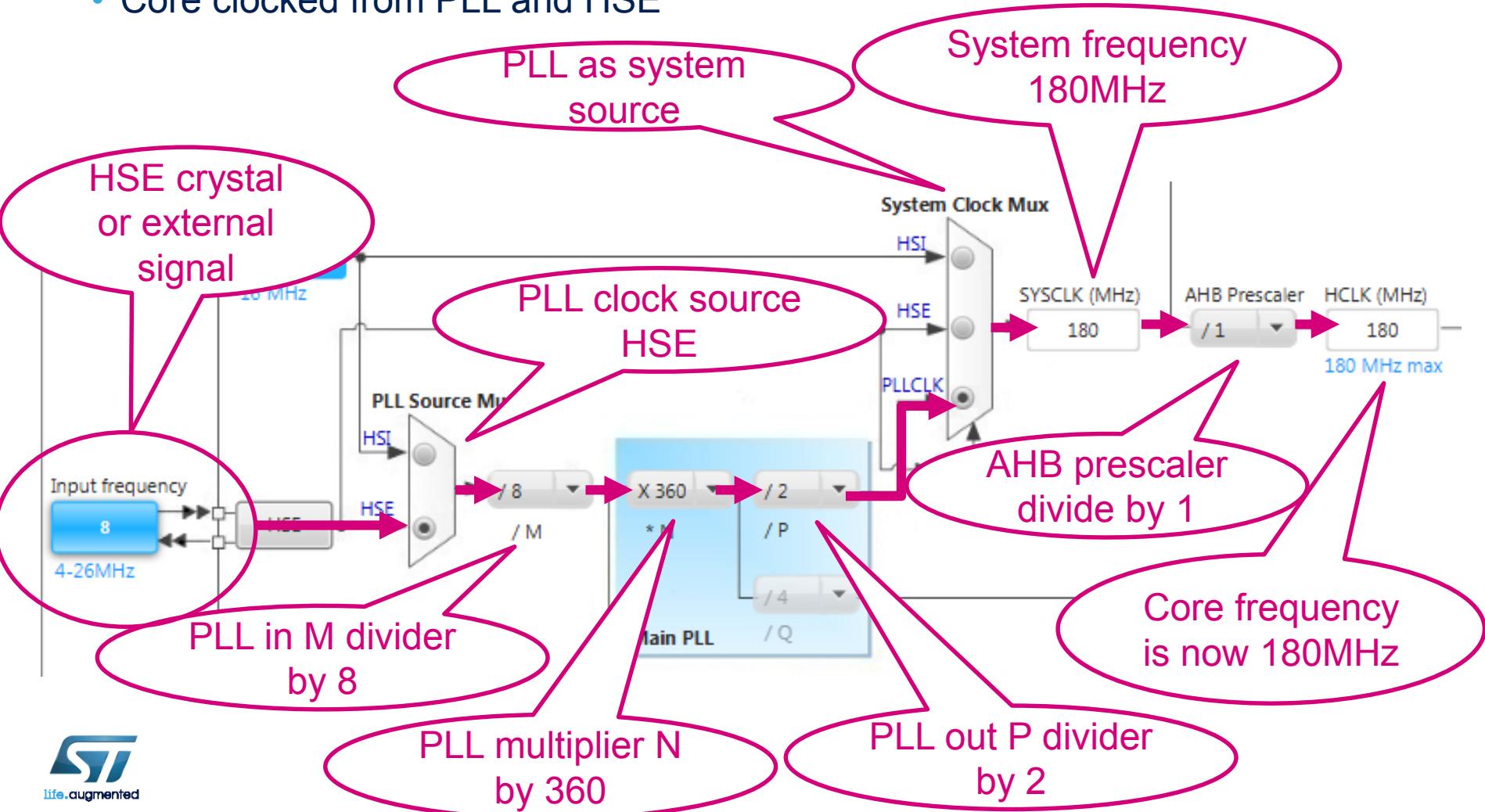
- Core clocked from PLL and HSI



# Configure GPIO for LED toggling

## Clock Configuration overview 16

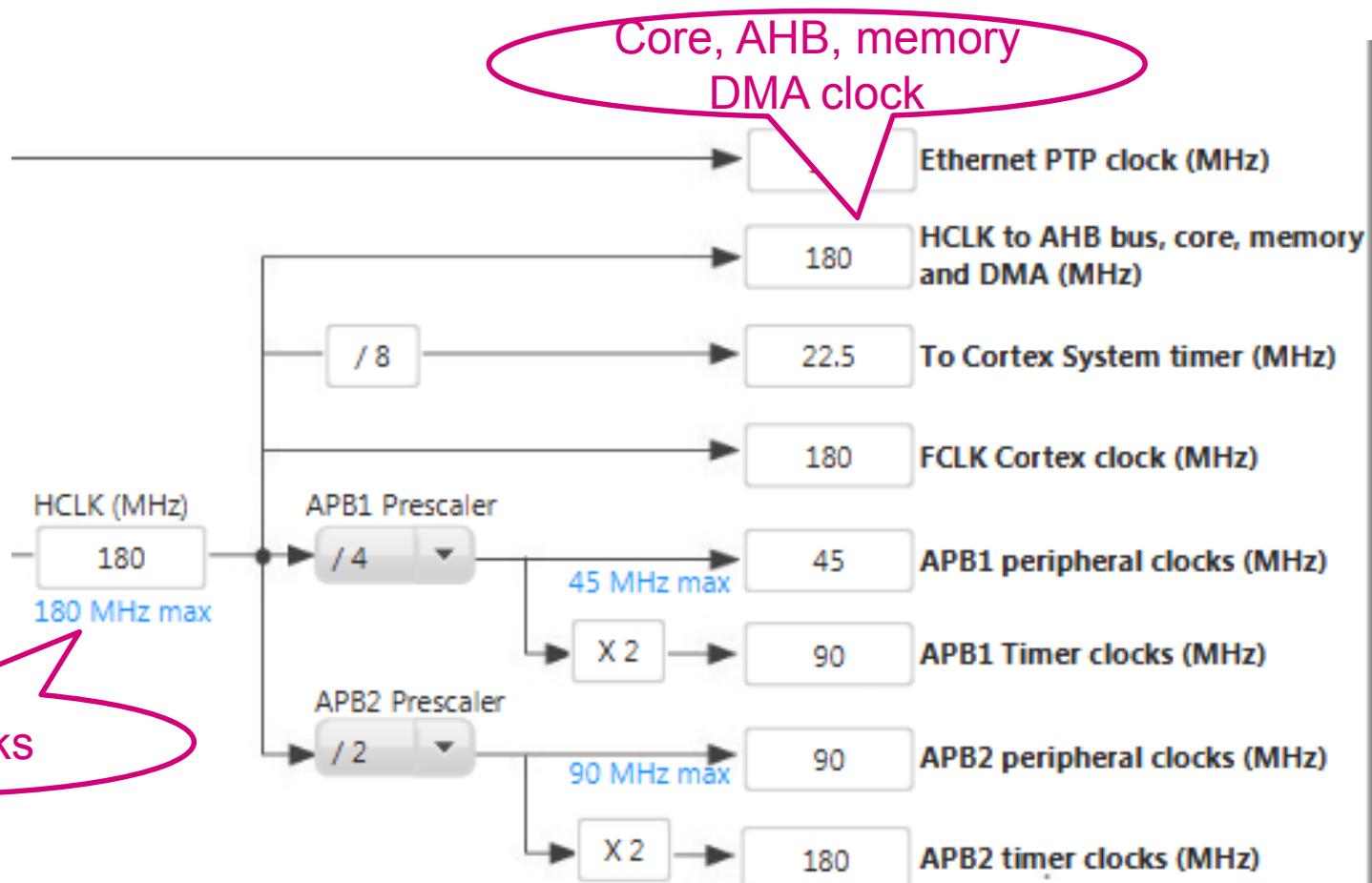
- Core clocked from PLL and HSE



# Configure GPIO for LED toggling

## Clock Configuration overview 17

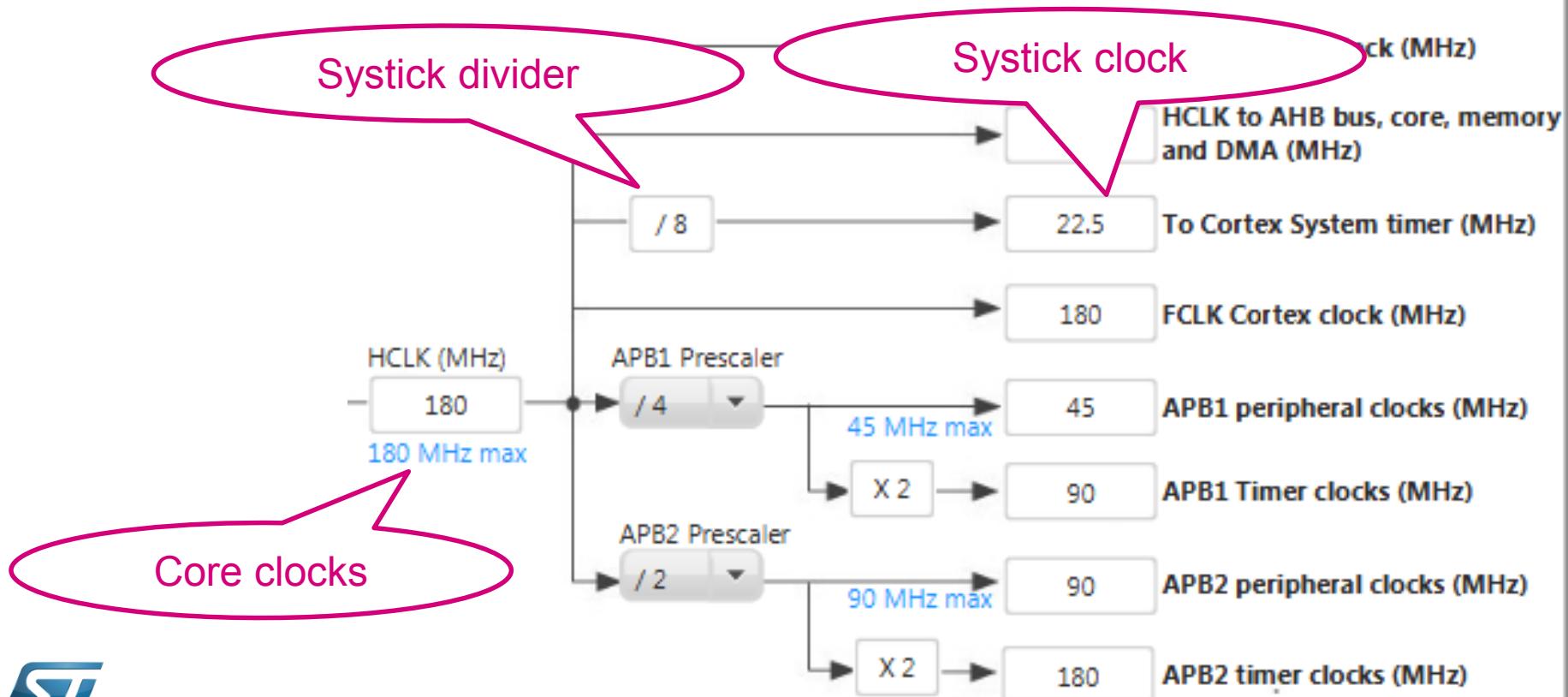
- AHB, APB prescalers and peripheral speed



# Configure GPIO for LED toggling

Clock Configuration overview 18

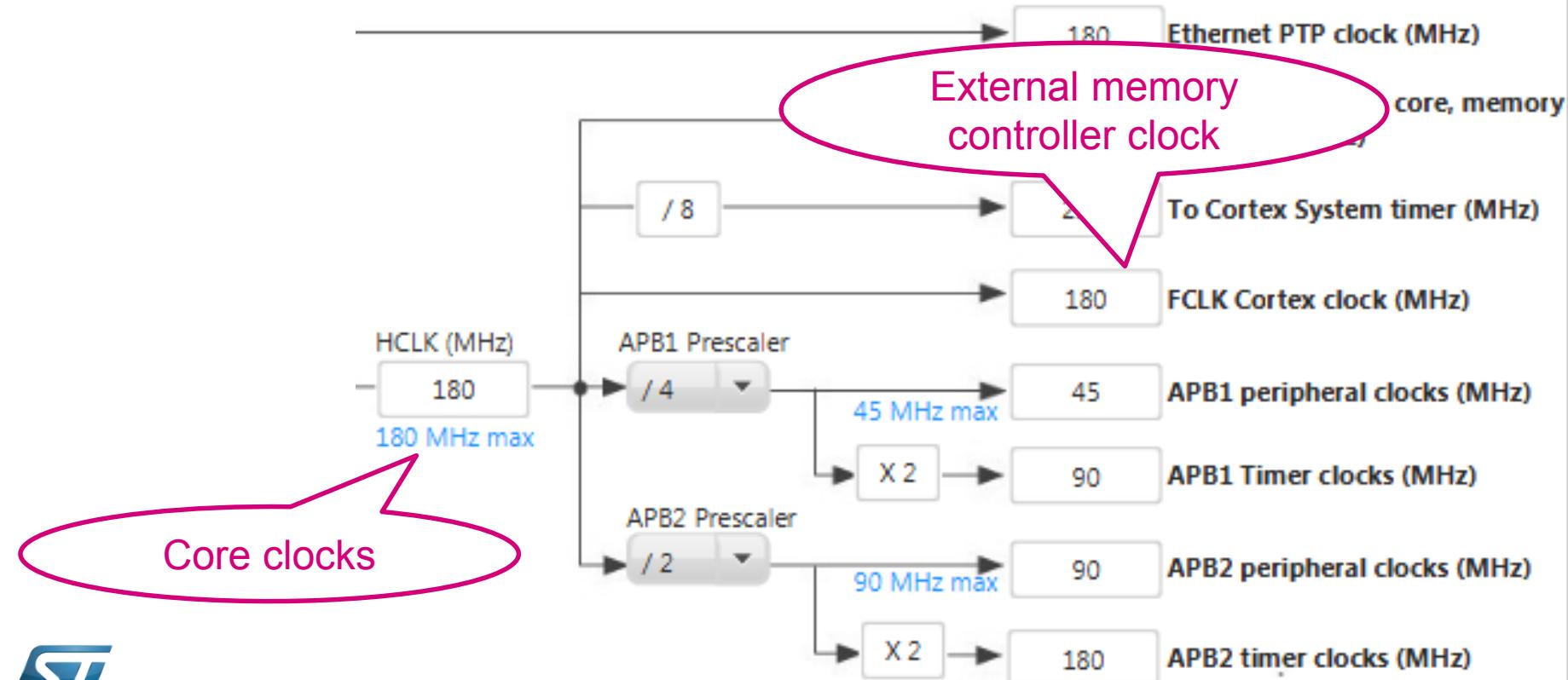
- AHB, APB prescalers and peripheral speed



# Configure GPIO for LED toggling

Clock Configuration overview 19

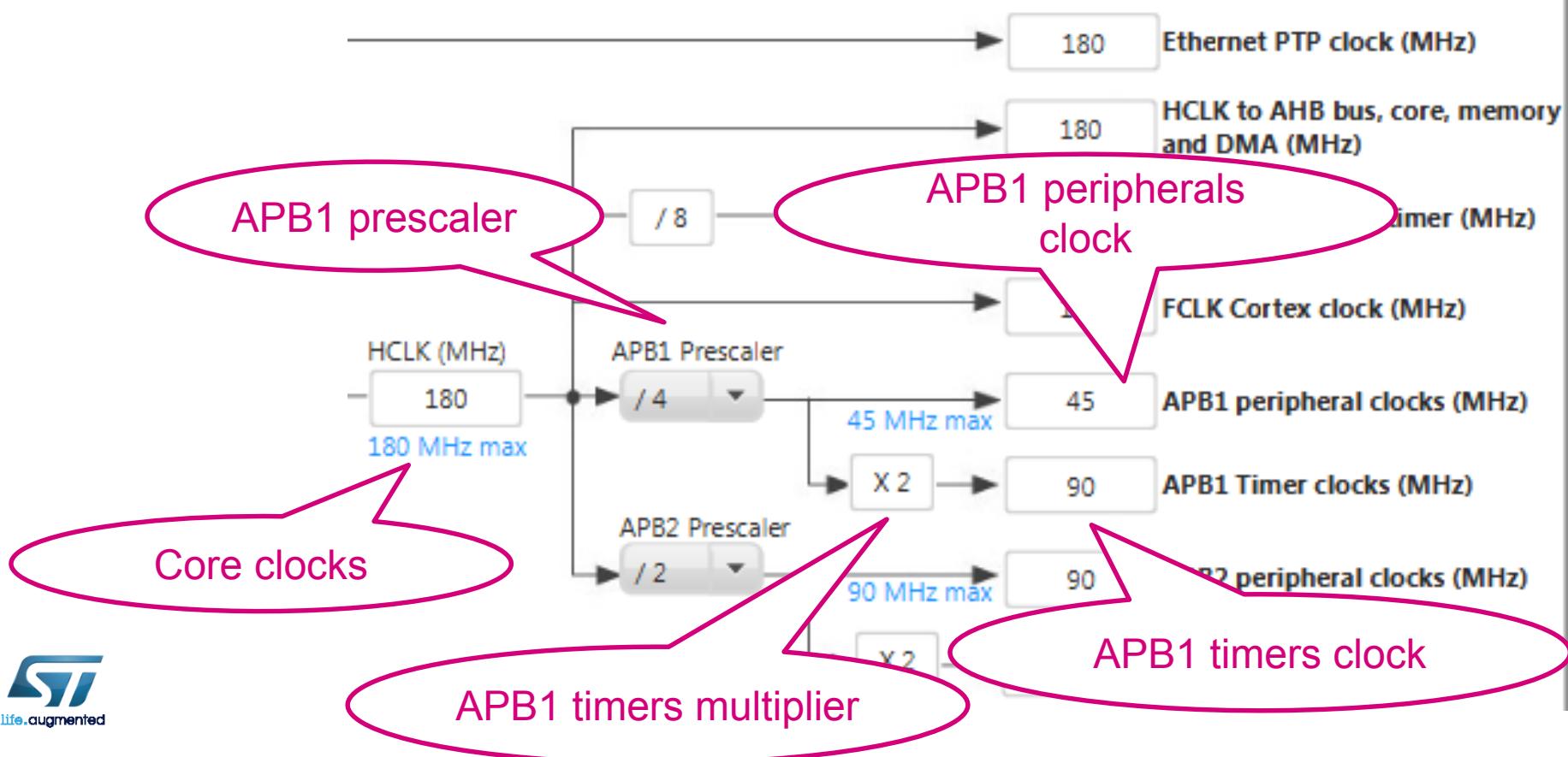
- APB prescalers and peripheral speed



# Configure GPIO for LED toggling

Clock Configuration overview 20

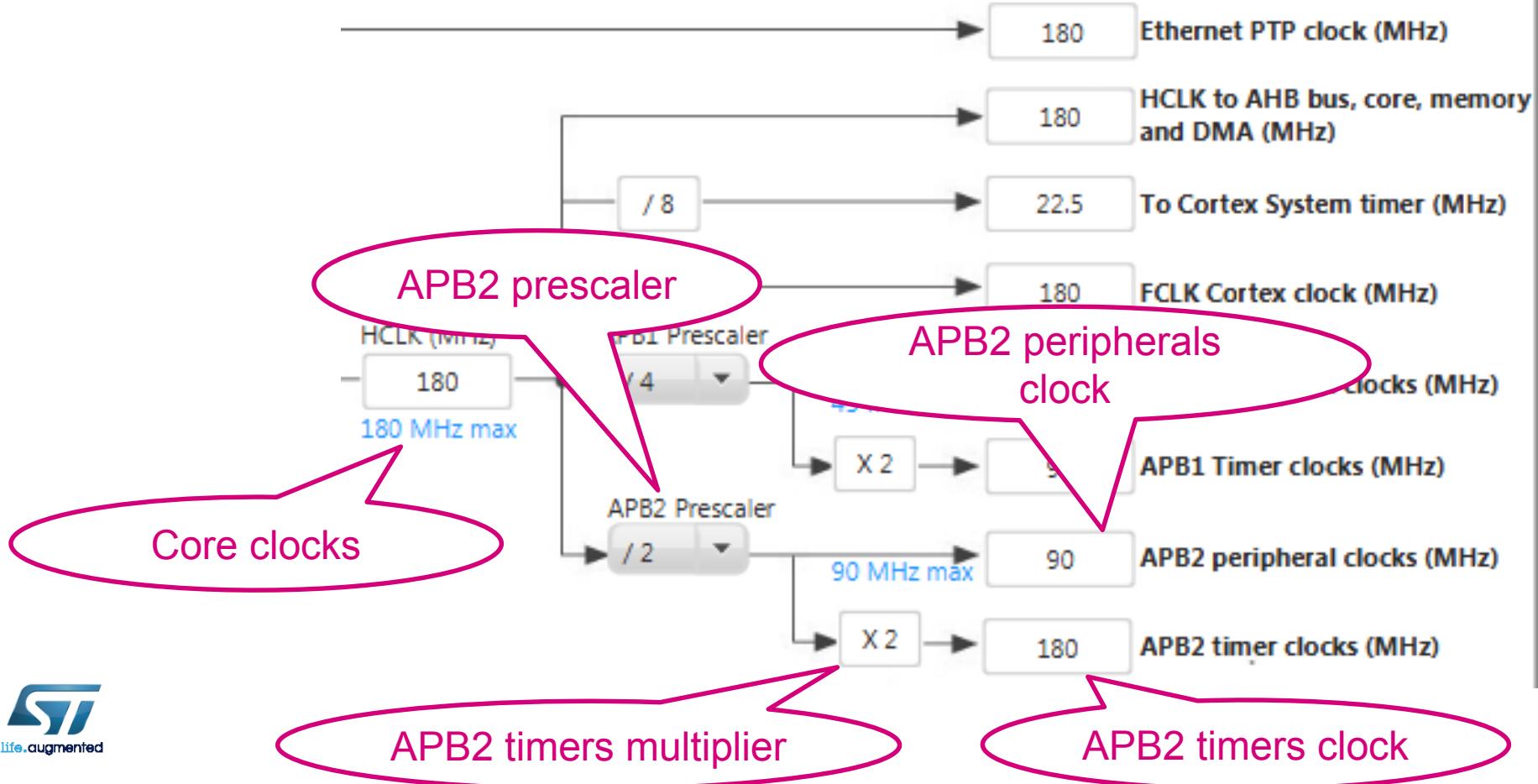
- AHB, APB prescalers and peripheral speed



# Configure GPIO for LED toggling

Clock Configuration overview 21

- AHB, APB prescalers and peripheral speed



# Configure GPIO for LED toggling

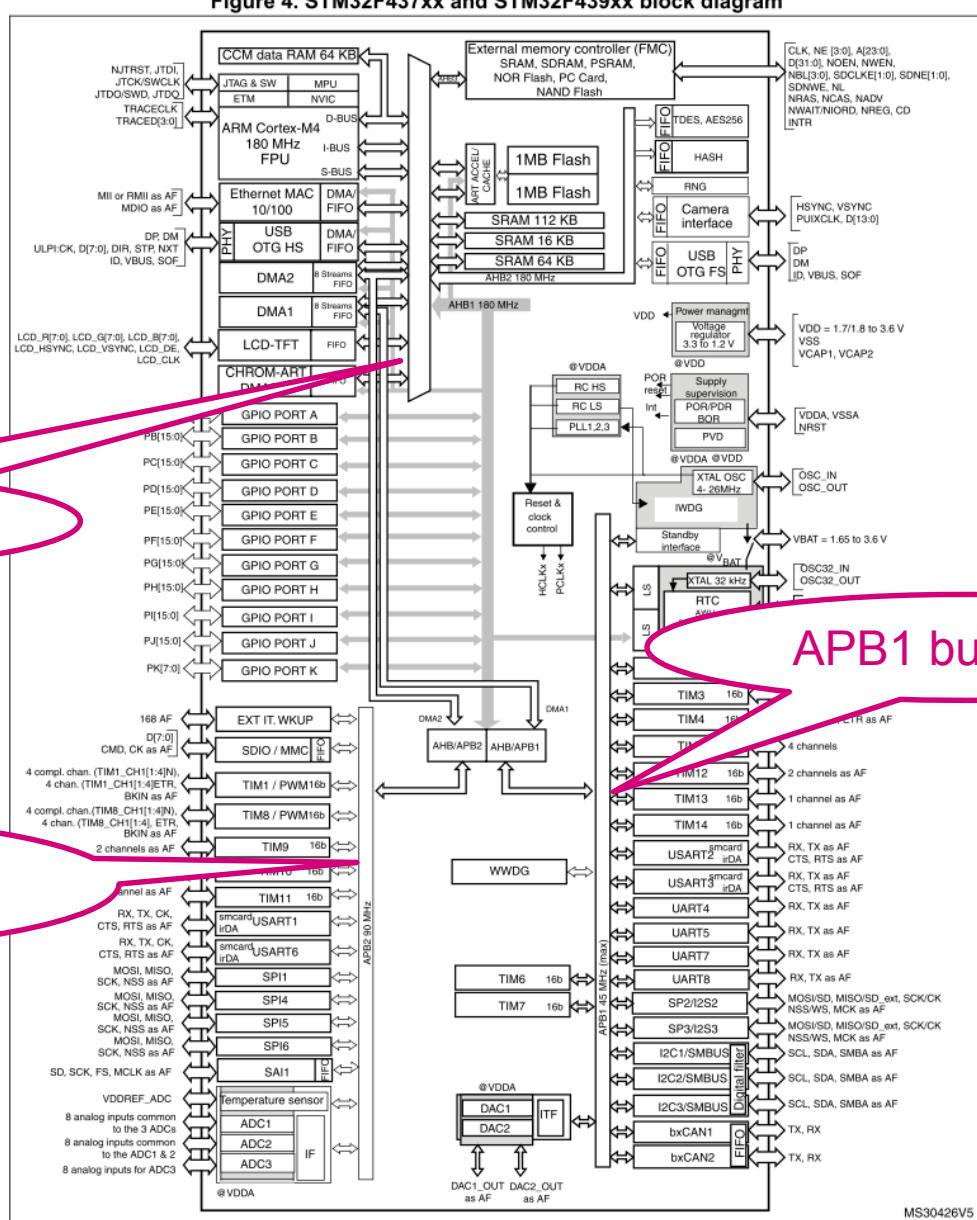
## Clock Configuration overview 22

- Data sheet  
Figure 4

AHB, Core, memory clock

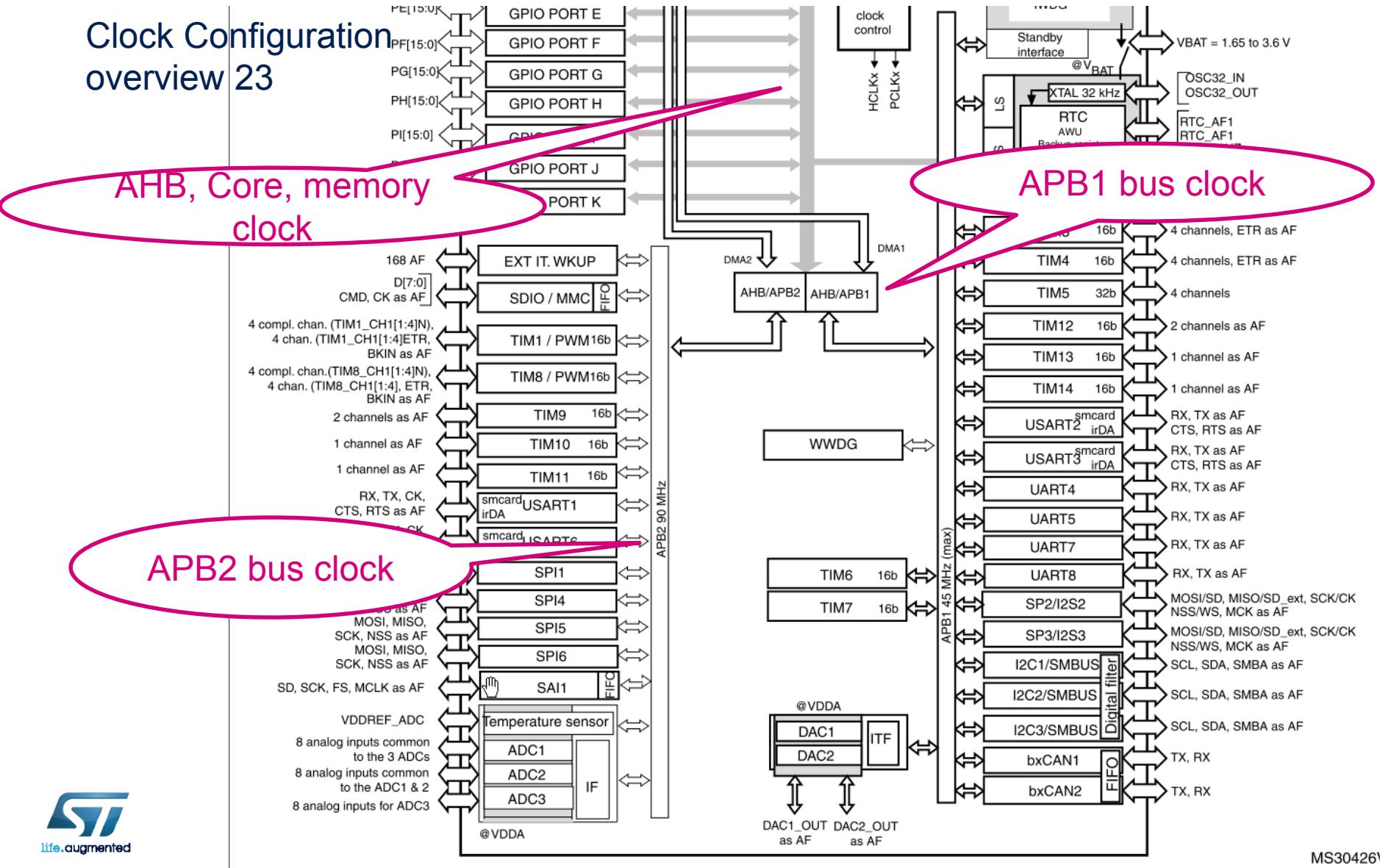
APB1 bus clock

APB2 bus clock



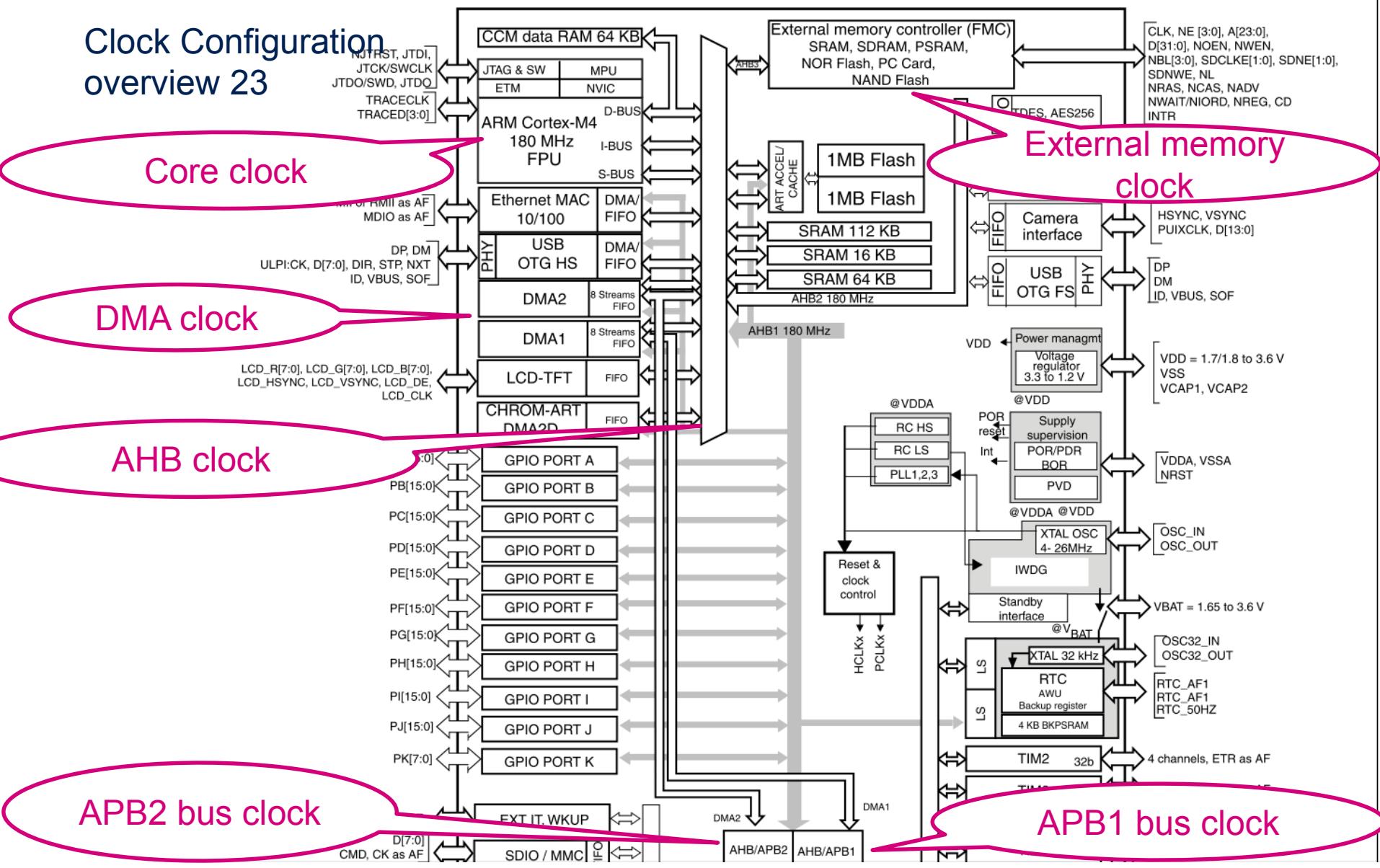
# Configure GPIO for LED toggling

Clock Configuration  
overview 23



# Configure GPIO for LED toggling

## Clock Configuration overview 23

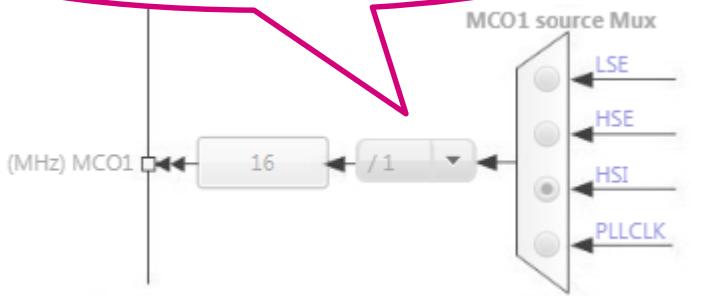


# Configure GPIO for LED toggling

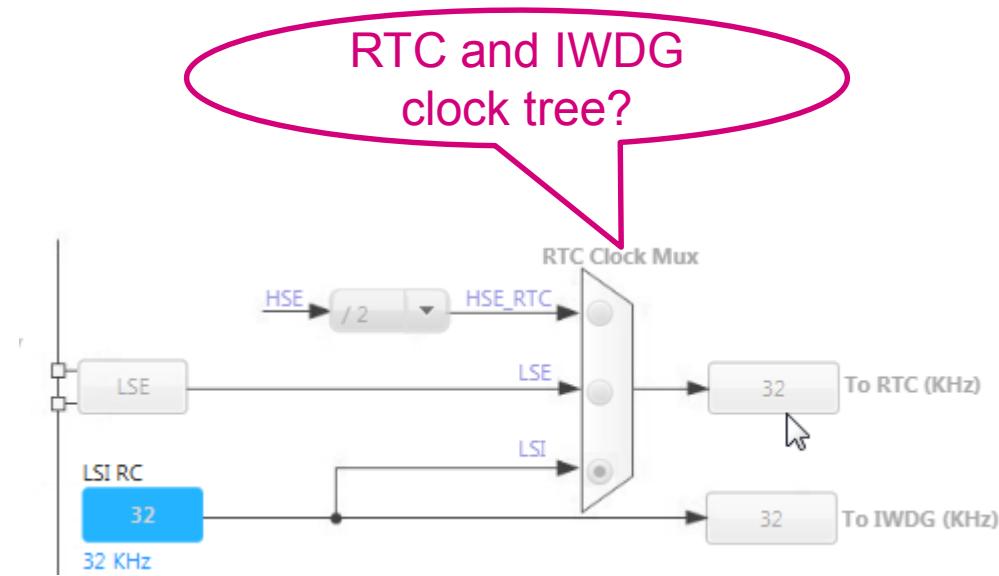
Clock Configuration overview 24

- Enable clocks which are gray
  - How to enable gray features?

MCO output?



RTC and IWDG  
clock tree?



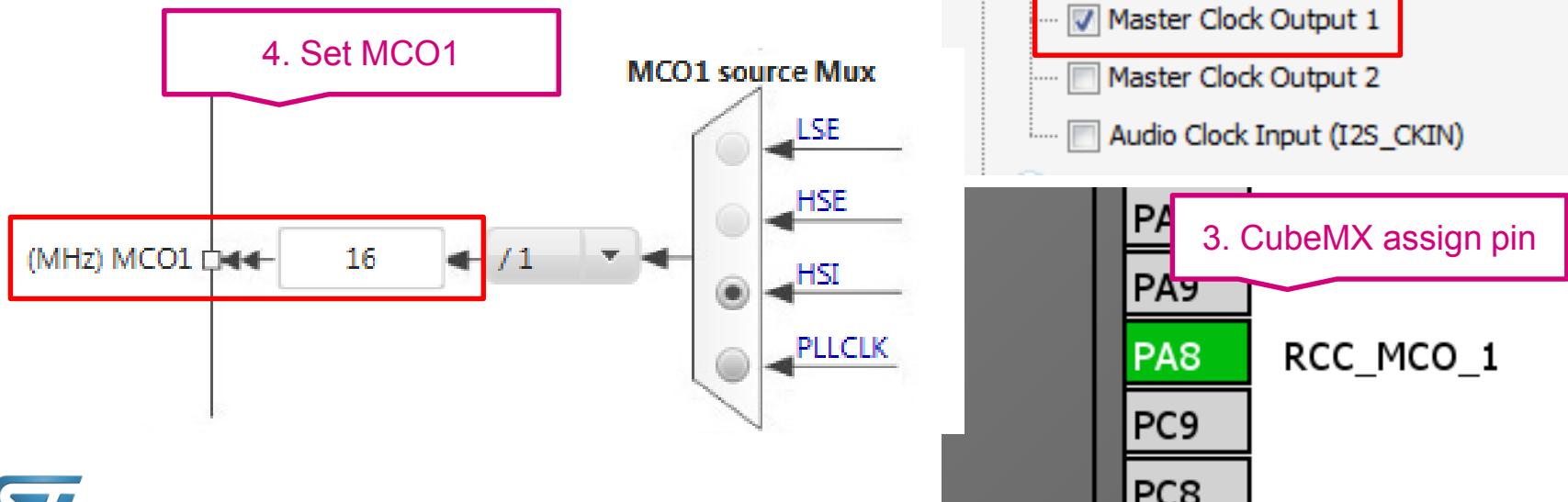
USB FS tree?



# Configure GPIO for LED toggling

## Clock Configuration overview 25

- MCO1 output
  - TAB>Pinout
  - RCC>MCO1 checkbox
  - TAB>Clock Configuration
  - Now the MCO1 output can be set

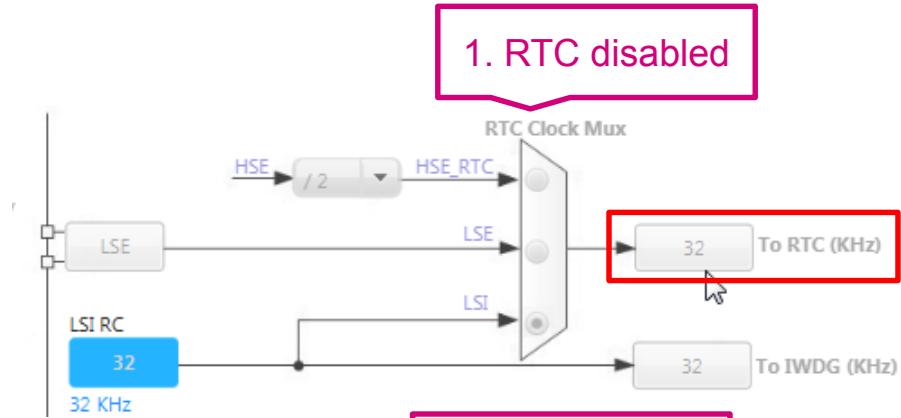


# Configure GPIO for LED toggling

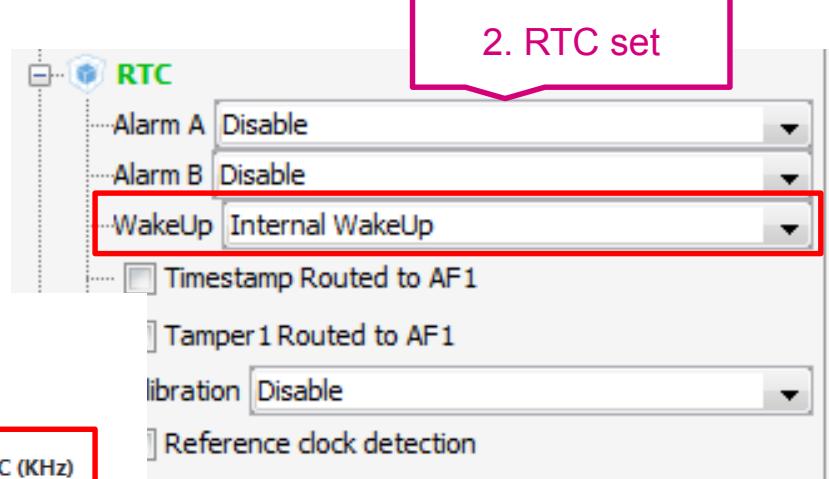
Clock Configuration overview 26

- RTC

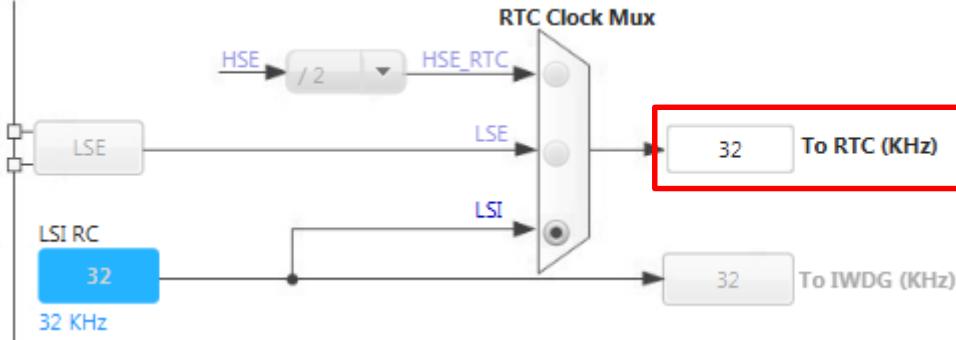
- TAB>Pinout
- RCC>RTC set RTC feature
- TAB>Clock Configuration
- Now the RTC can be set



1. RTC disabled



2. RTC set



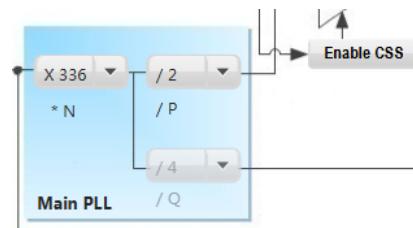
3. Set RTC

# Configure GPIO for LED toggling

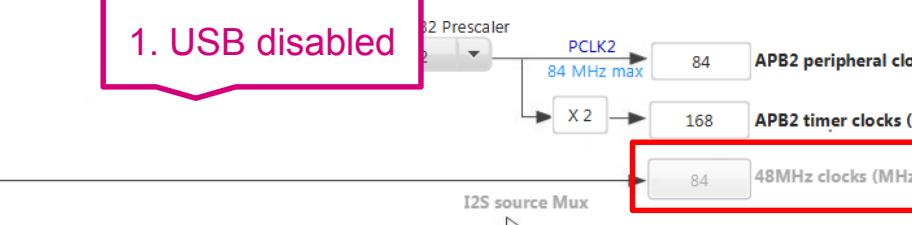
## Clock Configuration overview 26

- USB

- TAB>Pinout
- RCC>USB\_OTG\_FS set feature
- TAB>Clock Configuration
- Now the USB clock can be set



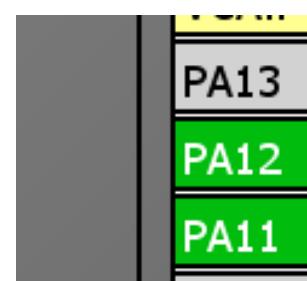
1. USB disabled



2. USB enable

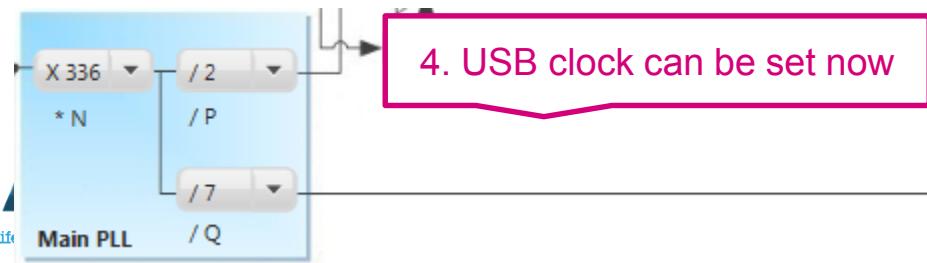


3. CubeMX assign pins

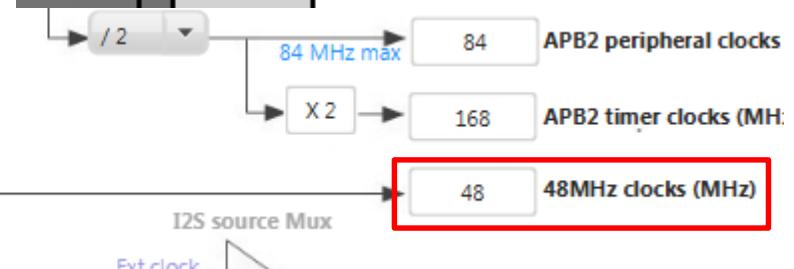


USB\_OTG\_FS\_DP

USB\_OTG\_FS\_DM

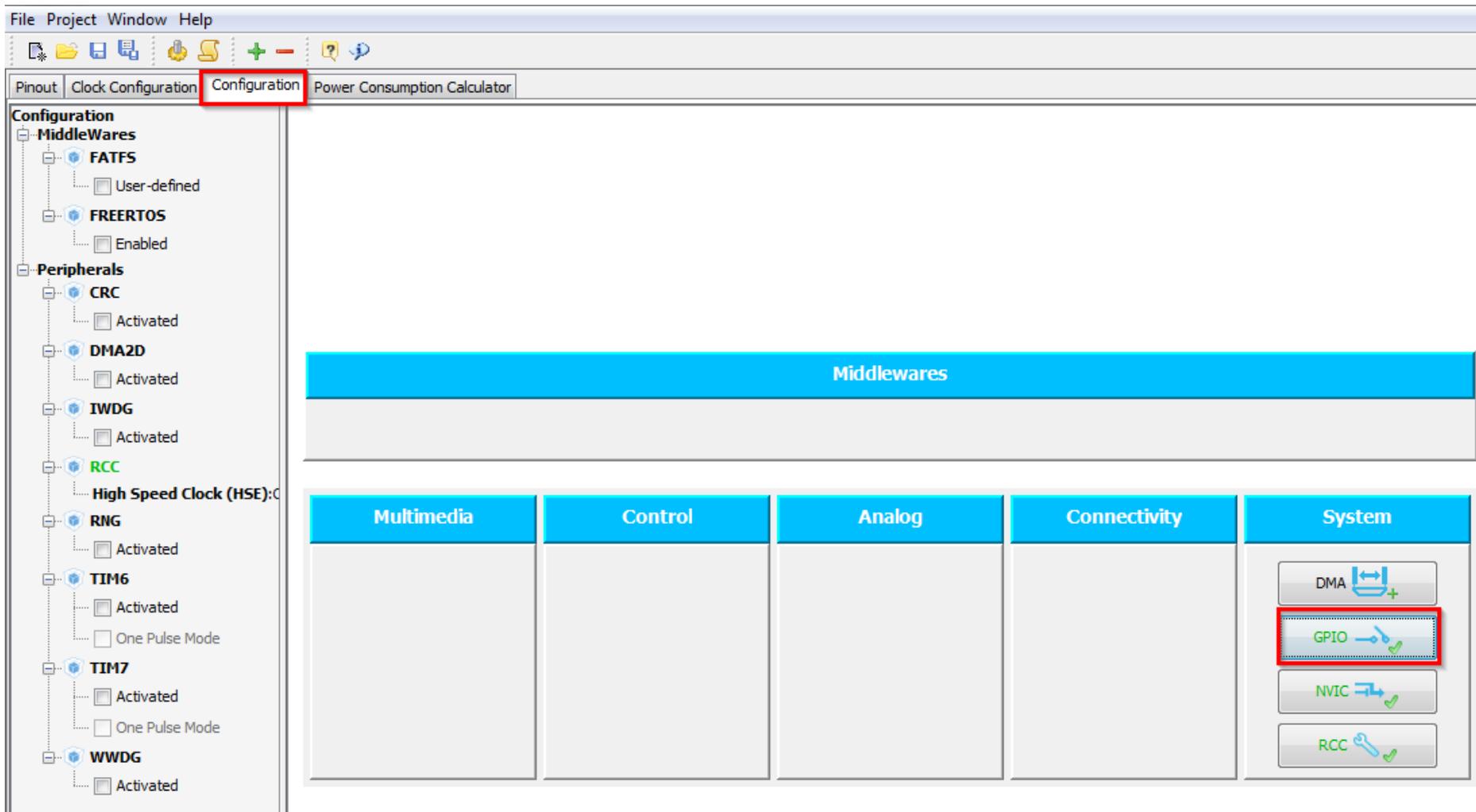


4. USB clock can be set now



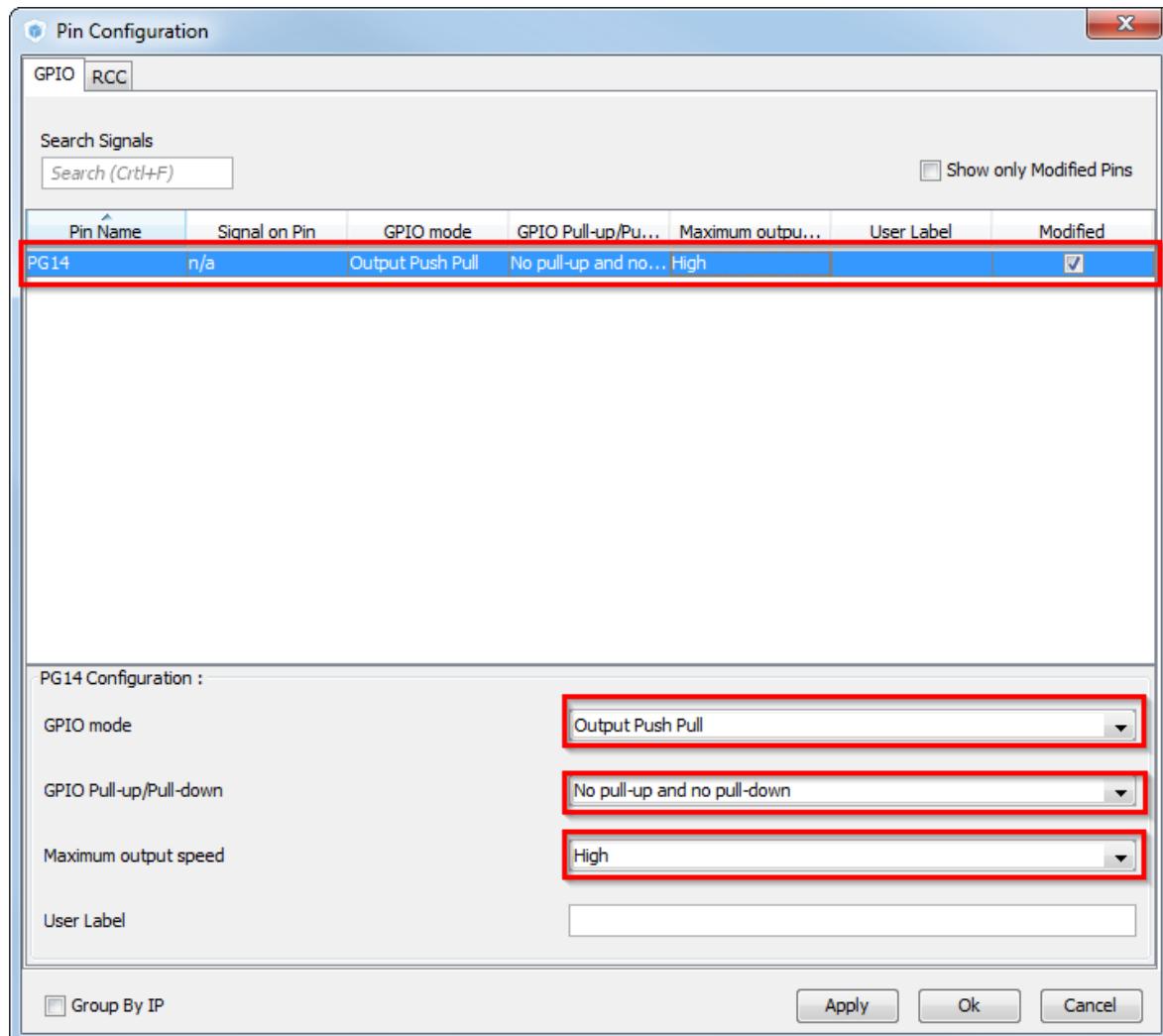
# Configure GPIO for LED toggling

- GPIO Configuration
  - TAB>Configuration>System>GPIO



# Configure GPIO for LED toggling

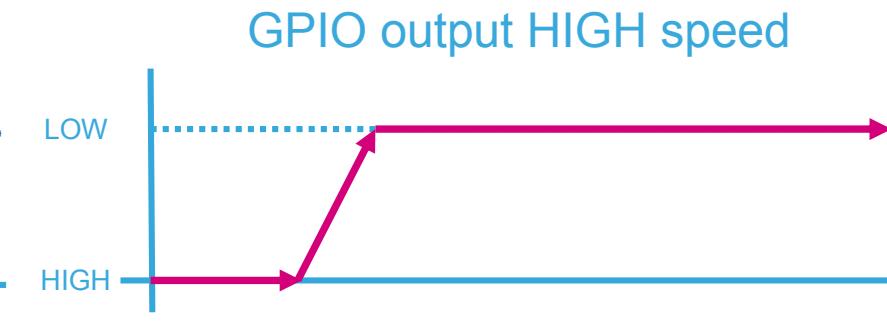
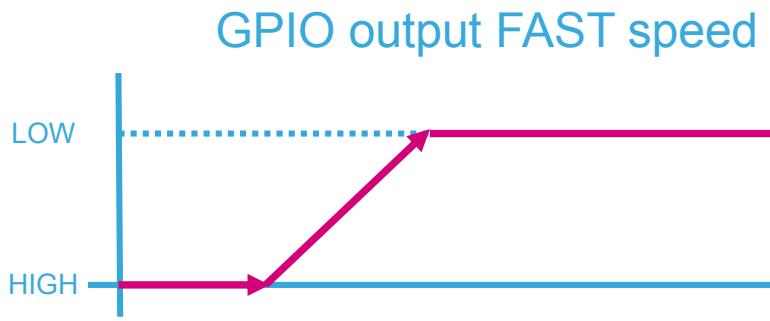
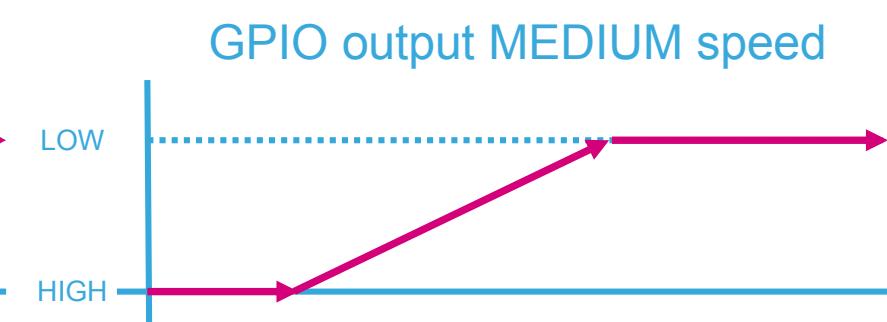
- GPIO(Pin) Configuration
  - Select Push Pull mode
  - No pull-up and pull-down
  - Output speed to HIGH  
Is important for faster peripheries like SPI, USART
  - Button OK



# Configure GPIO for LED toggling

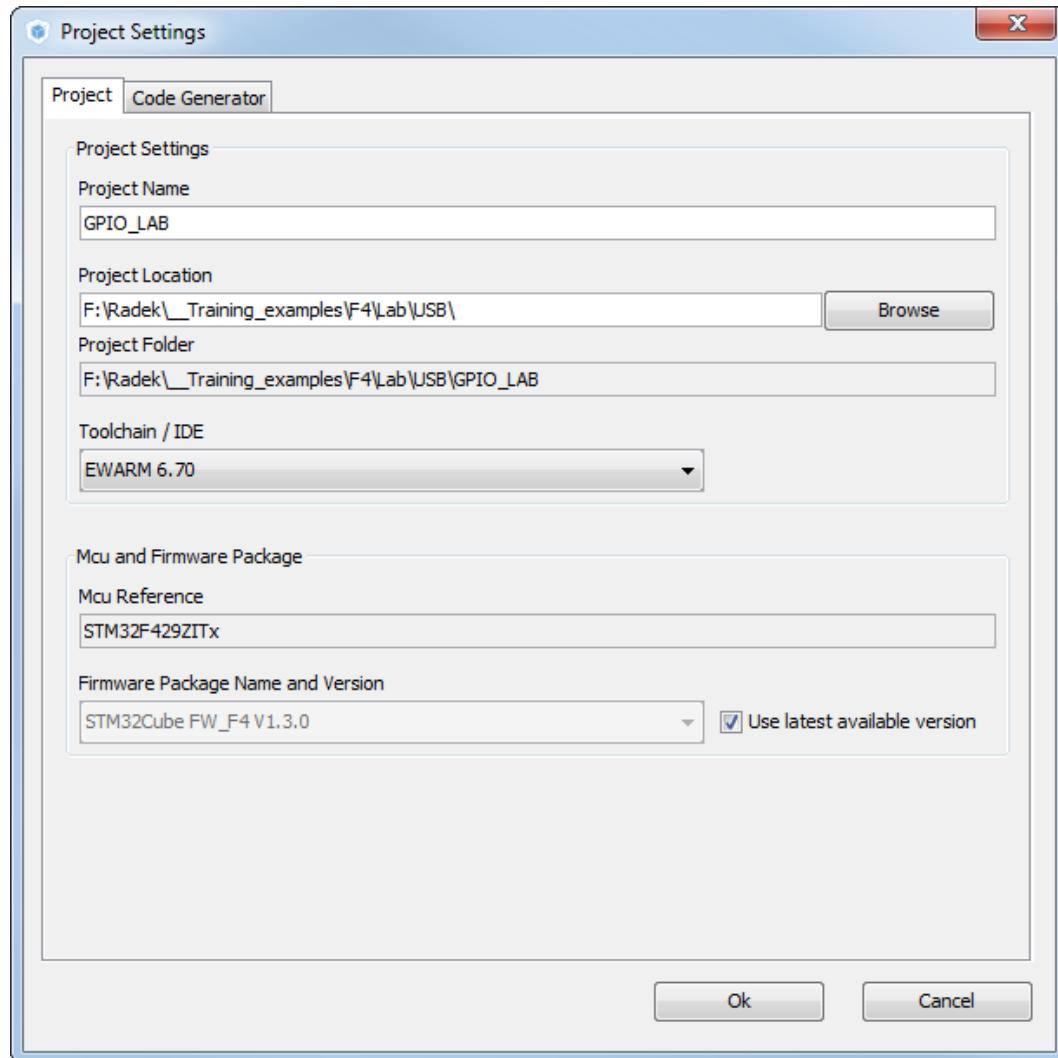
- GPIO(Pin) output speed configuration

- 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



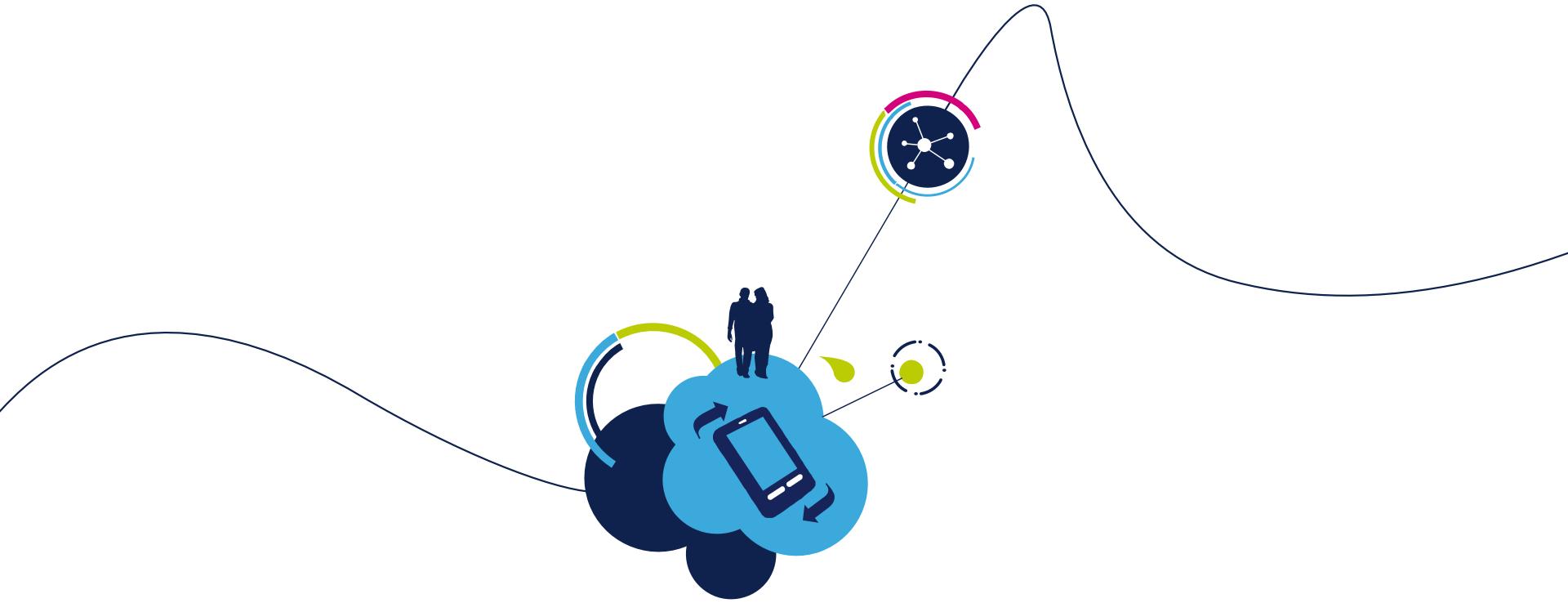
# Configure GPIO for LED toggling

- 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



# Configure GPIO for LED toggling

- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 3 */` and `/* USER CODE END 3 */` tags
  - Into infinite loop `while(1){ }`
- For toggling we need to use this functions
  - `HAL_HAL_Delay` which create specific delay
  - `HAL_GPIO_WritePin` or `HAL_GPIO_TogglePin`



# EXTI lab 2

# 2 Configure EXTI which turns on LED

45

- Objective

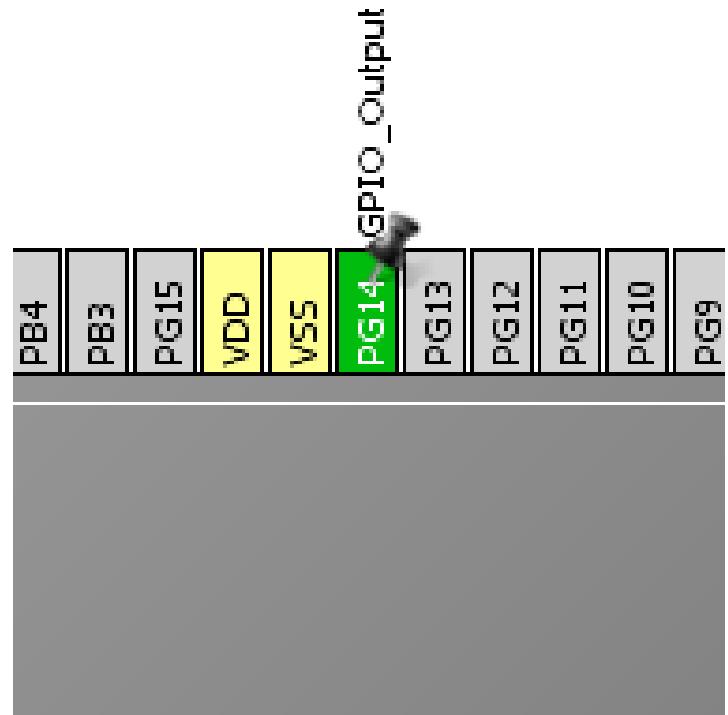
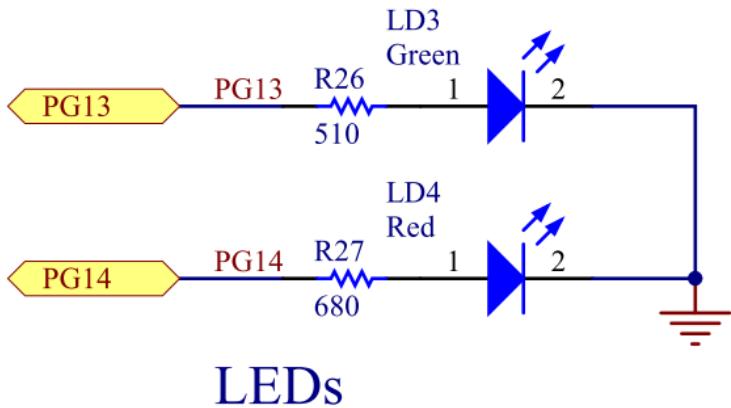
- Learn how to setup input pin with EXTI in CubeMX
- How to Generate Code in CubeMX and use HAL functions

- Goal

- Configure GPIO and EXTI pin in CubeMX and Generate Code
- Add into project Callback function and function which turn on led
- Verify the correct functionality by pressing button which turns on LED

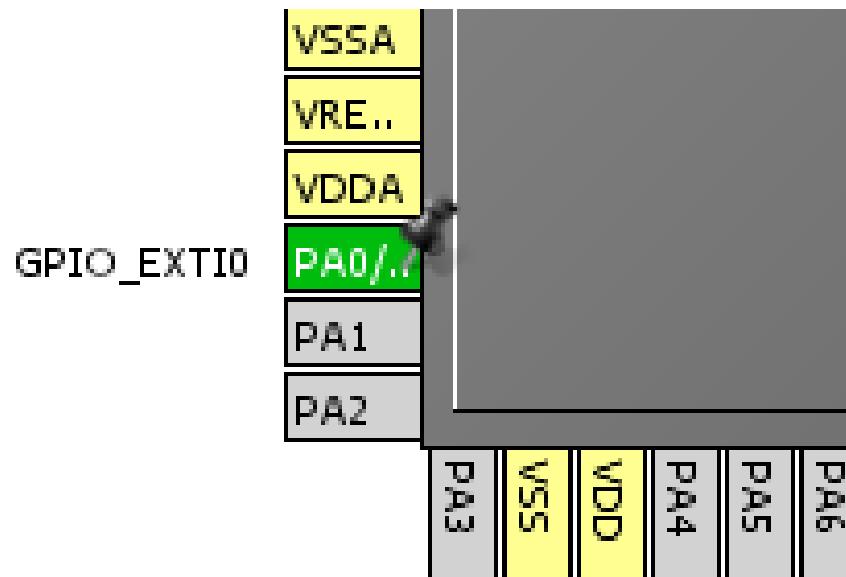
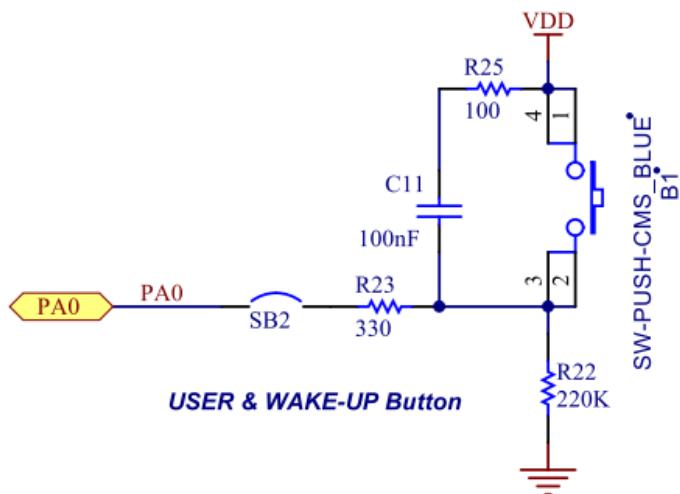
# Configure EXTI which turns on LED

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Configure LED pin as GPIO\_Output
- Configure Button pin as GPIO\_EXTIX



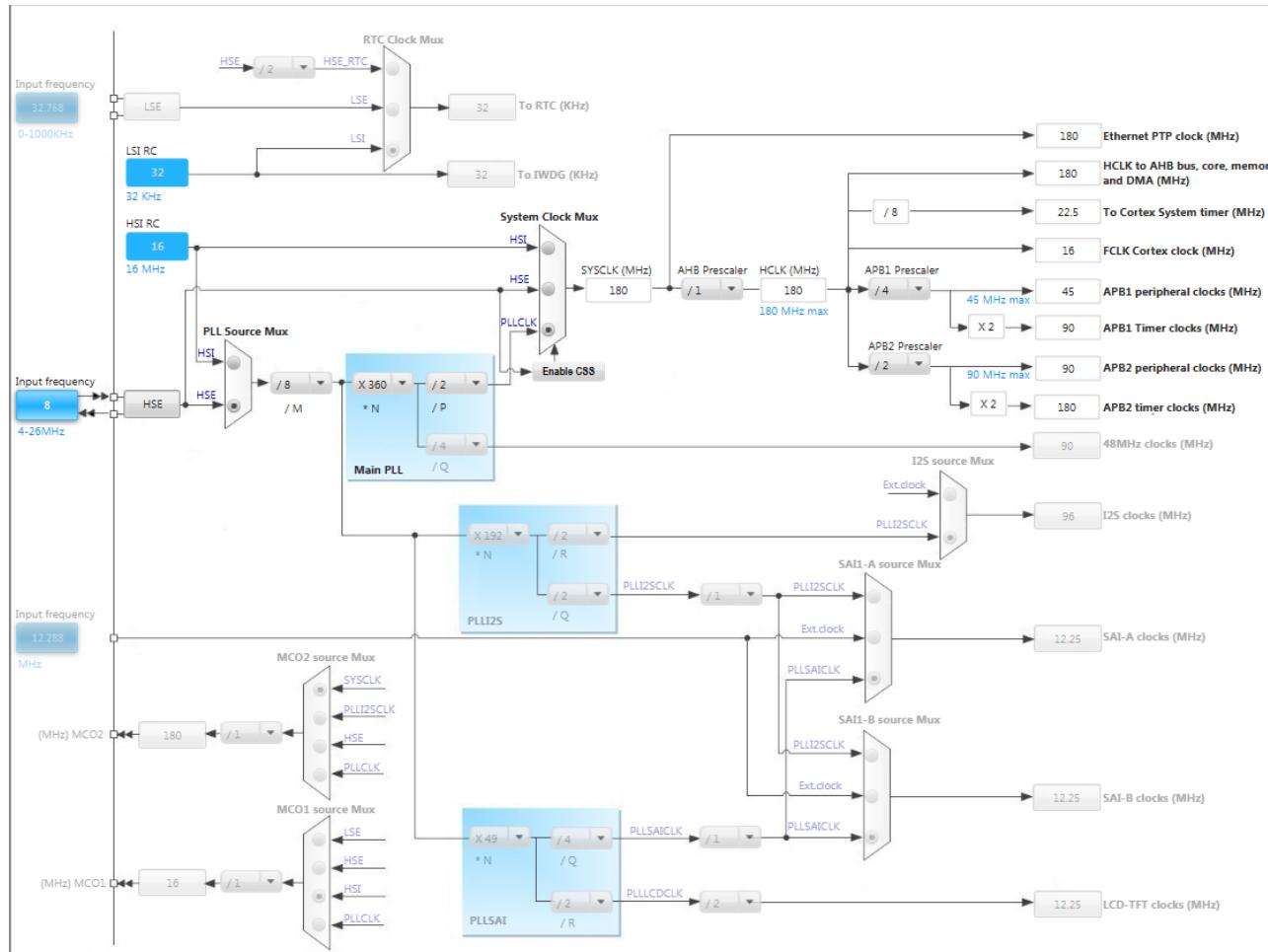
## 2 Configure EXTI which turns on LED

- Create project in CubeMX
    - Menu > File > New Project
    - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
  - Configure LED pin as GPIO\_Output
  - Configure Button pin as GPIO\_EXTIX



# Configure EXTI which turns on LED

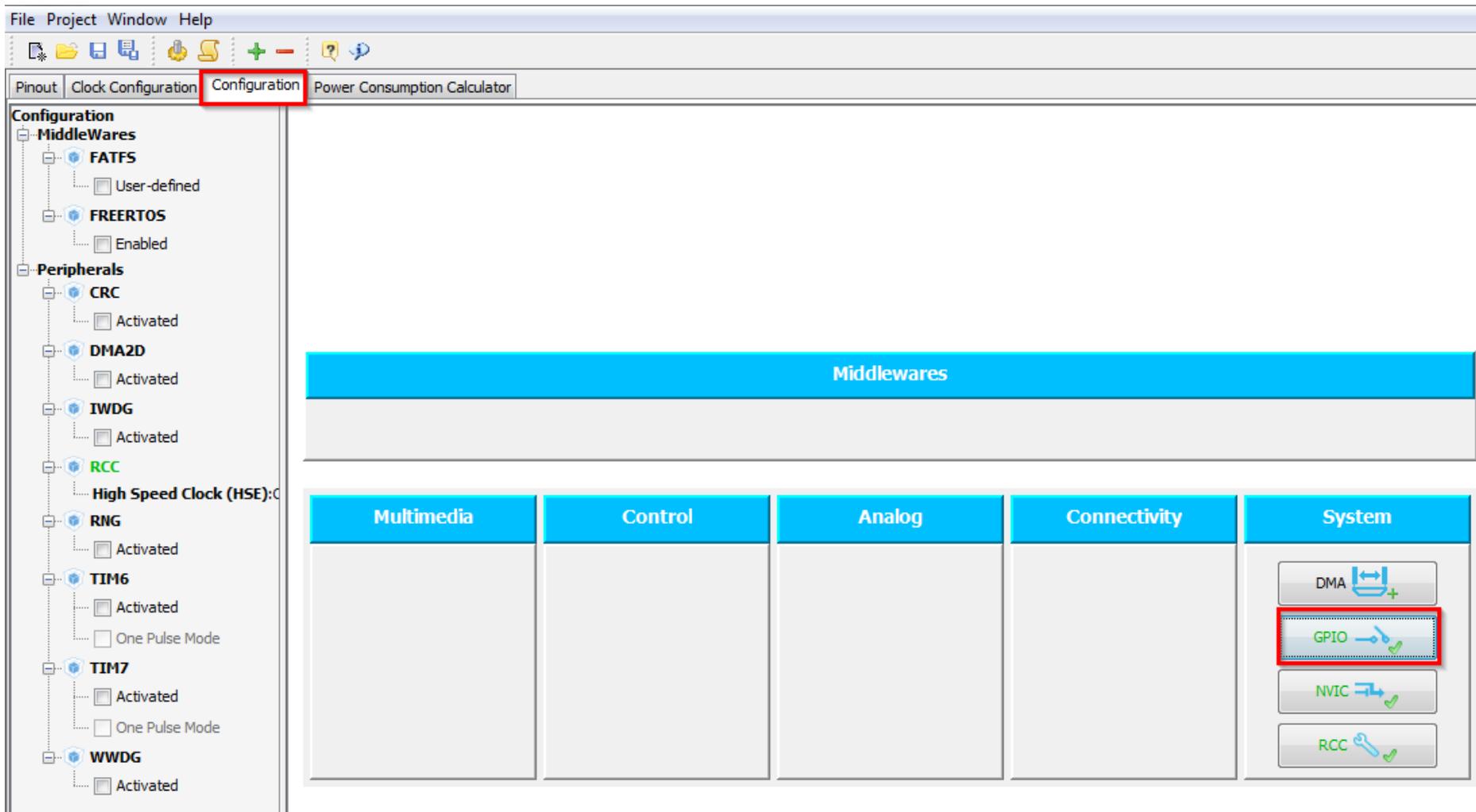
- In order to run on maximum frequency, setup clock system
- Details in lab 0



# 2 Configure EXTI which turns on LED

49

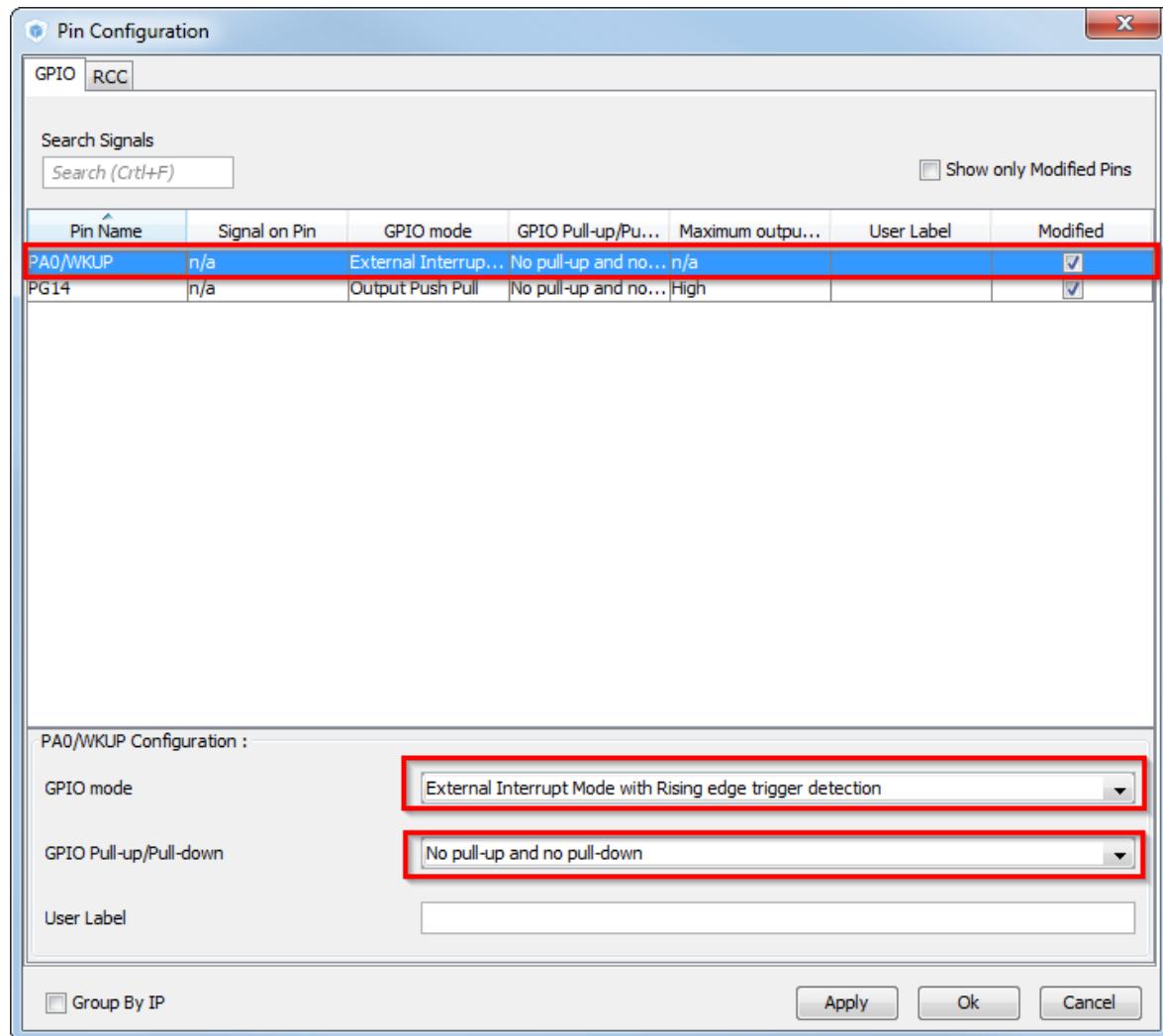
- GPIO Configuration
  - TAB>Configuration>System>GPIO



# Configure EXTI which turns on LED

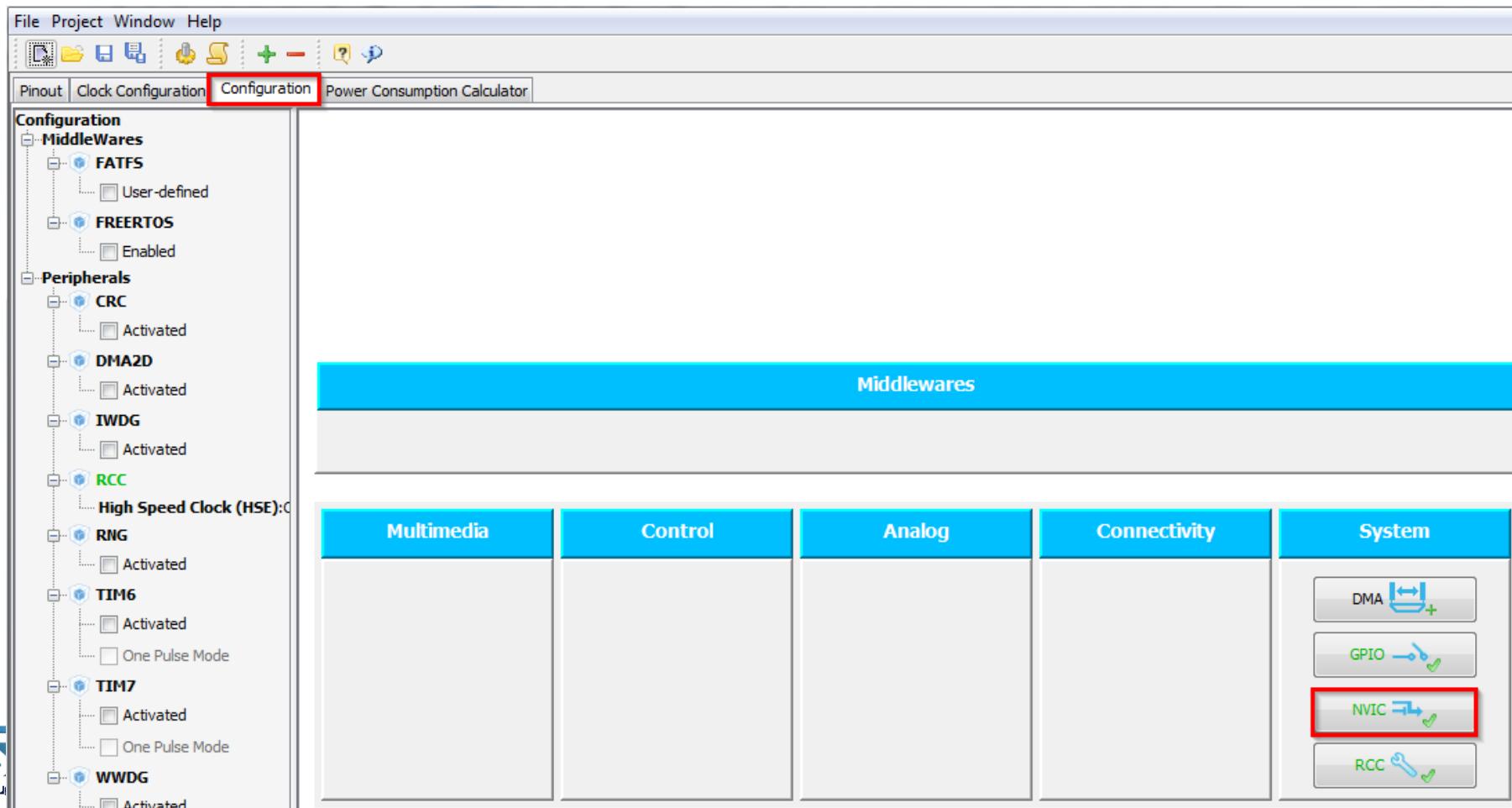
- GPIO(Pin) Configuration

- Select External Interrupt Mode with Rising edge trigger detection
- No pull-up or pull-down
- PG14 can be left in default settings
- Button OK



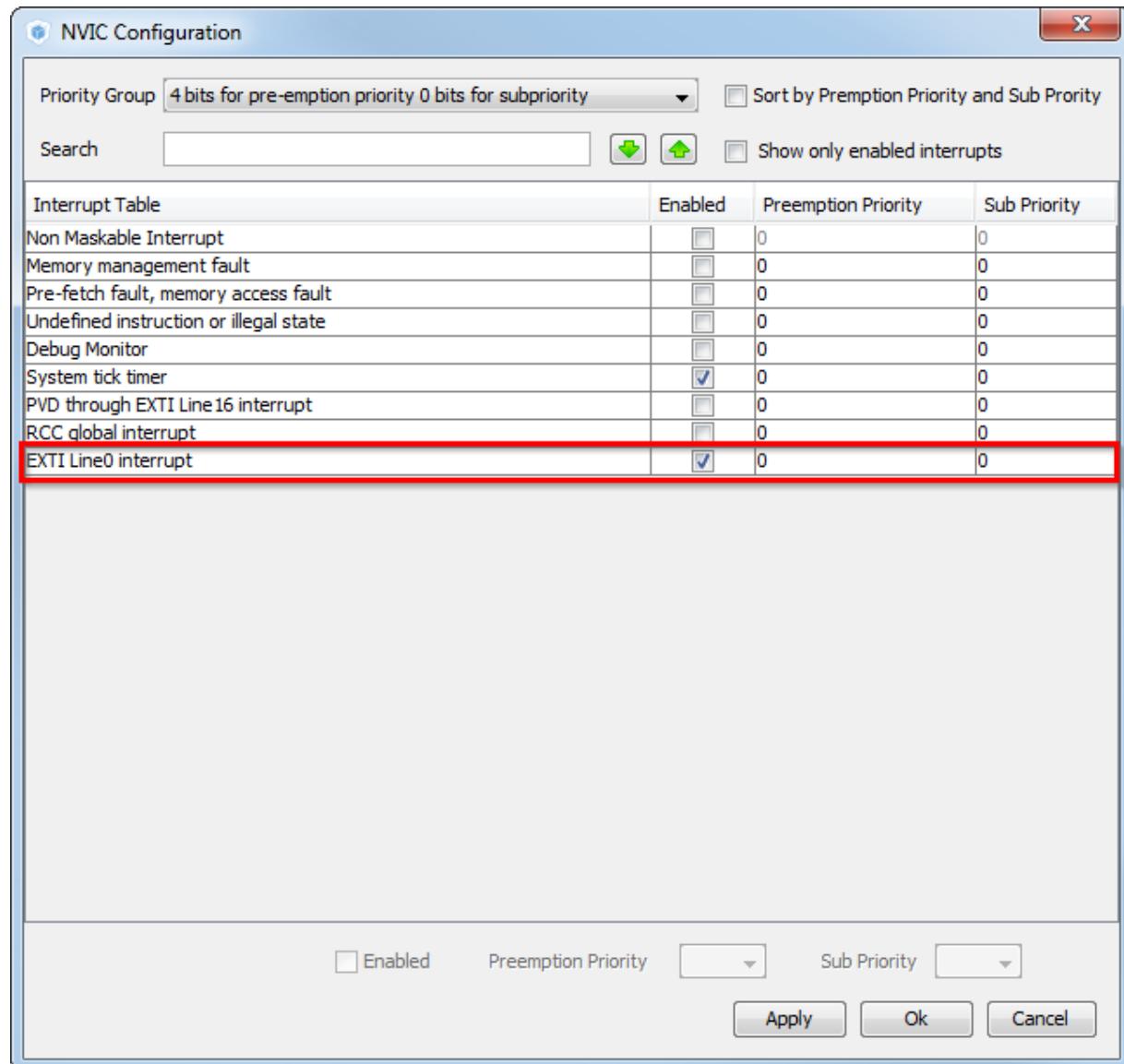
# Configure EXTI which turns on LED

- NVIC Configuration
  - We need to enable interrupts for EXTI
  - TAB>Configuration>System>NVIC



# Configure EXTI which turns on LED

- NVIC Configuration
  - Enable interrupt for EXTI Line0
  - Button OK



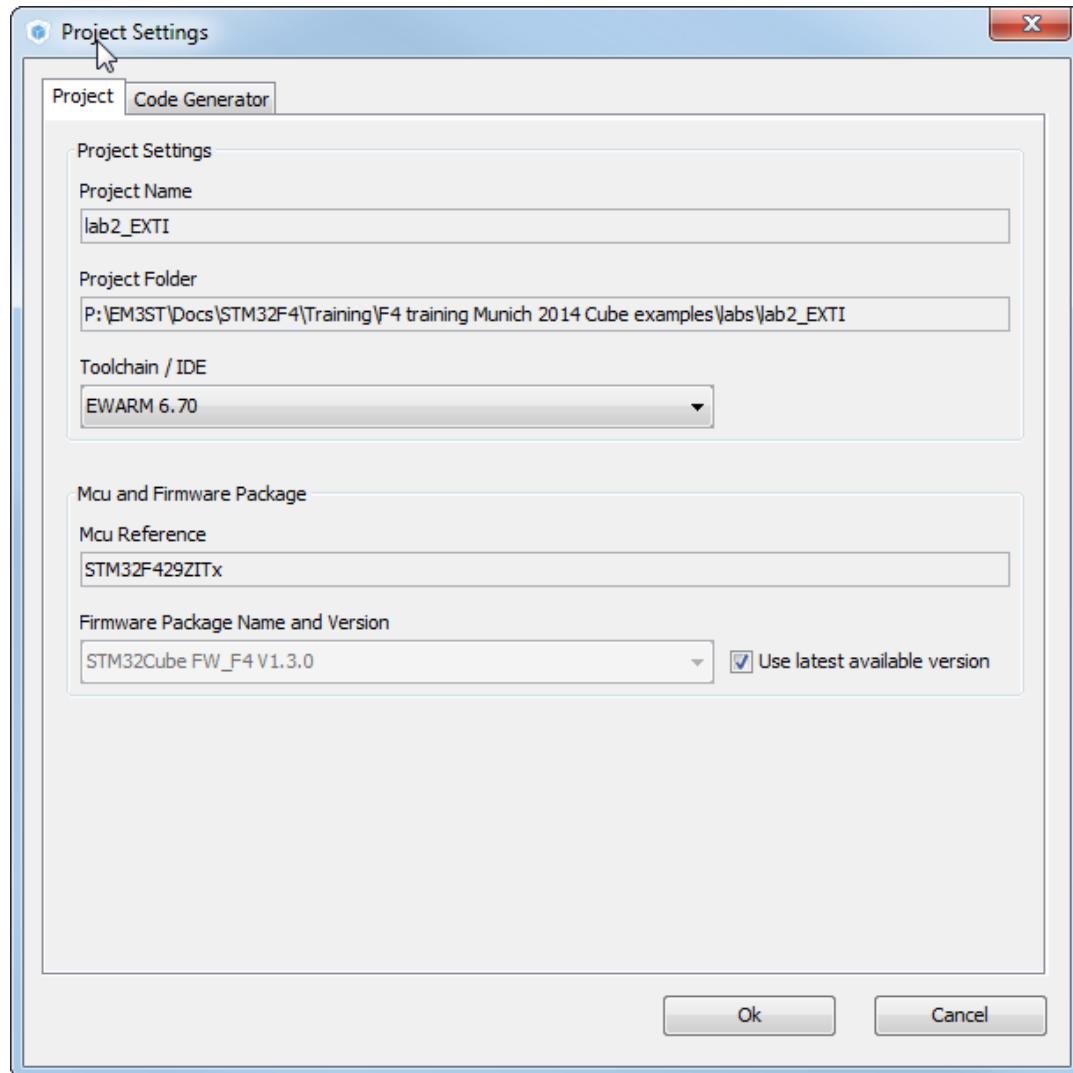
# Configure EXTI which turns on LED

- Now we set the project details for generation

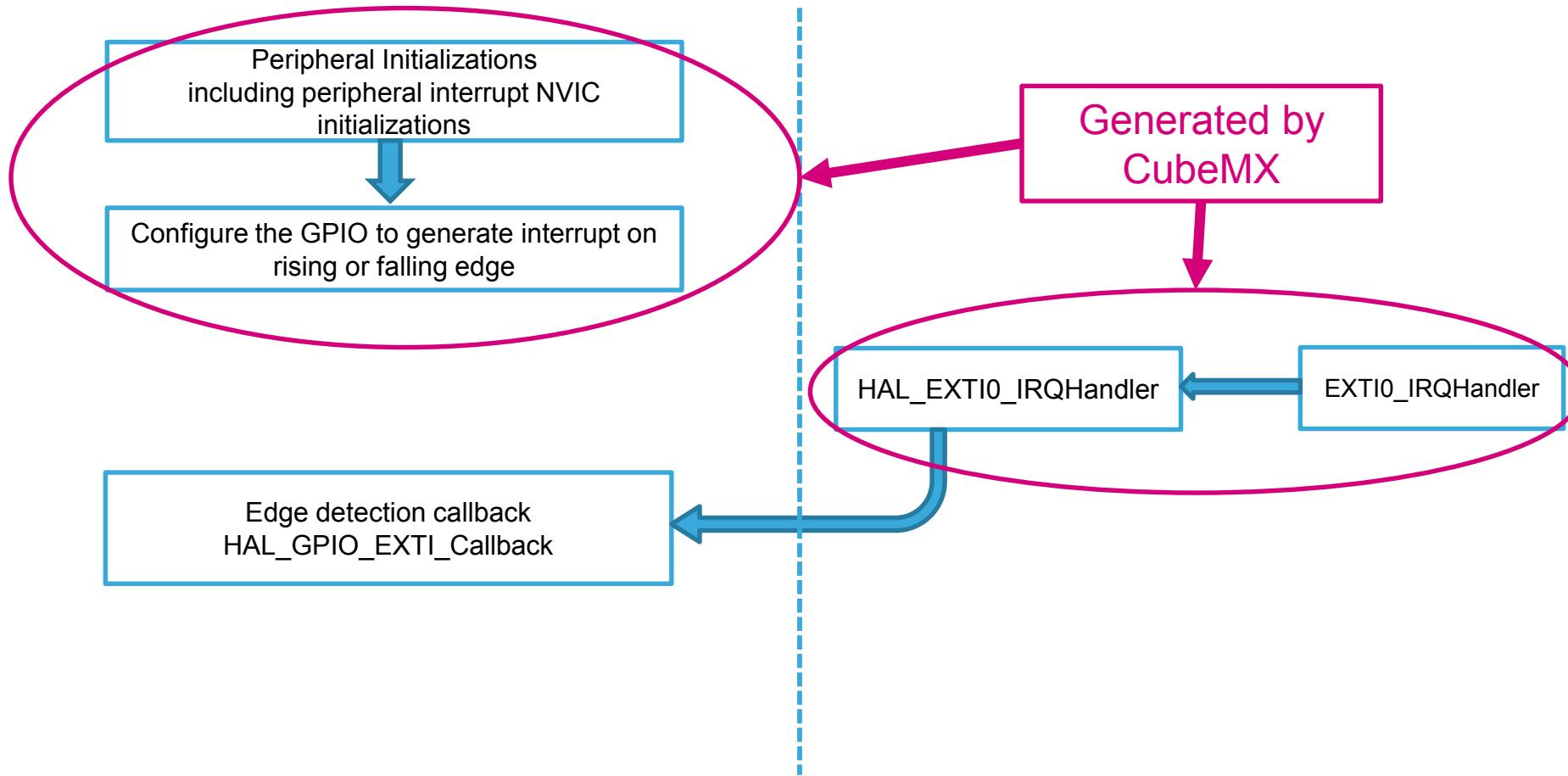
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

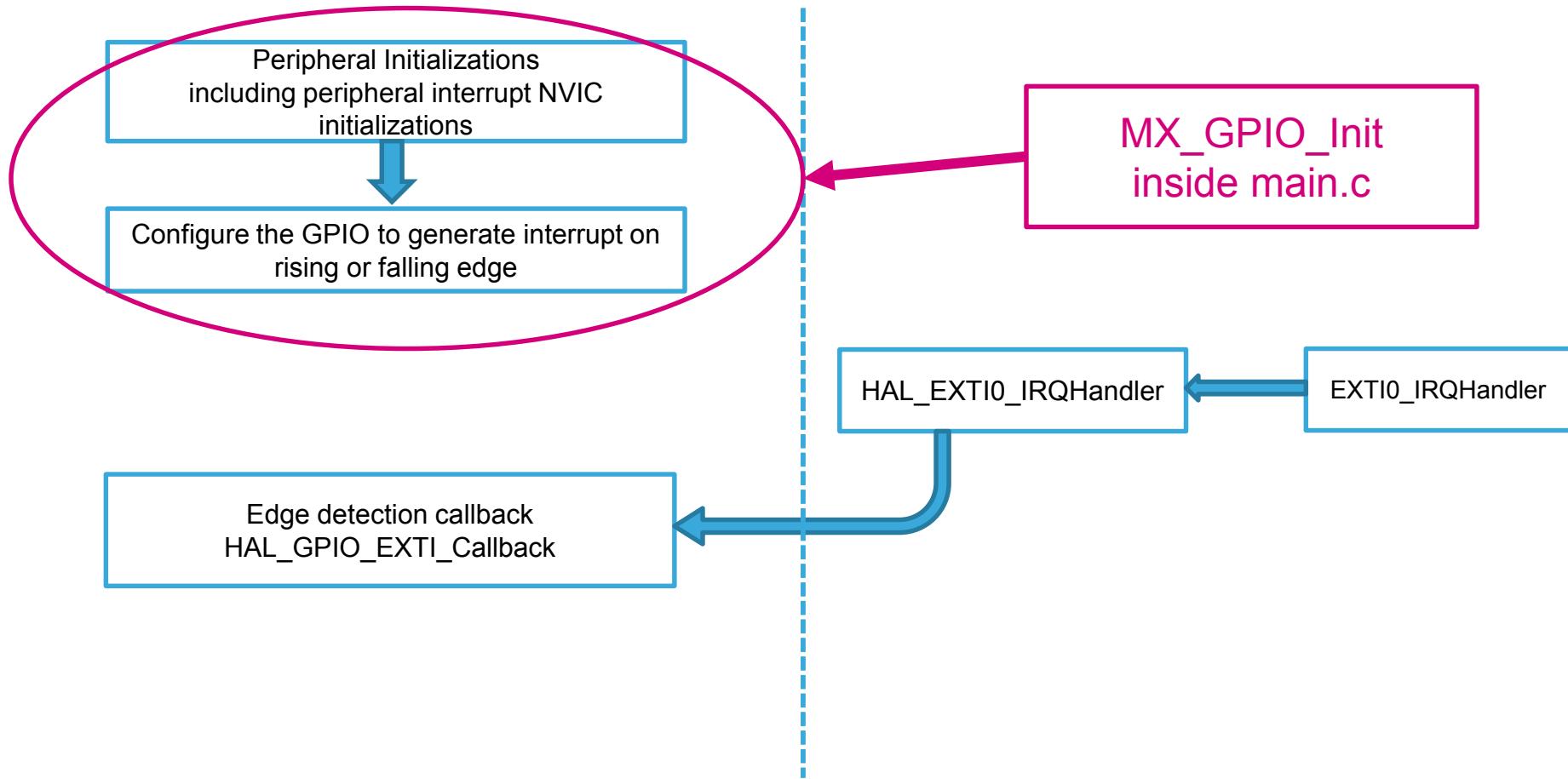
- Menu > Project > Generate Code



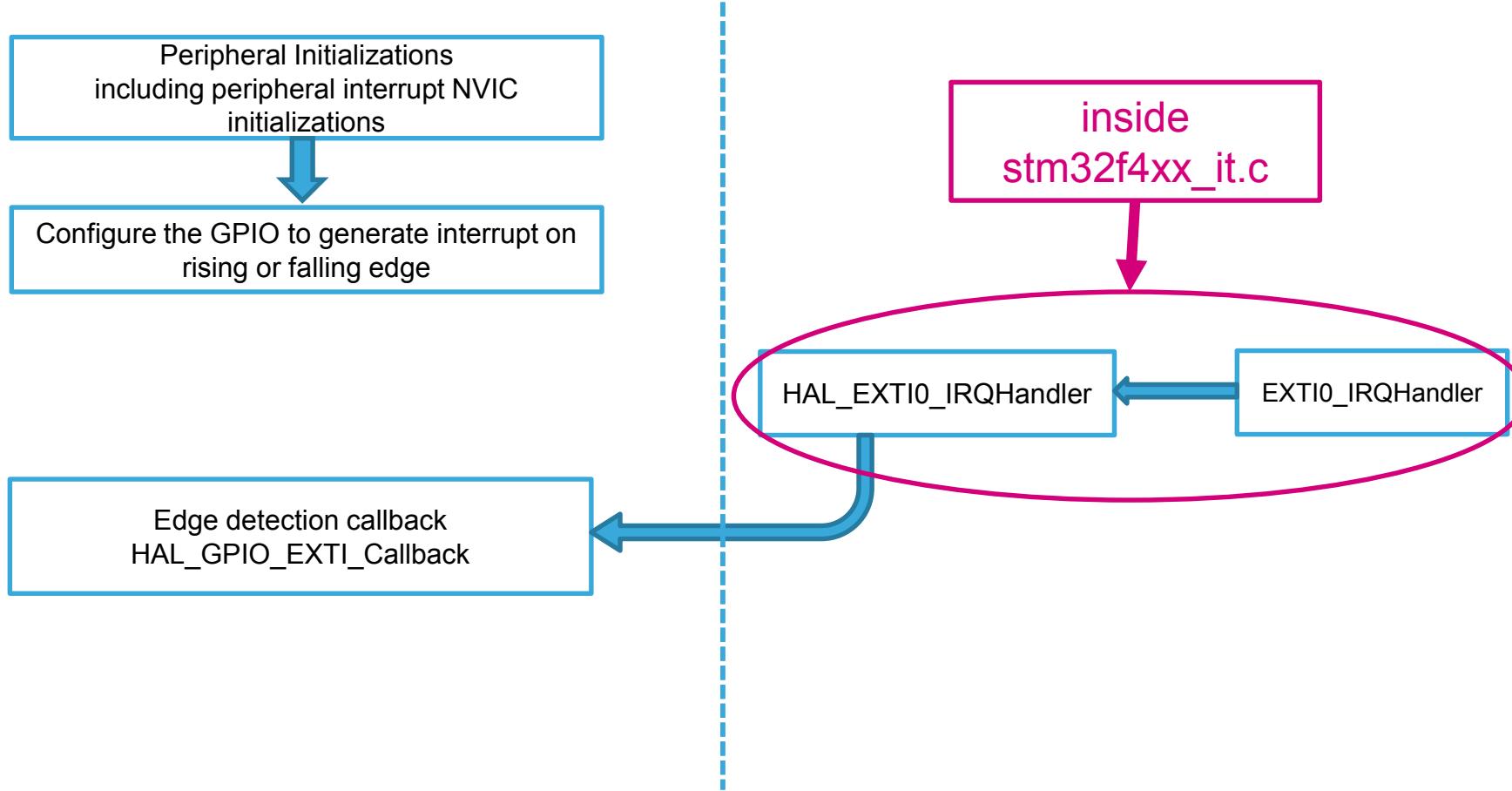
## HAL Library work flow 1



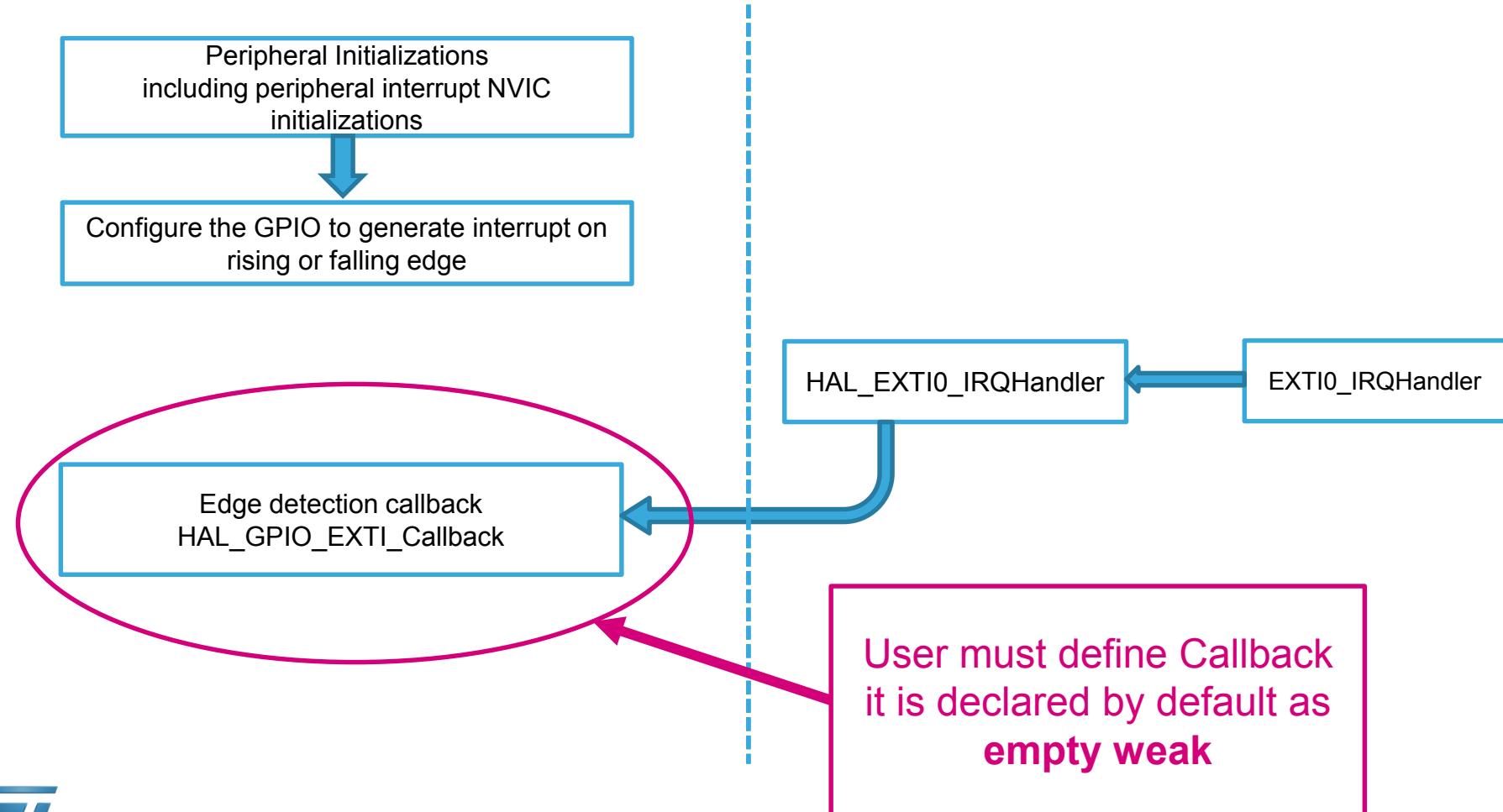
## HAL Library work flow 2



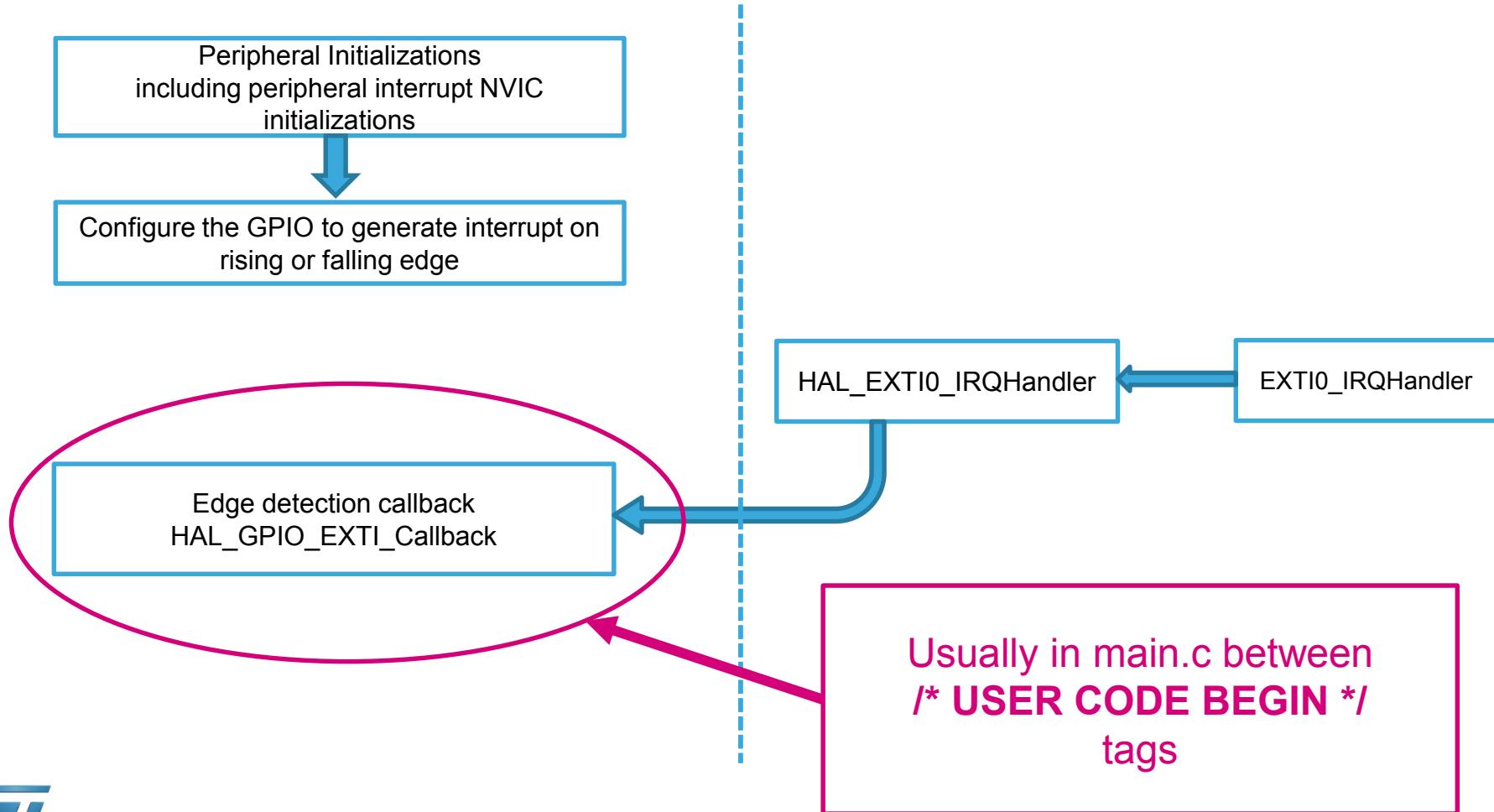
## HAL Library working flow 3



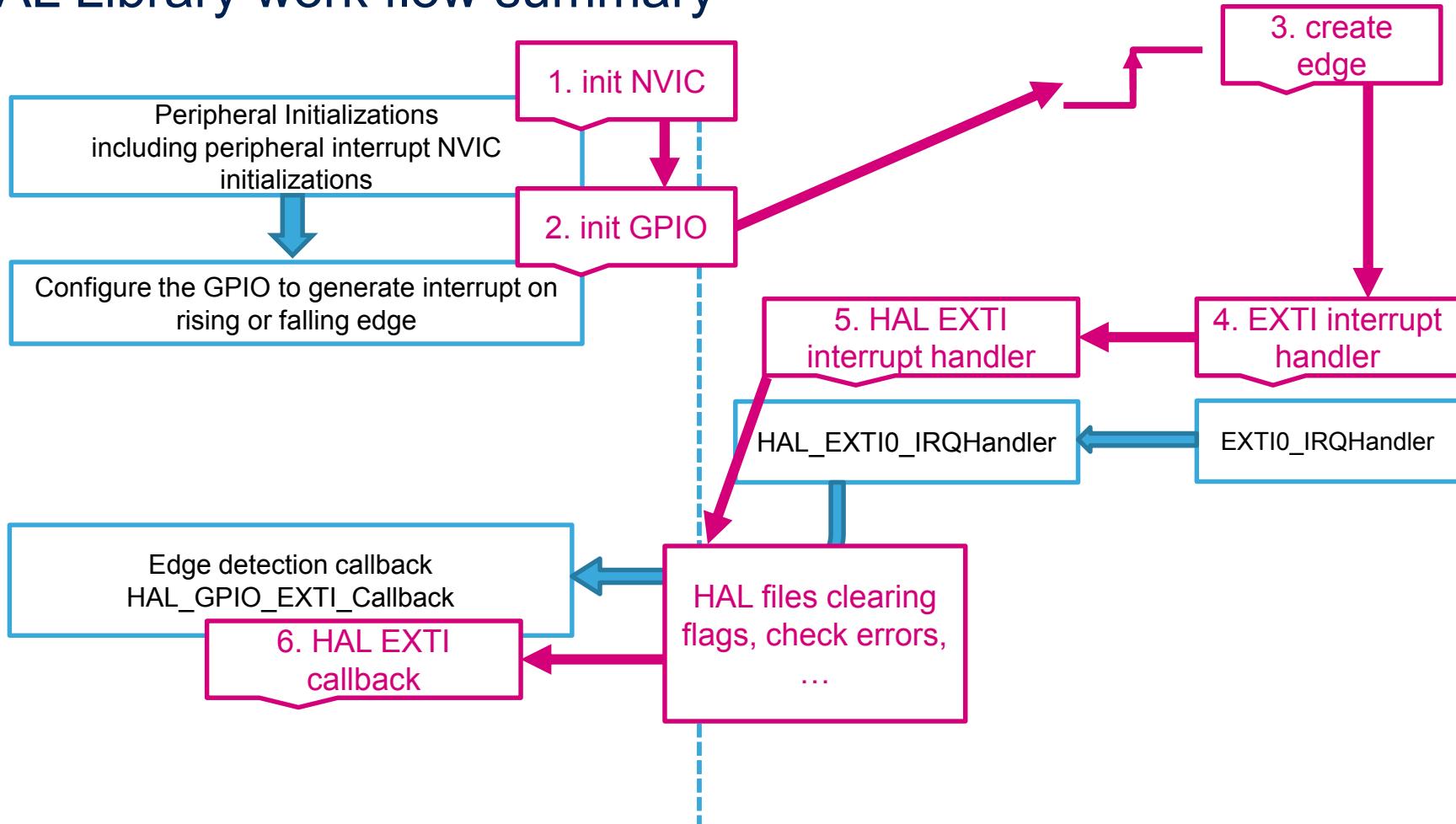
## HAL Library work flow 4



## HAL Library work flow 5



## HAL Library work flow summary



# 2 Configure EXTI which turns on LED

60

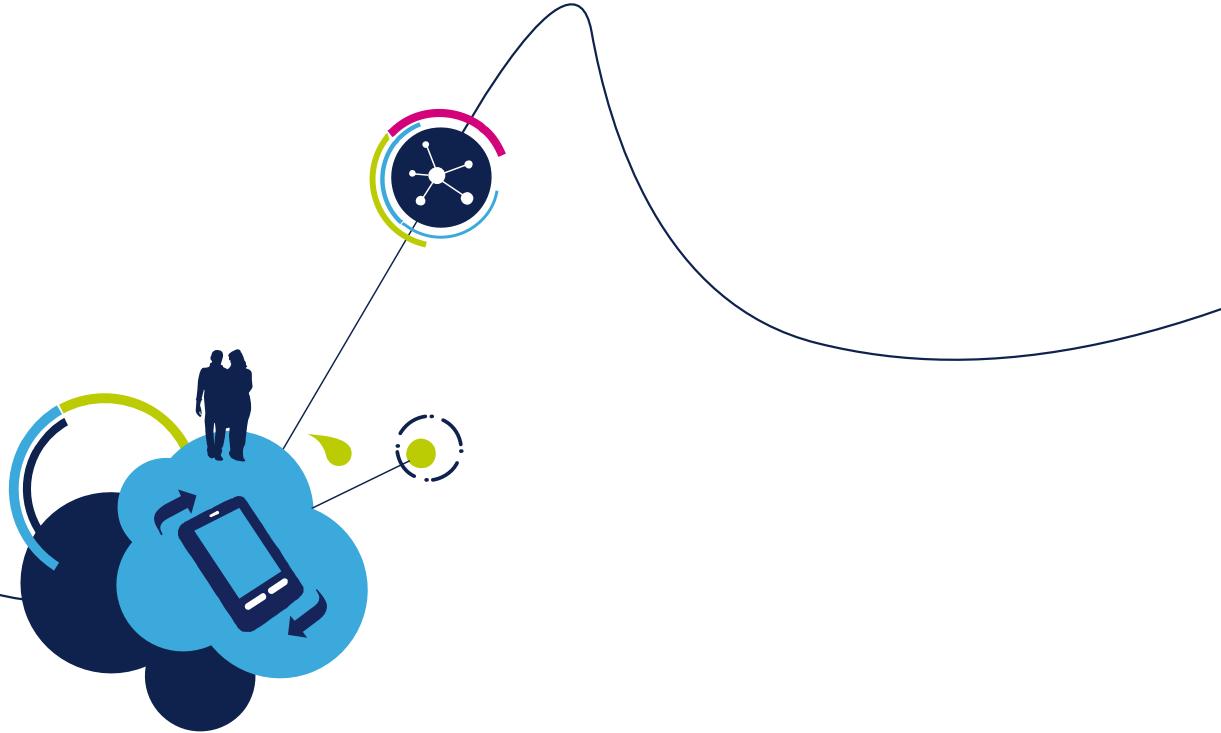
- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 4 */` and `/* USER CODE END 4 */` tags
  - We create function which will handle the EXTI interrupts
- The HAL callback function for EXTI
  - `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`
- For LED turn on we need to use this functions
  - `HAL_GPIO_WritePin`

# 2 Configure EXTI which turns on LED

61

- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 4 */` and `/* USER CODE END 4 */` tags
  - We create function which will handle the EXTI interrupts
- The HAL callback function for EXTI
  - `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`
- For LED turn on we need to use this functions
  - `HAL_GPIO_WritePin`

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(GPIO_Pin == GPIO_PIN_0) {
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
    } else {
        __NOP();
    }
}
/* USER CODE END 4 */
```



# Low Power mode SLEEP lab 3

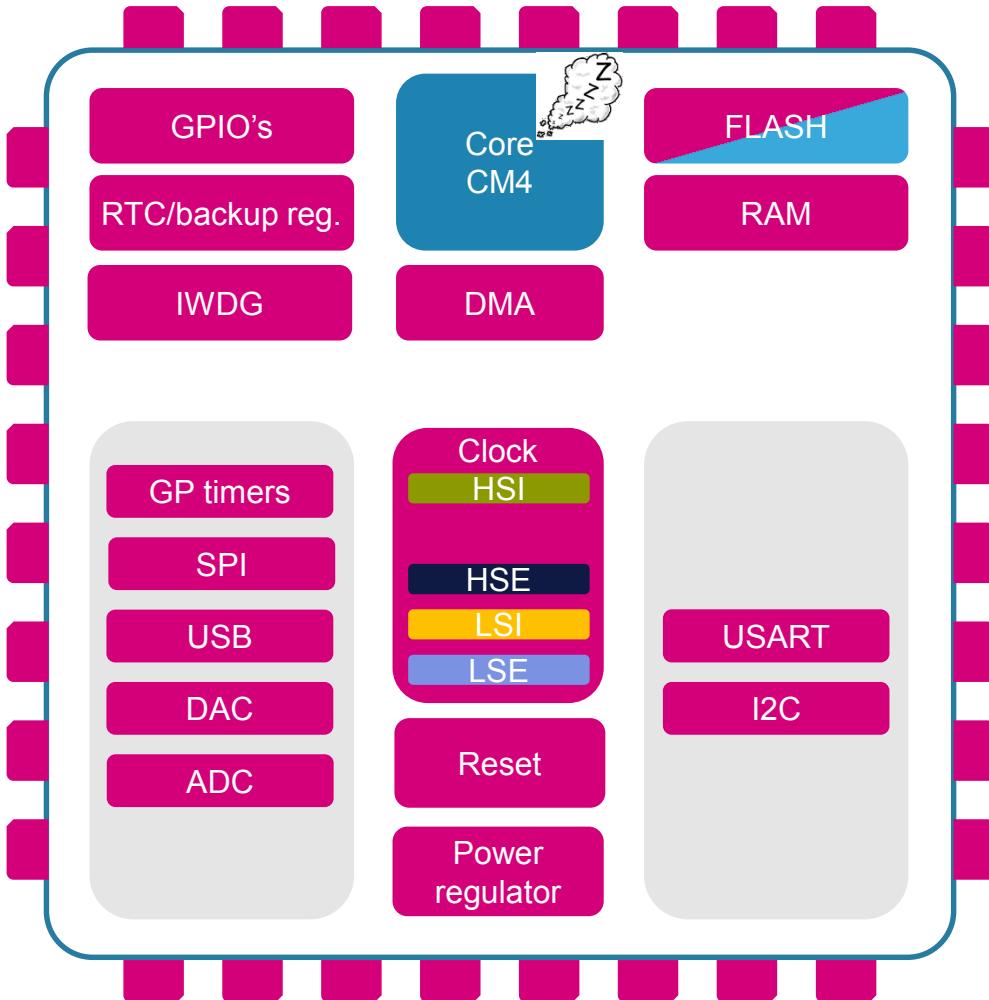
- Objective

- We use the EXTI setup from lab 1
- Learn how to setup SLEEP in HAL
- Create simple project with SLEEP mode with wake up on pin press

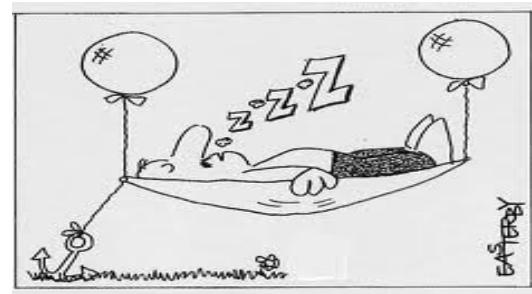
- Goal

- Use project from EXTI lab
- Learn how to setup the SLEEP in HAL, and which events can wake up MCU
- Verify the correct functionality by measuring consumption

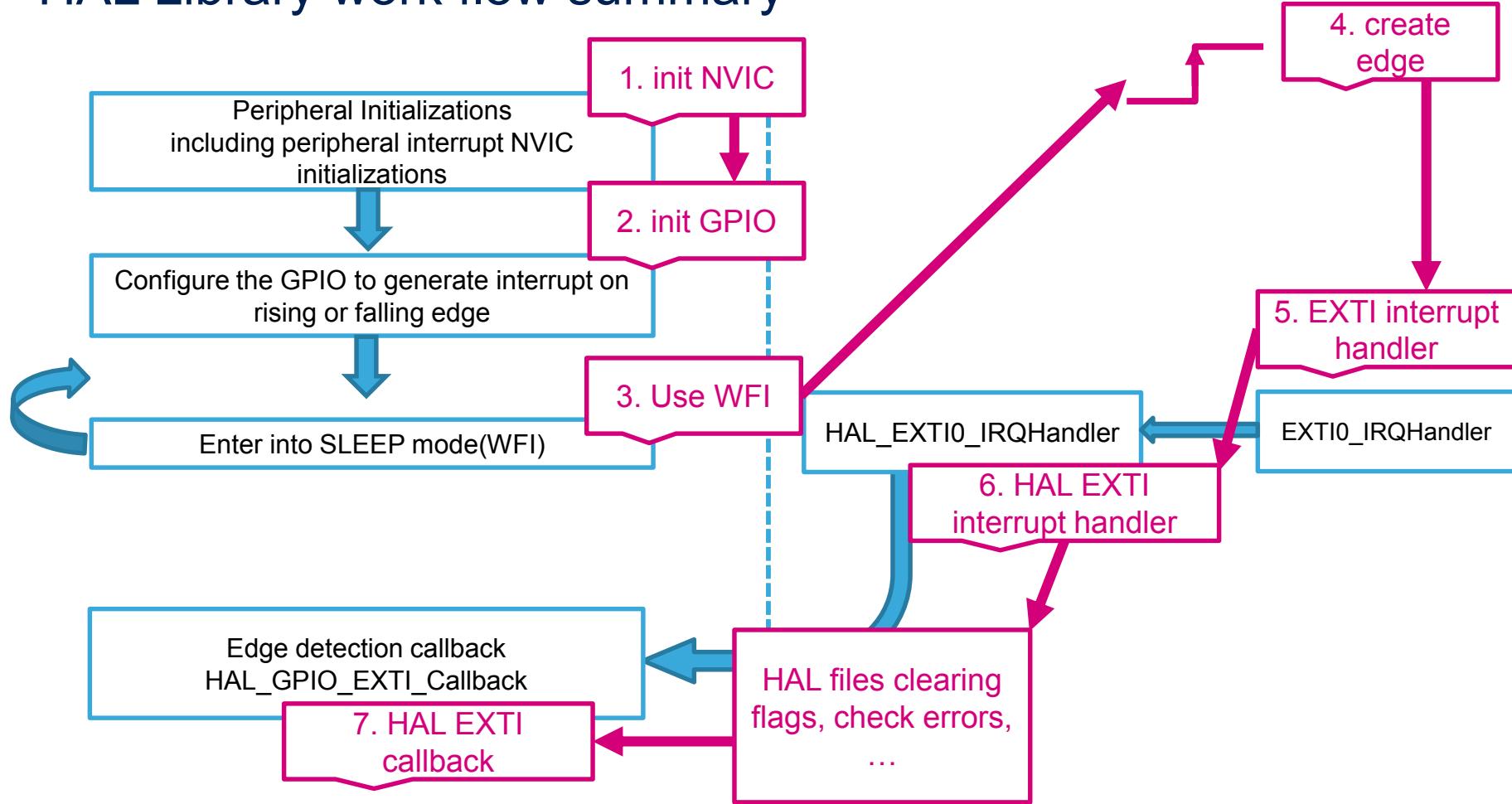
# SLEEP Mode



- Core is stopped
- Peripherals are running



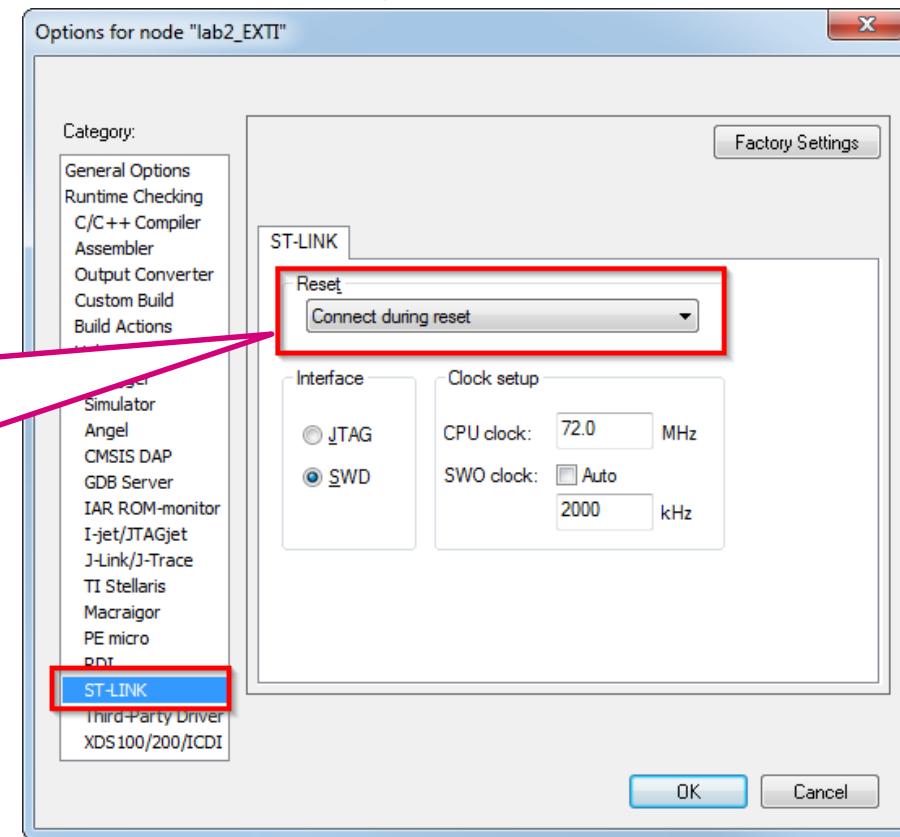
## HAL Library work flow summary



# Use SLEEP mode with EXTI

- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between /\* USER CODE BEGIN 3 \*/ and /\* USER CODE END 3 \*/ tags
- Function to enter SLEEP
  - HAL\_PWR\_EnterSLEEPMode(uint32\_t Regulator, uint8\_t SLEEPEntry)
  - We can measure consumption

To be able to reprogram the STM32 which is in LP mode, use connection during reset option



- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 3 */` and `/* USER CODE END 3 */` tags
- Function to enter SLEEP
  - `HAL_PWR_EnterSLEEPMode(uint32_t Regulator, uint8_t SLEEPEntry)`
  - We can measure consumption

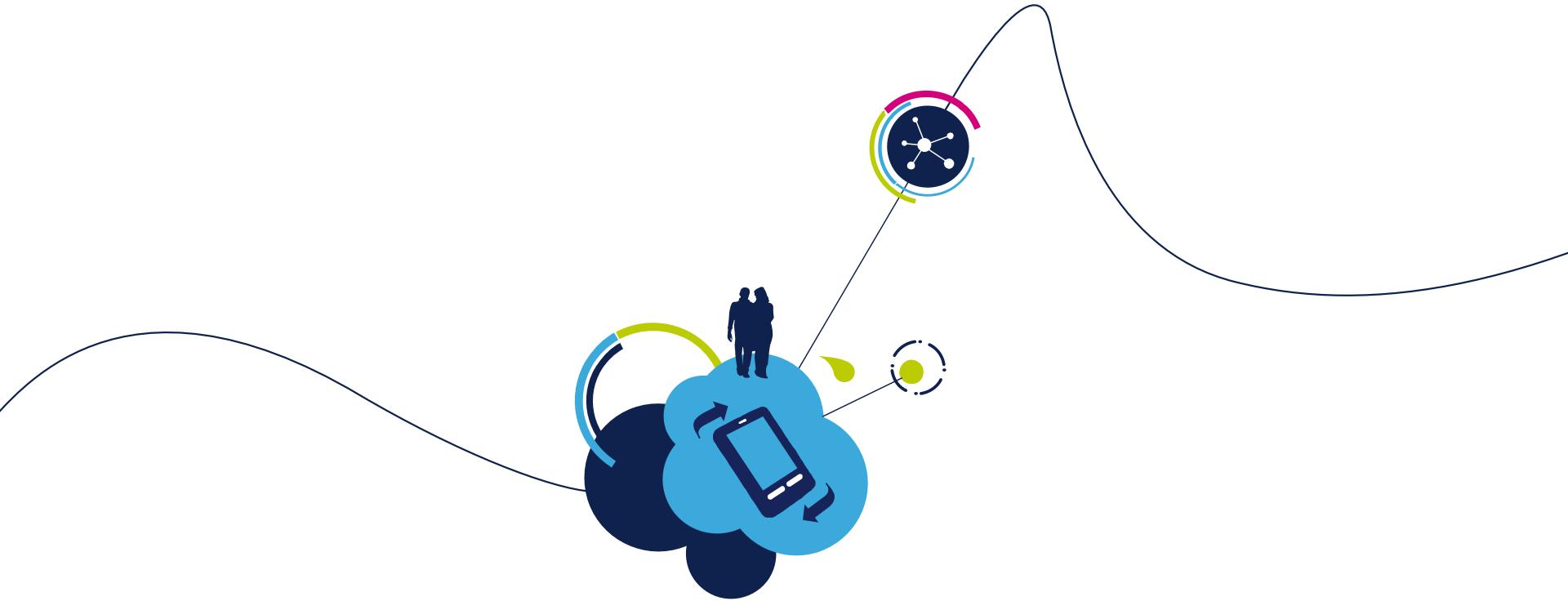
```
/* USER CODE BEGIN 3 */
/* Infinite loop */
while (1)
{
    HAL_Delay(1000);

    HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);
}
/* USER CODE END 3 */
```

- Consumption still to high?
  - Is STM32 really in SLEEP?
  - Is the Systick disabled?

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_Delay(1000);  
    HAL_SuspendTick();  
    HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON,PWR_SLEEPENTRY_WFI);  
    HAL_ResumeTick();  
}  
/* USER CODE END 3 */
```

- Is this better?



# Low Power mode STOP lab 4

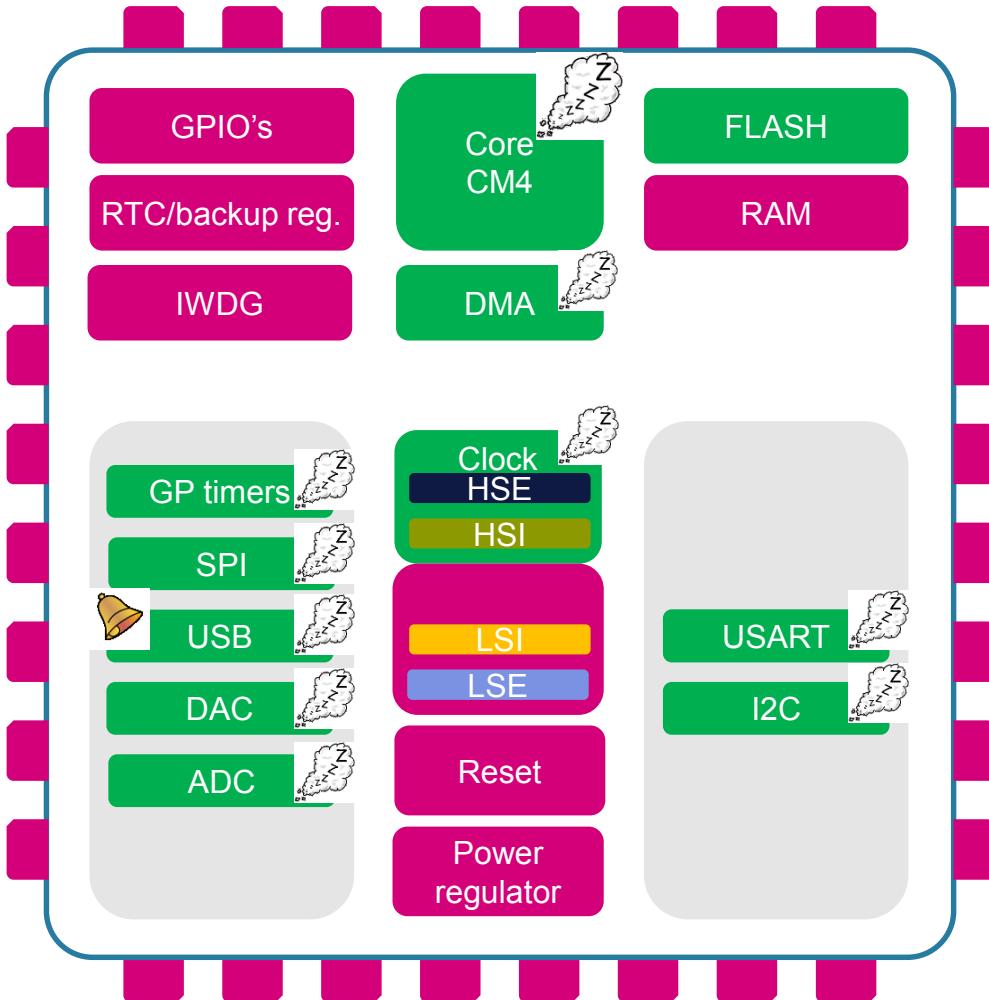
- Objective

- We use the EXTI setup from lab 1
- Learn how to setup STOP in HAL
- Create simple project with STOP mode with wake up on pin press

- Goal

- Use project from EXTI lab
- Learn how to setup the STOP in HAL, which events can wake up you
- Verify the correct functionality by measuring consumption

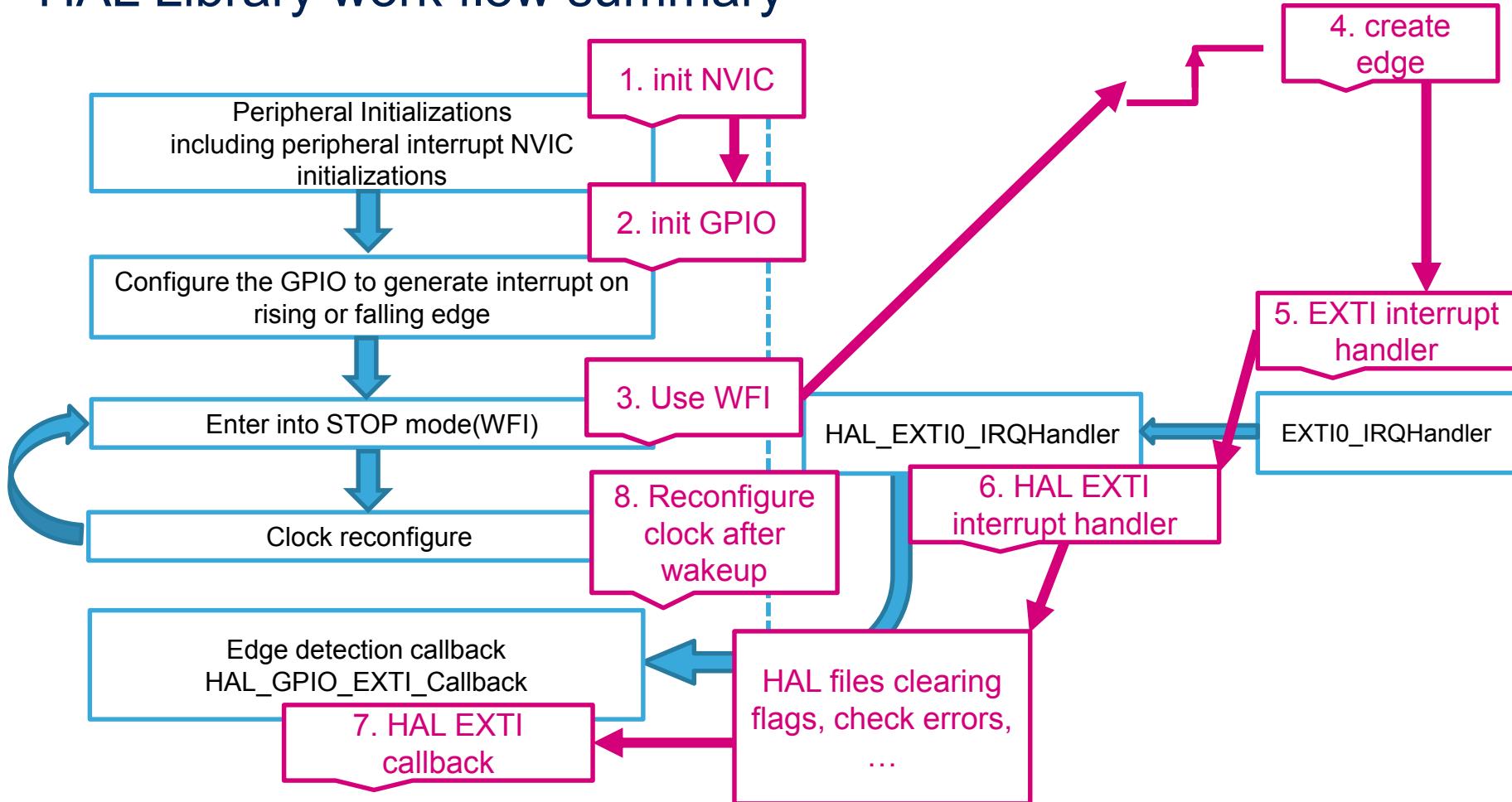
# STOP Mode



- Core is stopped
- HSE, MSI clocks are OFF
- SRAM and registers content is preserved
- Peripherals with HSI, LSI, LSE clock option can be ON
- GPIO's keep their setup



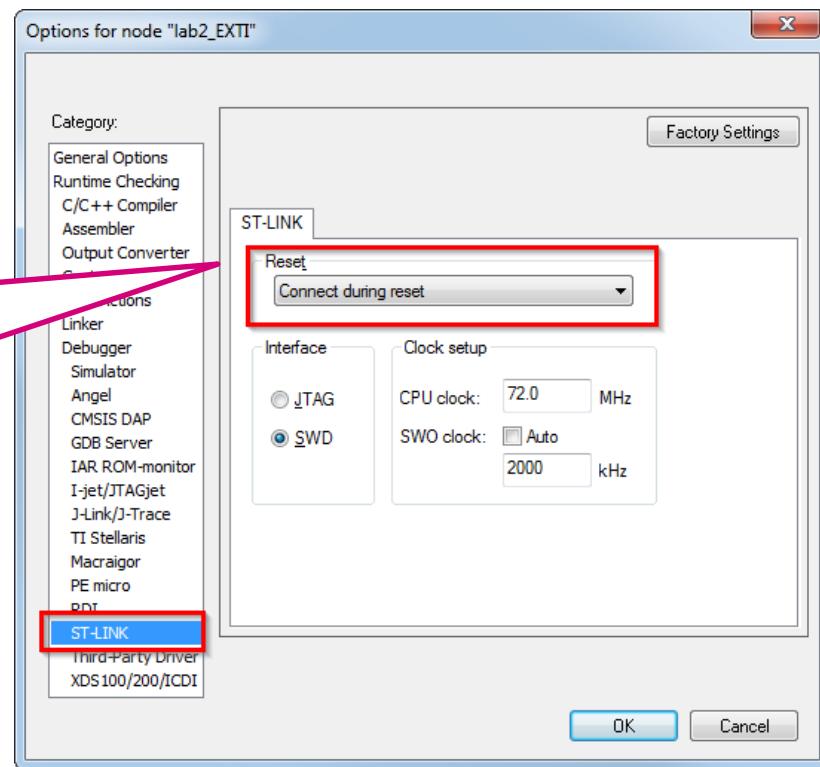
# HAL Library work flow summary



# Use STOP mode with EXTI

- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 3 */` and `/* USER CODE END 3 */` tags
- Function to enter SLEEP
  - `HAL_PWR_EnterSTOPMode(uint32_t Regulator, uint8_t STOPEntry)`
  - `HAL_PWREx_EnterUnderDriveSTOPMode(uint32_t Regulator, uint8_t STOPEntry)`
  - We can measure consumption

To be able to reprogram the STM32 which is in LP mode, use connection during reset option

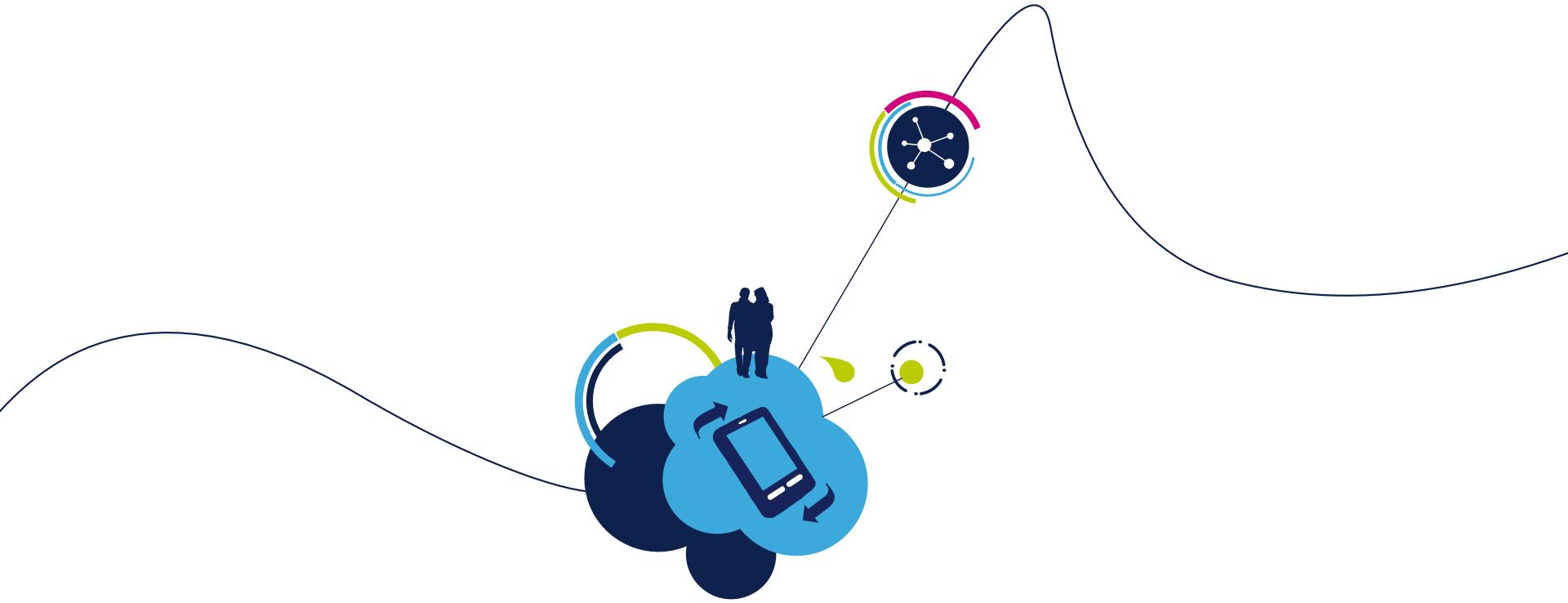


- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 3 \*/* and */\* USER CODE END 3 \*/* tags
- Function to enter SLEEP
  - HAL\_PWR\_EnterSTOPMode(uint32\_t Regulator, uint8\_t STOPEntry)
  - HAL\_PWREx\_EnterUnderDriveSTOPMode(uint32\_t Regulator, uint8\_t STOPEntry)
  - We can measure consumption

```
/* USER CODE BEGIN 3 */
/* Infinite loop */
while (1)
{
    HAL_Delay(1000);
    HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,PWR_STOPENTRY_WFI);
    SystemClock_Config();
}
/* USER CODE END 3 */
```

- Or different function (for STM32F42X/43X)

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_Delay(1000);  
    HAL_PWREx_EnterUnderDriveSTOPMode(PWR_LOWPOWERREGULATOR_UNDERDRIVE_ON,PWR_STOPENTRY_WFI);  
    SystemClock_Config();  
}  
/* USER CODE END 3 */
```



# Low Power mode STANDBY lab 5

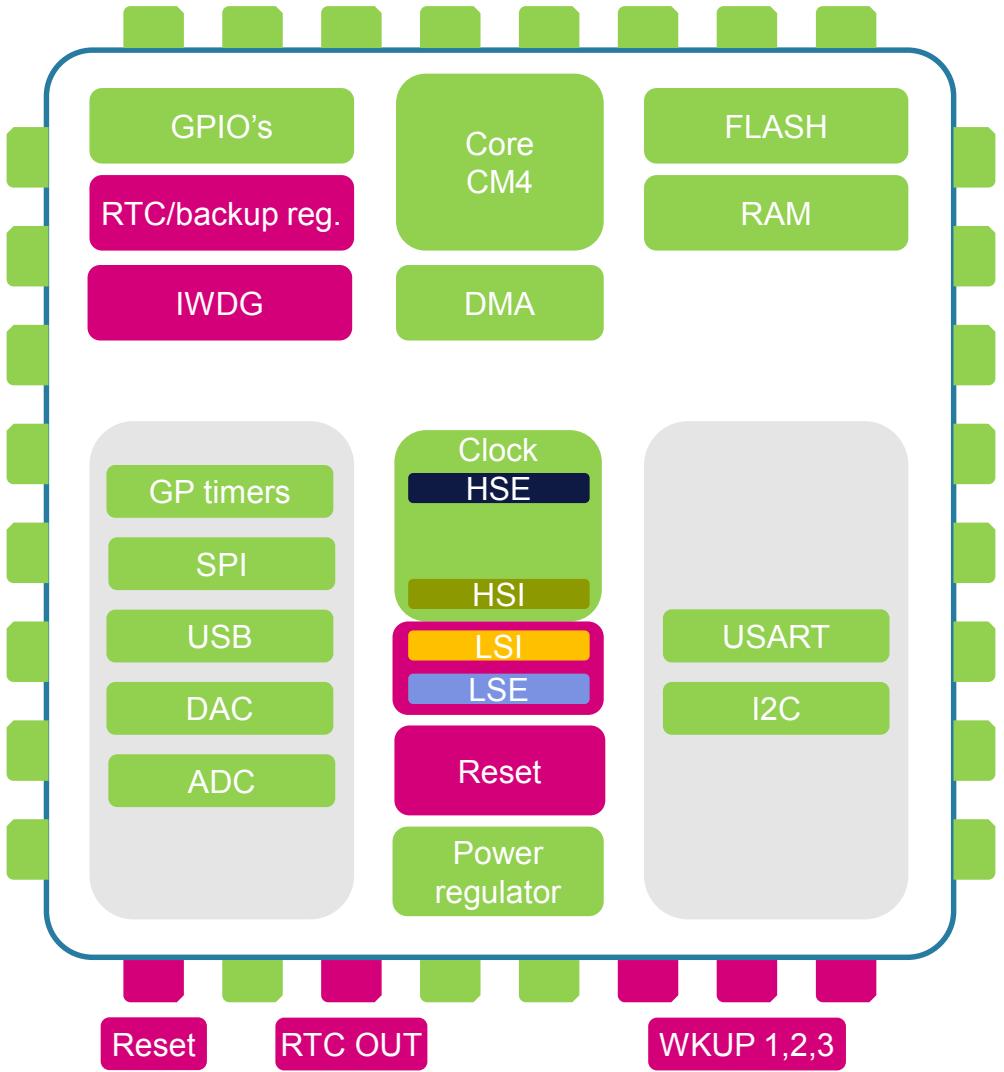
- Objective

- For this lab create CubeMX project
- For testing purpose enable LED on PG14
- Learn how to setup STANDBY in HAL
- Create simple project with STANDBY mode with wake up on pin press

- Goal

- Learn how to setup the STANDBY in HAL, which events can wake up you
- Verify the correct functionality by measuring consumption

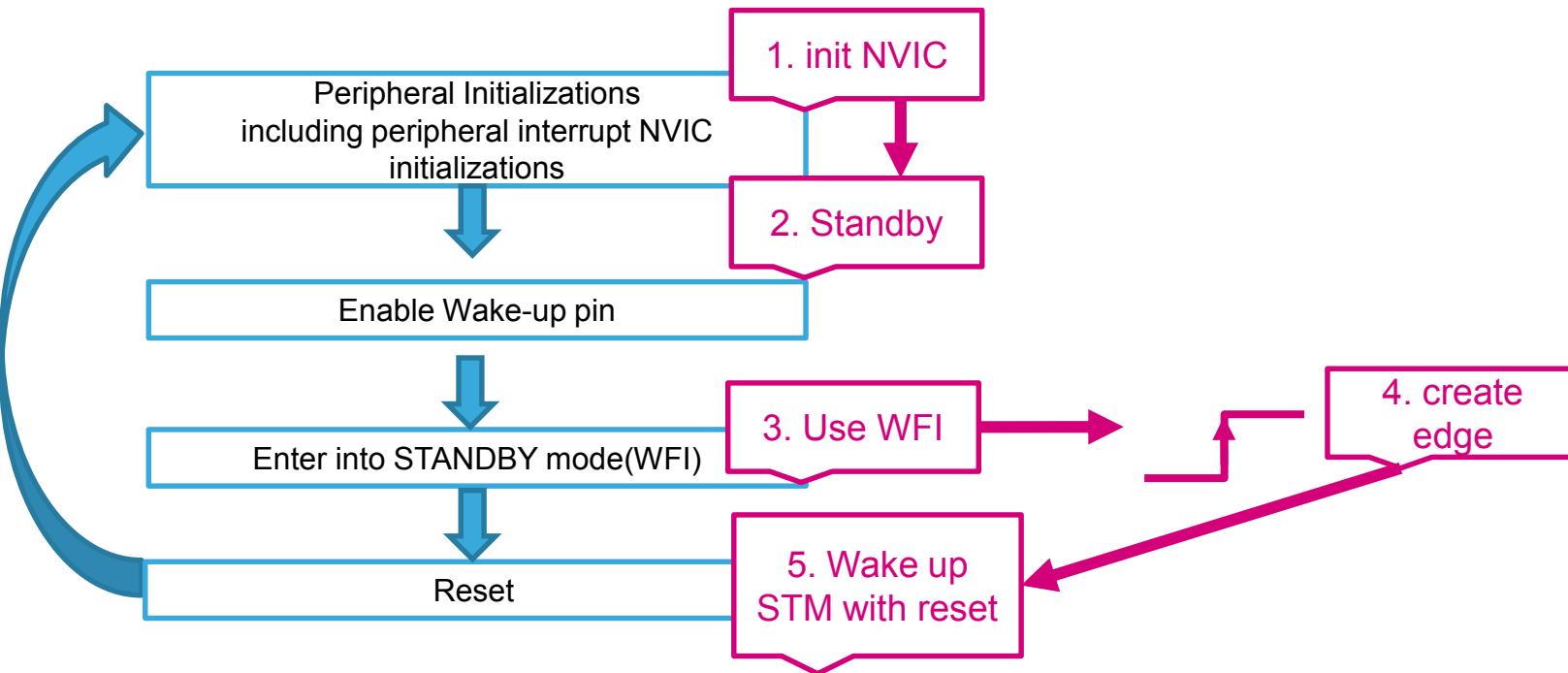
# STANDBY Mode



- Core and all peripherals are OFF, except RTC and IWDG if enabled
- HSE, HSI clocks are OFF, LSI LSE can be ON
- SRAM and registers content is lost, except RTC, and standby circuitry
- GPIO's are in high Z, except Reset, RTC OUT and WKUP 1,2,3

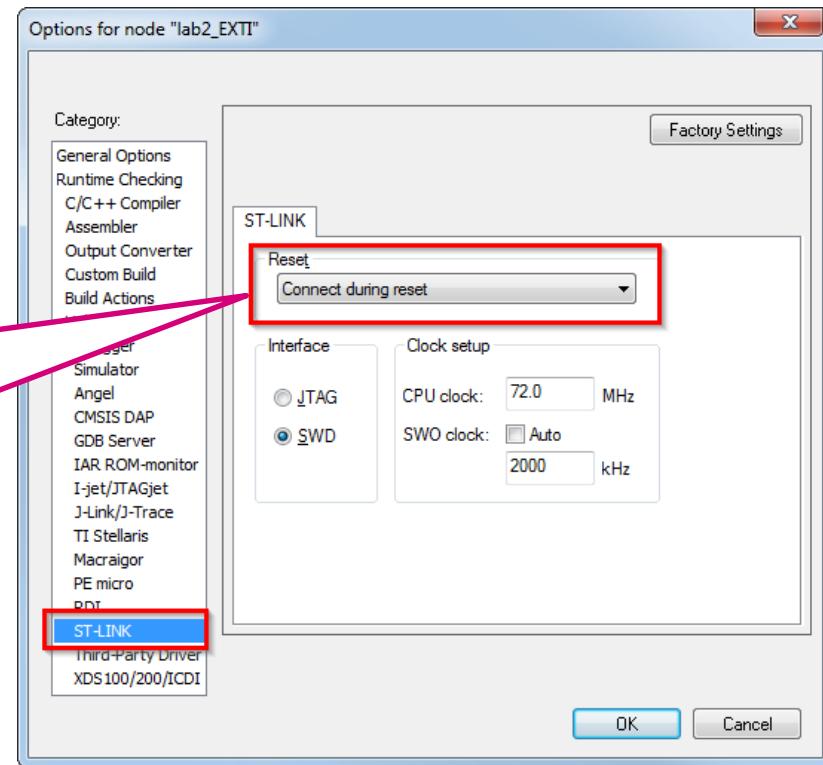


## HAL Library work flow summary



- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 3 */` and `/* USER CODE END 3 */` tags
- For Wake up we need to setup wake up pin
  - `HAL_PWR_EnableWakeUpPin(uint32_t WakeUpPinx)`
- Function to enter STANDBY
  - `HAL_PWR_EnterSTANDBYMode();`
  - We can measure consumption

To be able to reprogram the STM32 which is in LP mode, use connection during reset option



- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 3 \*/* and */\* USER CODE END 3 \*/* tags
- Function to enter SLEEP
  - HAL\_PWR\_EnterSTOPMode(uint32\_t Regulator, uint8\_t STOPEntry)
  - HAL\_PWREx\_EnterUnderDriveSTOPMode(uint32\_t Regulator, uint8\_t STOPEntry)
  - We can measure consumption

```
/* USER CODE BEGIN 3 */
/* Infinite loop */
while (1)
{
    HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_14);
    HAL_Delay(2000);
    HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN1);
    HAL_PWR_EnterSTANDBYMode();
}
/* USER CODE END 3 */
```

- We cannot go into STANDBY again?
- Try to clear wake up flag
  - `_HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);`

```
/* USER CODE BEGIN 2 */  
  
_HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);  
/* USER CODE END 2 */
```



# Data transfer over DMA lab 6

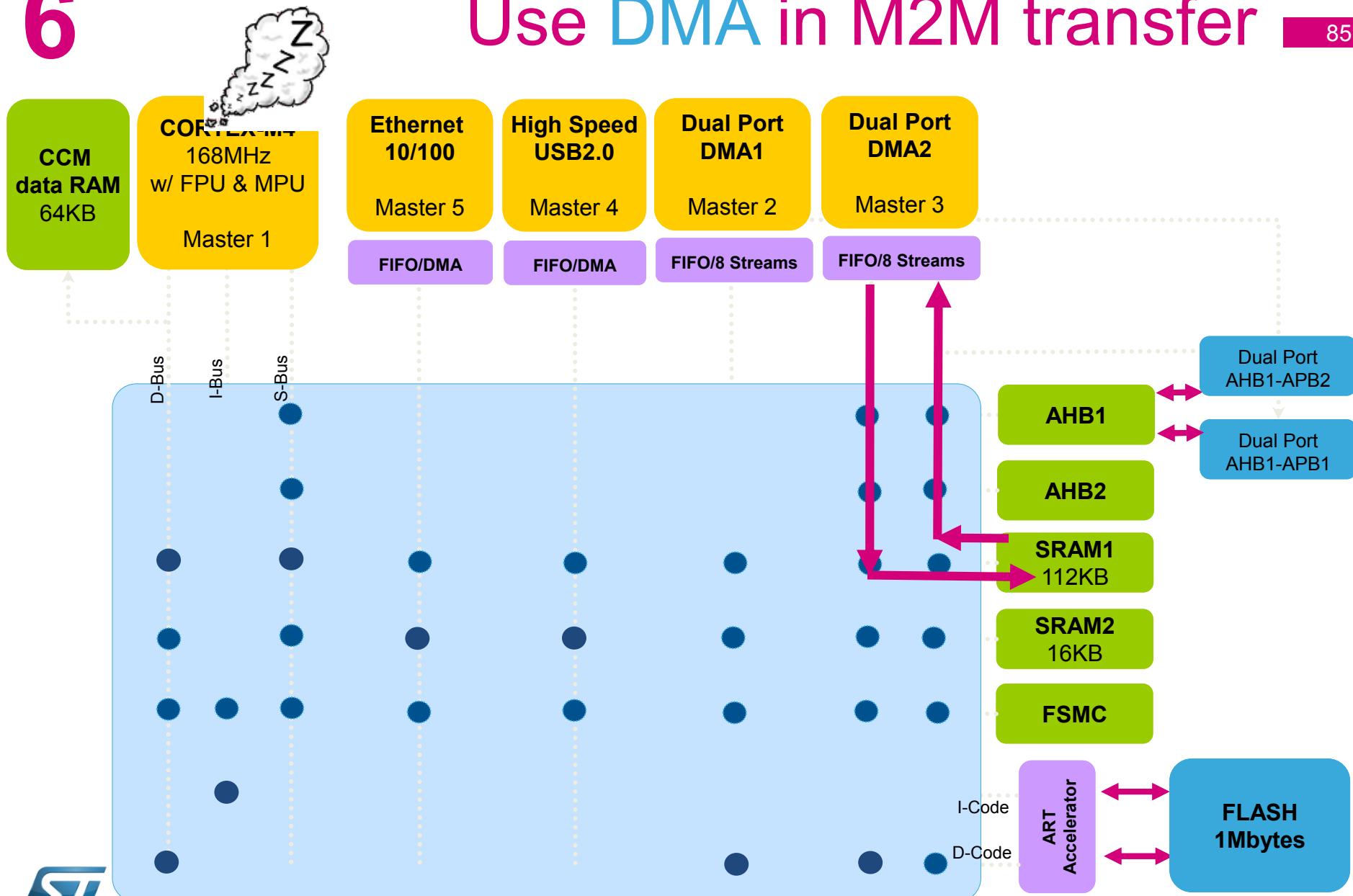
- Objective

- Learn how to setup DMA transfer in CubeMX
- Create simple DMA memory to memory transfer from RAM to RAM

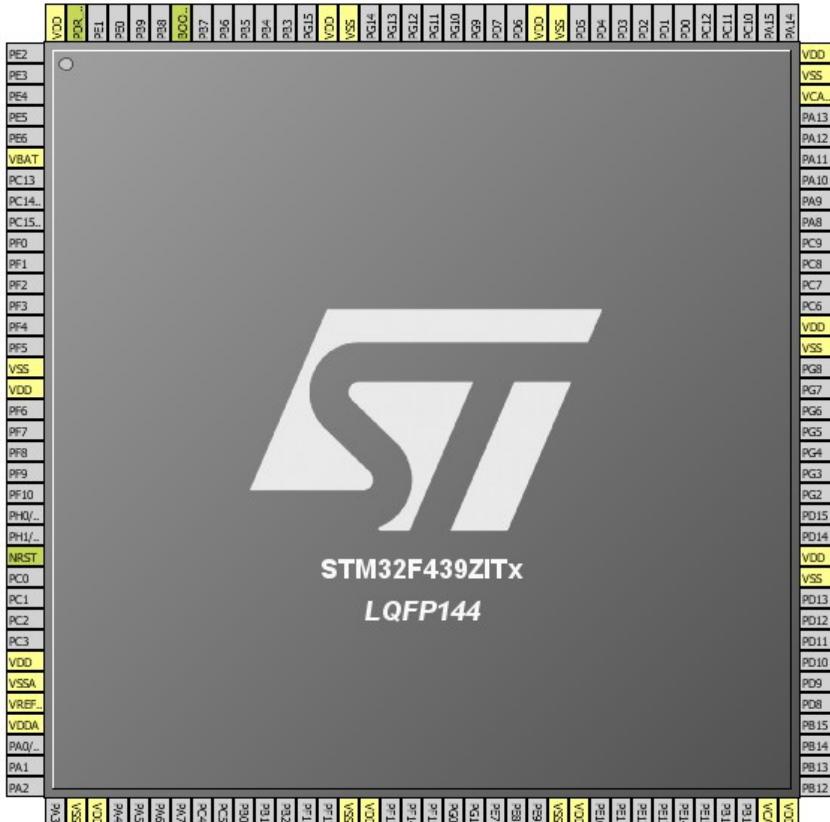
- Goal

- Use CubeMX and Generate Code with DMA
- Learn how to setup the DMA in HAL
- Verify the correct functionality by comparing transferred buffers

# Use DMA in M2M transfer

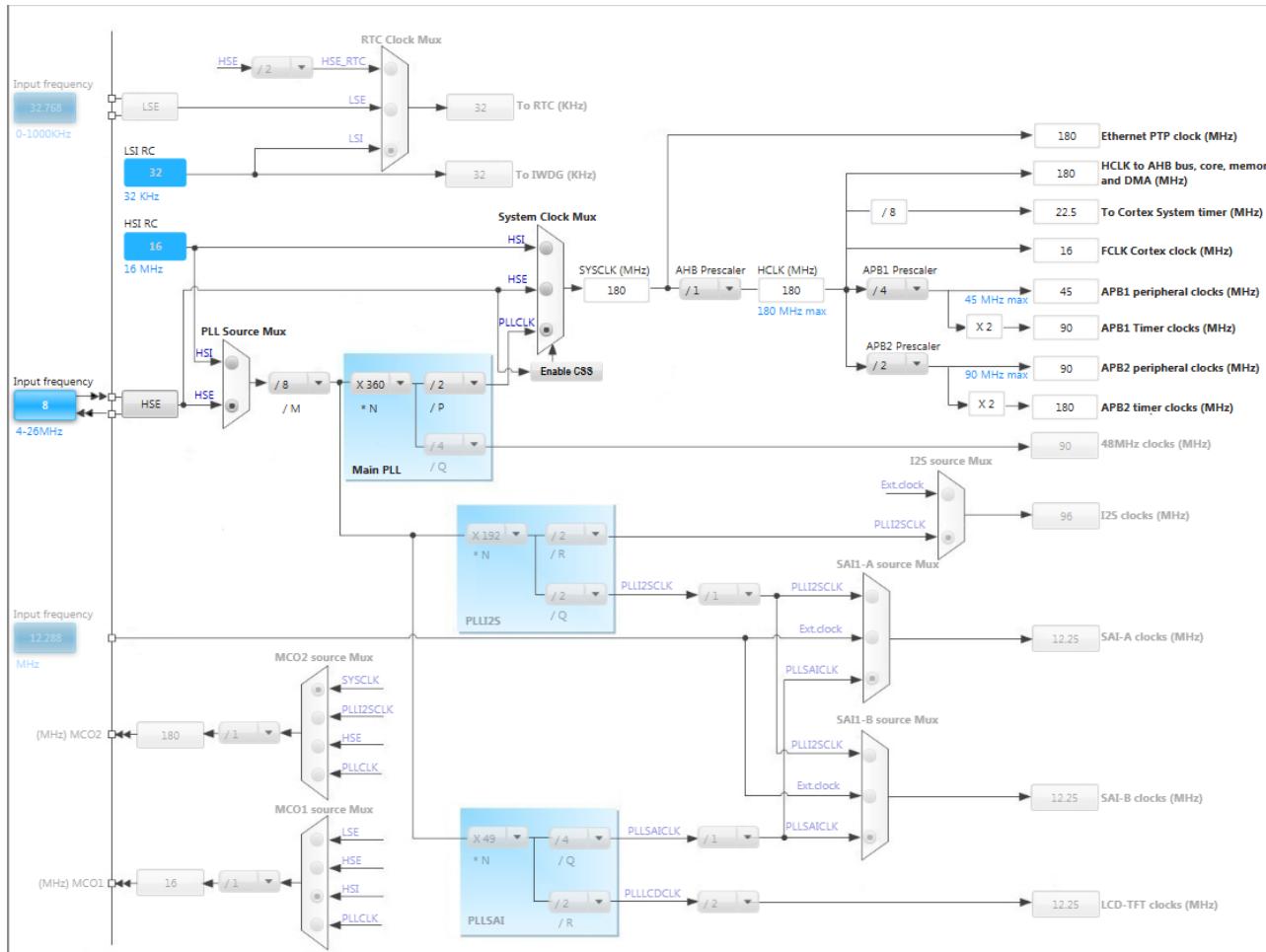


- Create project in CubeMX
    - Menu > File > New Project
    - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
  - For DMA we don't need to configure any pins



# Use DMA in M2M transfer

- In order to run on maximum frequency, setup clock system
- Details in lab 0

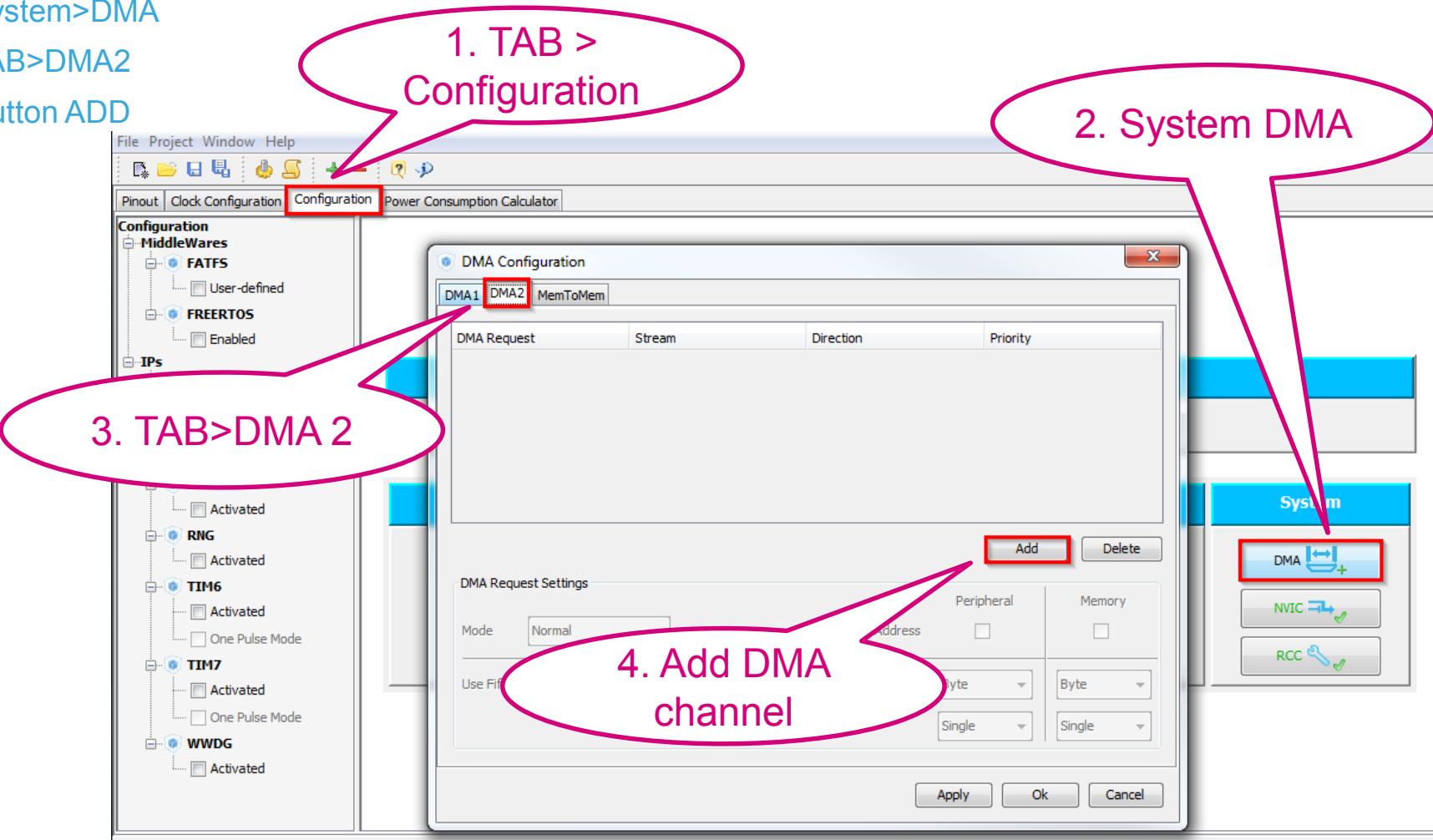


# 6

# Use DMA in M2M transfer

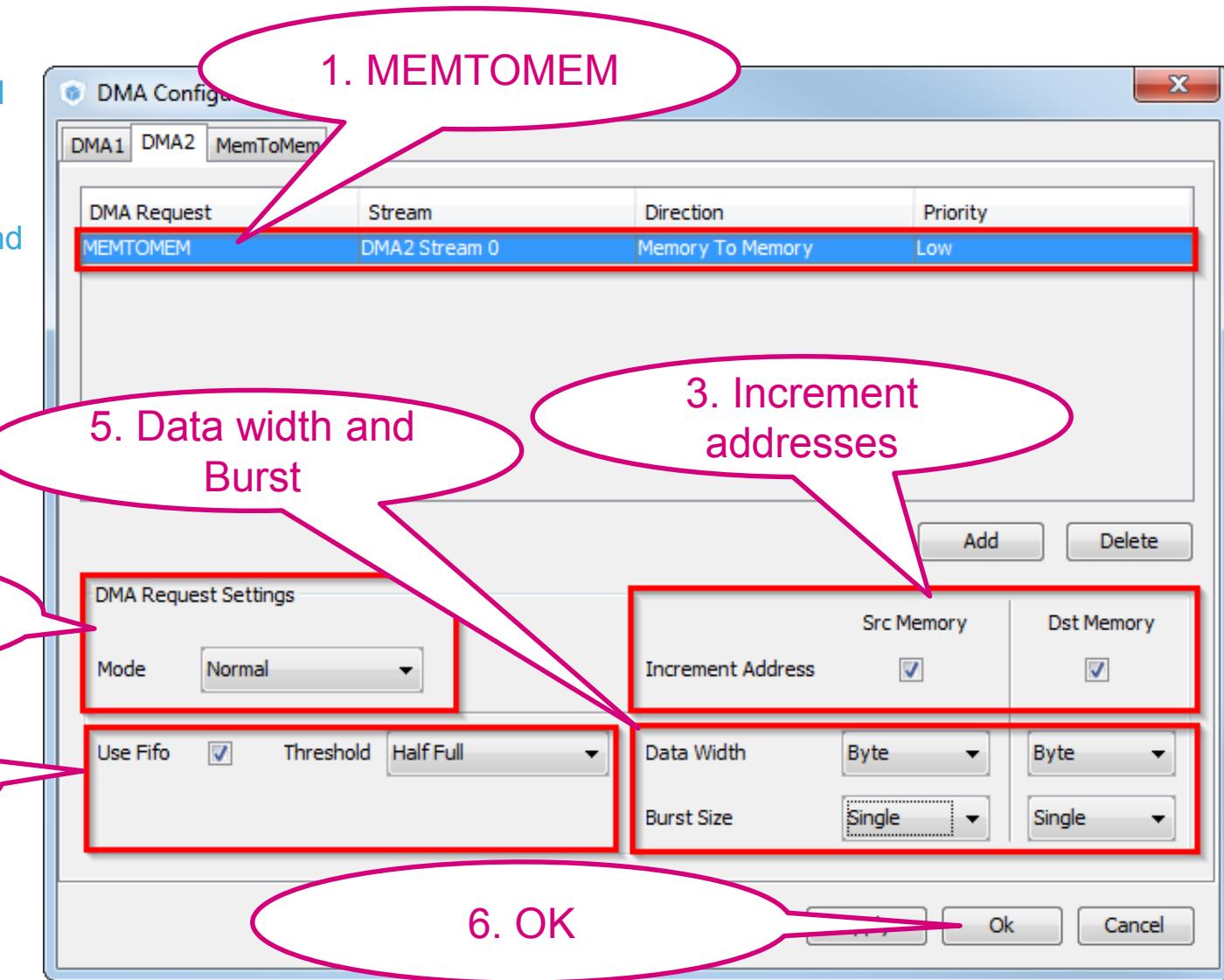
88

- DMA configuration
  - TAB>Configuration
  - System>DMA
  - TAB>DMA2
  - Button ADD



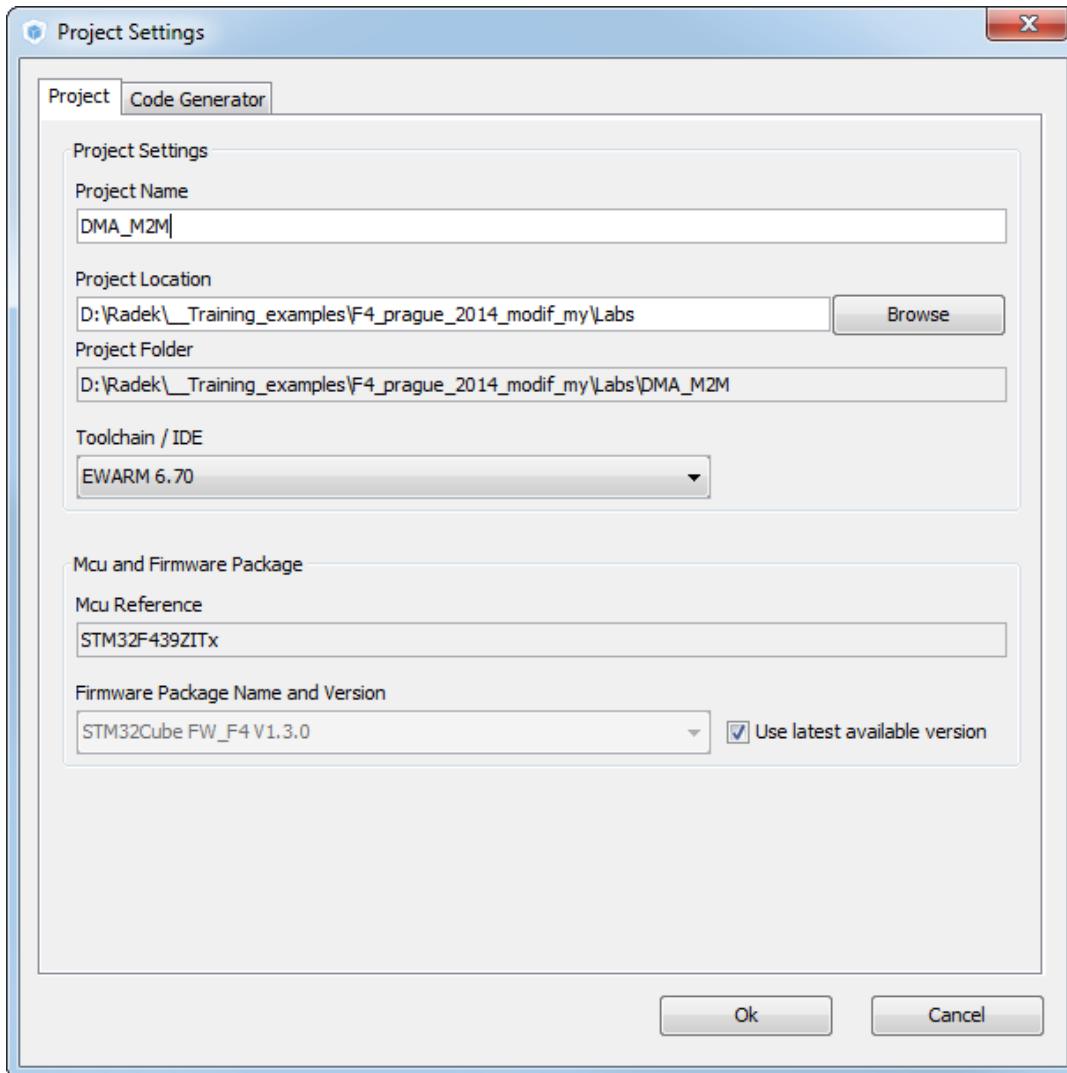
# Use DMA in M2M transfer

- DMA configuration
  - Select MEMTOMEM DMA request
  - Normal mode
  - Increment source and destination address
  - FIFO setup
  - Byte data width
  - Burst size
  - Button OK



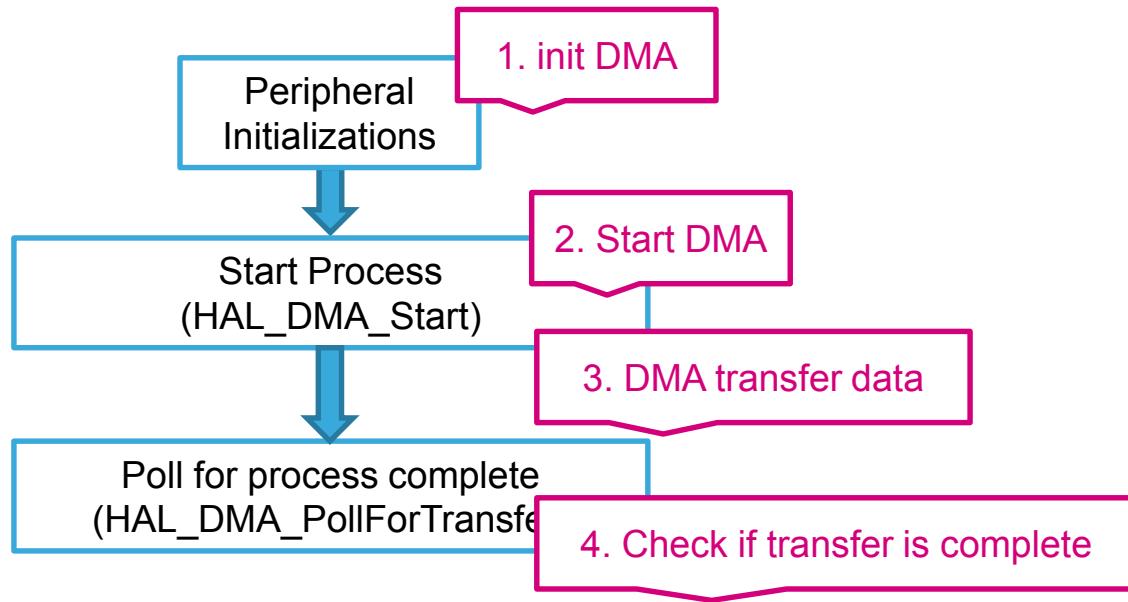
# Use DMA in M2M transfer

- 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



# Use DMA in M2M transfer

- Start process DMA (same for TIM, ADC)
  - Non blocking start process
  - The end of the process must be checked by polling



- Return values

- Most of CubeMX functions have return values, which indicate, if operation was successful, timeout occurs or function end with error
- Is recommended handle this return values to be sure that program working as expected

Ex: Poll for process complete  
(HAL\_DMA\_PollForTransfer)

HAL\_OK

HAL\_ERROR

HAL\_BUSY

DMA transfer was successfully finished and data was transferred to destination without error

- Return values

- Most of CubeMX functions have return values, which indicate, if operation was successful, timeout occurs or function end with error
- Is recommended handle this return values to be sure that program working as expected

Ex: Poll for process complete  
(HAL\_DMA\_PollForTransfer)

HAL\_OK

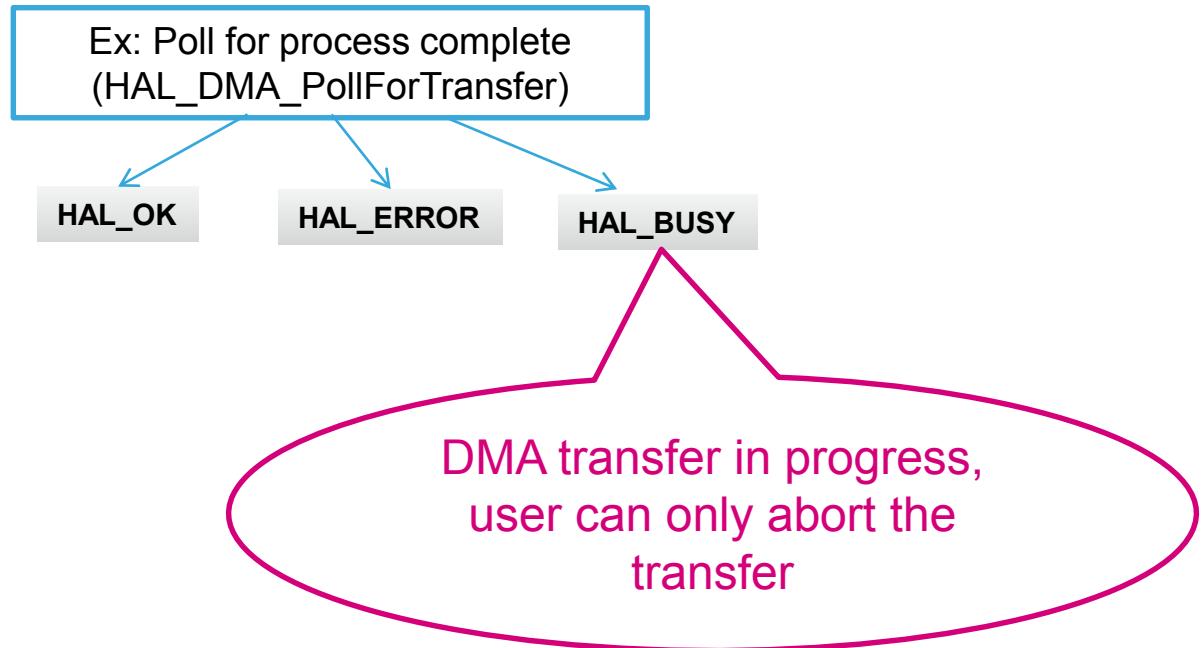
HAL\_ERROR

HAL\_BUSY

Error occurs during DMA  
transfer you use  
`HAL_DMA_GetError` for  
details what happened

- Return values

- Most of CubeMX functions have return values, which indicate, if operation was successful, timeout occurs or function end with error
- Is recommended handle this return values to be sure that program working as expected



- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 2 */` and `/* USER CODE END 2 */` tags
- HAL functions for DMA
  - `HAL_DMA_Start(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)`
  - `HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t CompleteLevel, uint32_t Timeout)`

- We create two buffers
  - One with source data
  - Second as destination buffer

```
/* 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 */
```

- HAL\_DMA\_Start start the M2M data transfer
- HAL\_DMA\_PollForTransfer check if the transfer ends successfully

```
/* 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 */
```



# Data transfer over DMA with interrupt

## lab 7

# 7 Use DMA M2M transfer with interrupt

99

- Objective

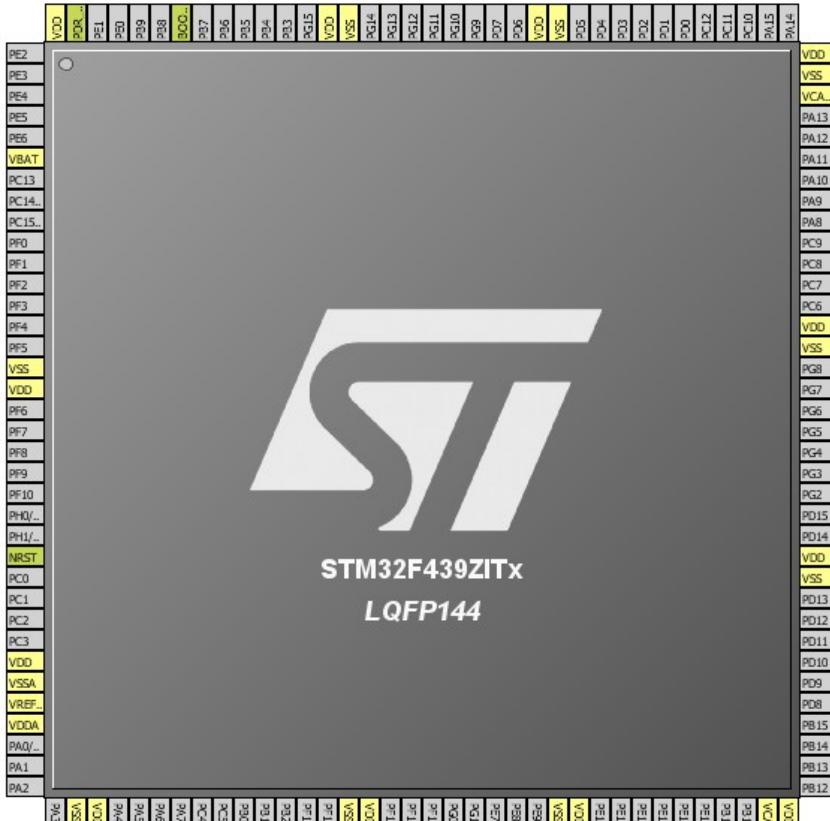
- Learn how to setup DMA transfer with interrupt in CubeMX
- Create simple DMA memory to memory transfer from RAM to RAM

- Goal

- Use CubeMX and Generate Code with DMA
- Learn how to setup the DMA in HAL
- Verify the correct functionality by comparing transferred buffers

# 7 Use DMA M2M transfer with interrupt

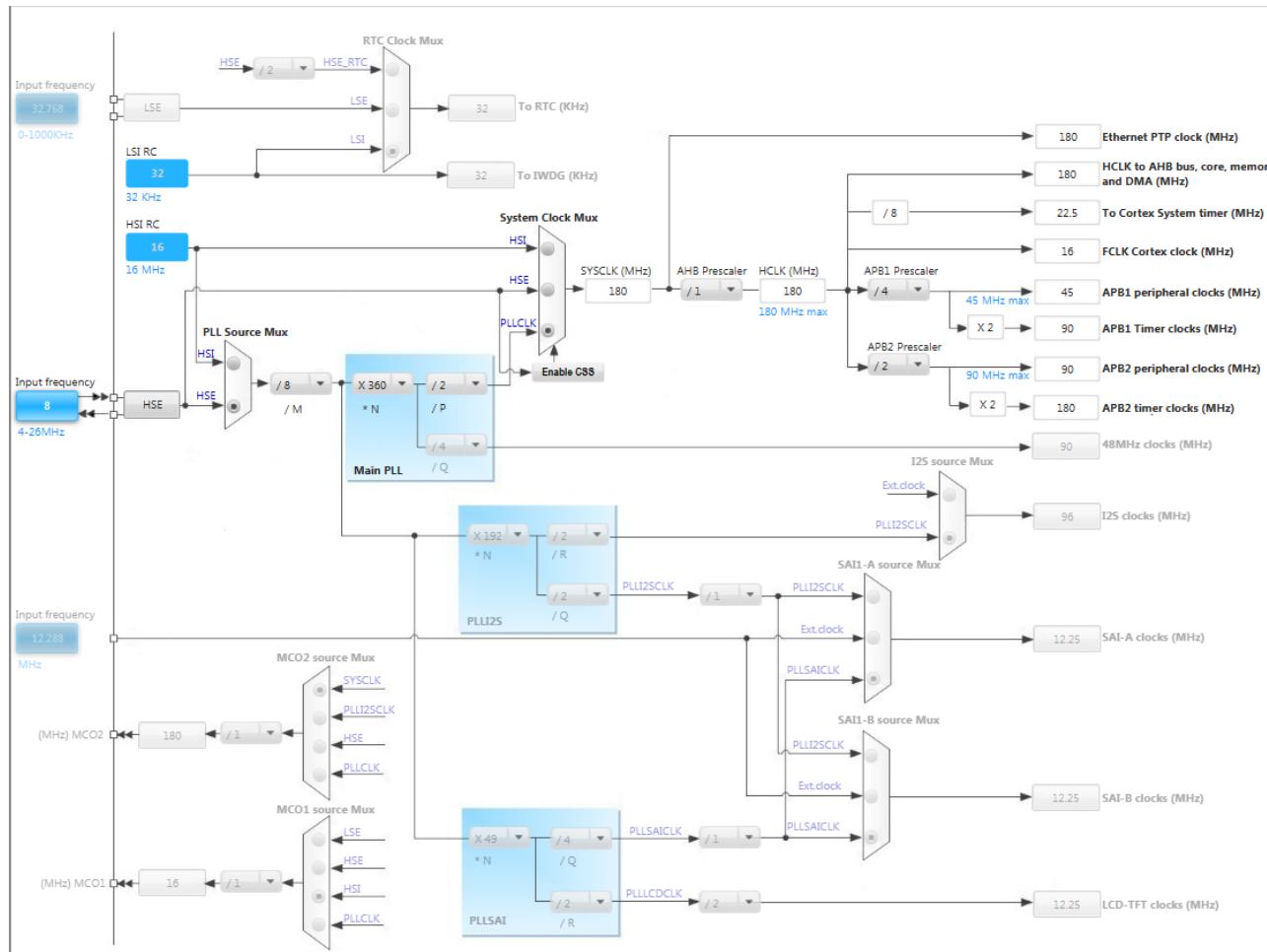
- Create project in CubeMX
    - Menu > File > New Project
    - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
  - For DMA we don't need to configure any pins



# 7 Use DMA M2M transfer with interrupt

101

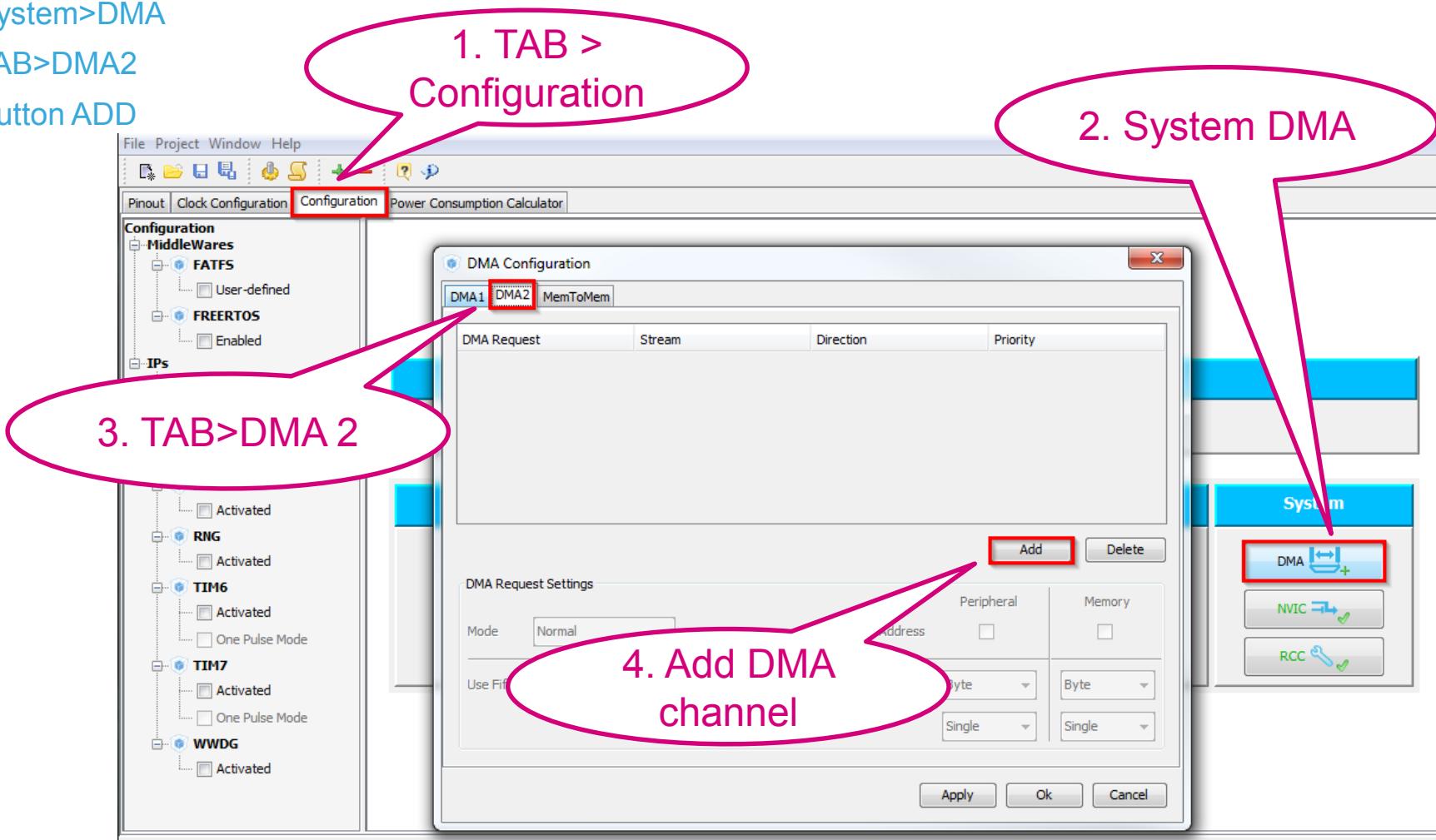
- In order to run on maximum frequency, setup clock system
- Details in lab 0



# 7 Use DMA M2M transfer with interrupt

102

- DMA configuration
  - TAB>Configuration
  - System>DMA
  - TAB>DMA2
  - Button ADD

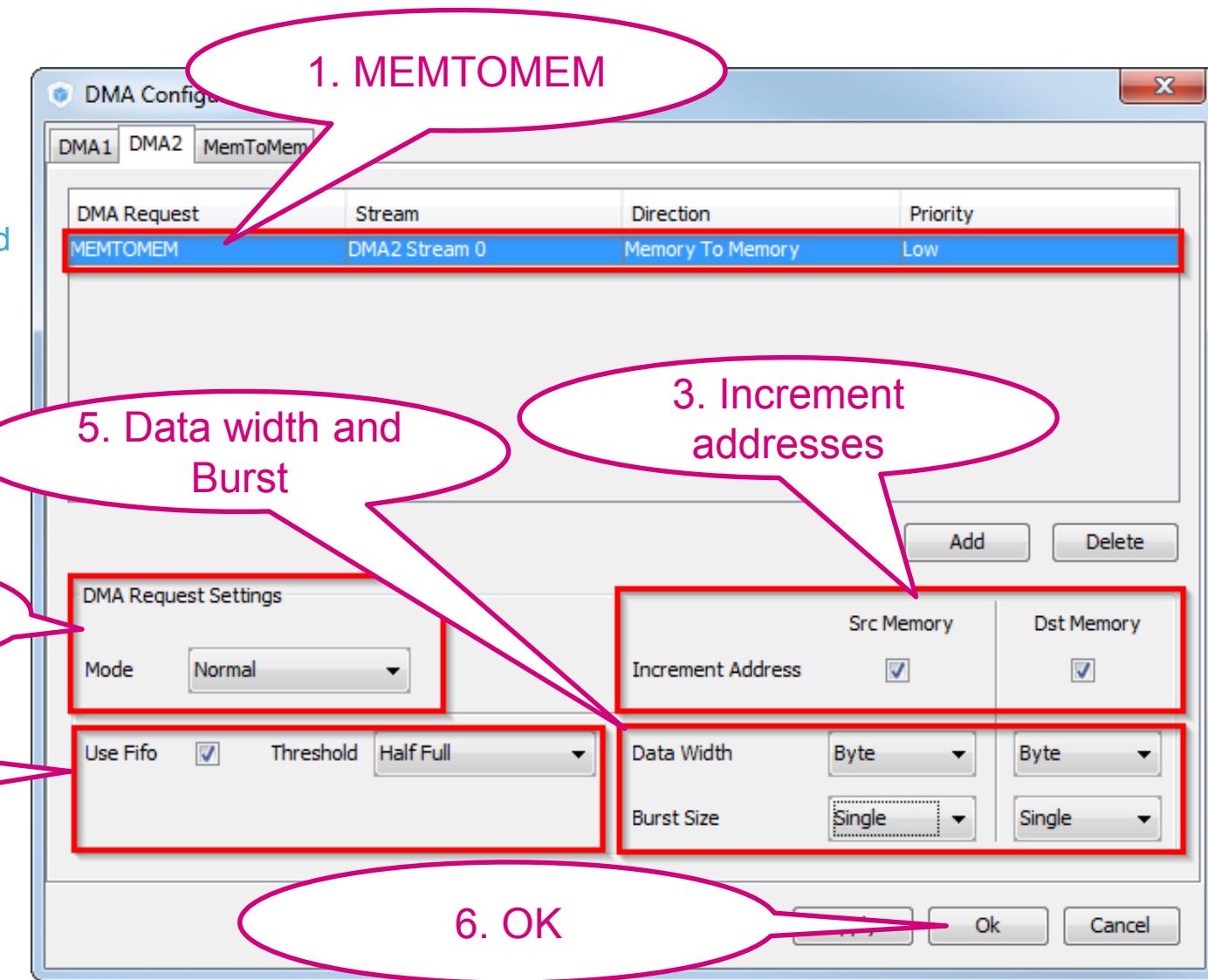


# 7 Use DMA M2M transfer with interrupt

103

- DMA configuration

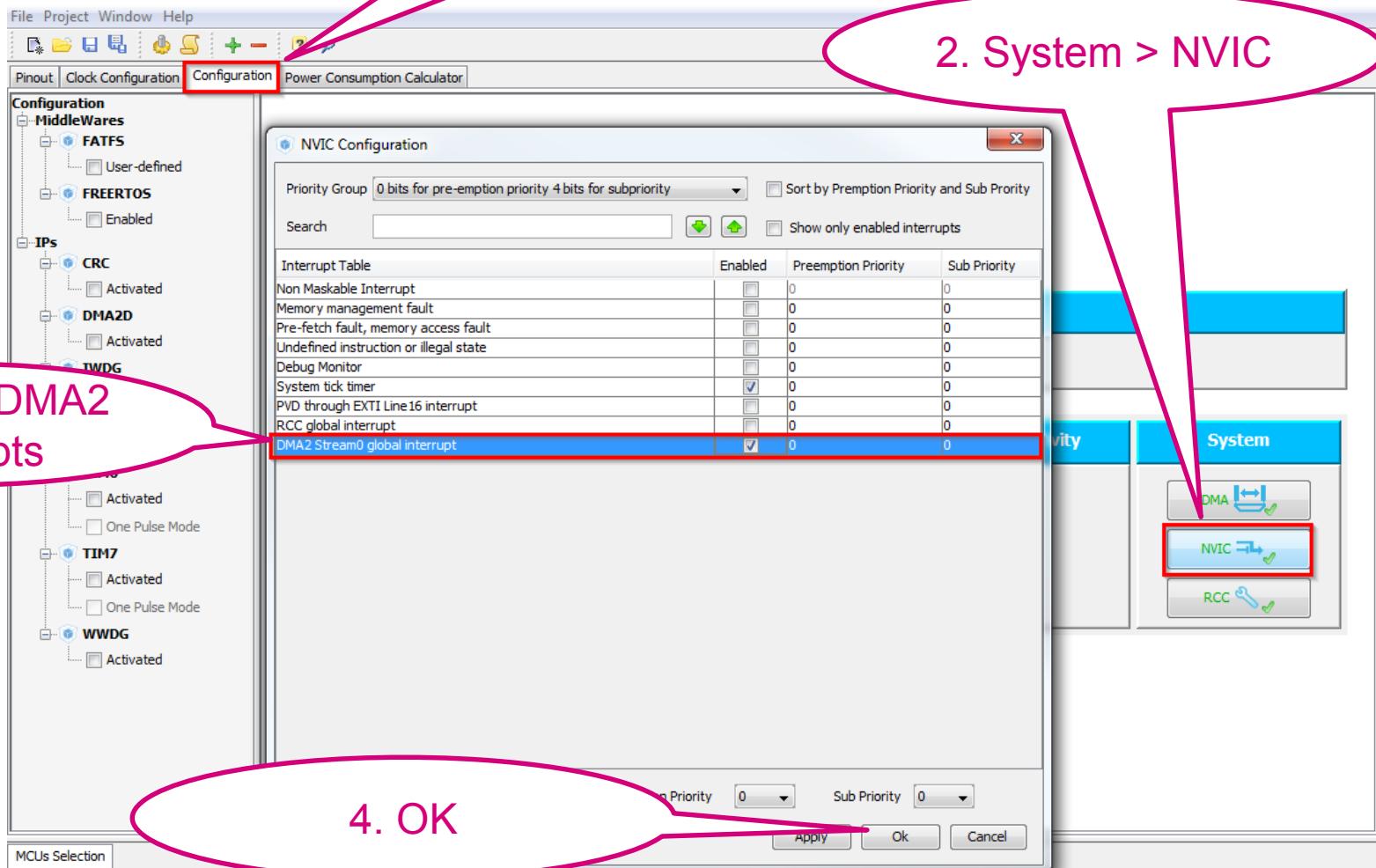
- Select MEMTOMEM DMA request
- Normal mode
- Increment source and destination address
- FIFO setup
- Byte data width
- Burst size
- Button OK



# 7 Use DMA M2M transfer with interrupt

104

- DMA configuration
  - System > NVIC
  - Enable DMA2 Stream interrupt
  - Button OK



# 7 Use DMA M2M transfer with interrupt

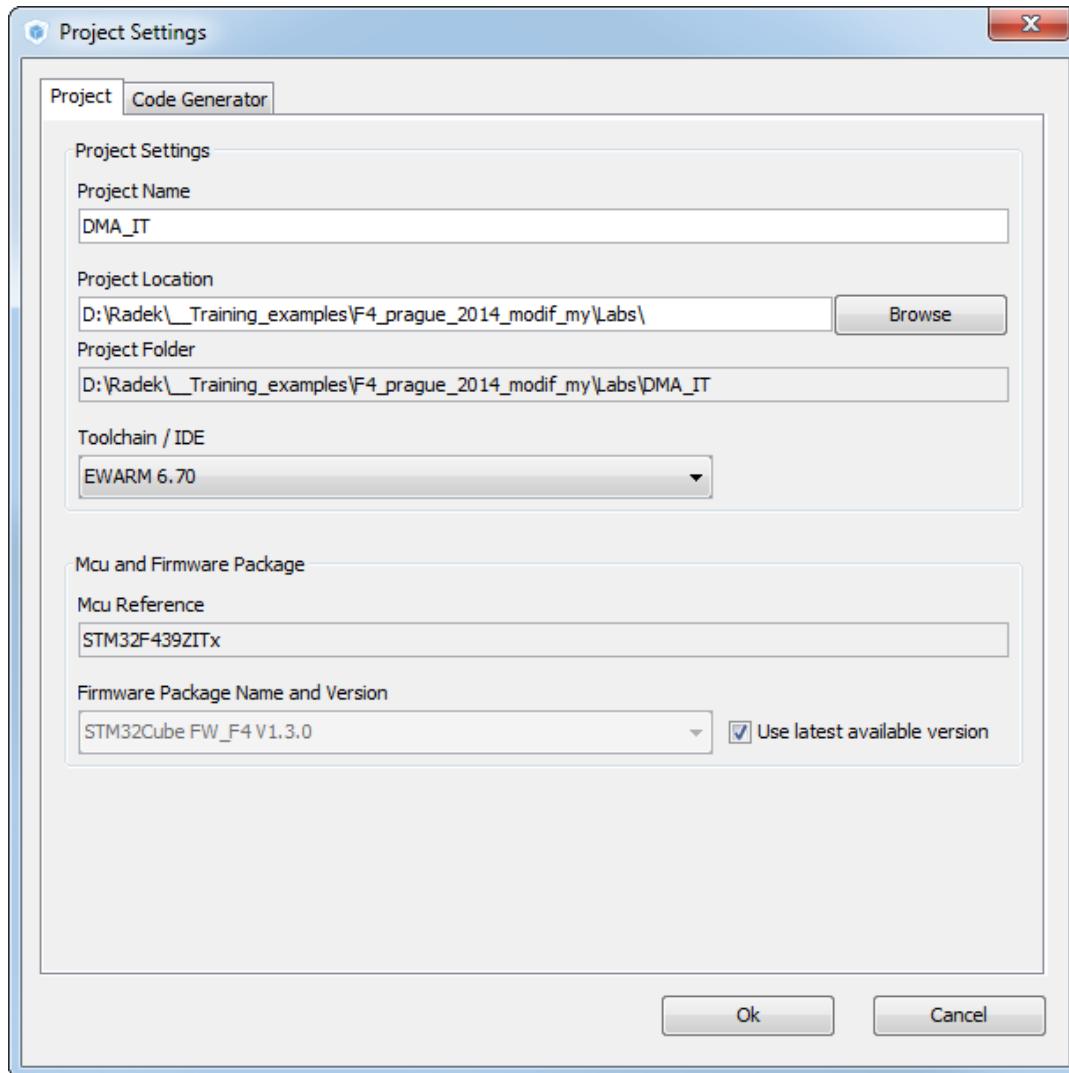
105

- 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

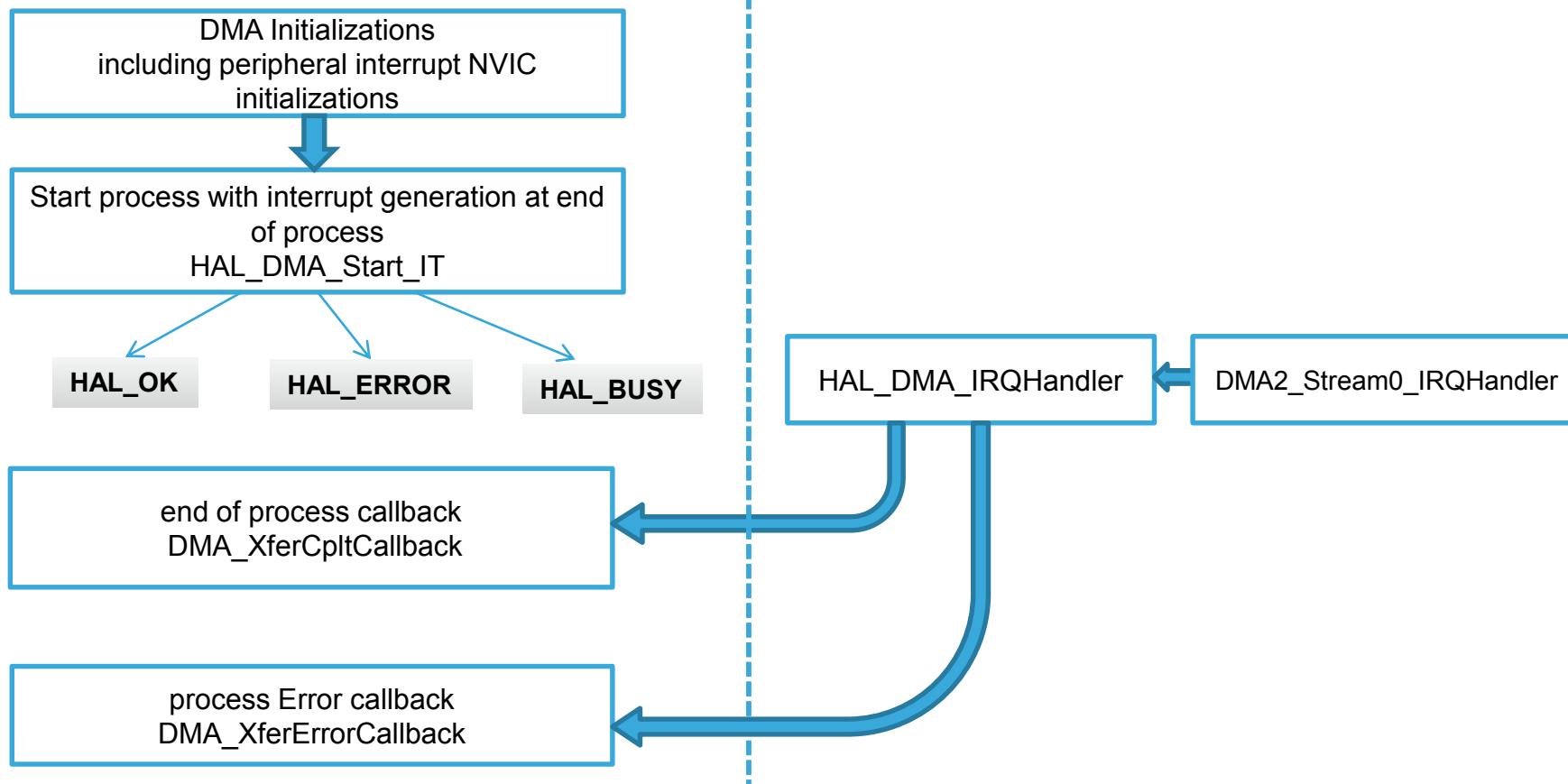
- Menu > Project > Generate Code



# 7 Use DMA M2M transfer with interrupt

106

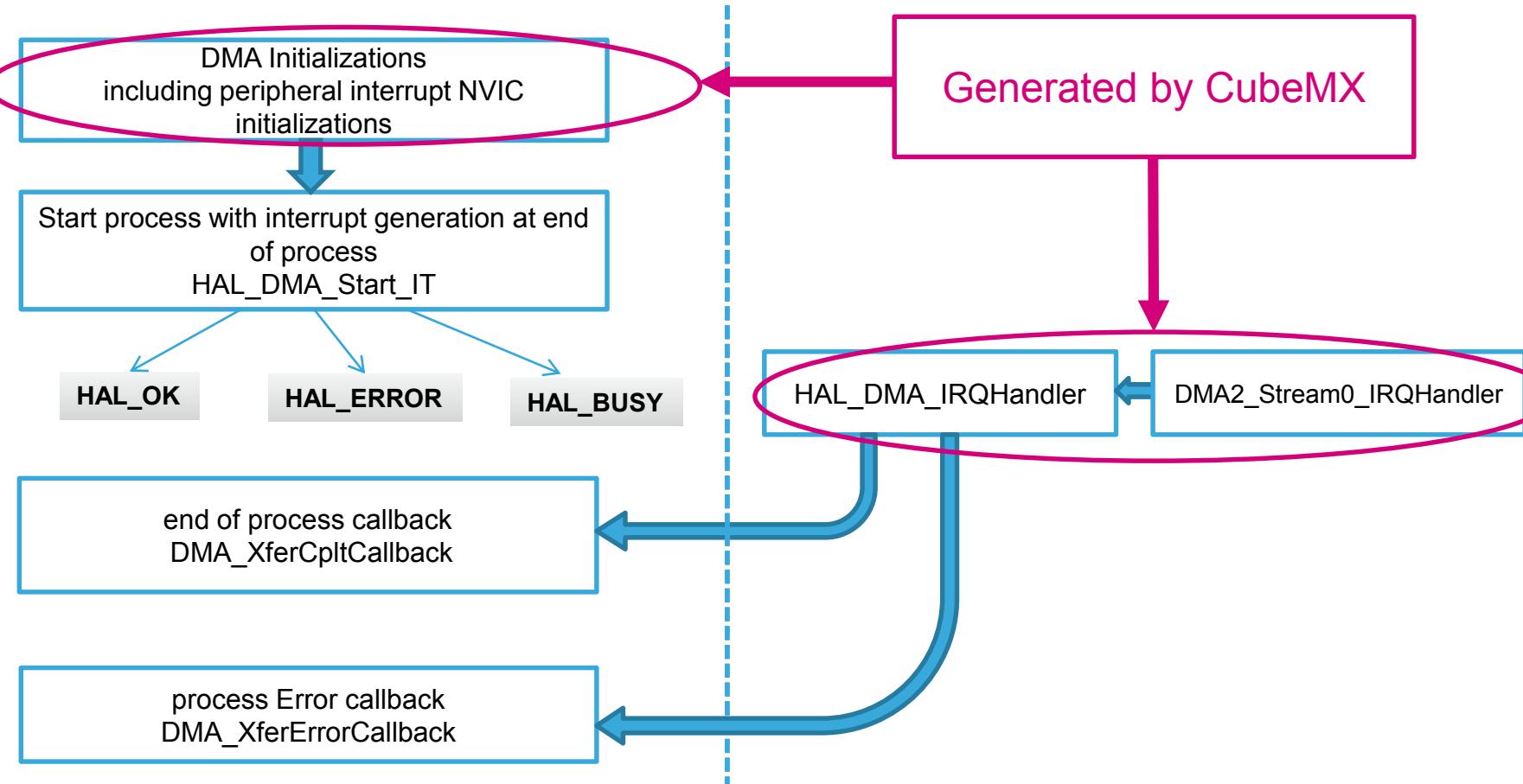
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

107

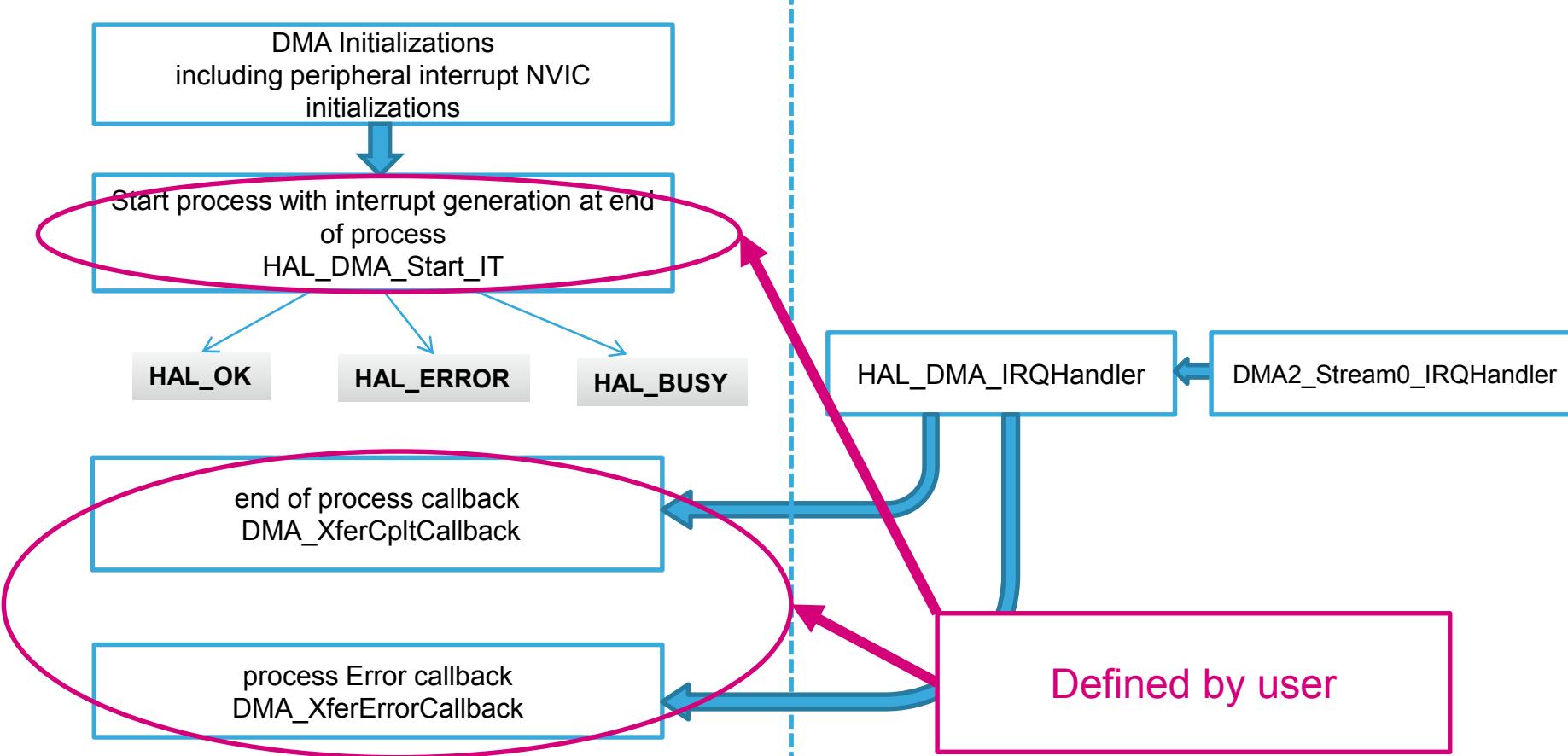
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

108

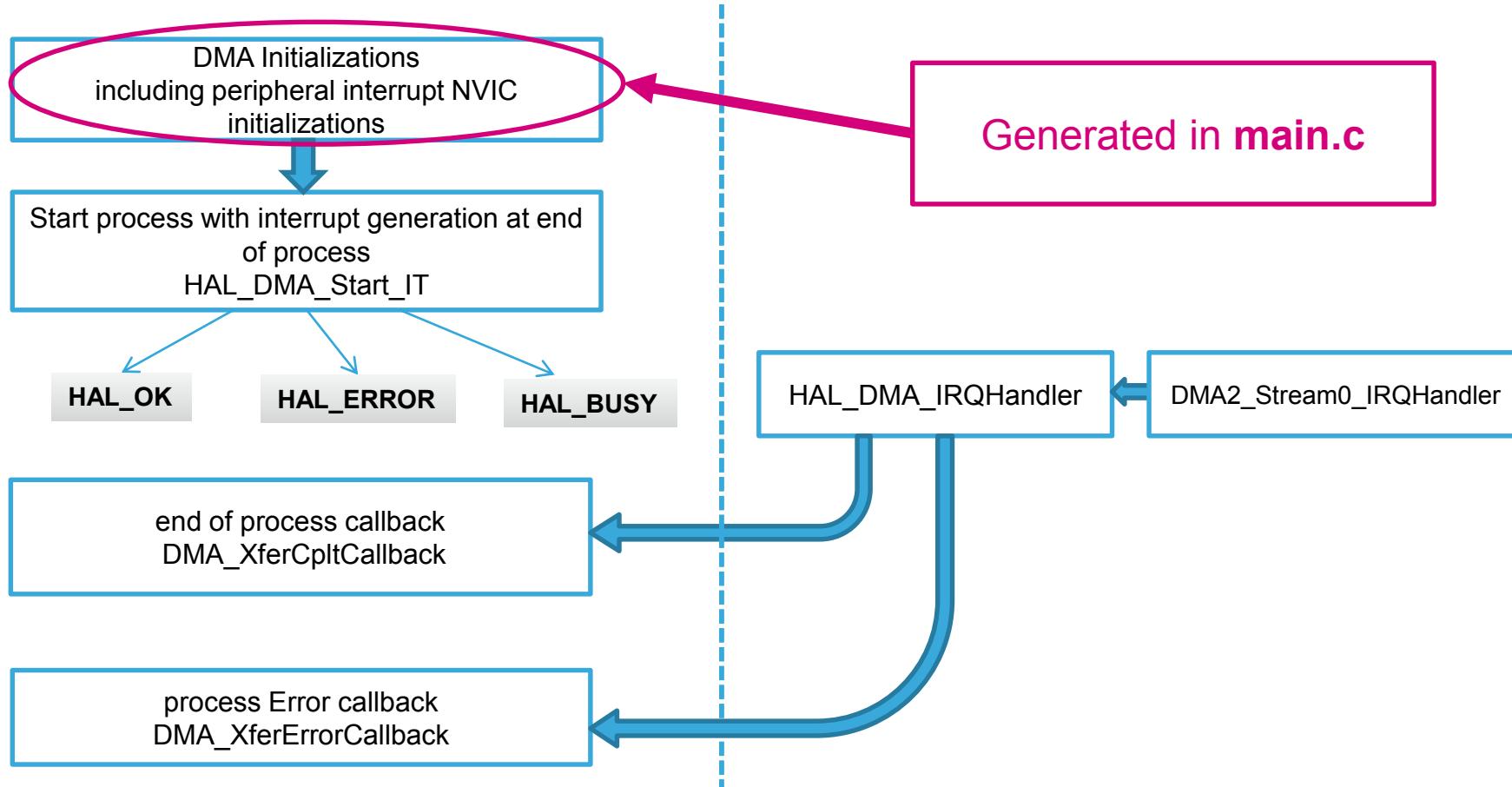
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

109

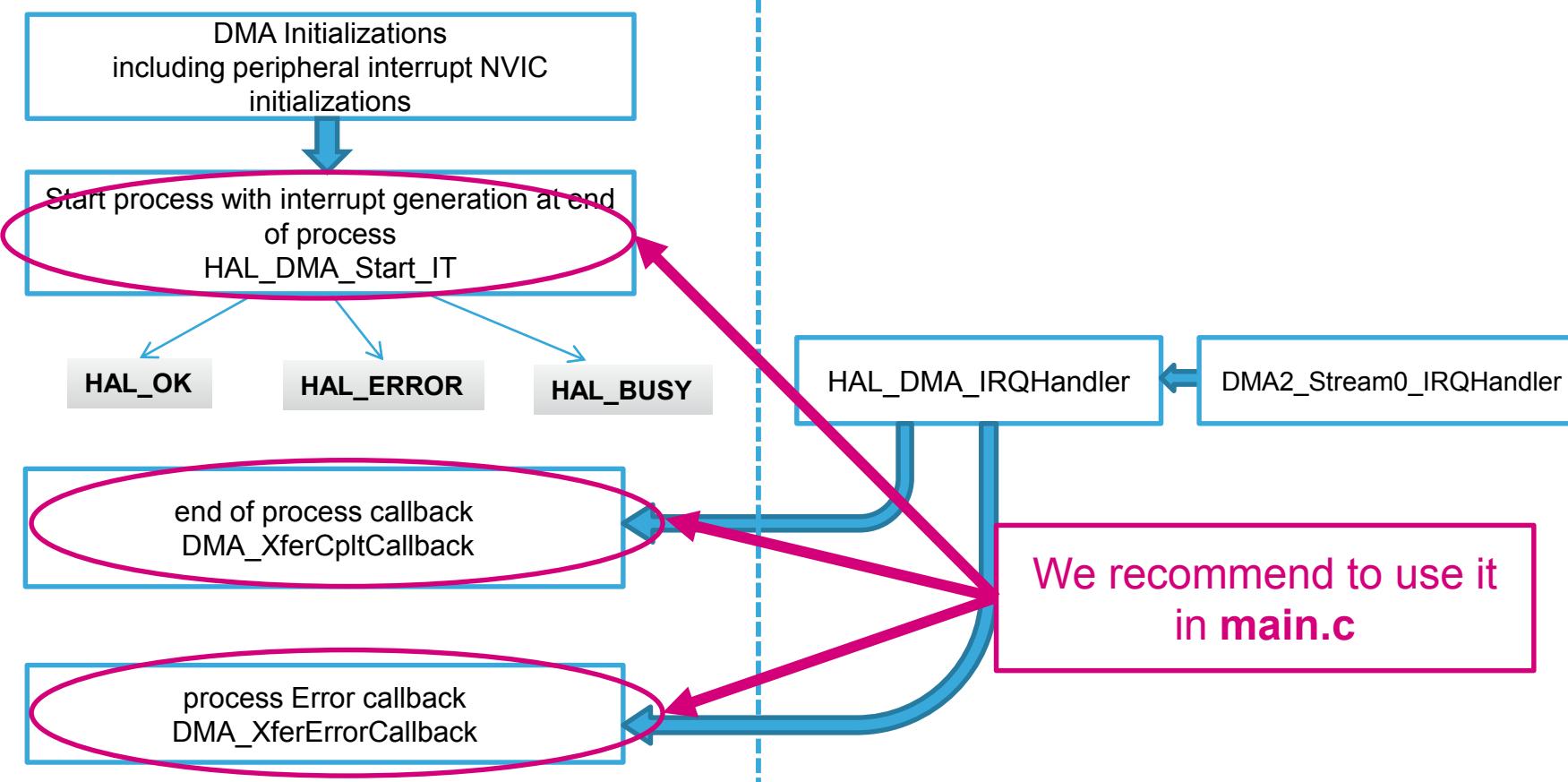
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

110

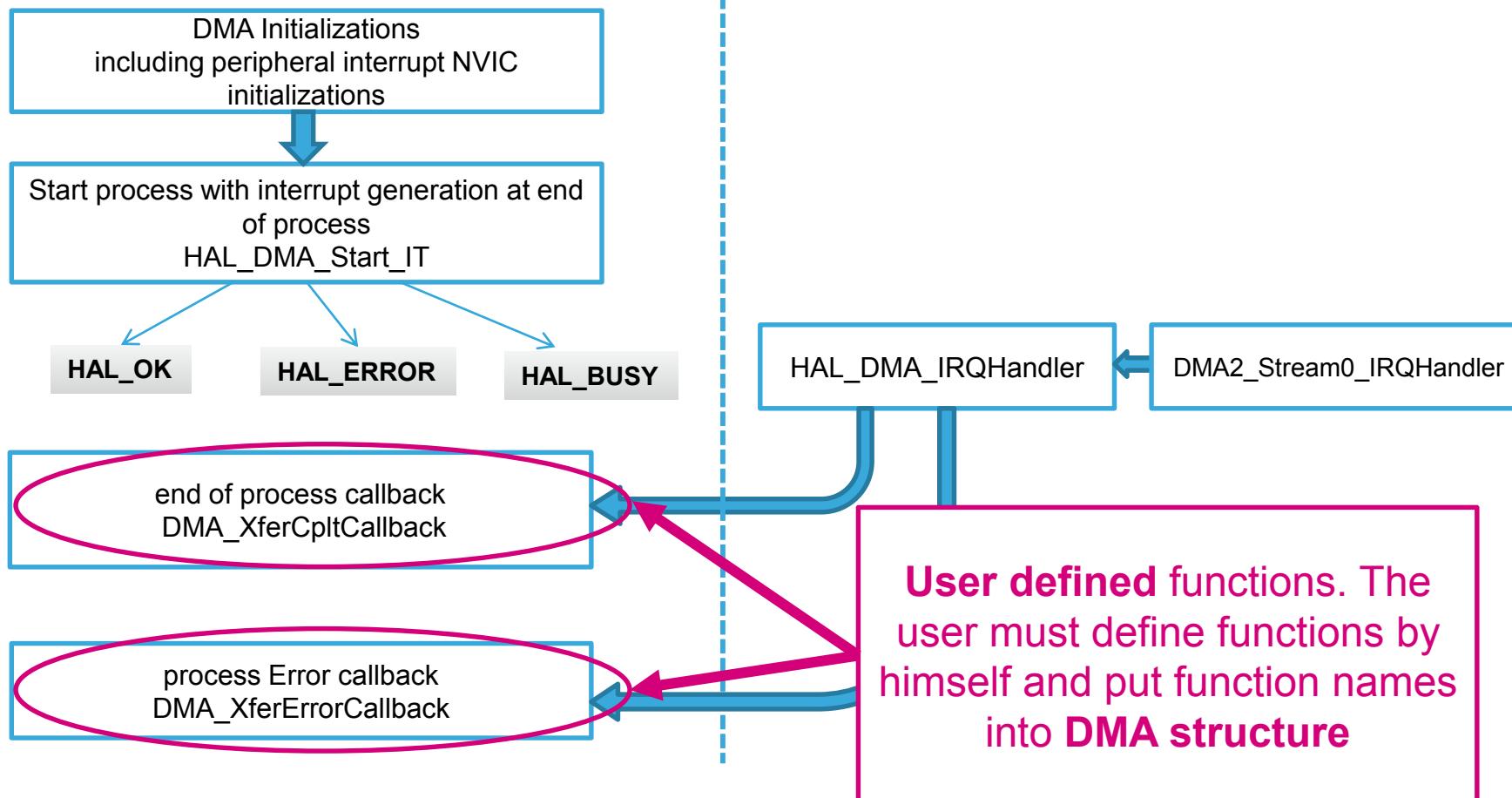
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

111

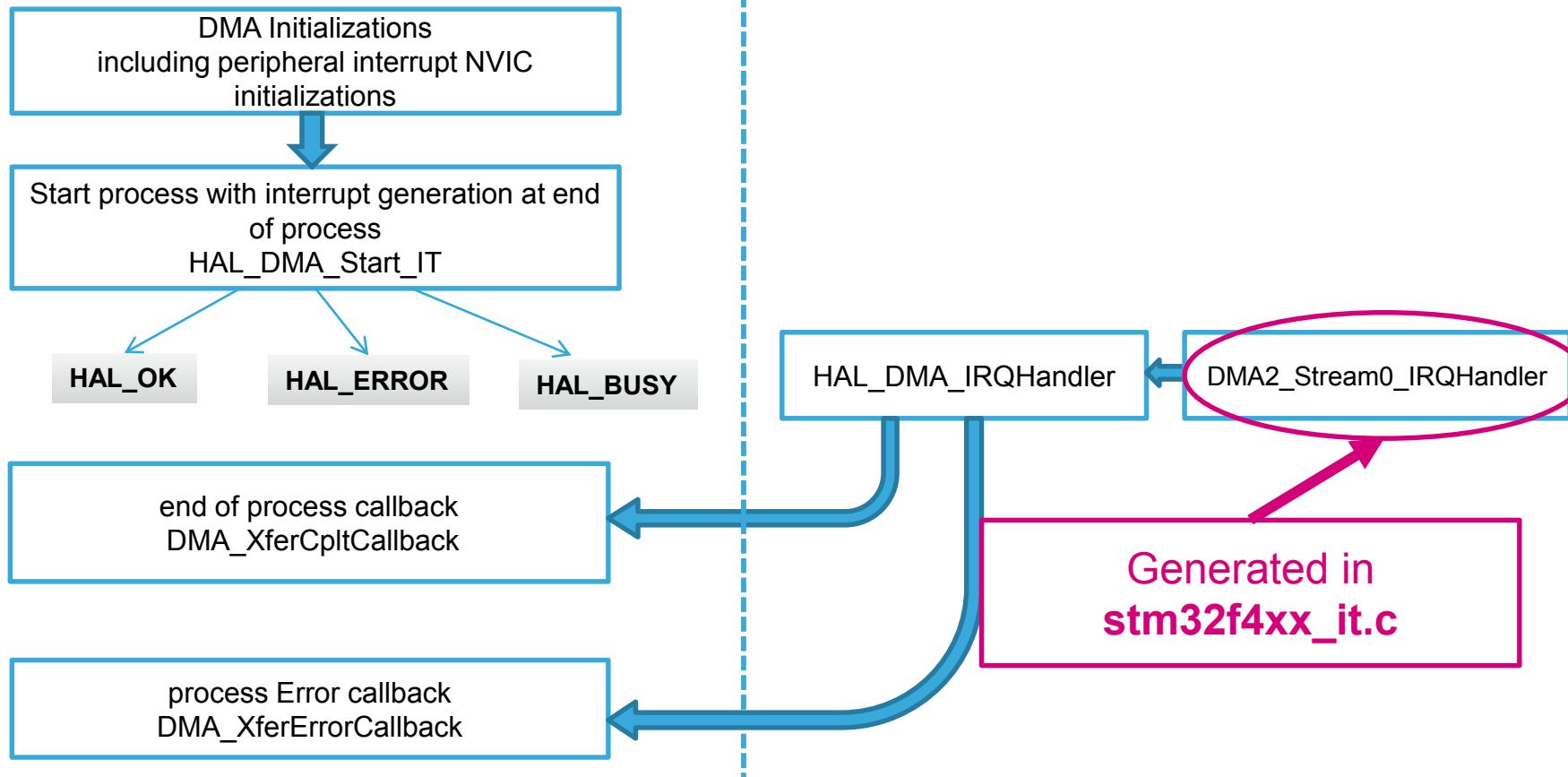
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

112

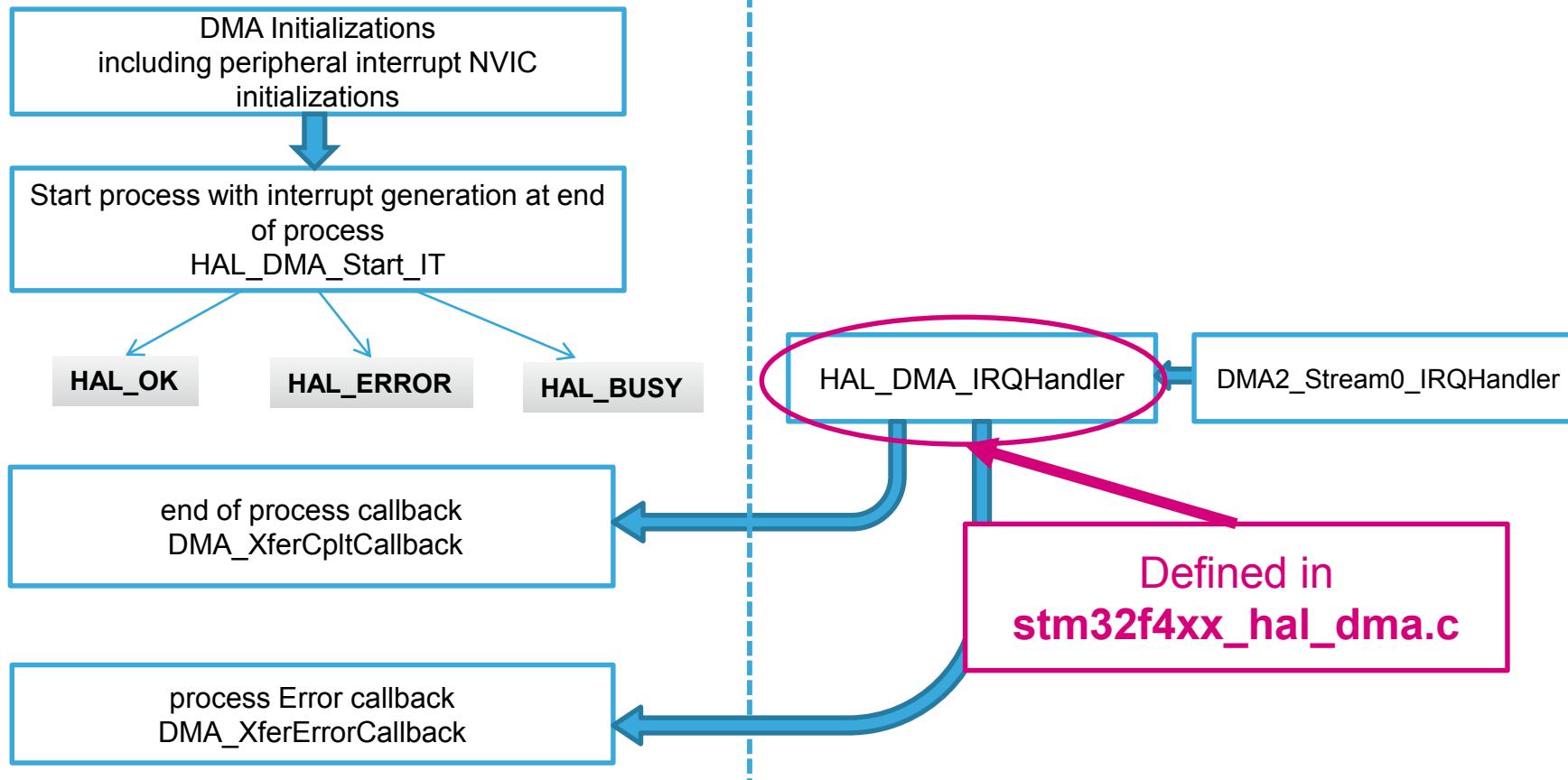
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

113

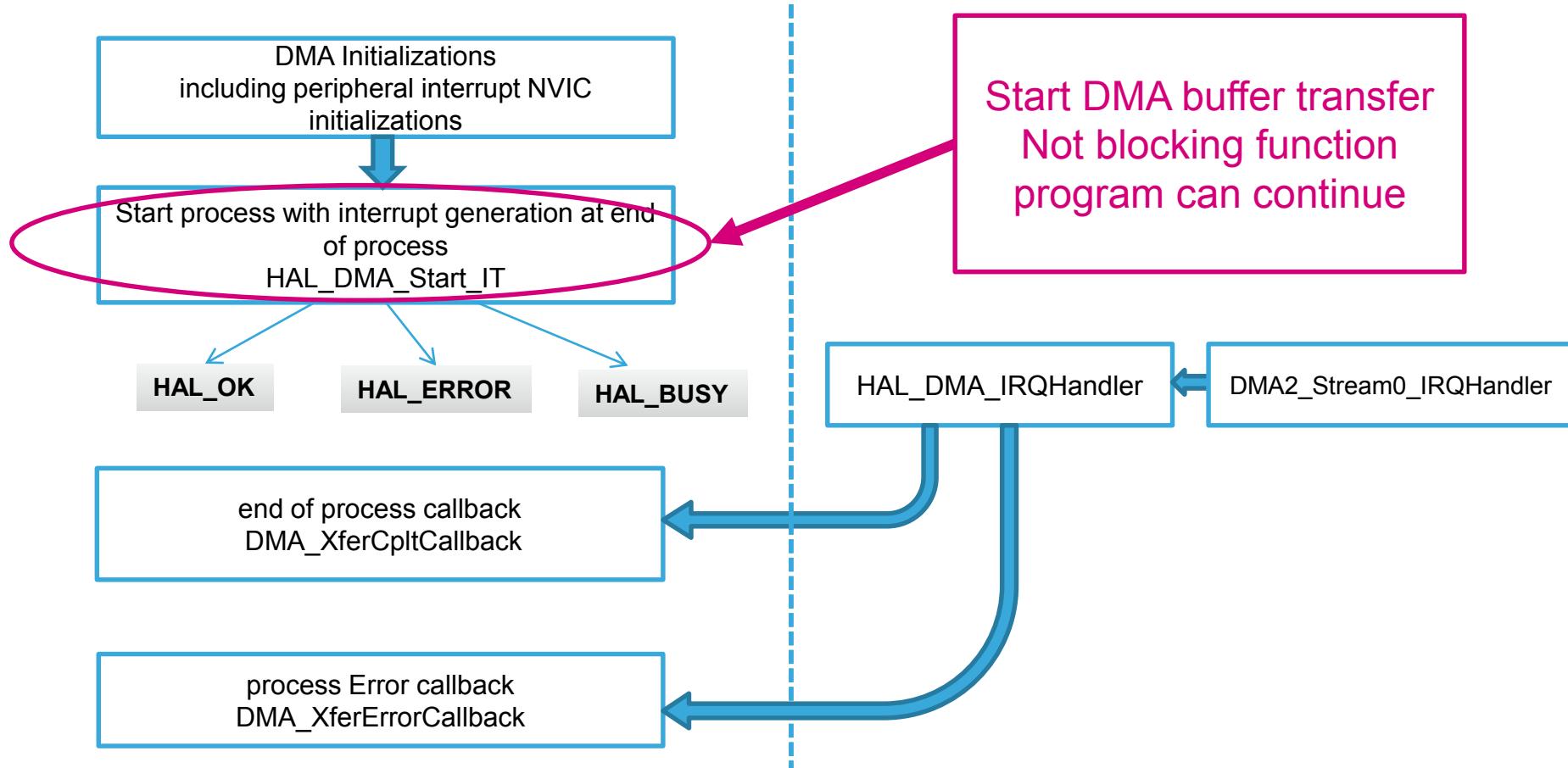
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

114

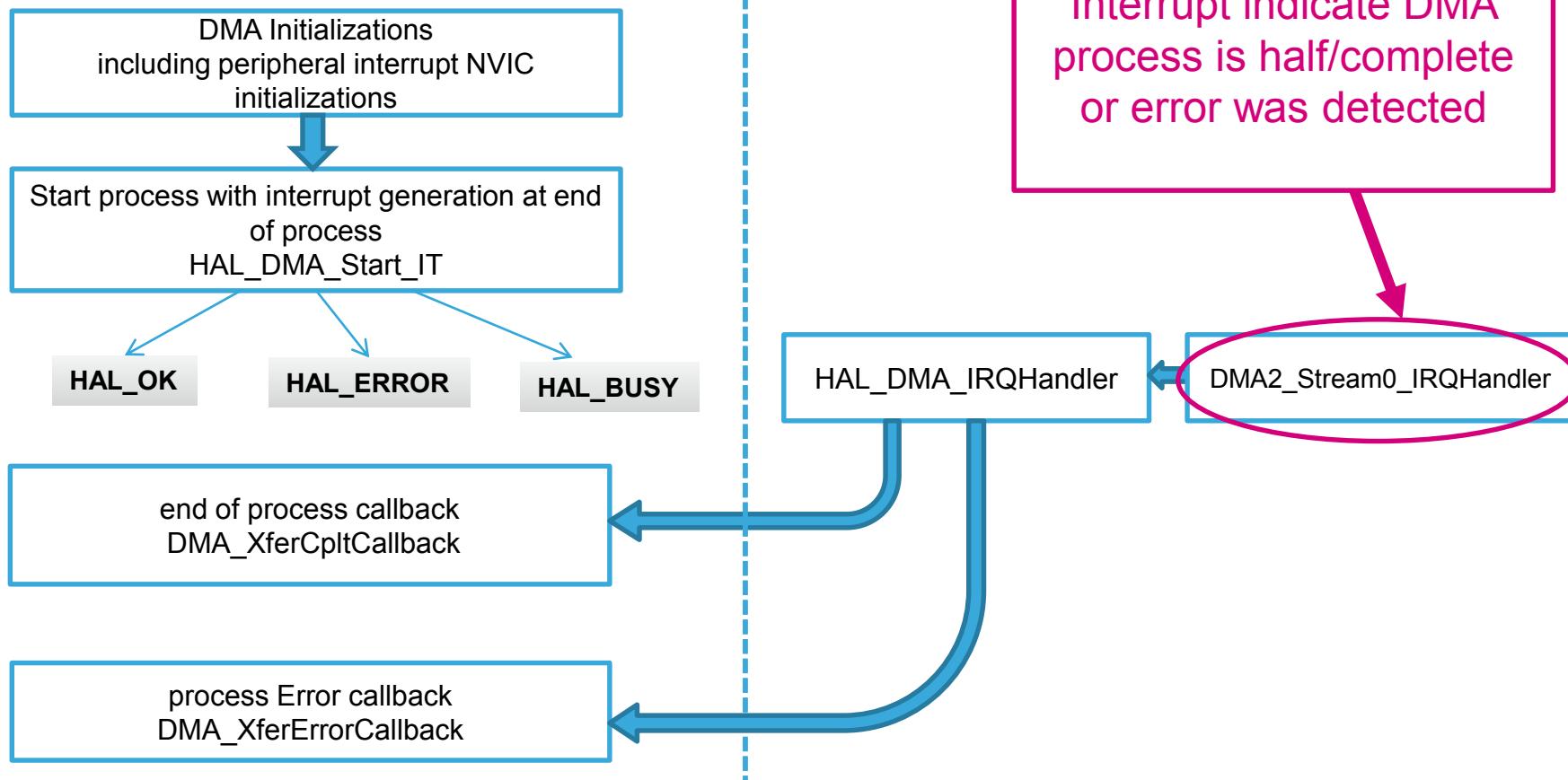
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

115

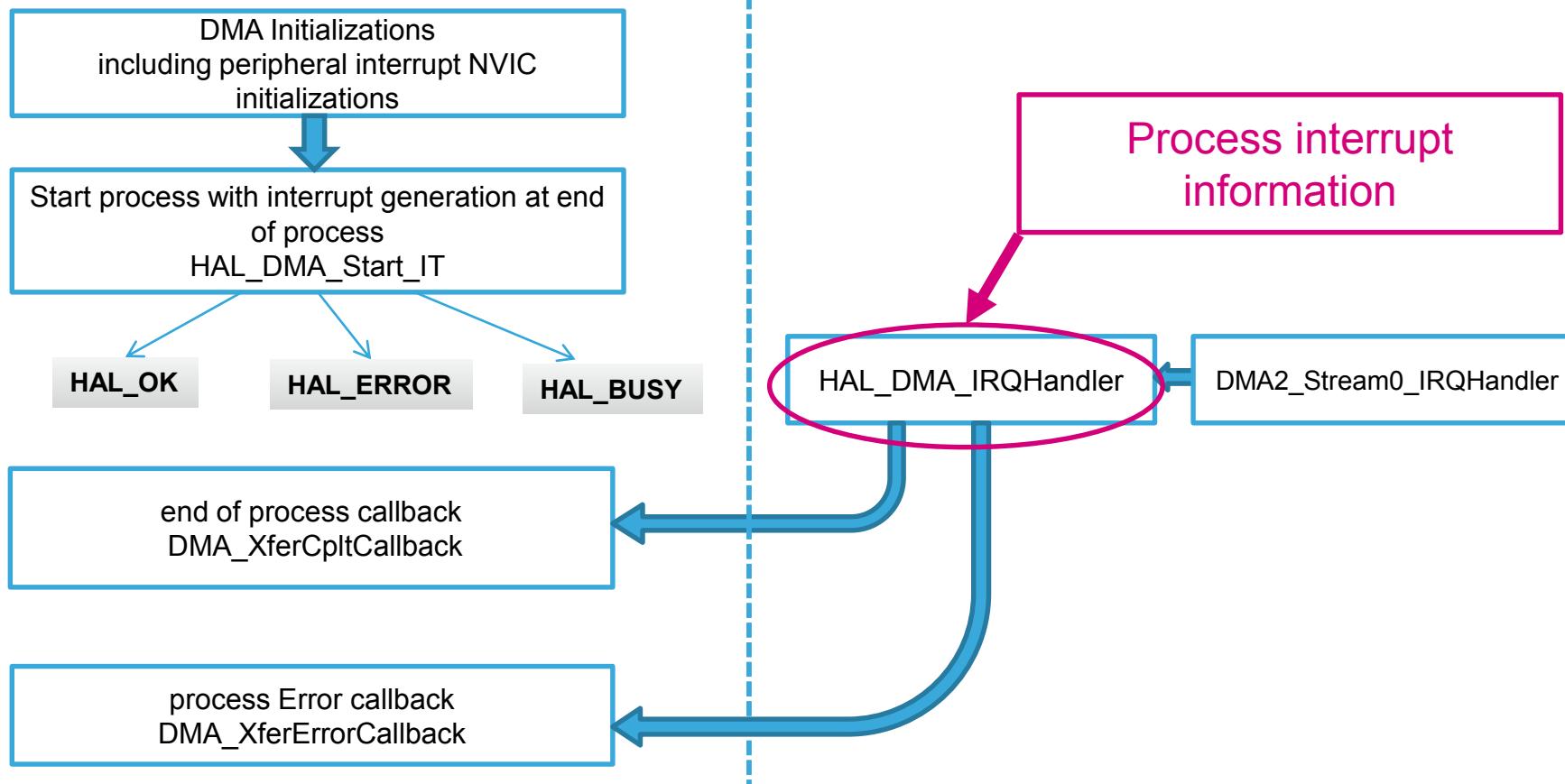
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

116

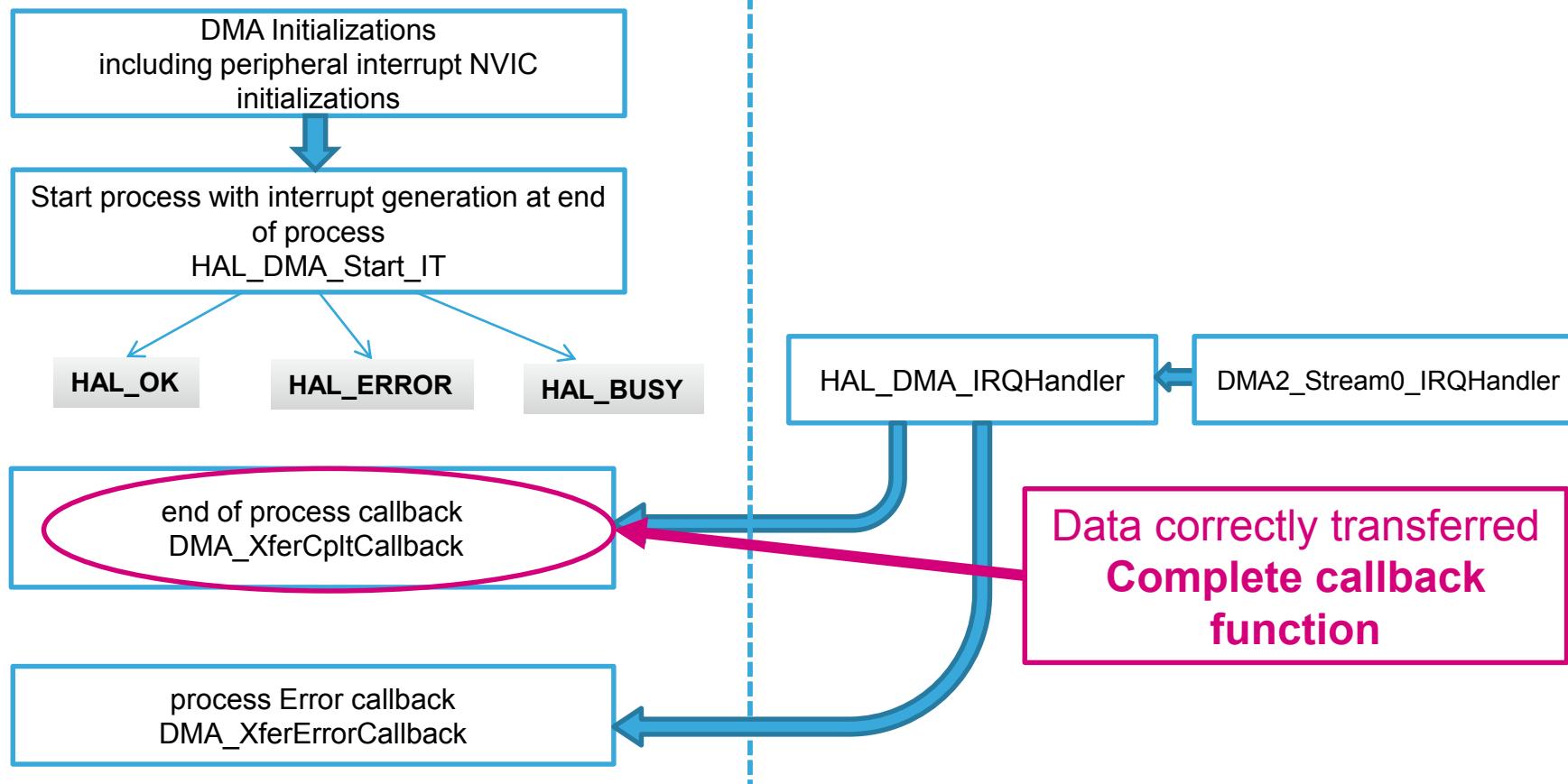
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

117

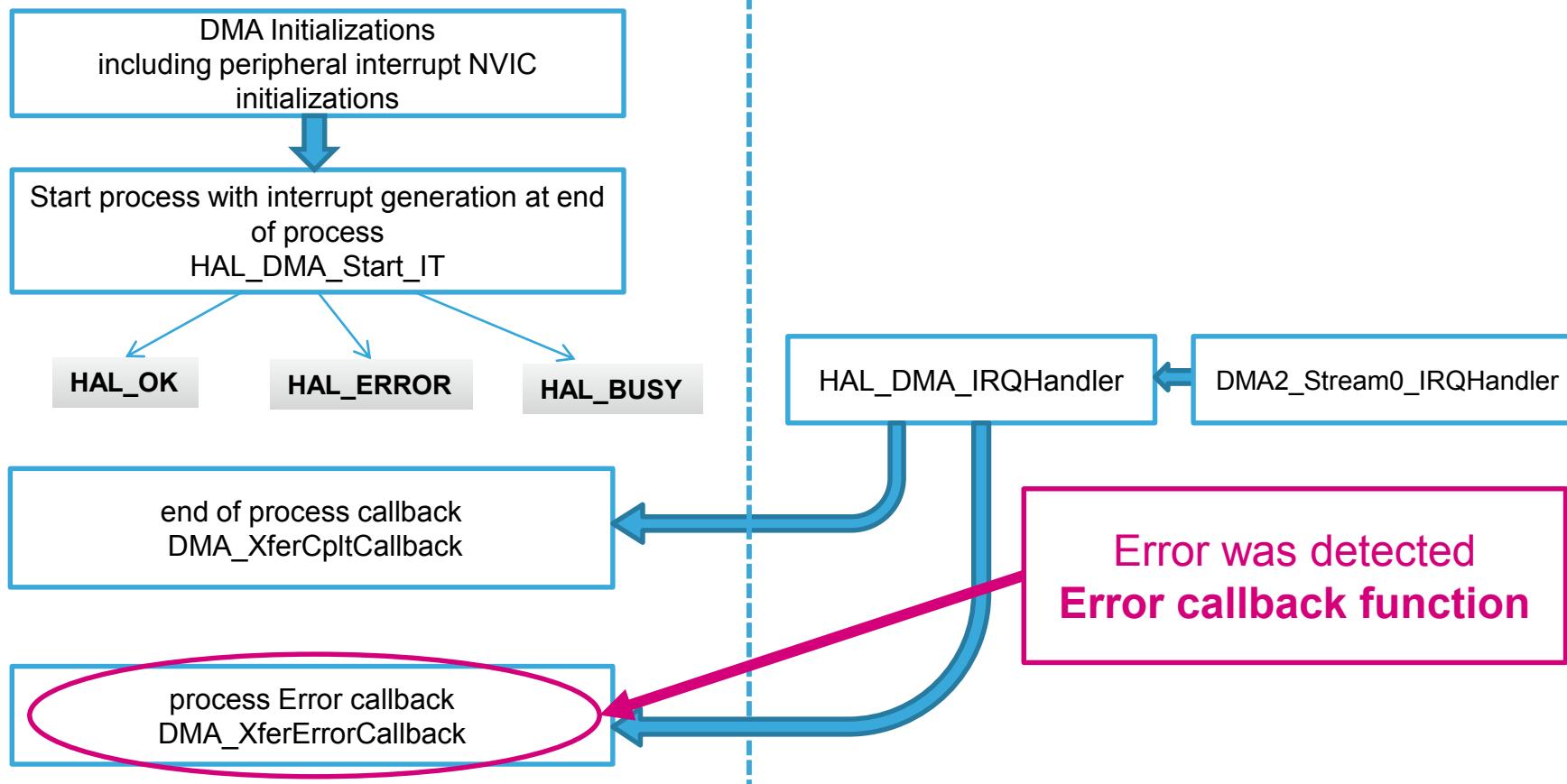
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

118

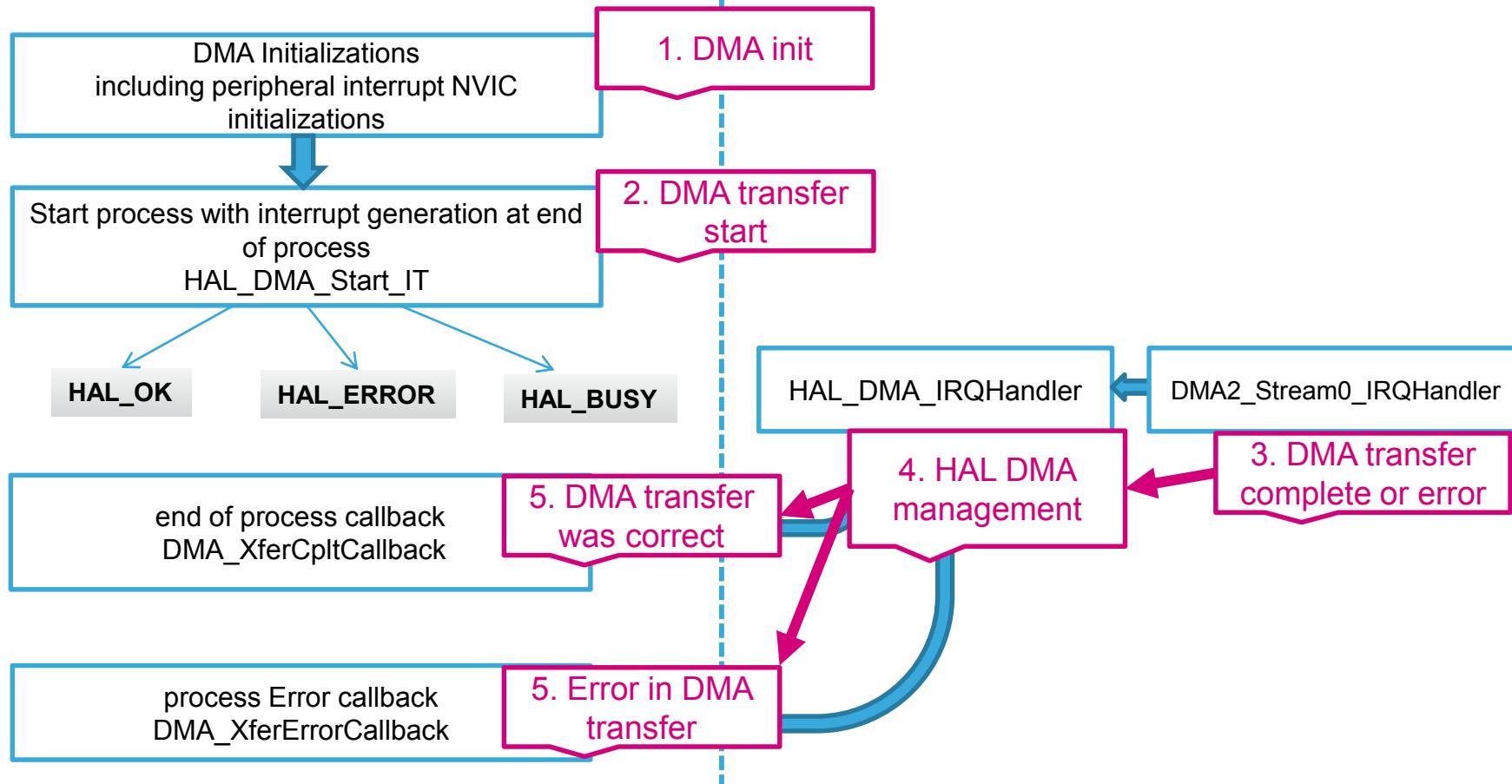
## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

119

## HAL Library DMA with IT flow



# 7 Use DMA M2M transfer with interrupt

120

- Now we open the project in our IDE
  - The functions we want to put into main.c
  - Between `/* USER CODE BEGIN 2 */` and `/* USER CODE END 2 */` tags
- DMA callback function
  - We need to add the name of callback function into DMA structure
- HAL functions for DMA
  - `HAL_DMA_Start_IT(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)`

# 7 Use DMA M2M transfer with interrupt

121

- We create two buffers
  - One with source data
  - Second as destination buffer

```
/* 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 */
```

# 7 Use DMA M2M transfer with interrupt

122

- 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

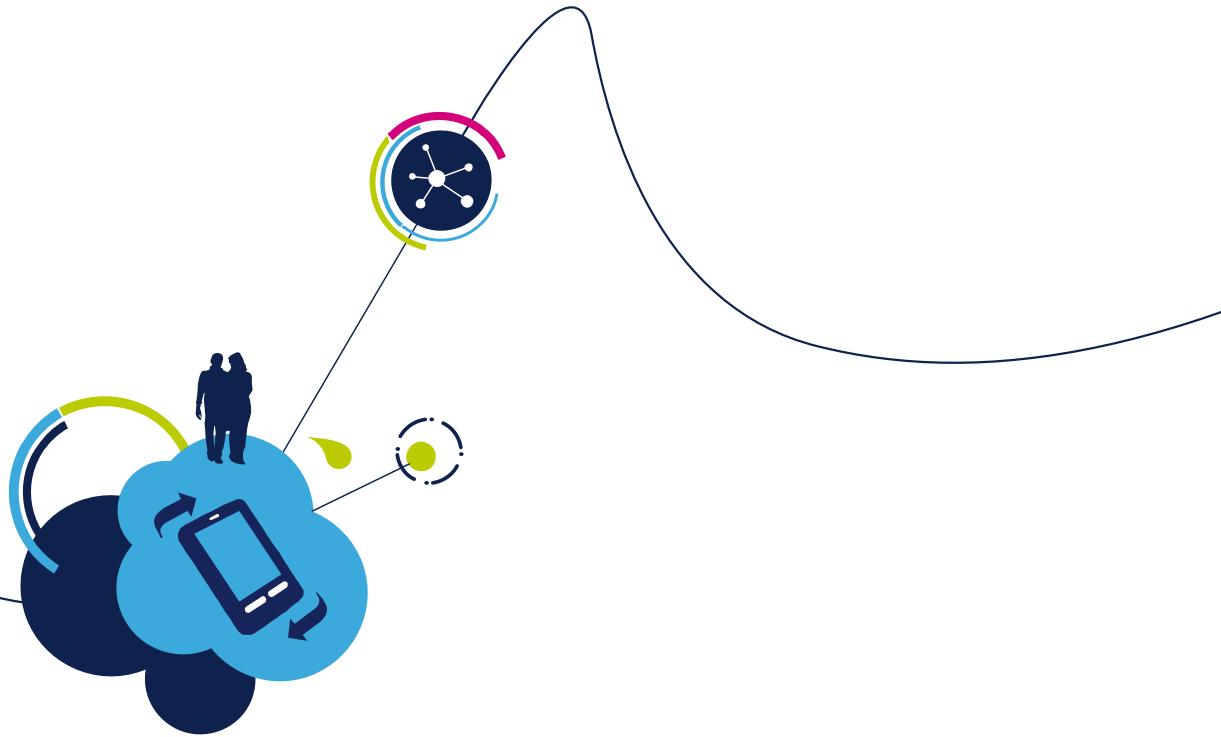
```
/* USER CODE BEGIN 4 */  
void XferCpltCallback(DMA_HandleTypeDef *hdma)  
{  
    __NOP(); //we reach this only if DMA transfer was correct  
}  
/* USER CODE END 4 */
```

# 7 Use DMA M2M transfer with interrupt

123

- DMA Start
  - Before we start the DMA with interrupt we need to set the callback into DMA structure
  - Then is possible use the `HAL_DMA_Start_IT` to begin DMA transfer

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



# Use RTC Alarm lab 8

- Objective

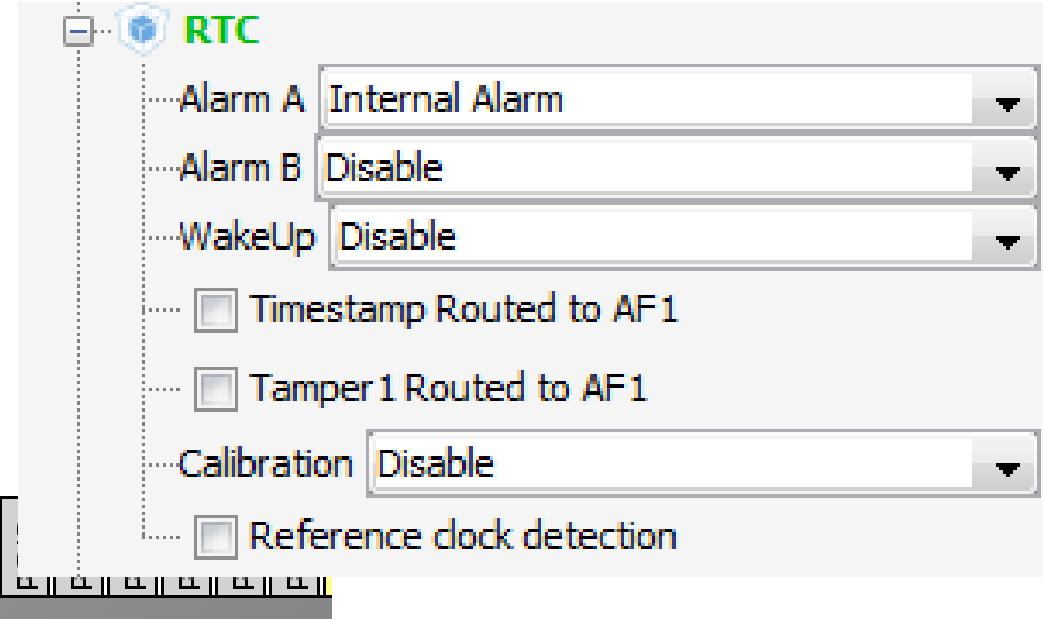
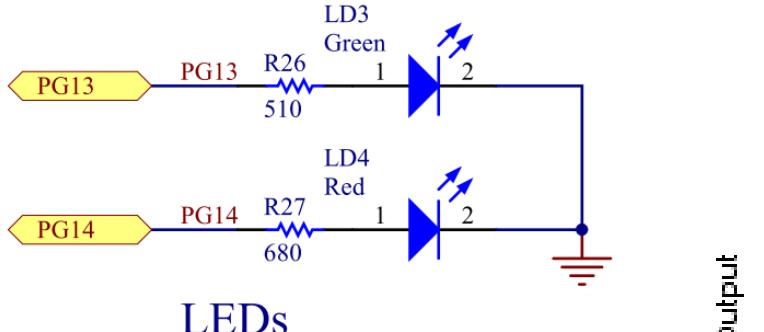
- Learn how to setup RTC with interrupt in CubeMX
- Create simple RTC project with periodic alarm interrupt

- Goal

- Use CubeMX and Generate Code with RTC
- Learn how to setup the RTC in HAL
- Verify the correct functionality by periodic RTC alarm interrupts

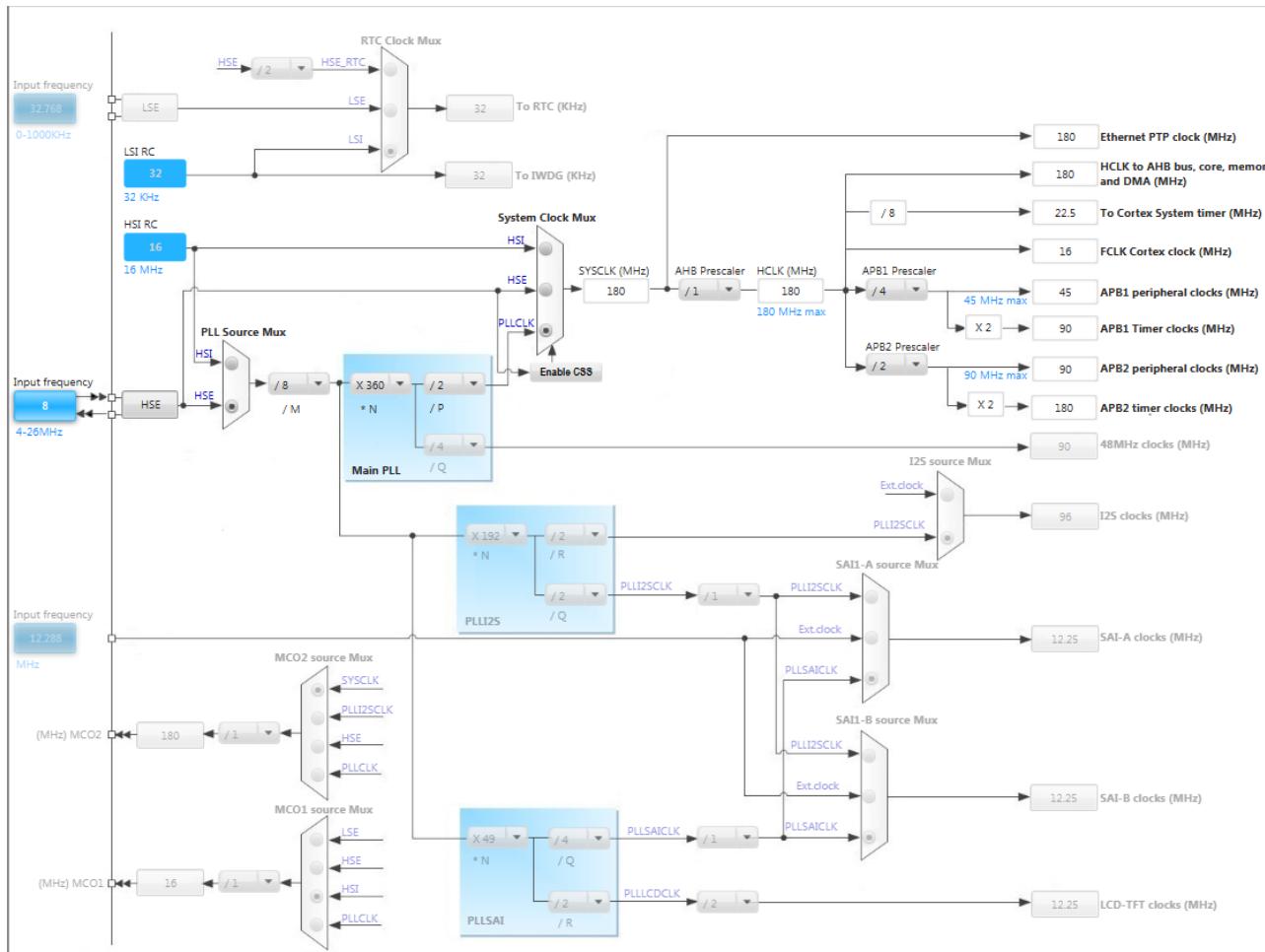
# Use RTC and Alarm with interrupt

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Set Internal Alarm on Alarm A or Alarm B
- Set GPIO to toggle with LED as alarm indication



# Use RTC and Alarm with interrupt

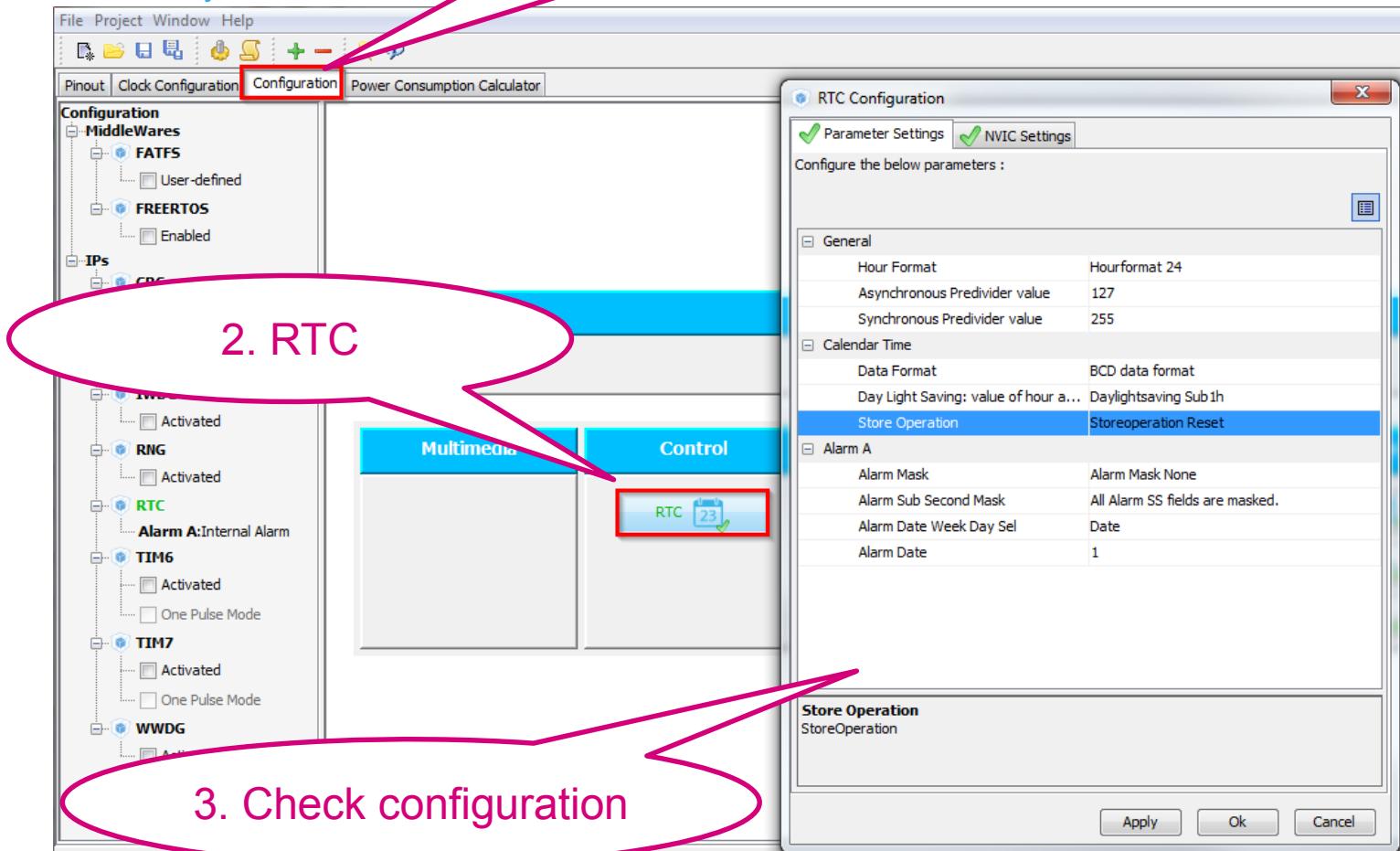
- In order to run on maximum frequency, setup clock system
- Details in lab 0



# Use RTC and Alarm with interrupt

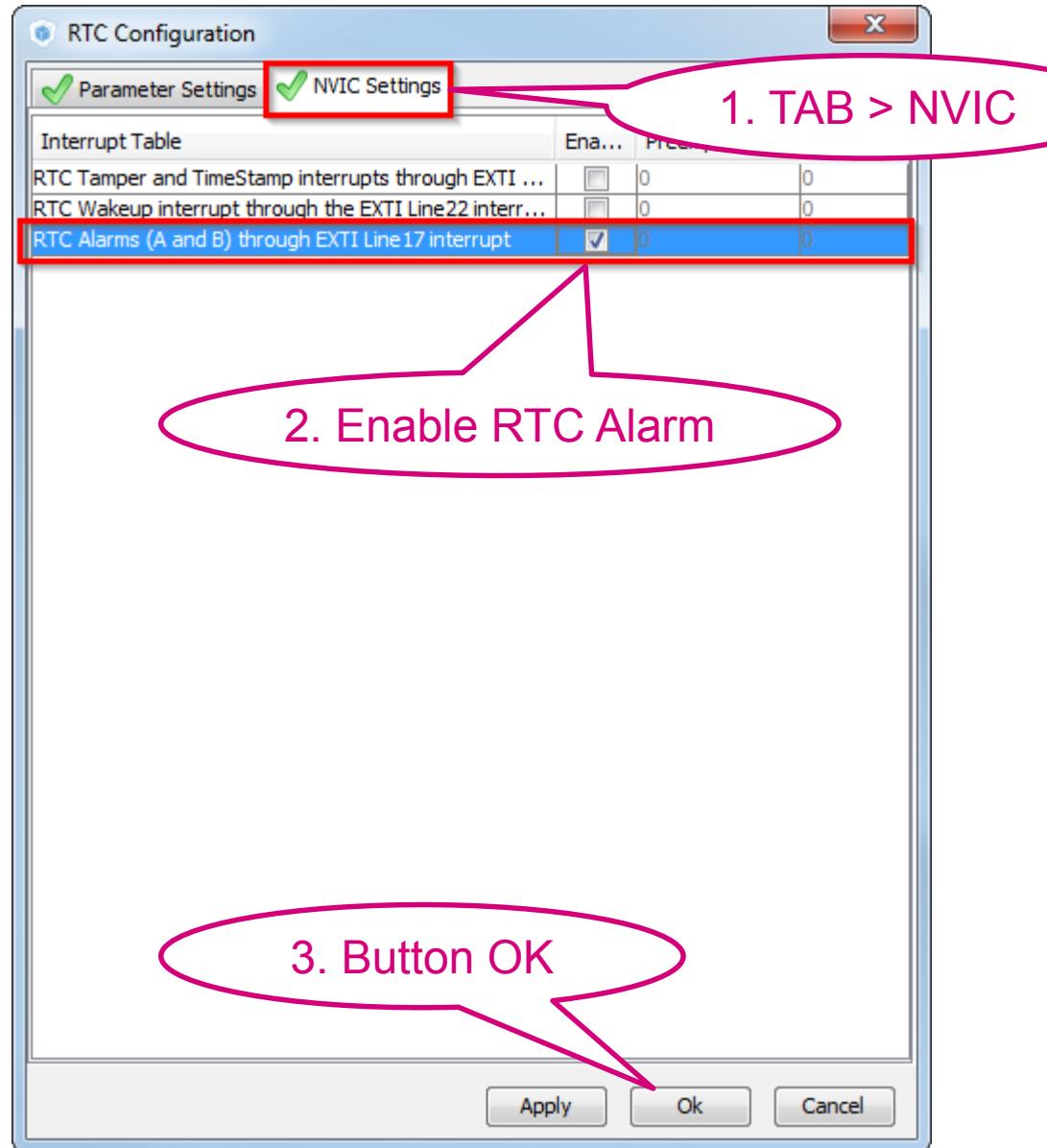
- RTC Configuration

- TAB>Configuration
- Control > RTC
- Set parameters which you want



# Use RTC and Alarm with interrupt

- RTC Configuration NVIC
  - TAB>NVIC Setup
  - Enable Alarm interrupt
  - Button OK



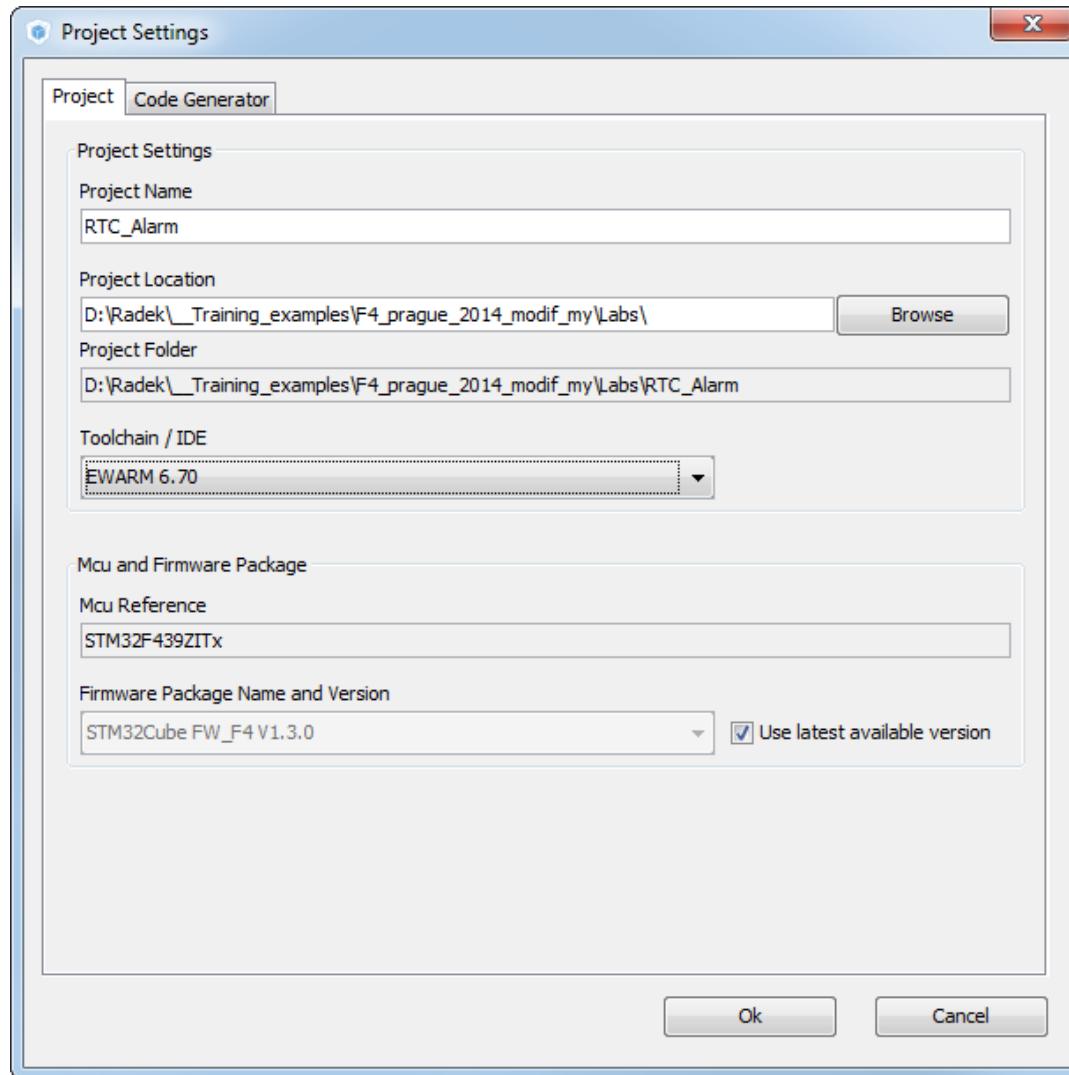
# Use RTC and Alarm with interrupt

- 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



- The RTC can be preserved during RESET(ok LP modes)
  - CubeMX not enable the RTC by default
  - We need to add `HAL_PWR_EnableBkUpAccess()` and `__HAL_RCC_RTC_ENABLE()` before we call `MX_RTC_Init()`
- Set the first alarm to 1s
  - In `MX_RTC_Init`
- We create the RTC interrupt handler and we reconfigure the Alarm A time
  - `HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)`
  - `HAL_RTC_GetAlarm(RTC_HandleTypeDef *hrtc, RTC_AlarmTypeDef *sAlarm, uint32_t Alarm, uint32_t Format)`
  - `HAL_RTC_SetAlarm_IT(RTC_HandleTypeDef *hrtc, RTC_AlarmTypeDef *sAlarm, uint32_t Format)`
- RTC alarm indication will be done by LED
  - `HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

- RTC enable

```
/* Initialize all configured peripherals */  
HAL_PWR_EnableBkUpAccess(); //enable PWR backup domain access (RTC,BKReg)  
__HAL_RCC_RTC_ENABLE(); //Enable RTC. not created by cube because the RTC can run.  
MX_GPIO_Init();  
MX_RTC_Init();
```

- In MX\_RTC\_Init we set first Alarm to 1s

```
/**Enable the Alarm A  
 */  
sAlarm.AlarmTime.Hours = 0;  
sAlarm.AlarmTime.Minutes = 0;  
sAlarm.AlarmTime.Seconds = 1;  
sAlarm.AlarmTime.SubSeconds = 0;
```

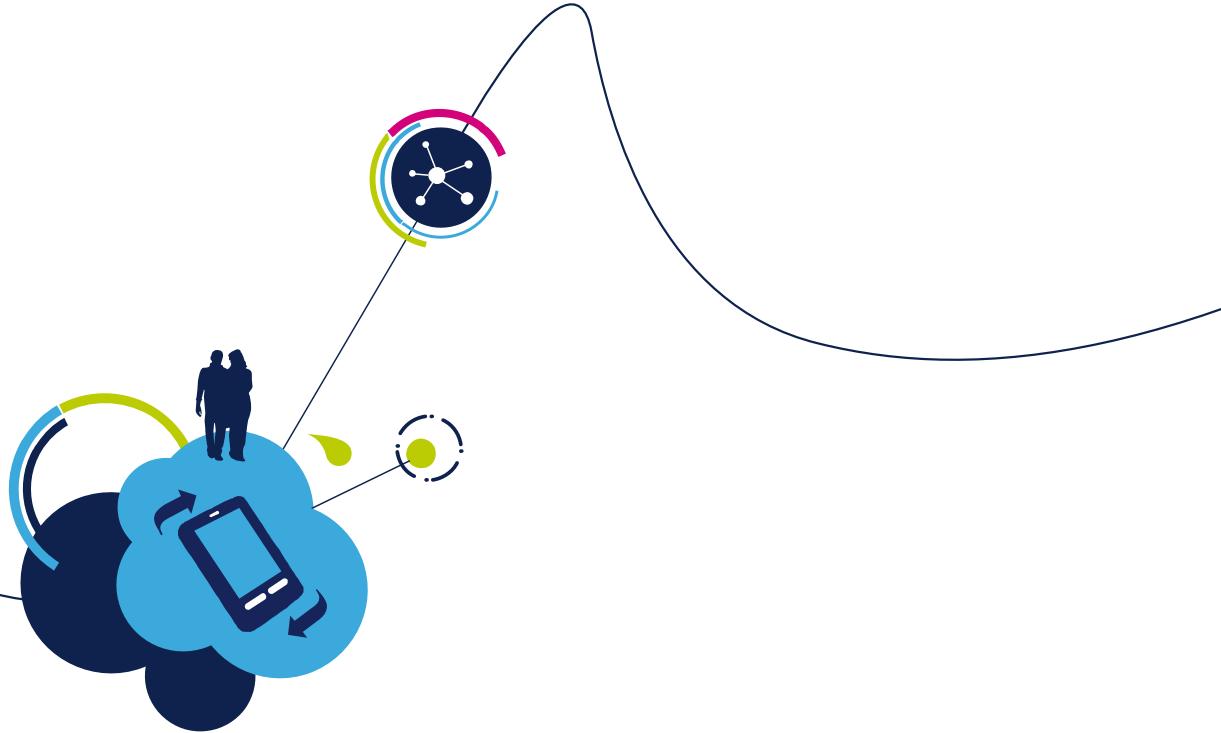
- RTC enable

```
/* USER CODE BEGIN 4 */  
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc){  
    RTC_AlarmTypeDef sAlarm;  
    HAL_RTC_GetAlarm(hrtc,&sAlarm,RTC_ALARM_A,FORMAT_BIN);  
    if(sAlarm.AlarmTime.Seconds>58){  
        sAlarm.AlarmTime.Seconds=0;  
    }else{  
        sAlarm.AlarmTime.Seconds=sAlarm.AlarmTime.Seconds+1;  
    }  
    while(HAL_RTC_SetAlarm_IT(hrtc, &sAlarm, FORMAT_BIN)!=HAL_OK){}  
    HAL_GPIO_TogglePin(GPIOG,GPIO_PIN_14);  
}  
/* USER CODE END 4 */
```

- Advanced task
  - The counting stops after 1minute
  - Modify the project to create alarm every 1s for infinite time

```
/**Enable the Alarm A
 */
sAlarm.AlarmTime.Hours = 0;
sAlarm.AlarmTime.Minutes = 0;
sAlarm.AlarmTime.Seconds = 1;
sAlarm.AlarmTime.SubSeconds = 0;
sAlarm.AlarmTime.TimeFormat = RTC_HOURFORMAT12_AM;
sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
sAlarm.AlarmMask = RTC_ALARMMASK_DATEWEEKDAY|RTC_ALARMMASK_HOURS|RTC_ALARMMASK_MINUTES;
sAlarm.AlarmSubSecondMask = RTC_ALARMSECONDMASK_ALL;
sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
sAlarm.AlarmDateWeekDay = 1;
sAlarm.Alarm = RTC_ALARM_A;
HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, FORMAT_BCD);
```

- We only need to modify the Alarm mask to ignore Days, Hours and Minutes



# UART Poll lab 9

- Objective

- Learn how to setup UART in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Work in pairs, one will create transmitter and second receiver

- Goal

- Configure UART in CubeMX and Generate Code
- Learn how to send and receive data over UART without interrupts
- Verify the correct functionality

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Pin selection
  - We are looking for free pins where is possible to create wire loopback connection

# Simple UART communication

- Pin selection

- We are looking for free pins where is possible to create wire loopback connection

Image from  
STM32F429-Discovery  
user manual

Table 6. MCU pin description versus board function (page 1 of 7)

MCU pin	Board function										Free I/O	Power supply		
	Main function	LQFP144	System	SDRAM	LCD-TFT	LCD-RGB	LCD-SPI	L3GD20	USB	LED	Puchbutton	ACP/RF	Touch panel	
BOOT0	138		BOOT0											21
NRST	25	NRST											5	12
PA0	34				RESET									18
PA1	35				RESET									17
PA2	36				RESET									20
PA3	37													19
PA4	40													22
PA5	41													21
PA6	42													24
PA7	43													23
PA8	100												3	53
PA9	101							SCL	ACP_RST					52
PA10	102							SCL						51
PA11	103													50
PA12	104													49
PA13	105	SWDIO												48

# Simple UART communication

- Pin selection
  - We are looking for free pins where is possible to create wire loopback connection

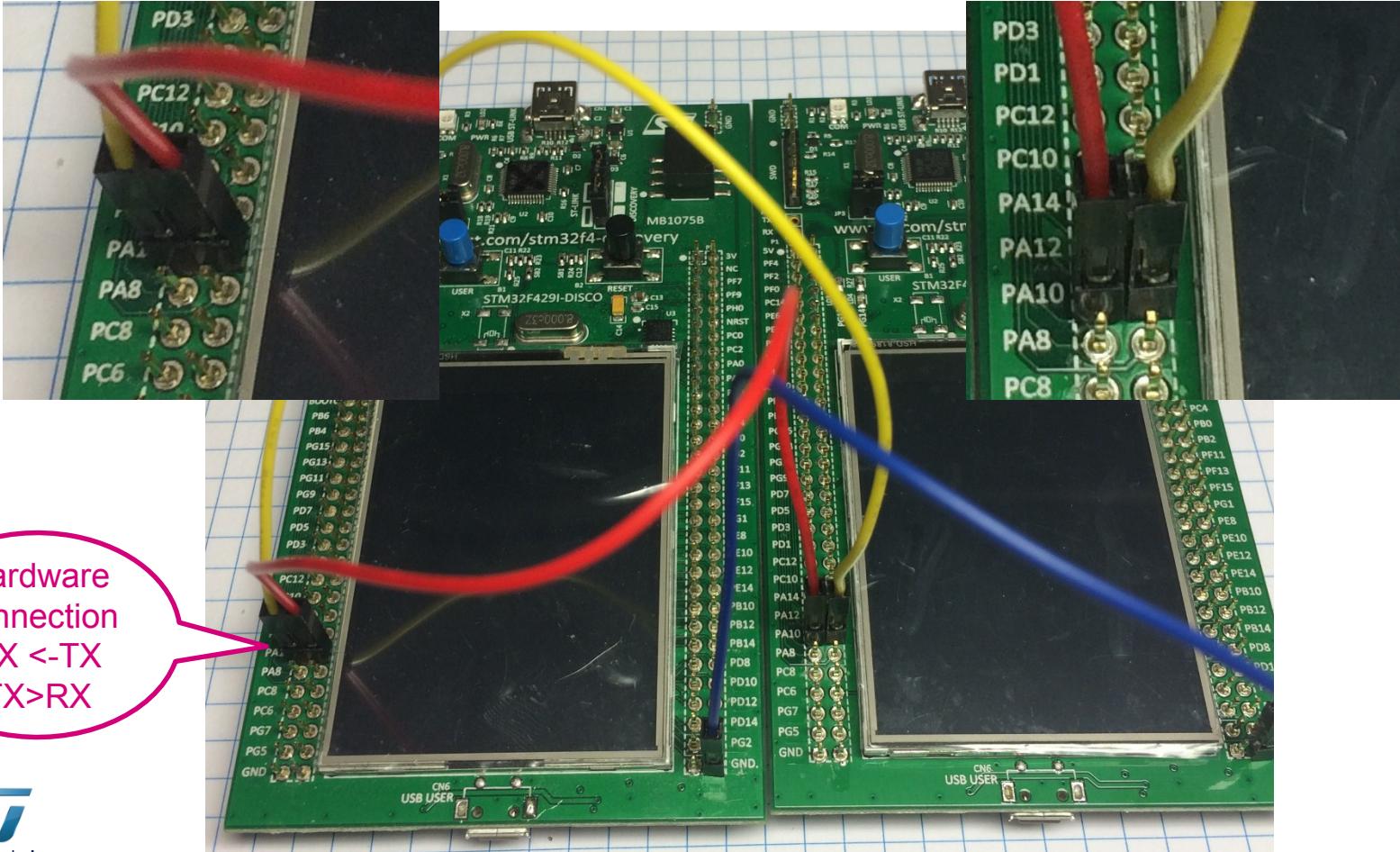
Table 12. STM32F427xx and STM32F429xx alternate function mapping

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/ 10/11	I2C1/ 2/3	SPI1/2/ 3/4/5/6	SPI2/3/S AI1	SPI3/US ART1/2/3	USART6/U ART4/5/7/8	CAN1/2/TIM 12/13/14/ LCD	OTG2_HS/ OTG1_FS	ETH	FMC/SDIO/ OTG2_FS	DCMI	LCD	SYS
Port A	PA0	-	TIM2_ CH1/TIM2_ ETR	TIM5_ CH1	TIM8_ ETR	-	-	-	USART2_ CTS	UART4_TX	-	-	ETH_MII_ CRS	-	-	-	EVEN TOUT
	PA1	-	TIM2_ CH2	TIM5_ CH2	-	-	-	-	USART2_ RTS	UART4_RX	-	-	ETH_MII_ RX_CLK/E TH_RMII_ REF_CLK	-	-	-	EVEN TOUT
	PA2	-	TIM2_ CH3	TIM5_ CH3	TIM9_ CH1	-	-	-	USART2_ TX	-	-	-	ETH_MDIÖ	-	-	-	EVEN TOUT
	PA3	-	TIM2_ CH4	TIM5_ CH4	TIM9_ CH2	-	-	-	USART2_ RX	-	-	OTG_HS_ ULPI_D0	ETH_MII_ COL	-	-	LCD_B5	EVEN TOUT
	PA4	-	-	-	-	-	SPI1_ NSS	SPI3_ NSS/ I2S3_WS	USART2_ CK	-	-	-	OTG_HS_ SOF	DCMI_ HSYNC	LCD_ VSYNC	EVEN TOUT	
	PA5	-	TIM2_ CH1/TIM2_ ETR	-	TIM8_ CH1N	-	SPI1_ SCK	-	-	-	-	OTG_HS_ ULPI_CK	-	-	-	EVEN TOUT	
	PA6	-	TIM1_ BKIN	TIM3_ CH1	TIM8_ BKIN	-	SPI1_ MISO	-	-	-	TIM13_CH1	-	-	-	DCMI_ PIXCLK	LCD_G2	EVEN TOUT
	PA7	-	TIM1_ CH1N	TIM3_ CH2	TIM8_ CH1N	-	SPI1_ MOSI	-	-	-	TIM14_CH1	-	ETH_MII_ RX_DV/ ETH_RMII_ CRS_DV	-	-	-	EVEN TOUT
	PA8	MCO1	TIM1_ CH1	-	-	I2C3_ SCL	-	-	USART1_ CK	-	-	OTG_FS_ SOF	-	-	-	LCD_R6	EVEN TOUT
	PA9	-	TIM1_ CH2	-	-	I2C3_ SMBA	-	-	USART1_ TX	-	-	-	-	-	DCMI_ D0	-	EVEN TOUT
	PA10	-	TIM1_ CH3	-	-	-	-	-	USART1_ RX	-	-	OTG_FS_ ID	-	-	DCMI_ D1	-	EVEN TOUT
	PA11	-	TIM1_ CH4	-	-	-	-	-	USART1_ CTS	-	CAN1_RX	OTG_FS_ DM	-	-	-	LCD_R4	EVEN TOUT
	PA12	-	TIM1_ ETR	-	-	-	-	-	USART1_ RTS	-	CAN1_TX	OTG_FS_ DP	-	-	-	LCD_R5	EVEN TOUT

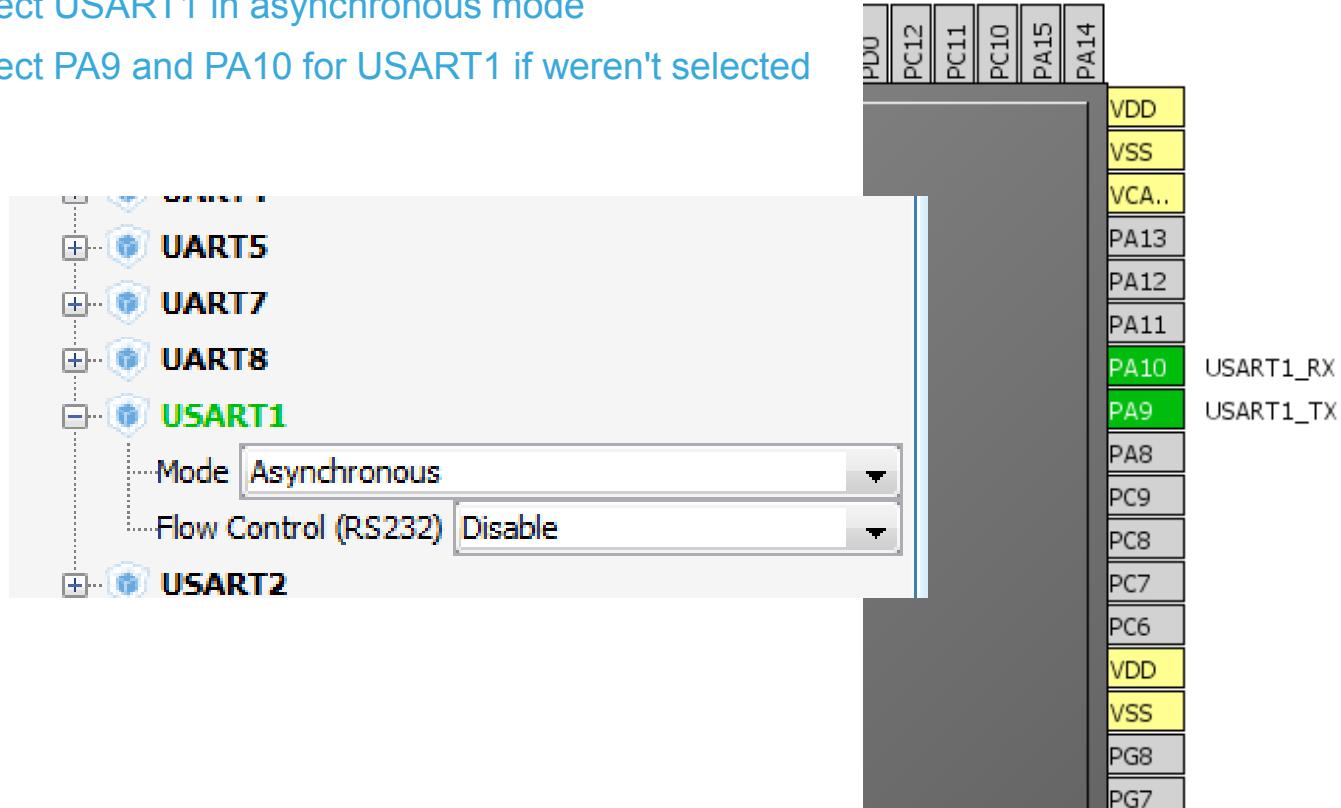
Image from  
STM32F429  
datasheet

# Simple UART communication

- Hardware preparation
  - We connect selected pins together by jumper, this help us to create loopback on UART

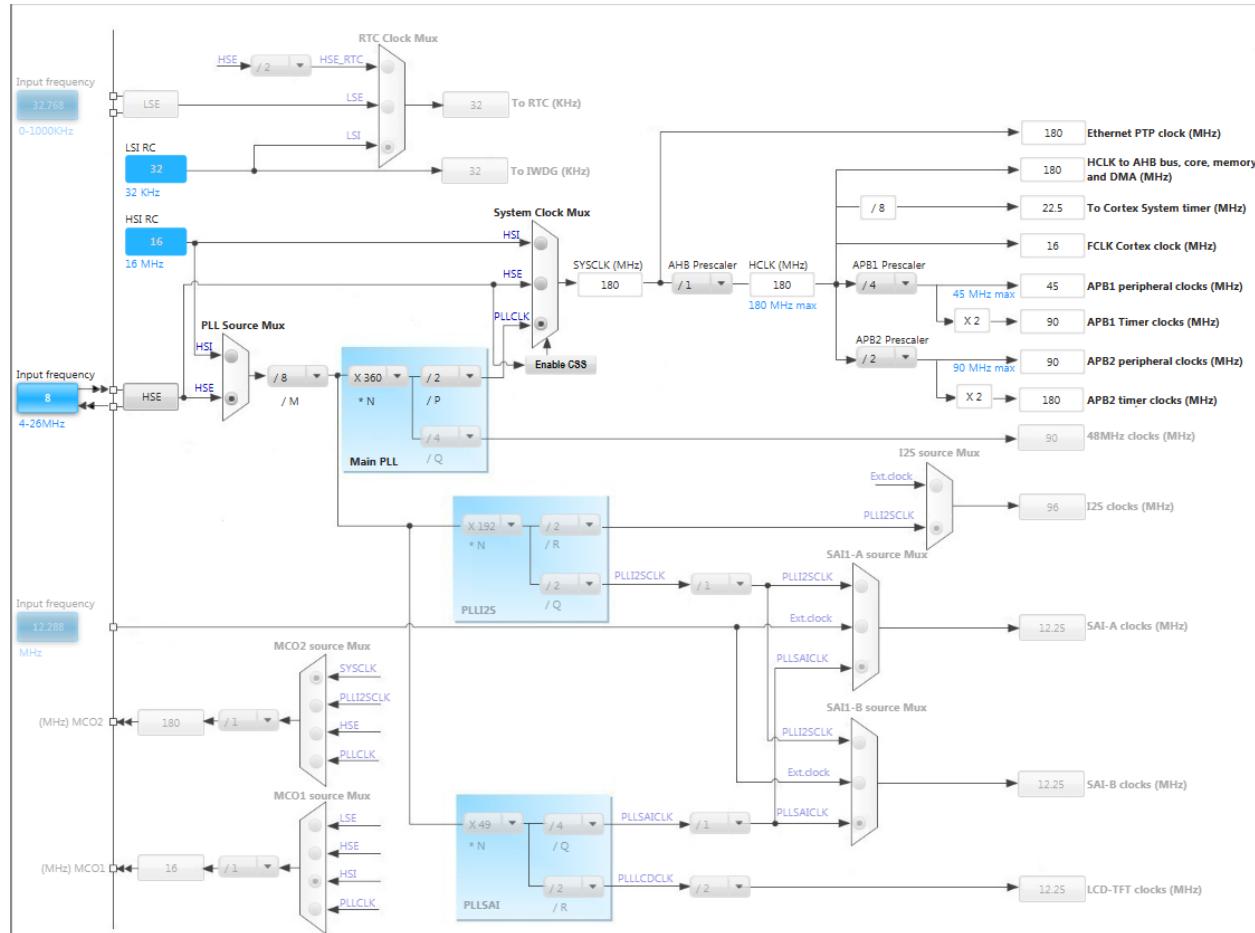


- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX UART selection
  - Select USART1 in asynchronous mode
  - Select PA9 and PA10 for USART1 if weren't selected



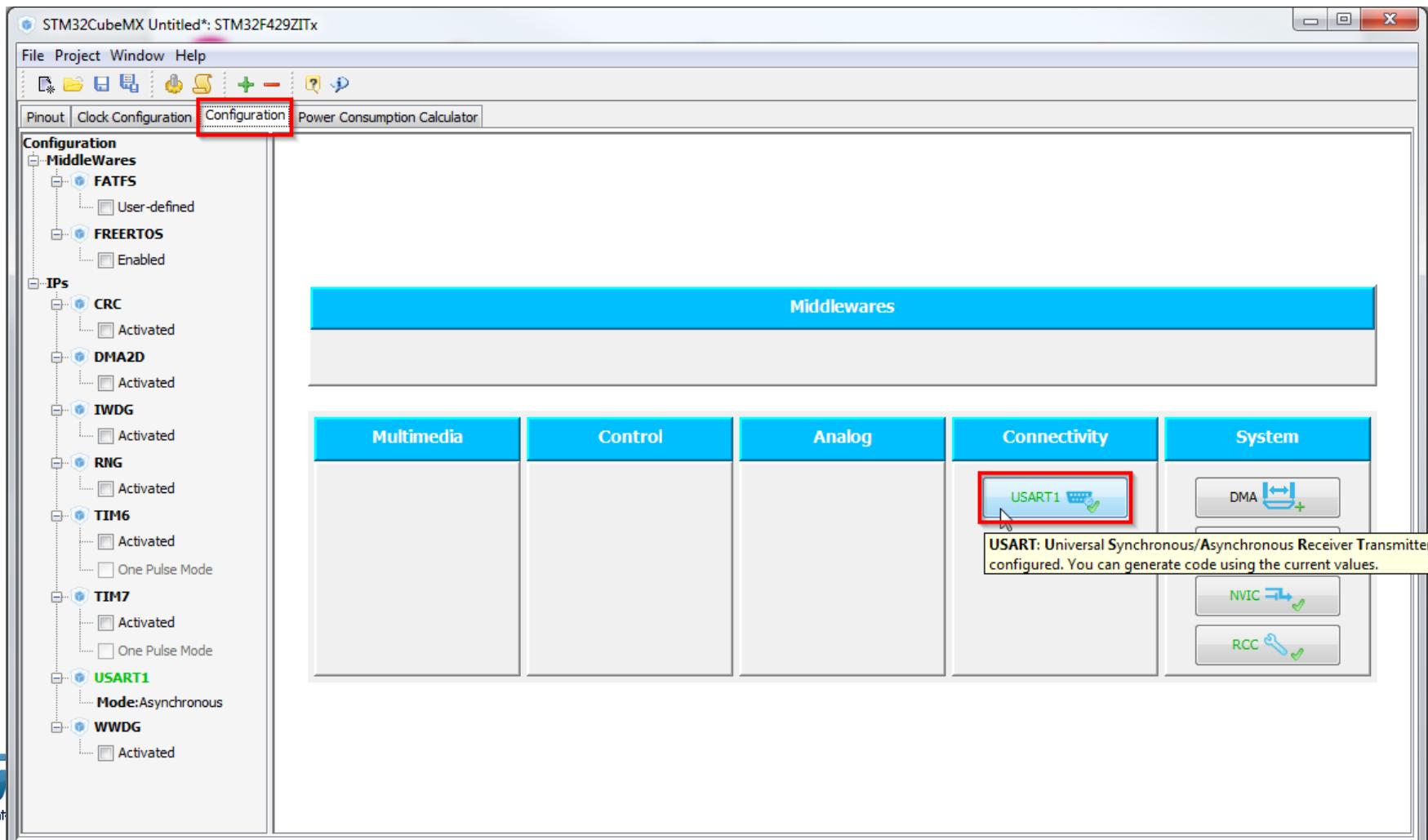
# Simple UART communication

- In order to run on maximum frequency, setup clock system
- Details in lab 0



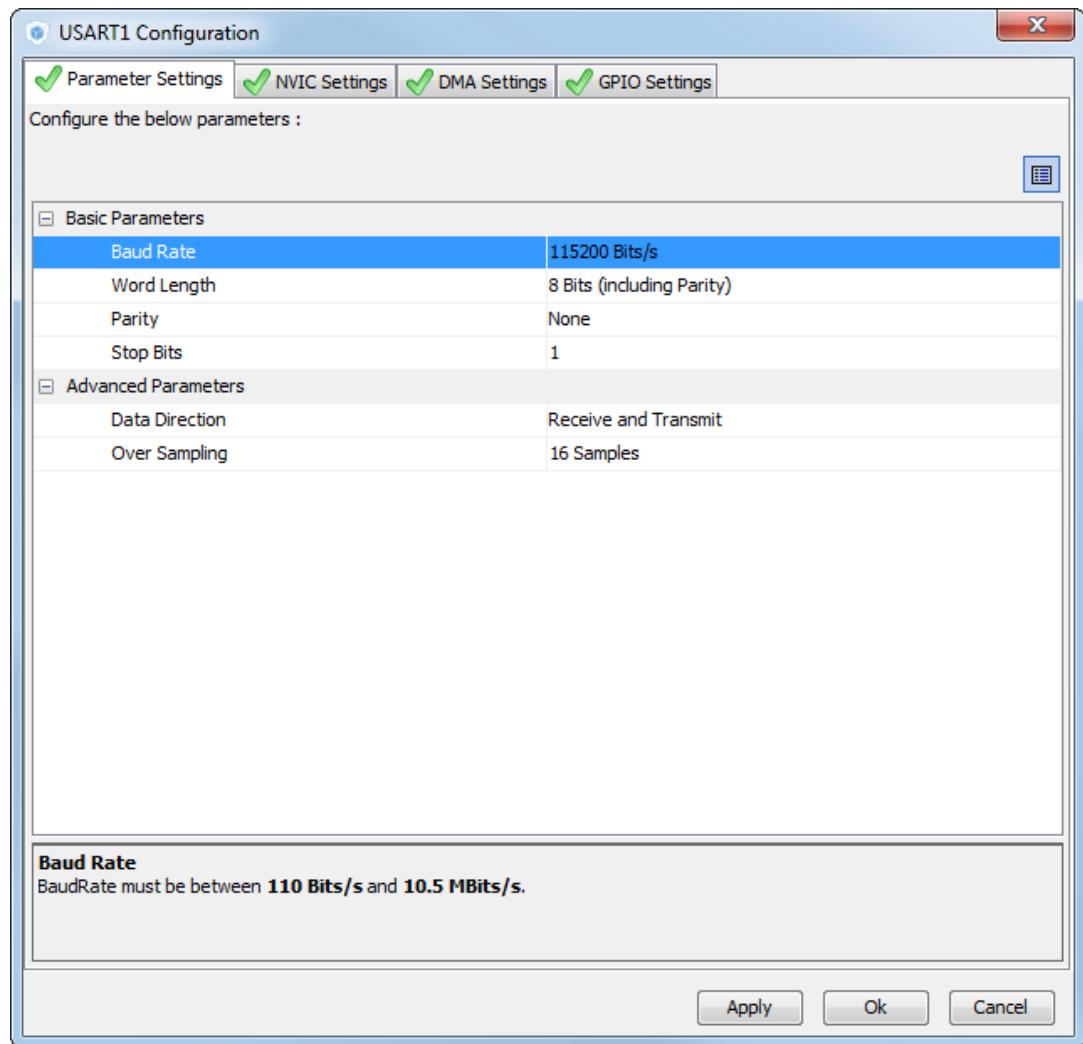
# Simple UART communication

- CubeMX UART configuration
  - Tab>Configuration>Connectivity>USART1



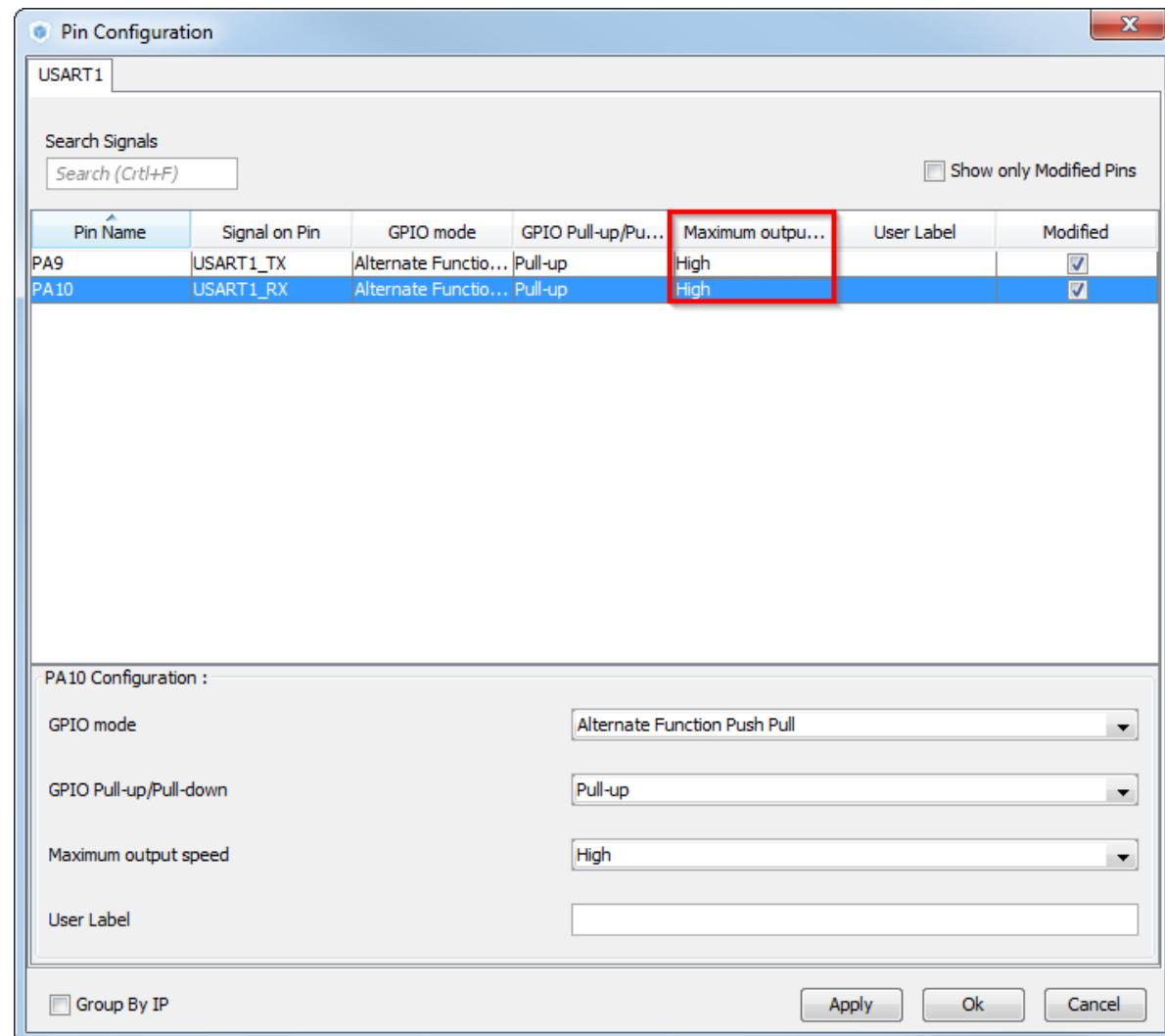
- CubeMX USART configuration check:

- BaudRate
- Word length
- Parity
- Stop bits
- Data direction
- Oversampling



- CubeMX USART GPIO configuration check:

- On high baud rate set the GPIO speed to HIGH
- TAB>Configuration>System>>GPIO
- Set the HIGH output speed  
Button OK

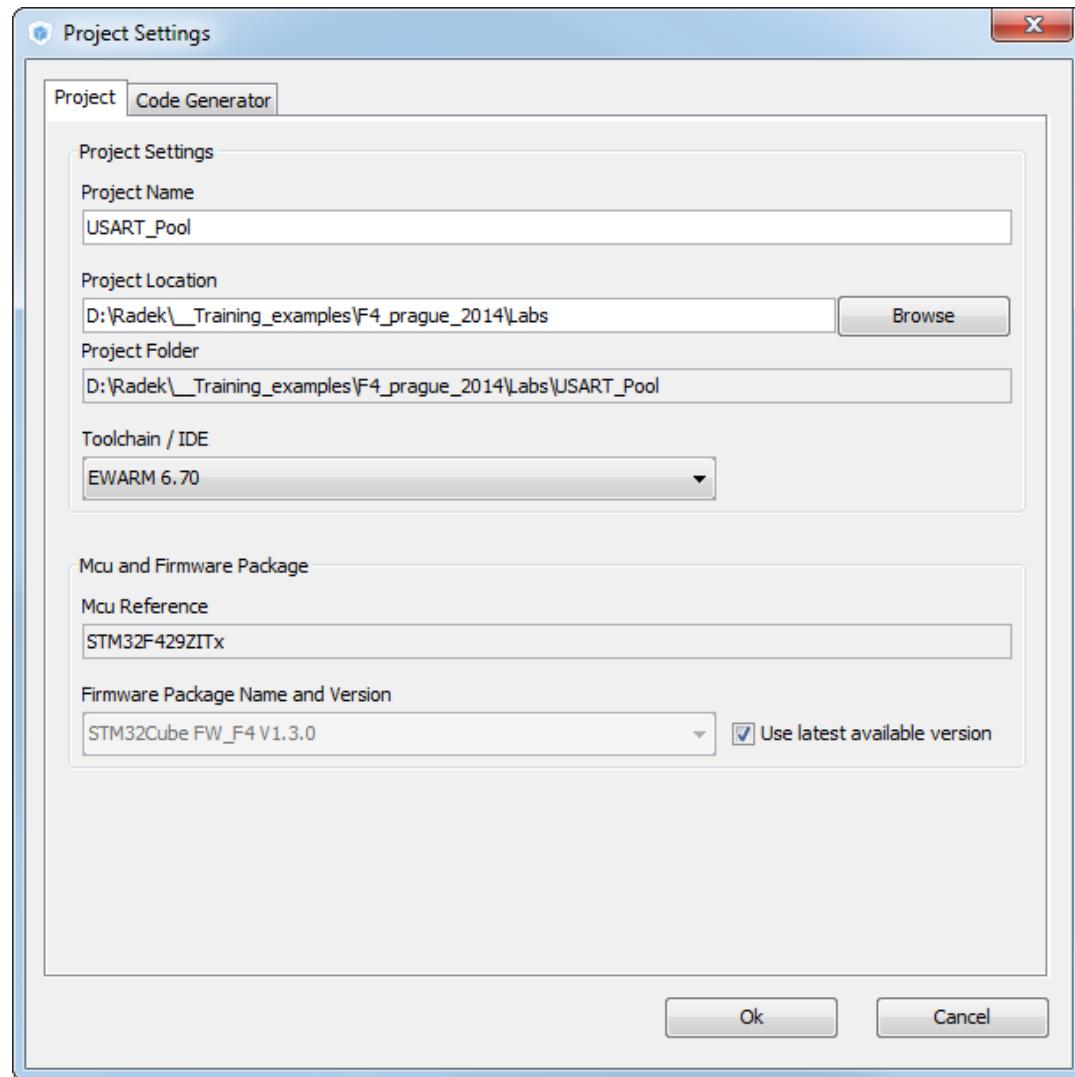


- Now we set the project details for generation

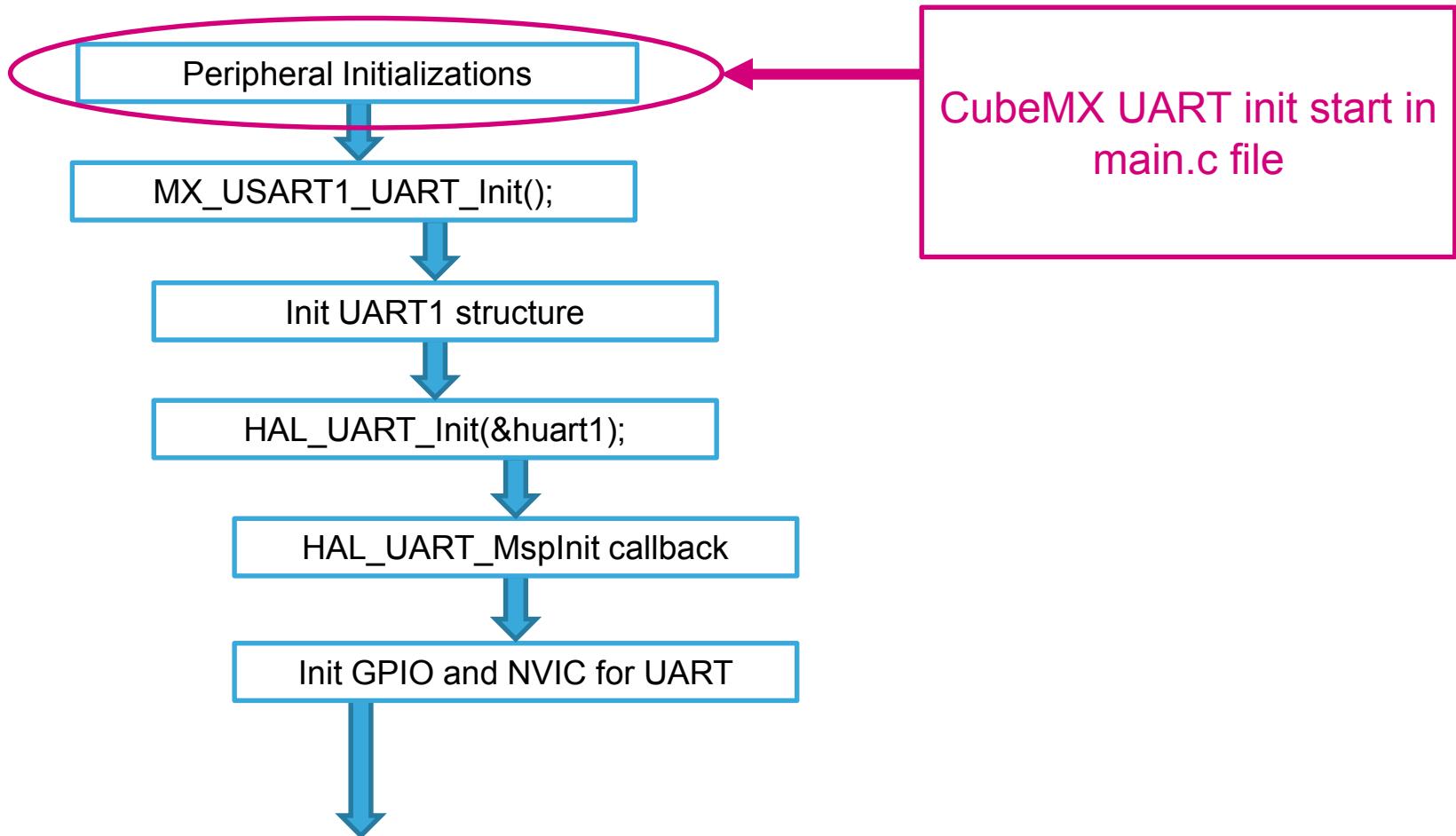
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

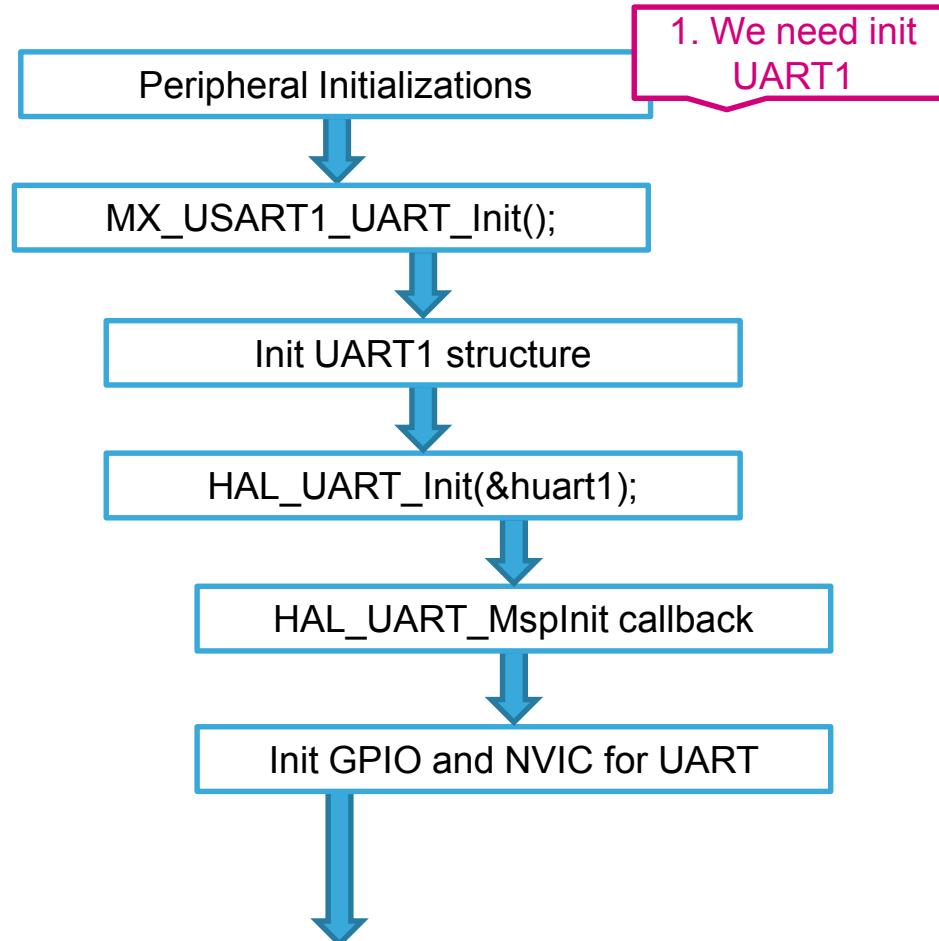
- Menu > Project > Generate Code



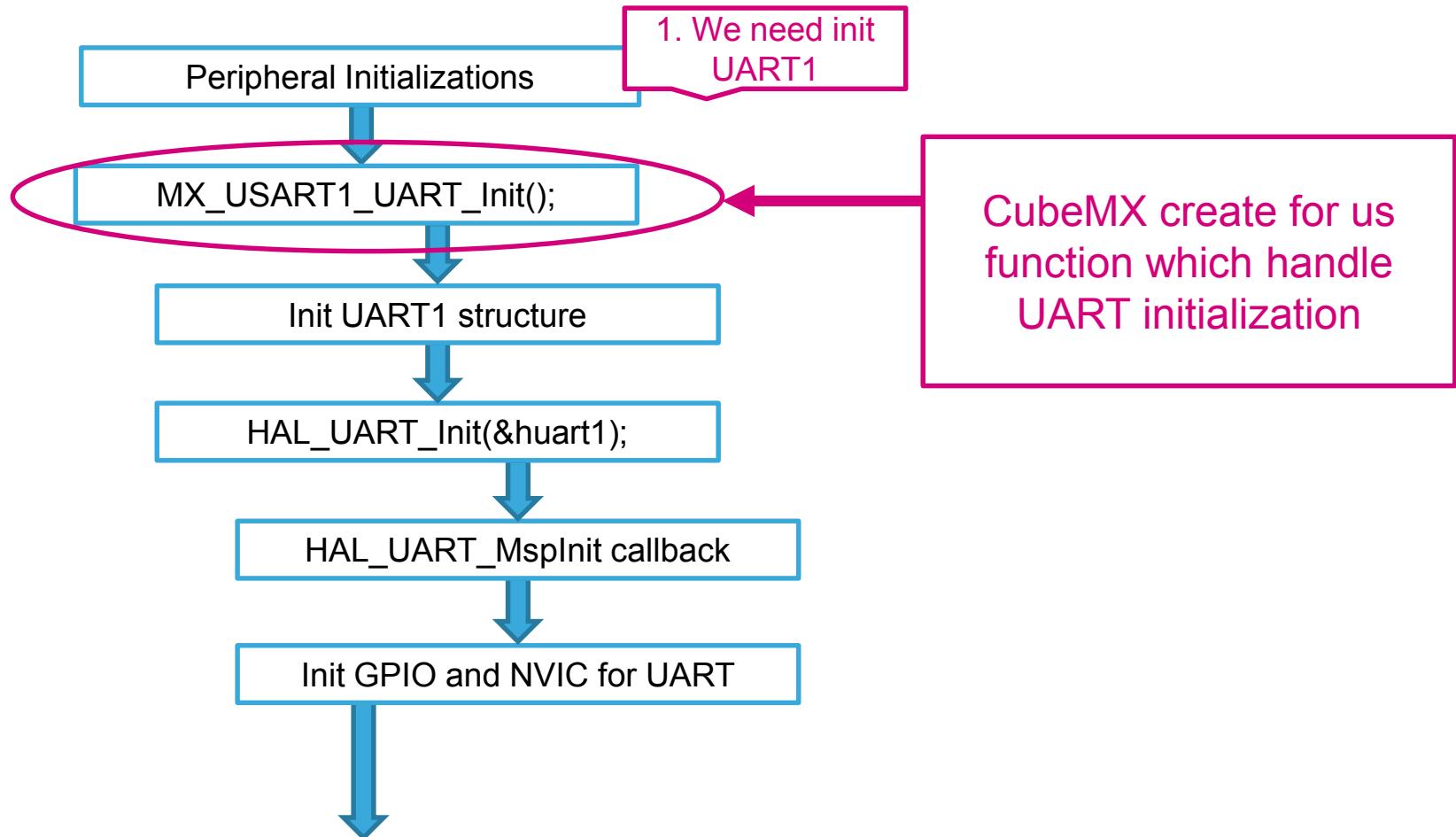
## HAL Library init flow



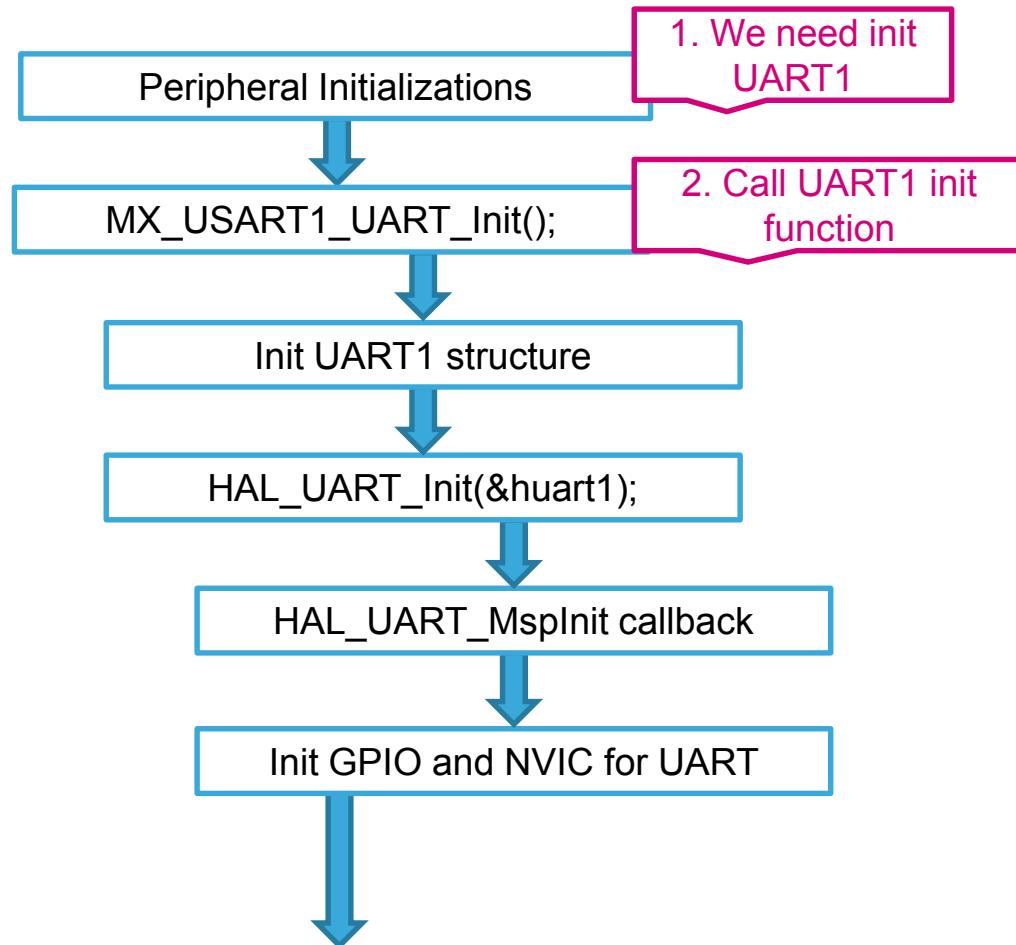
## HAL Library init flow



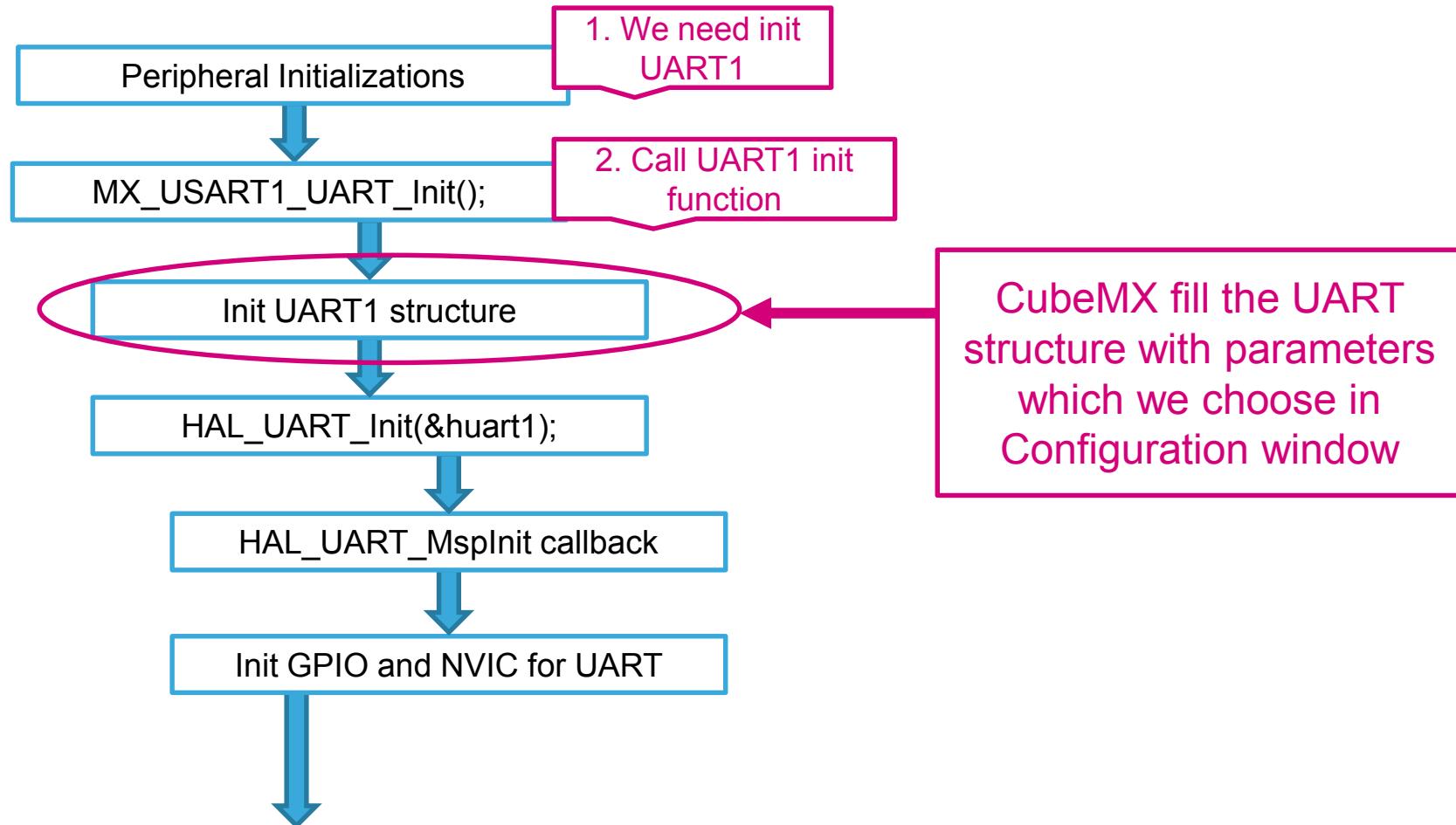
## HAL Library init flow



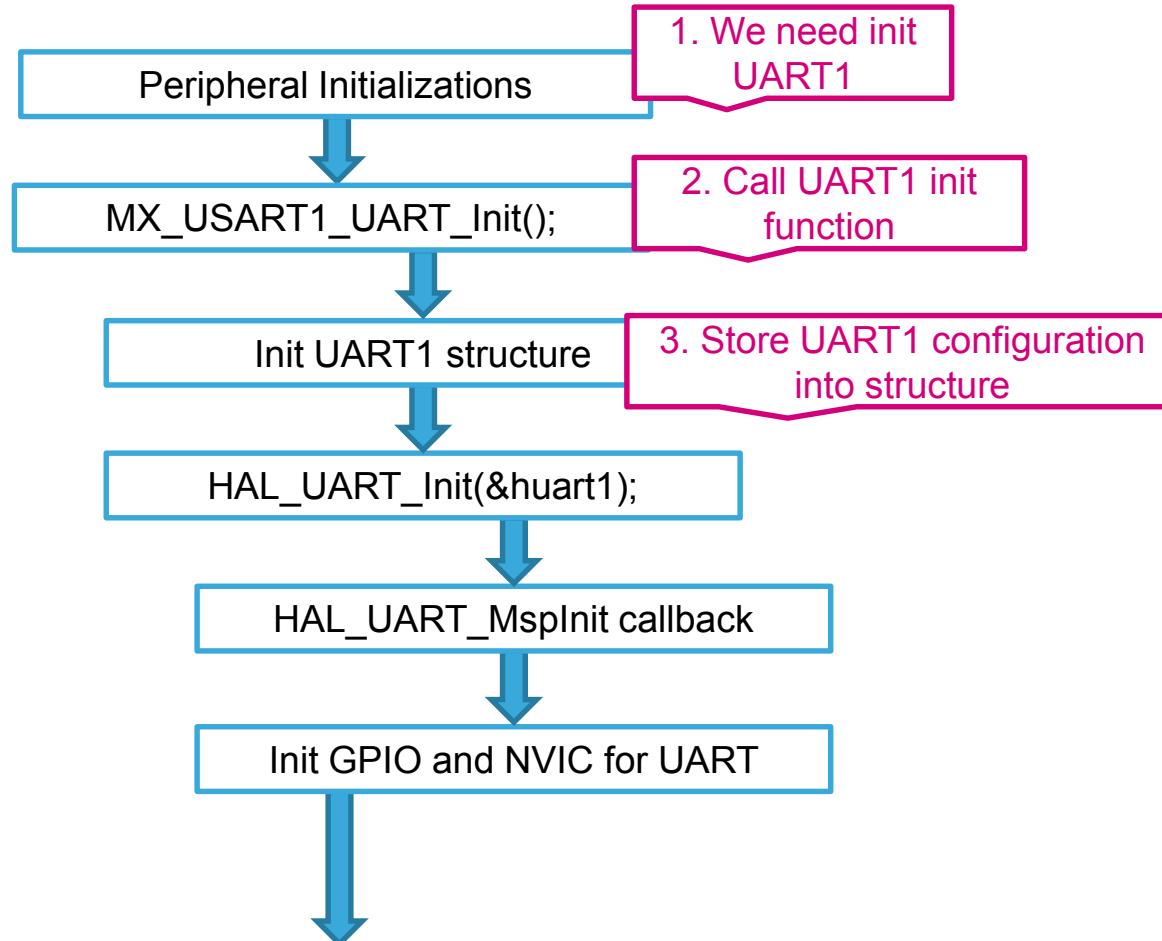
## HAL Library init flow



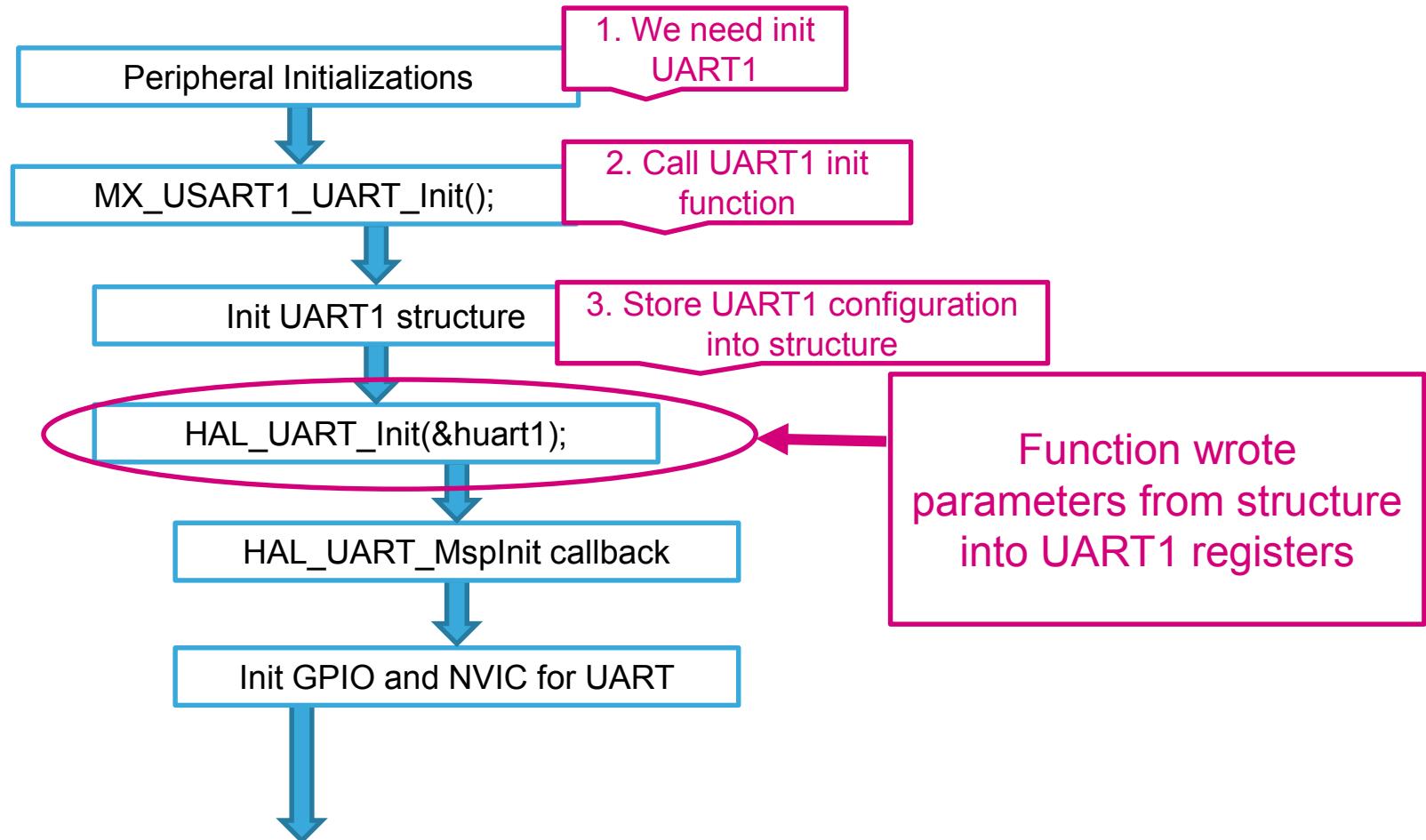
## HAL Library init flow



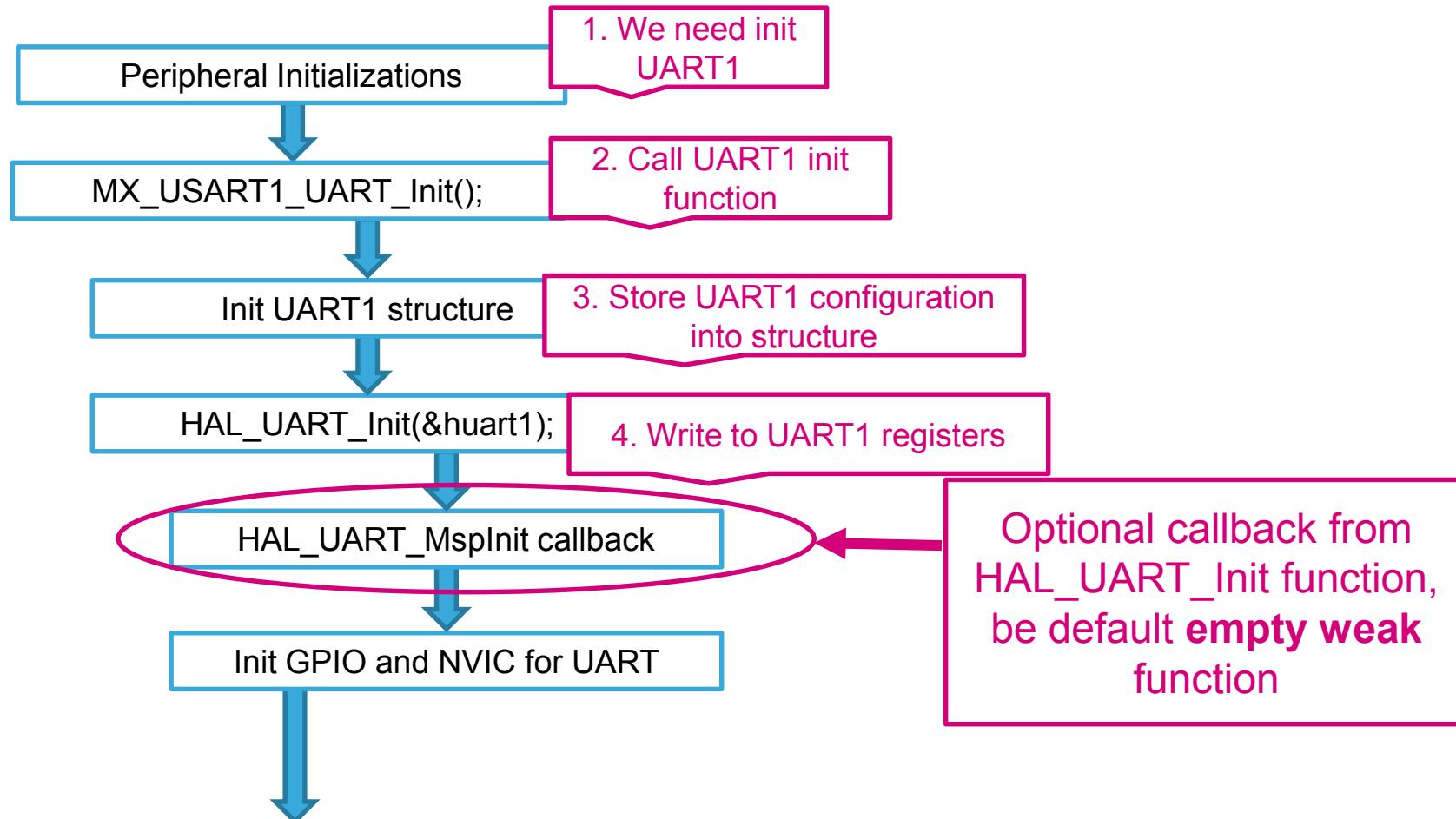
## HAL Library init flow



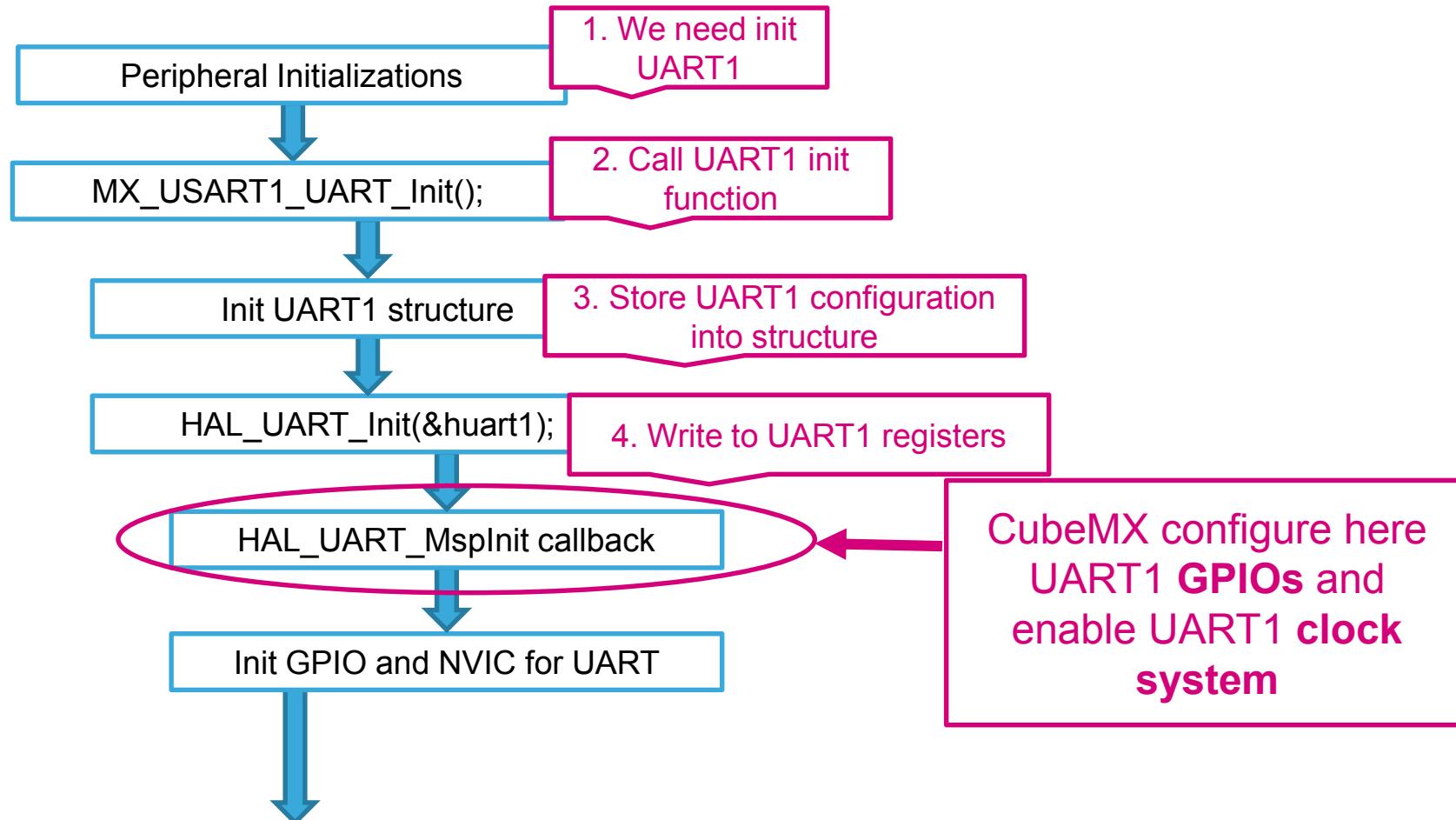
## HAL Library init flow



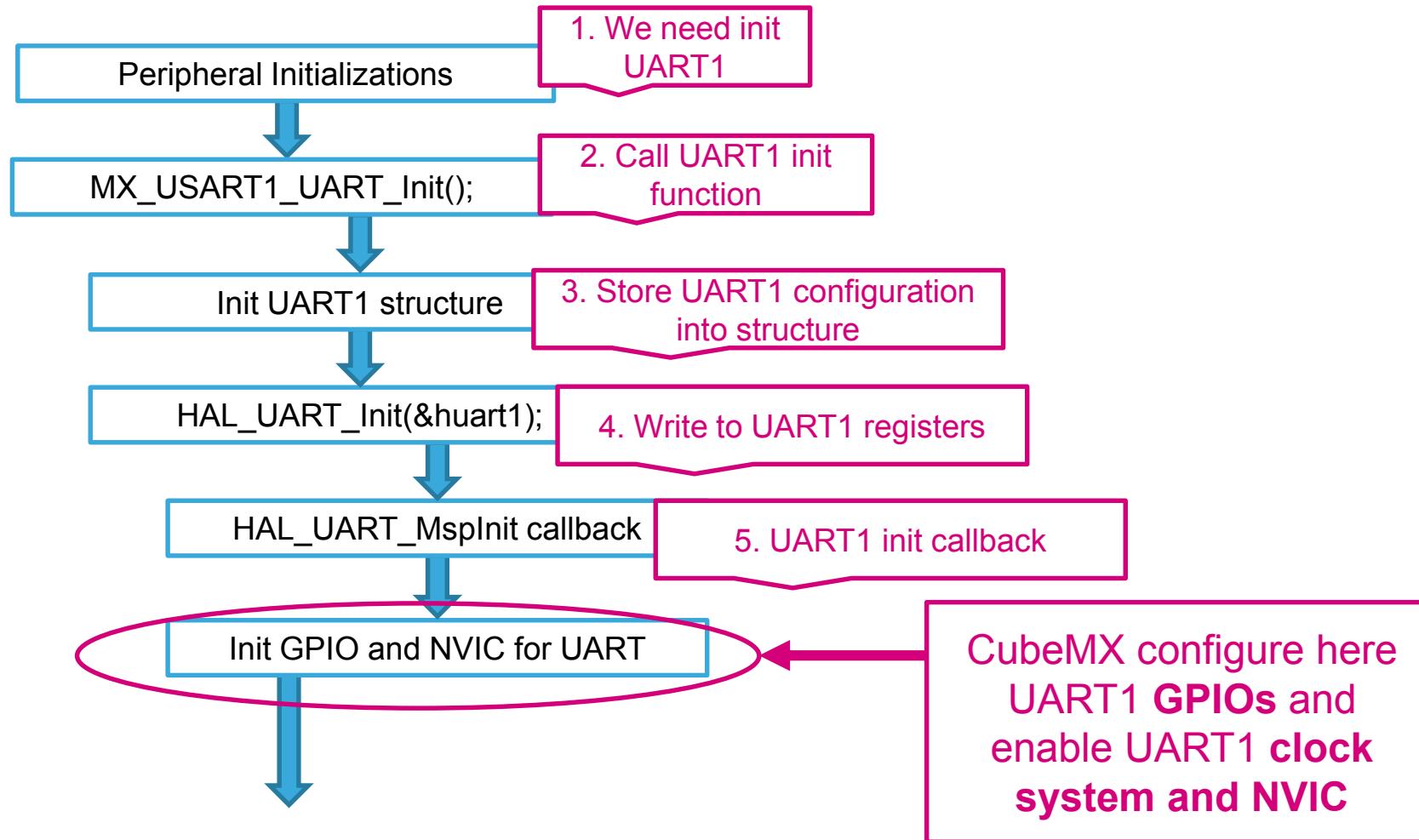
## HAL Library init flow



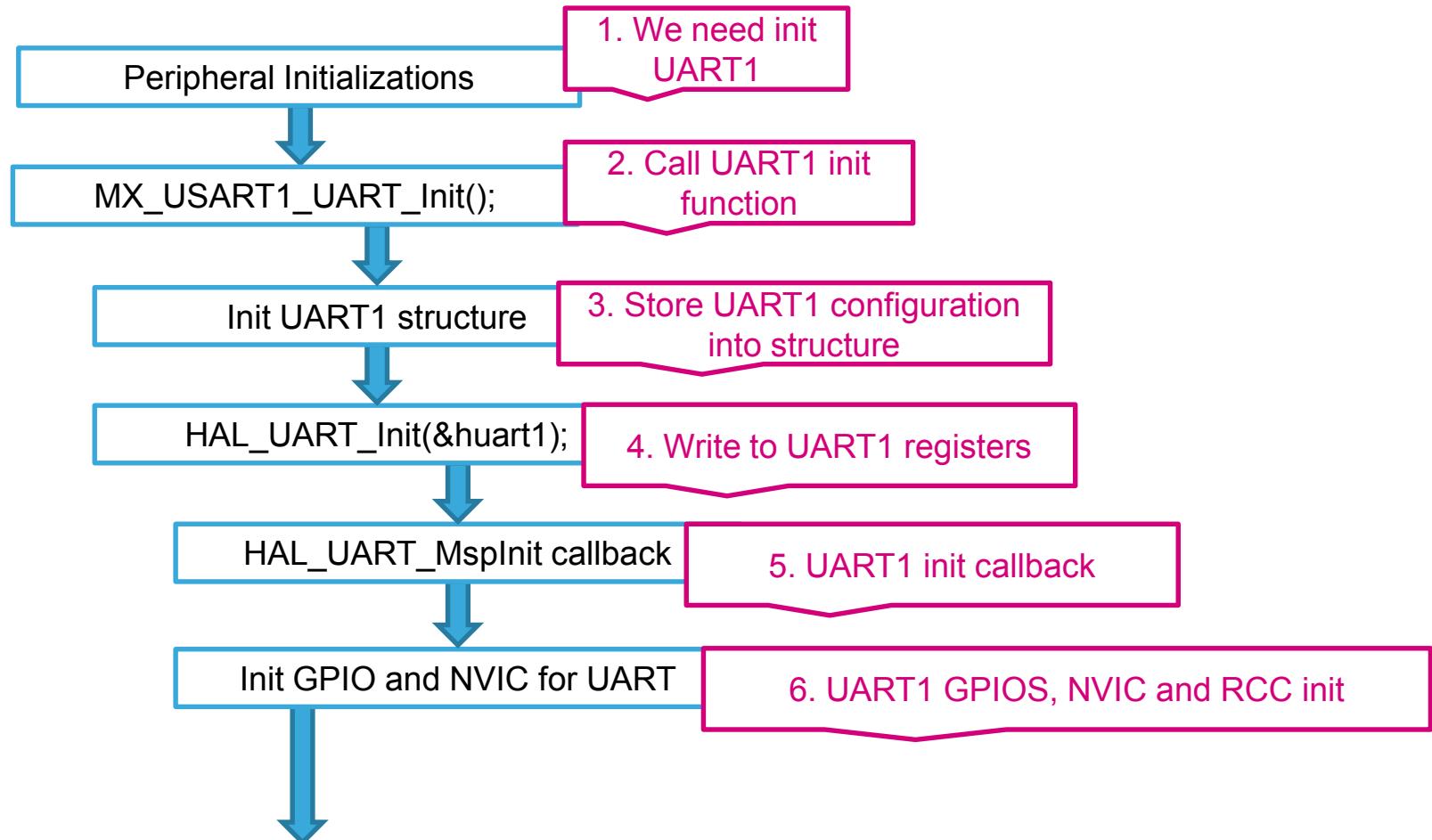
## HAL Library init flow



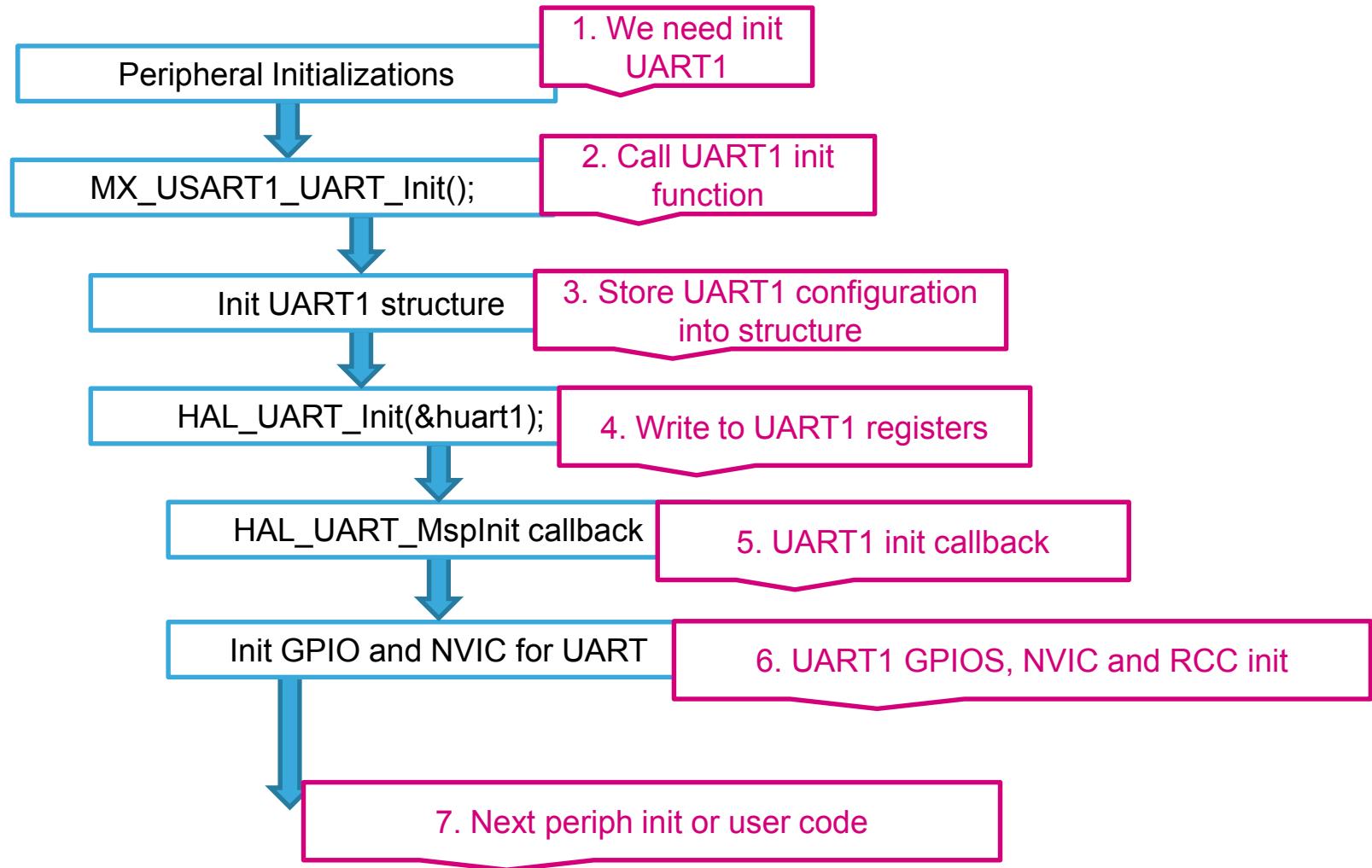
## HAL Library init flow



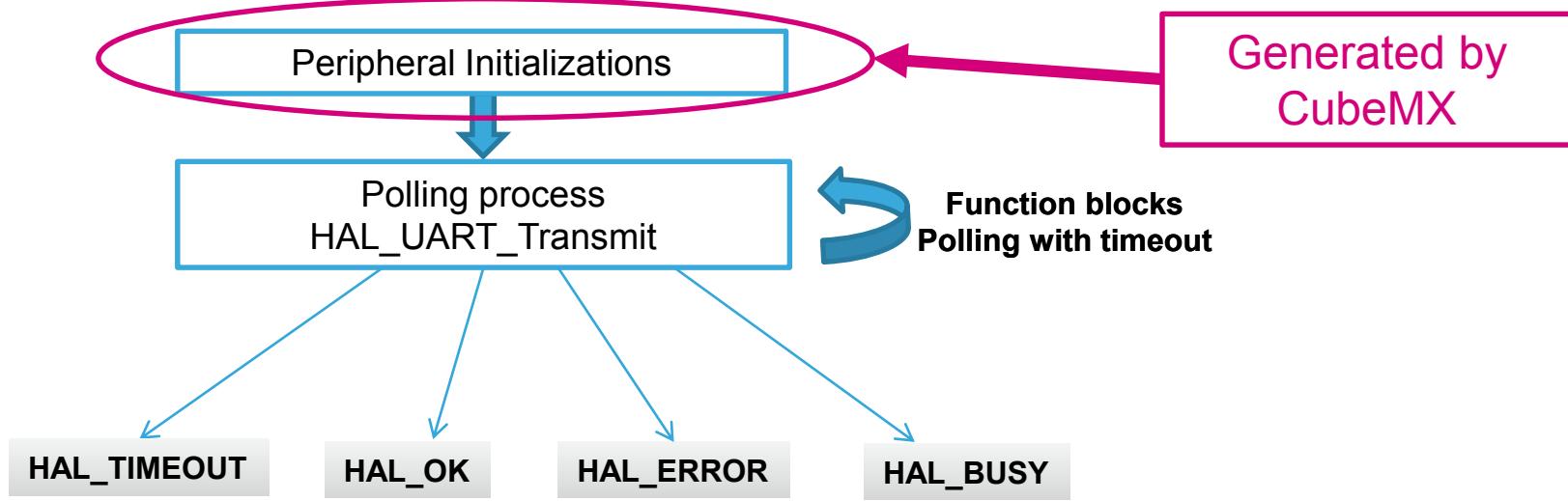
## HAL Library init flow



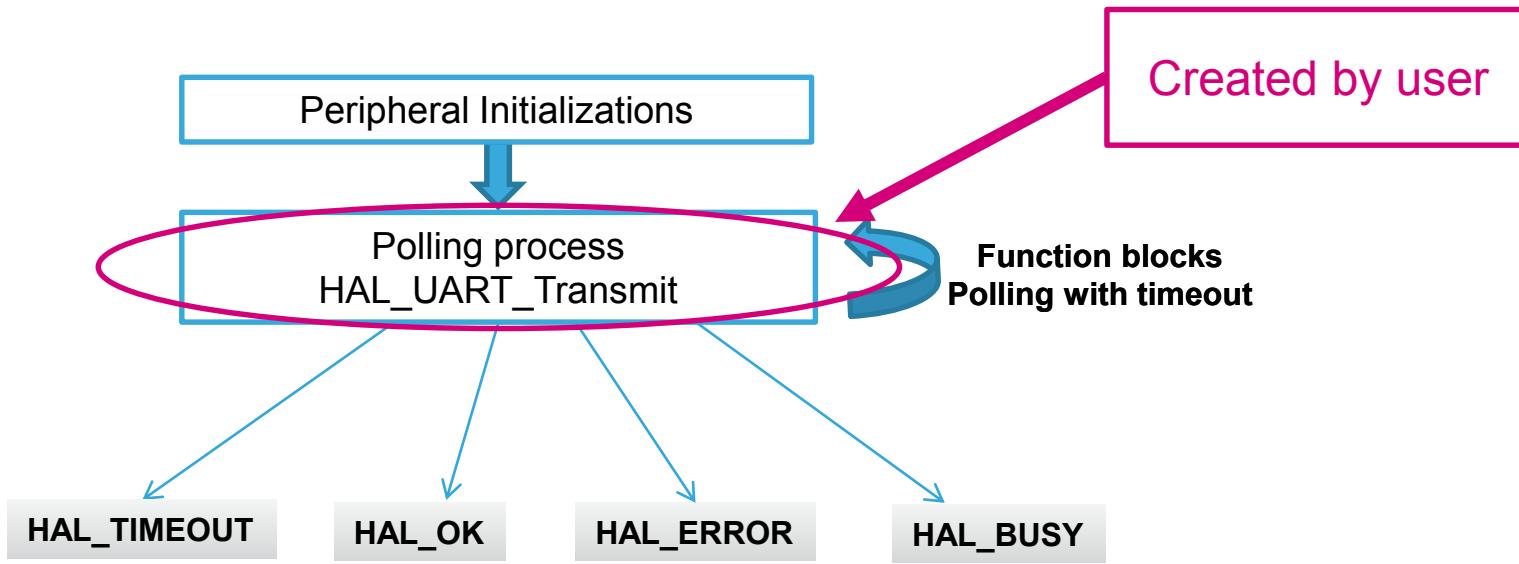
## HAL Library init flow



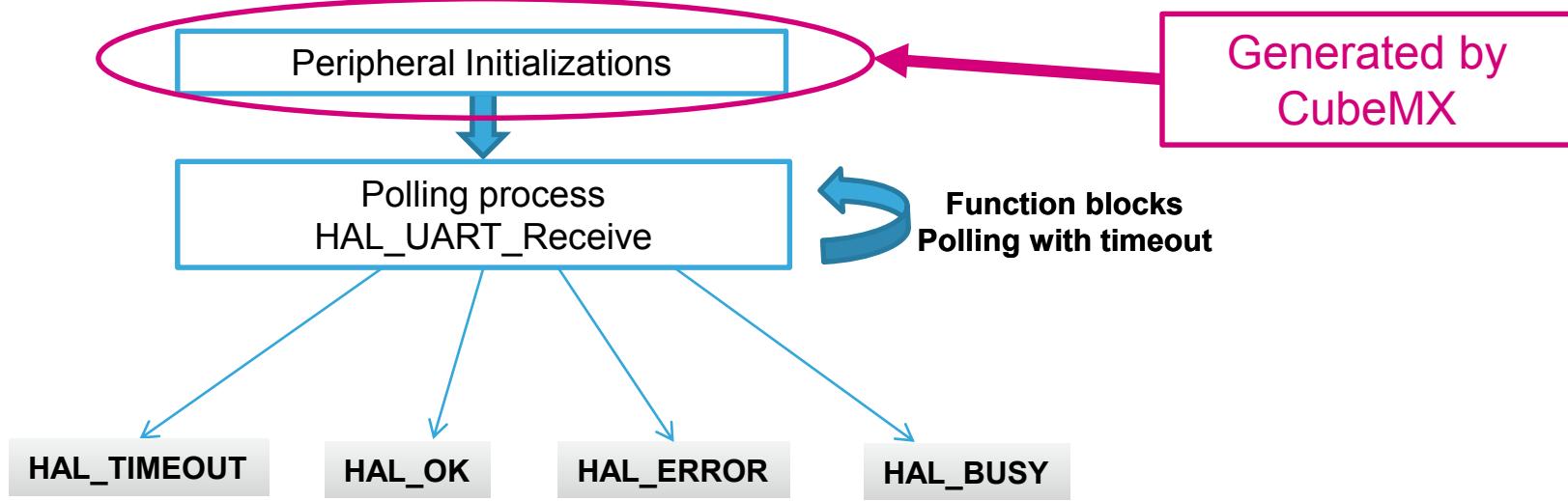
## HAL Library transmit flow



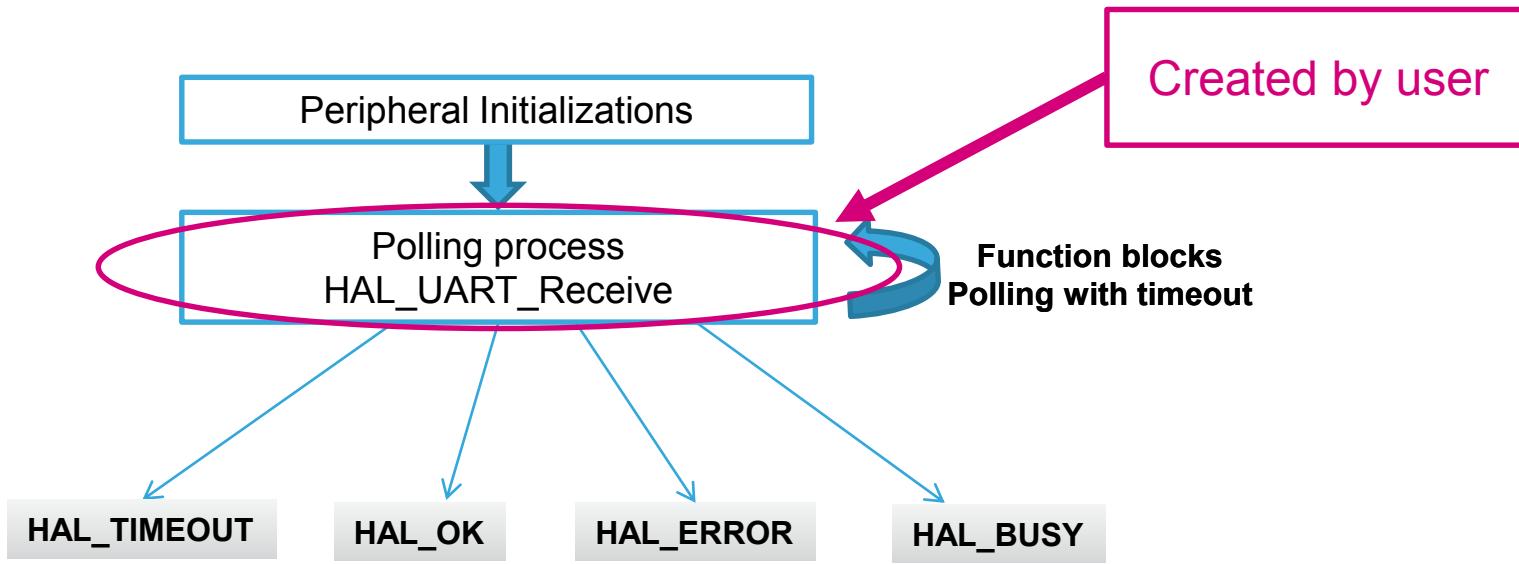
## HAL Library transmit flow



## HAL Library receive flow



## HAL Library receive flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 3 \*/* and */\* USER CODE END 3 \*/* tags
  - Into infinite while function
- For transmit use function
  - `HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
- For receive use function
  - `HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);`

- Transmit solution
  - Create data structure for data

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

- Call transmit function from while loop

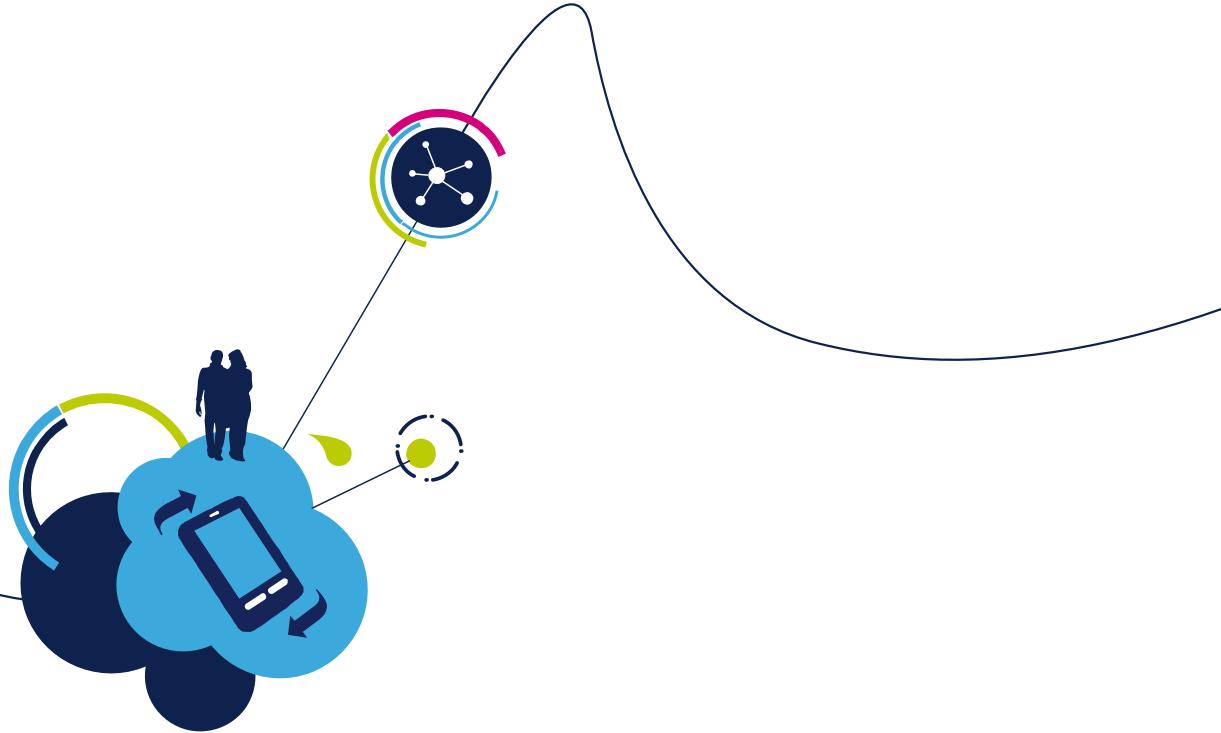
```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_UART_Transmit(&huart1,data,10,1000);  
}  
/* USER CODE END 3 */
```

- Receive solution
  - Create data structure for data

```
/* USER CODE BEGIN 0 */  
uint8_t data[10];  
/* USER CODE END 0 */
```

- Call transmit function from while loop

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_UART_Receive(&huart1,data,10,1000);  
}  
/* USER CODE END 3 */
```



# UART Interrupt lab 10

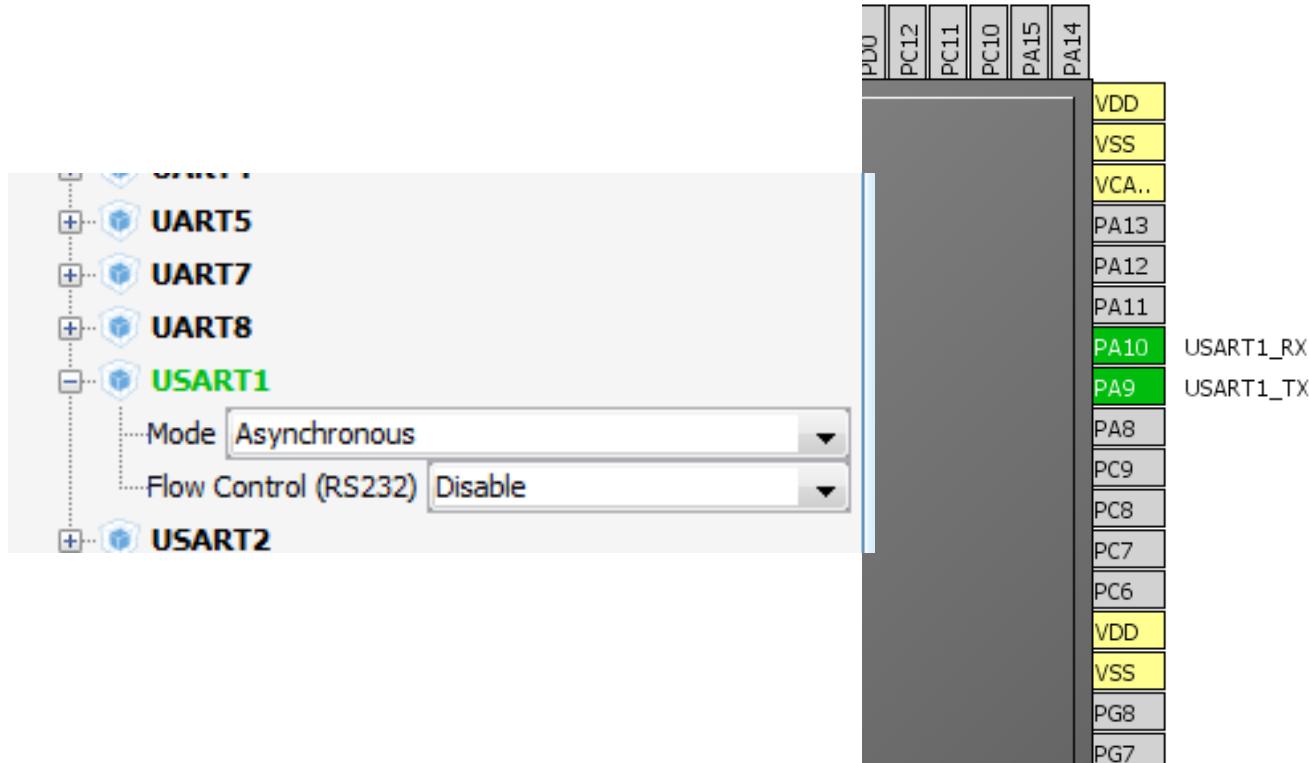
- Objective

- Learn how to setup UART with interrupts in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Create simple loopback example with interrupts

- Goal

- Configure UART in CubeMX and Generate Code
- Learn how to send and receive data over UART with interrupts
- Verify the correct functionality

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Pin selection
  - It will be same as previous lab we use again PA9 and PA10

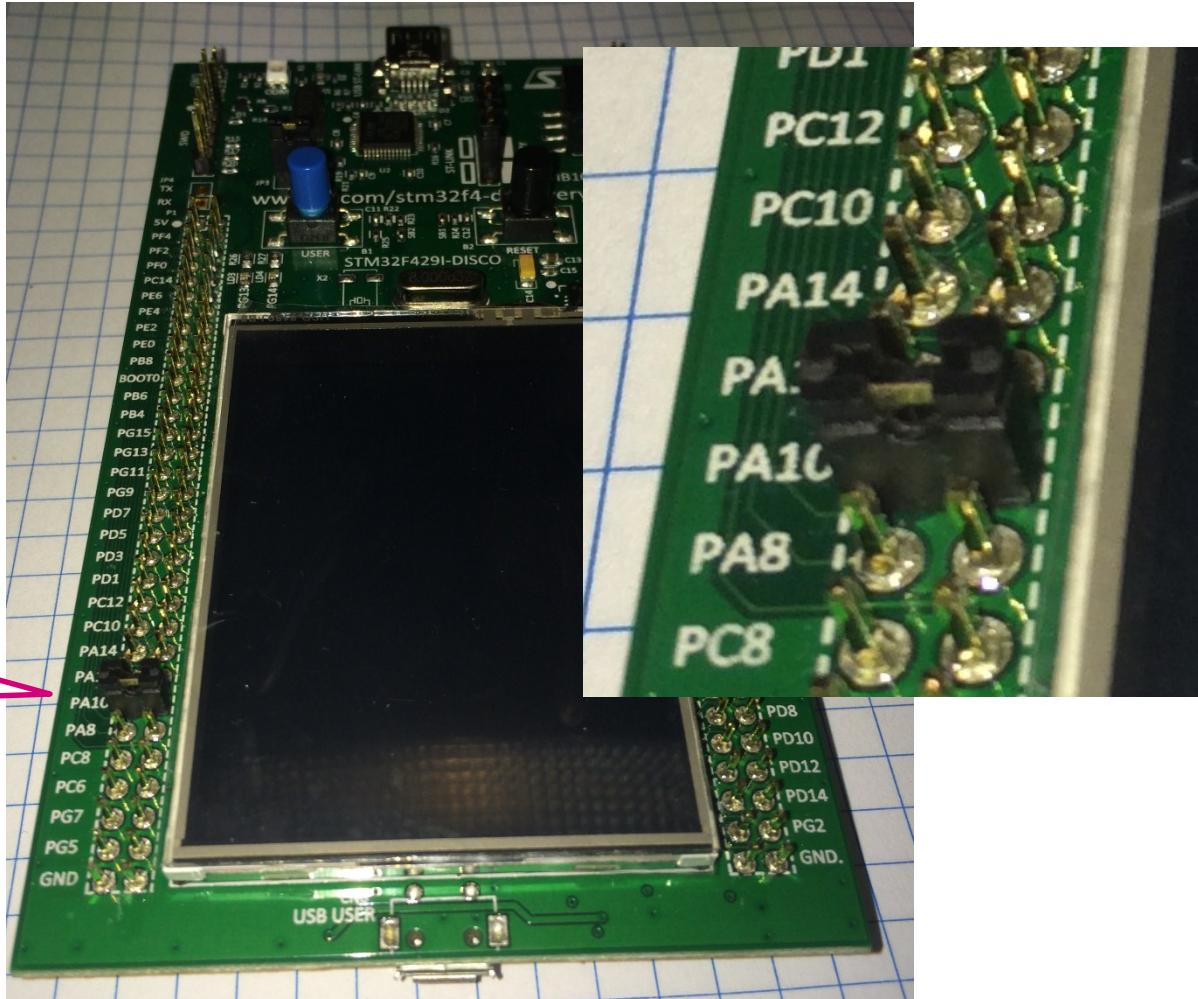


# 10

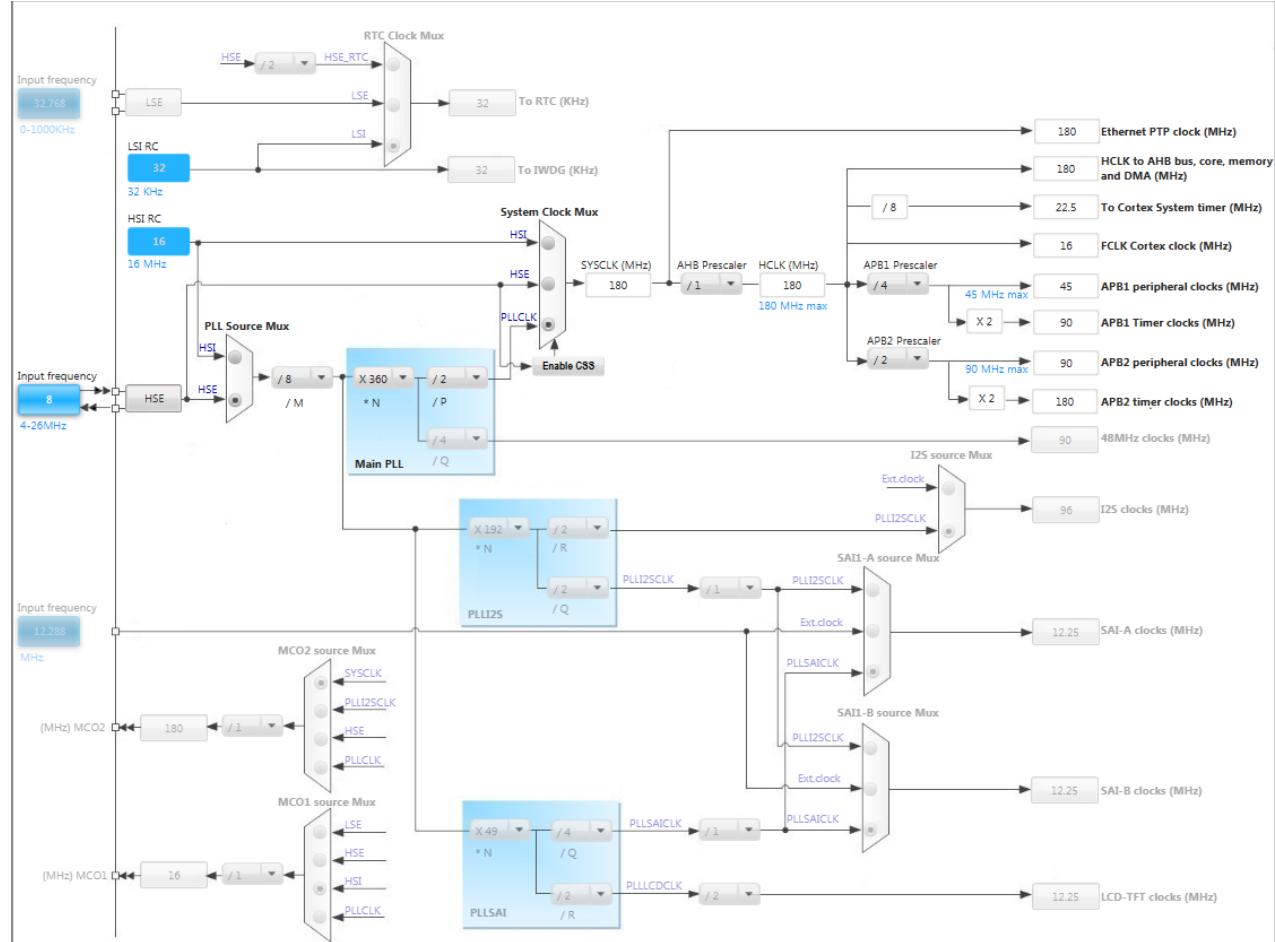
# Use UART with interrupt

169

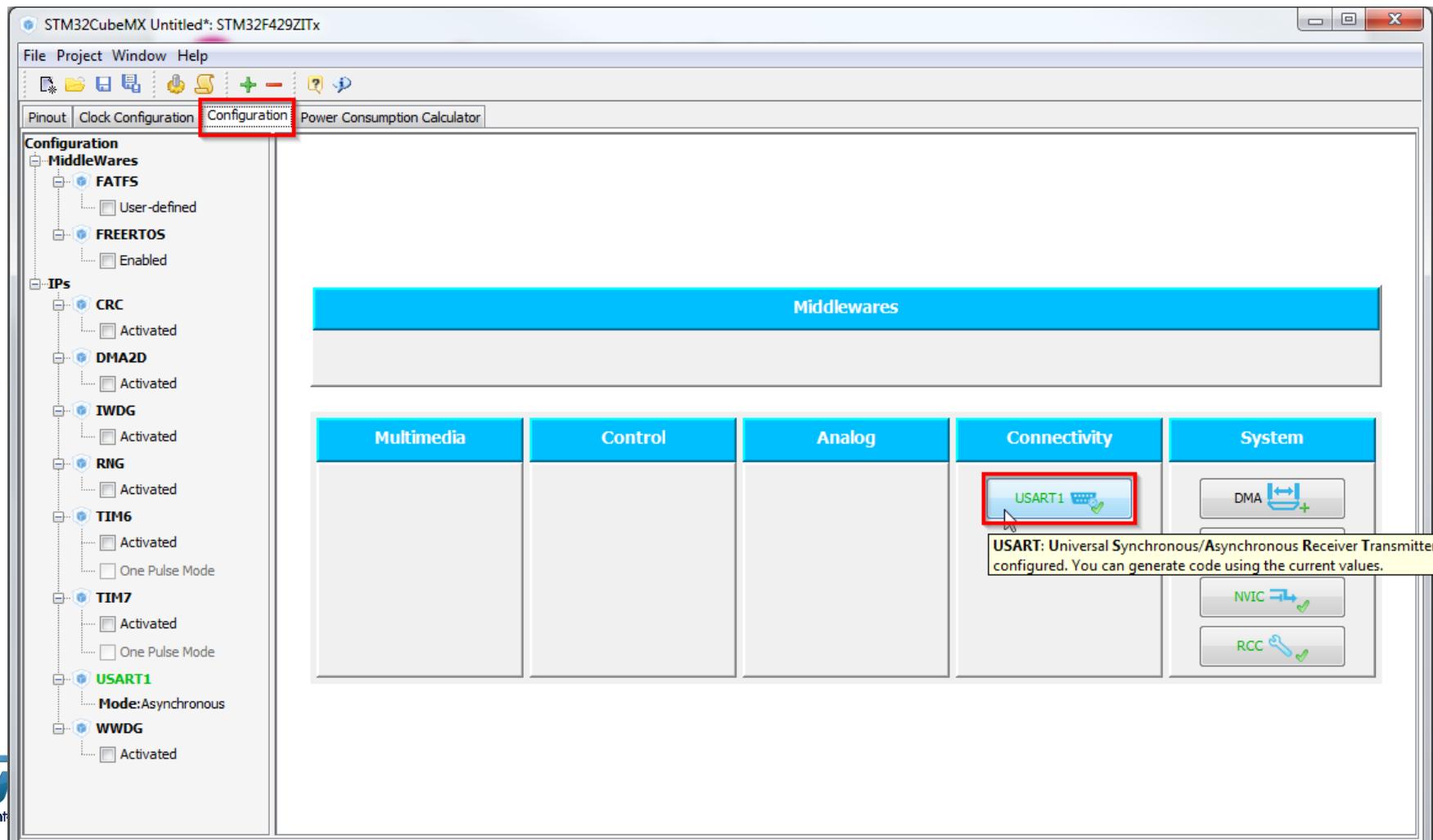
- Hardware preparation
  - We connect selected pins together by jumper, this help us to create loopback on UART



- In order to run on maximum frequency, setup clock system
- Details in lab 0

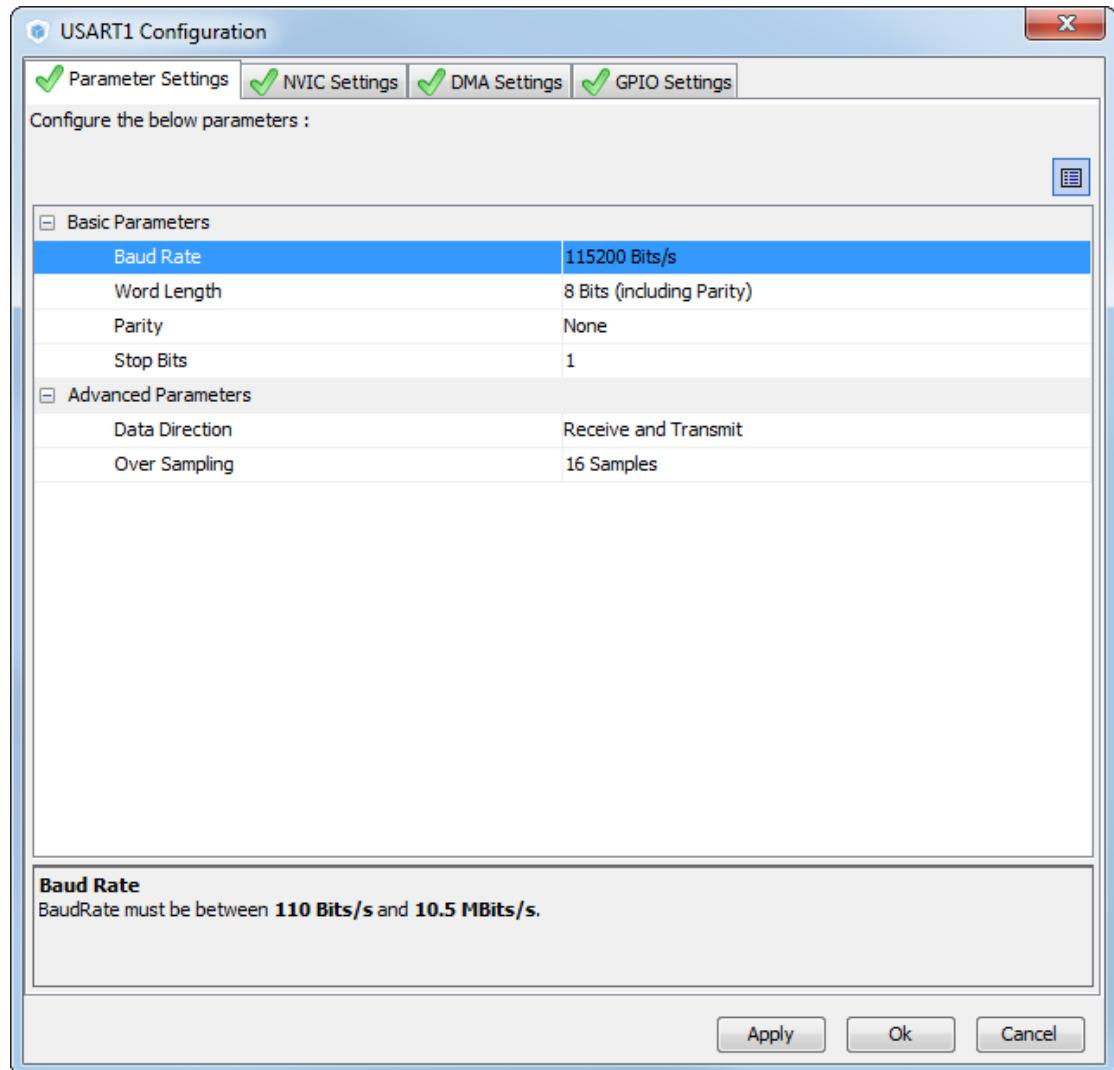


- CubeMX UART configuration
  - Tab>Configuration>Connectivity>USART1



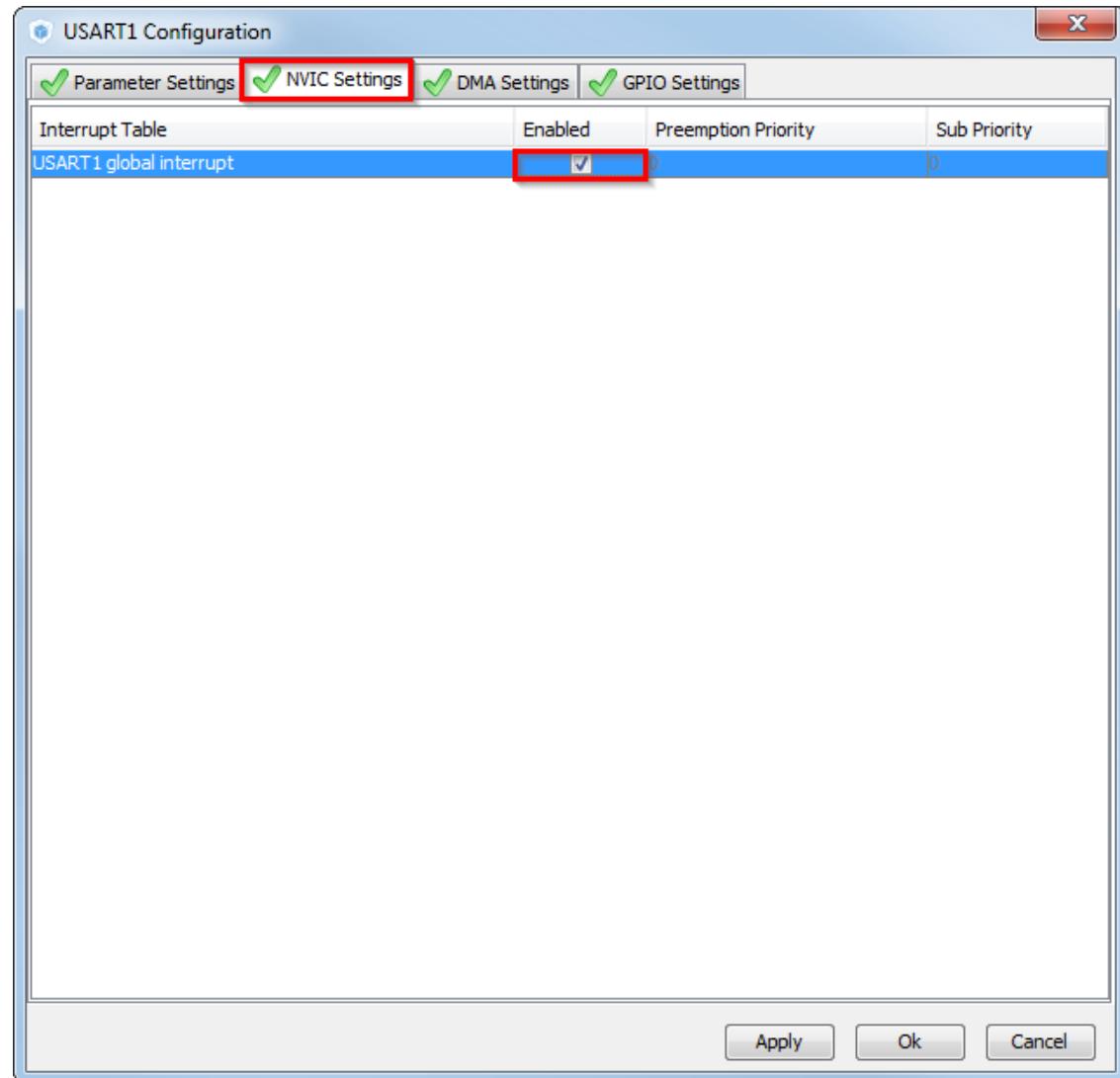
- CubeMX UART configuration check:

- BaudRate
- Word length
- Parity
- Stop bits
- Data direction
- Oversampling



- CubeMX USART configuration NVIC settings

- TAB>NVIC Settings
- Enable interrupts
- OK

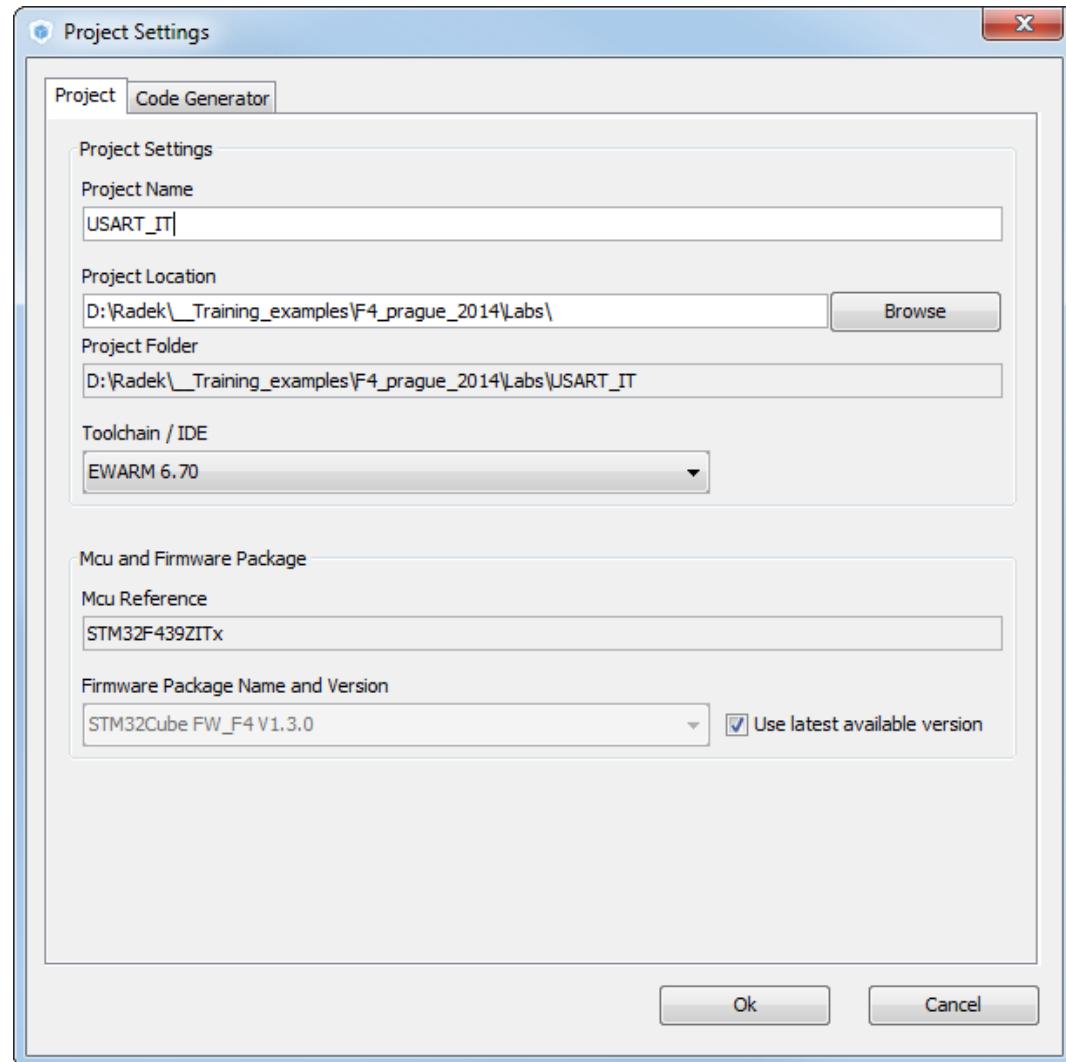


- Now we set the project details for generation

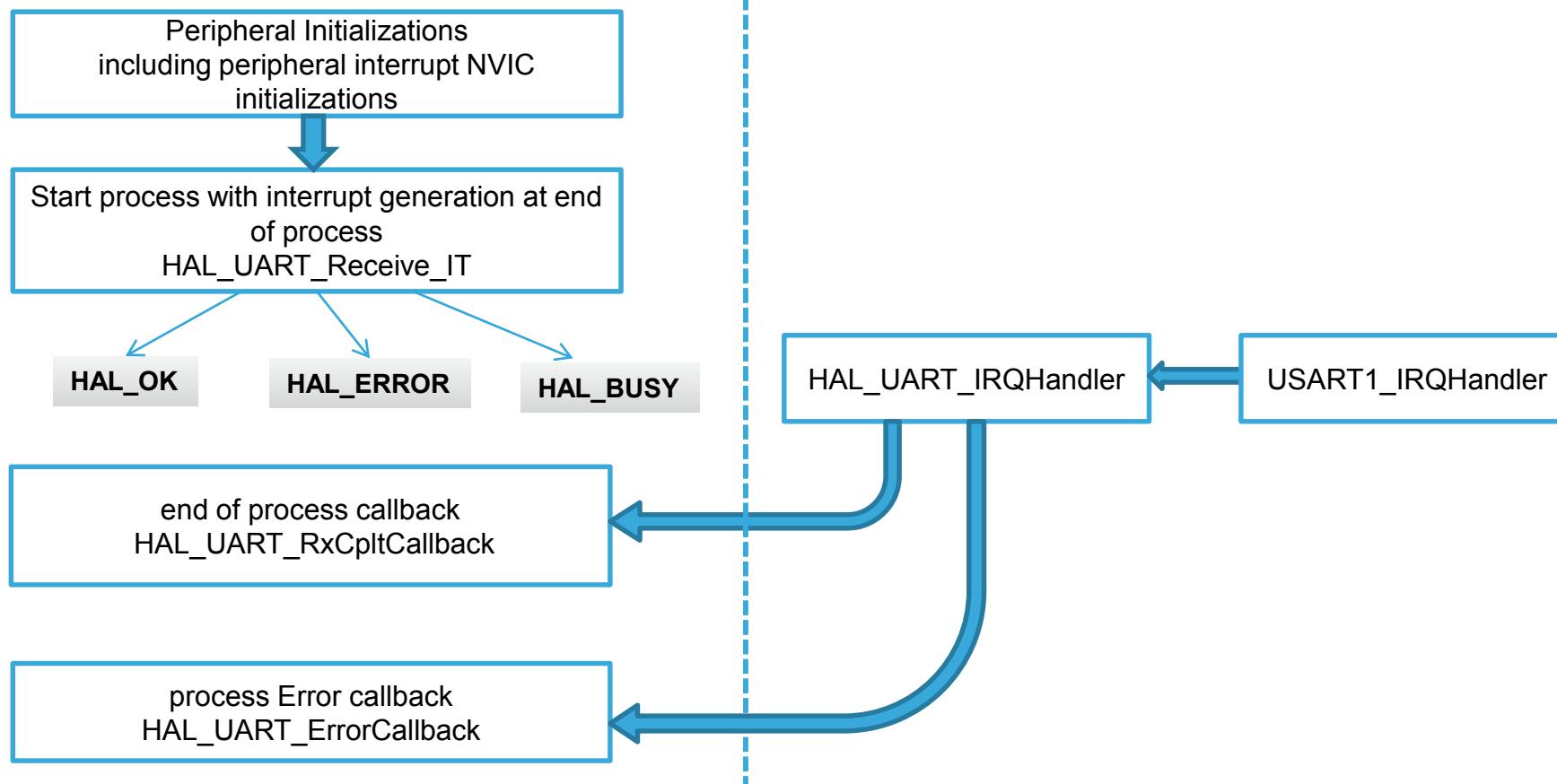
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

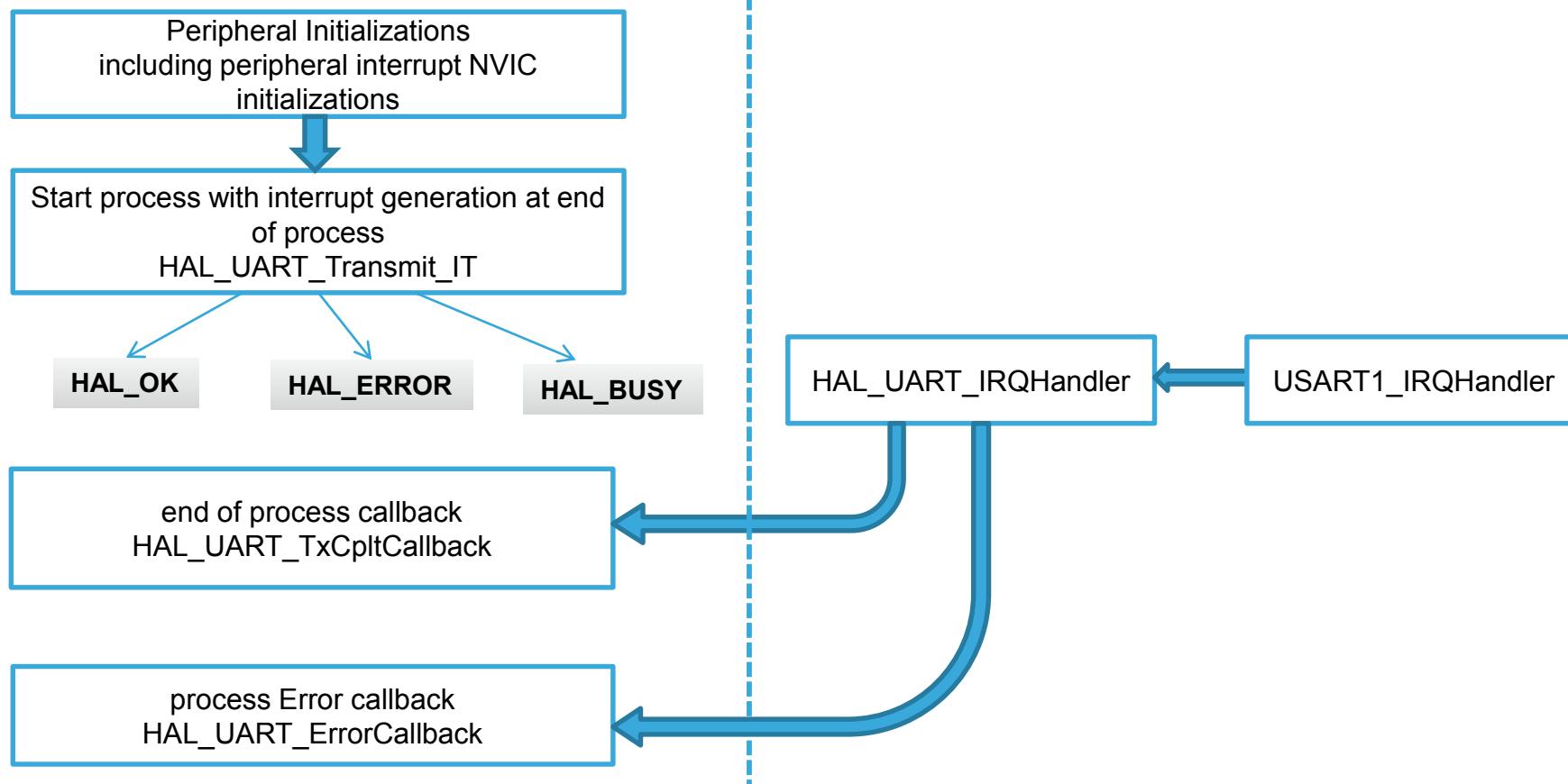
- Menu > Project > Generate Code



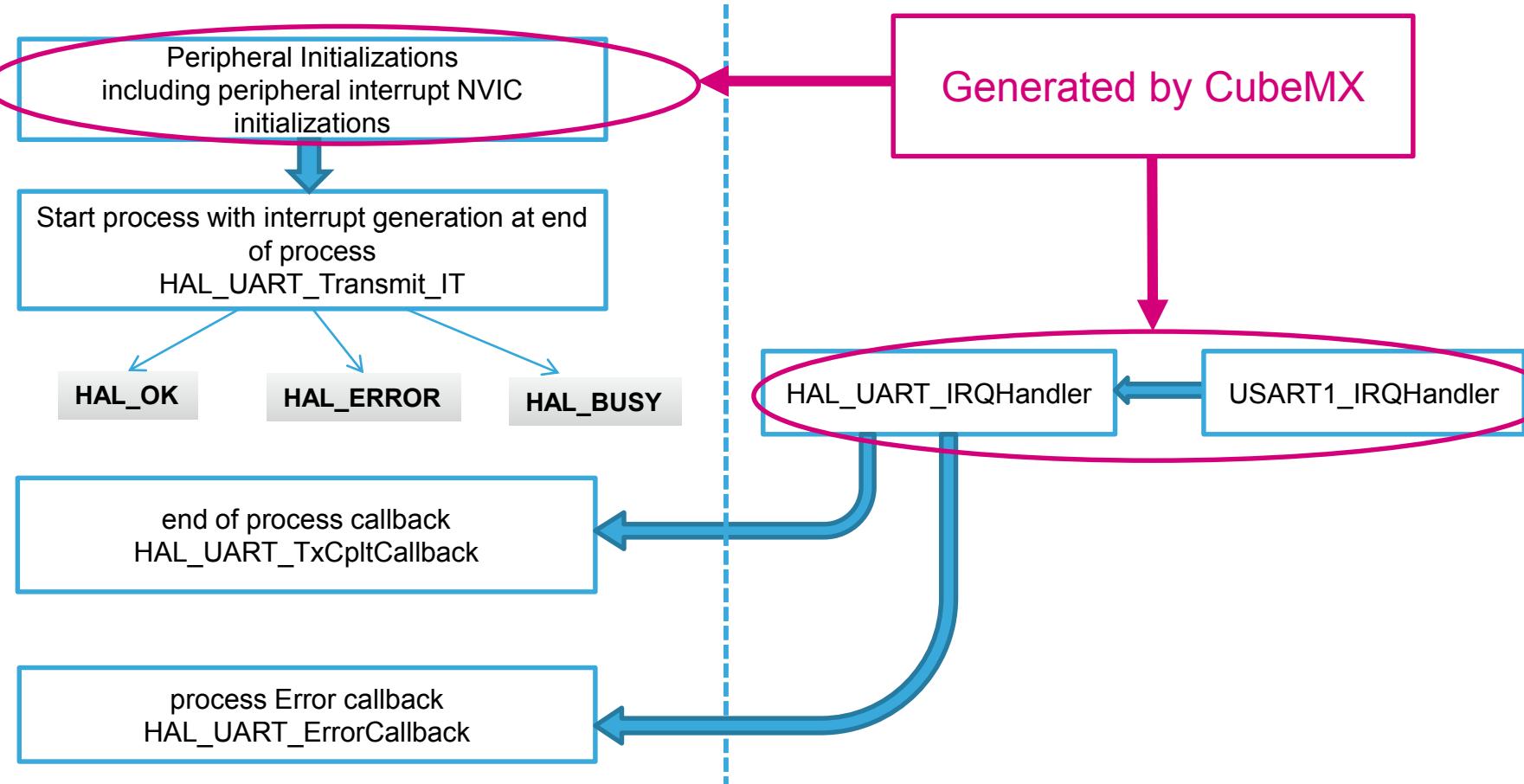
## HAL Library UART with IT receive flow



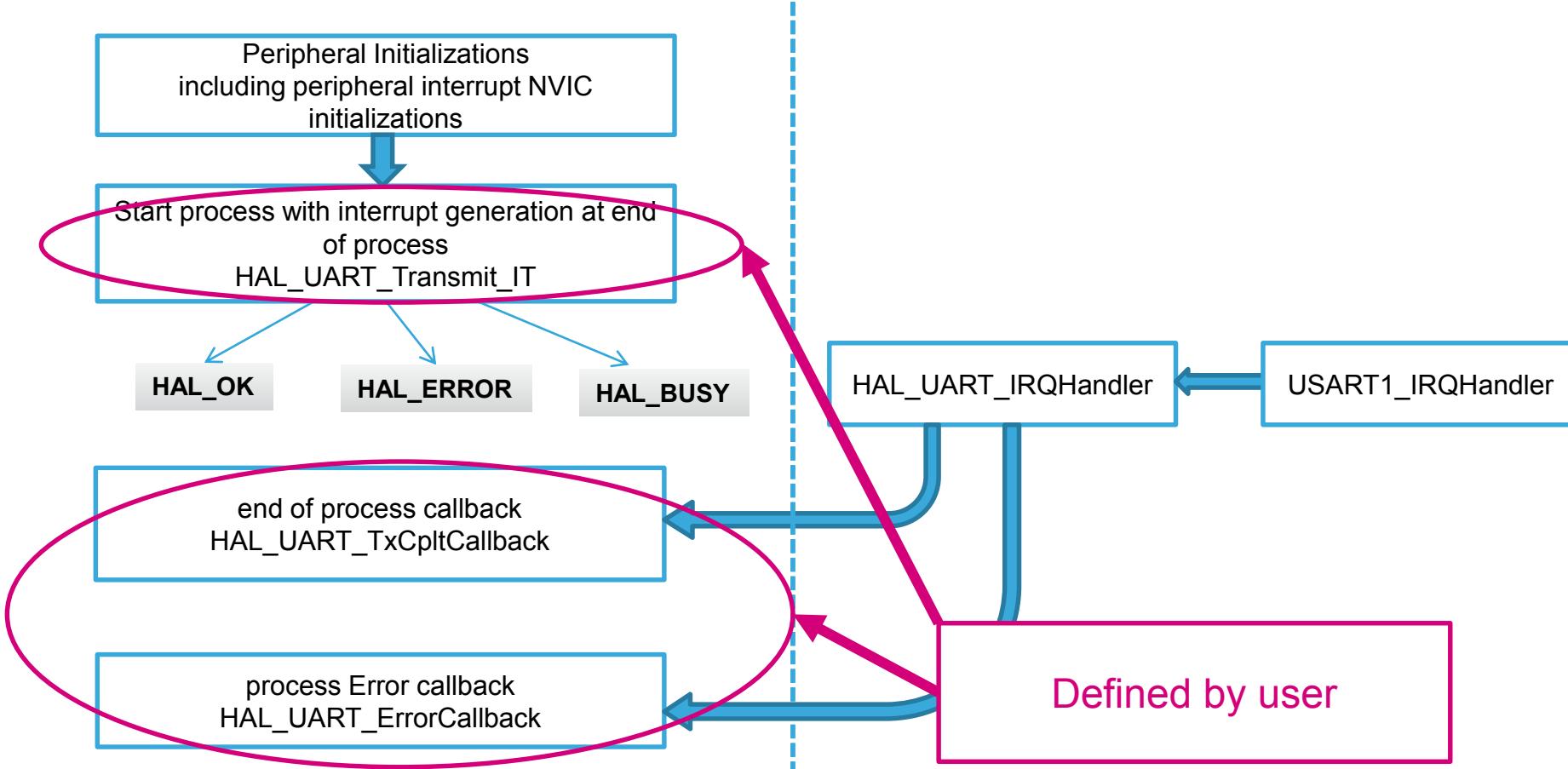
## HAL Library UART with IT transmit flow



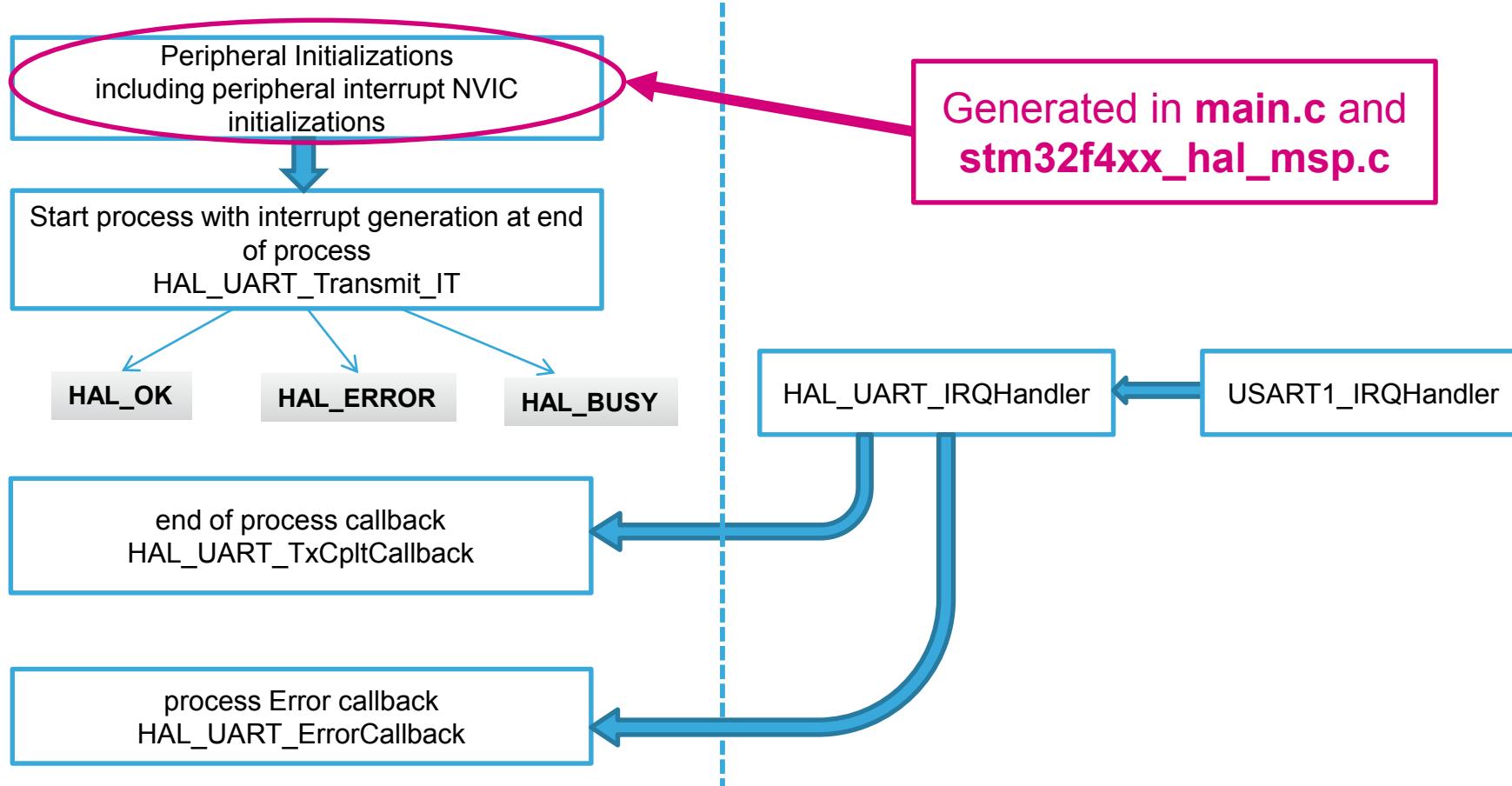
## HAL Library UART with IT transmit flow



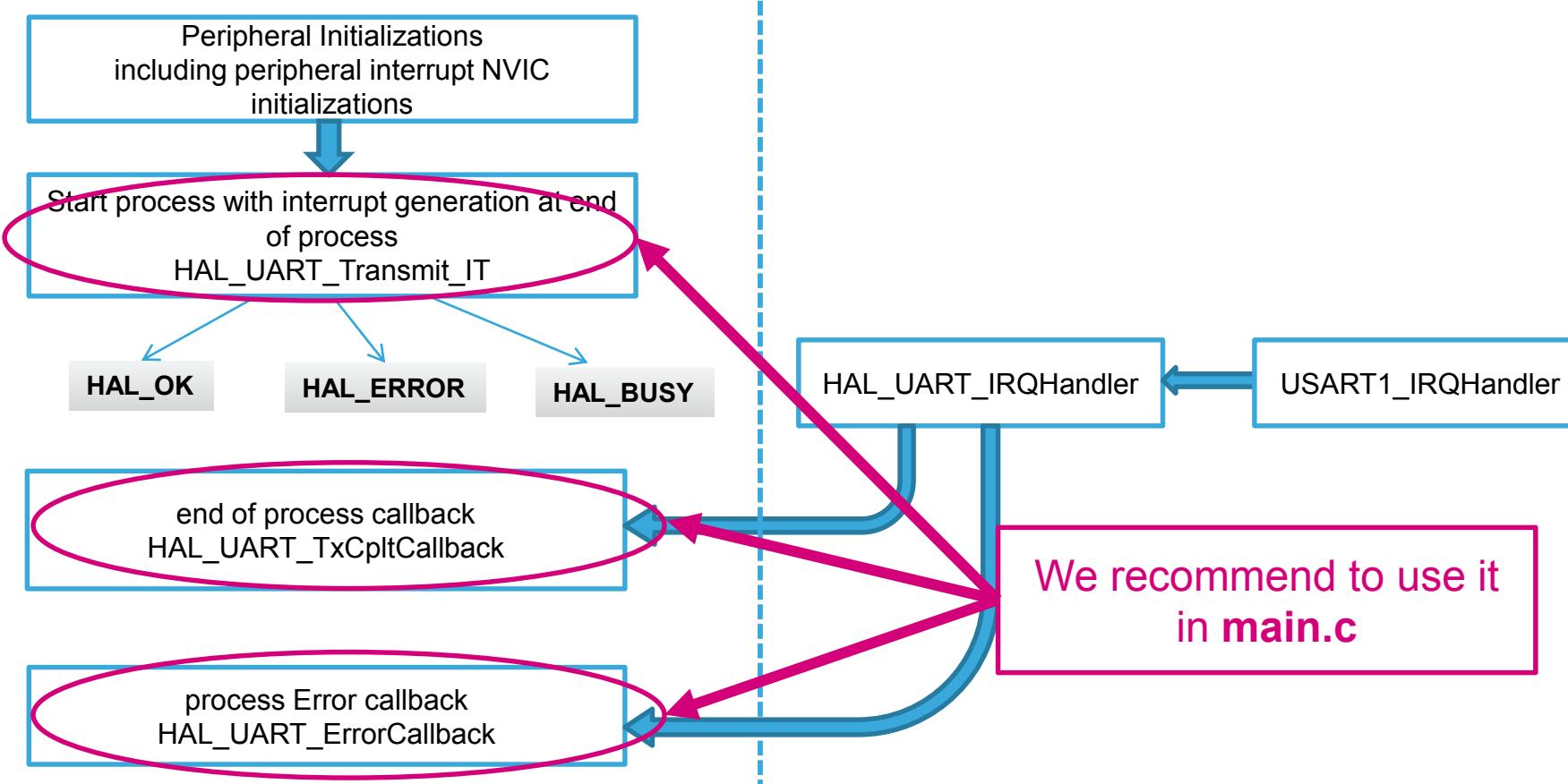
## HAL Library UART with IT receive flow



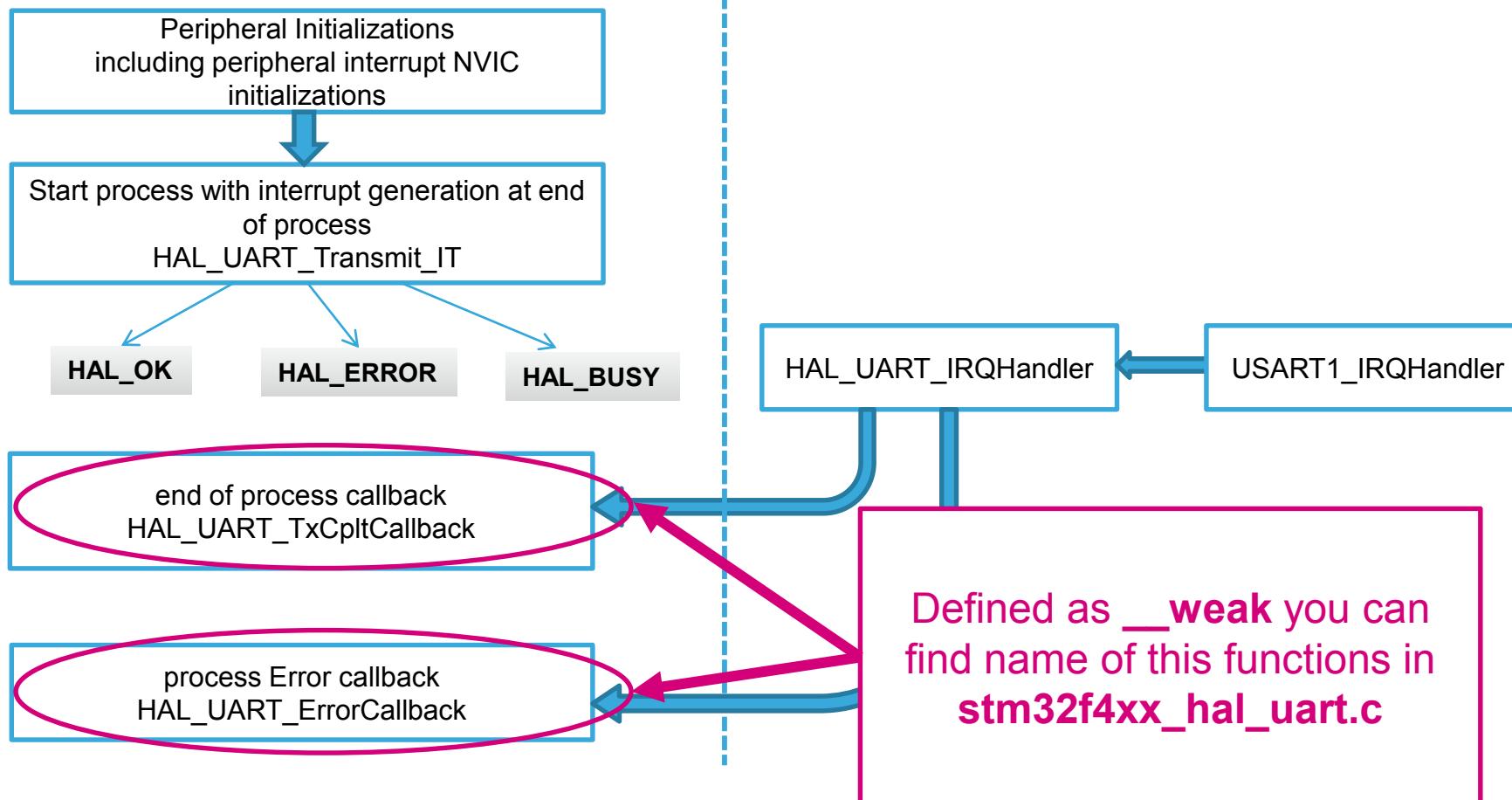
## HAL Library UART with IT receive flow



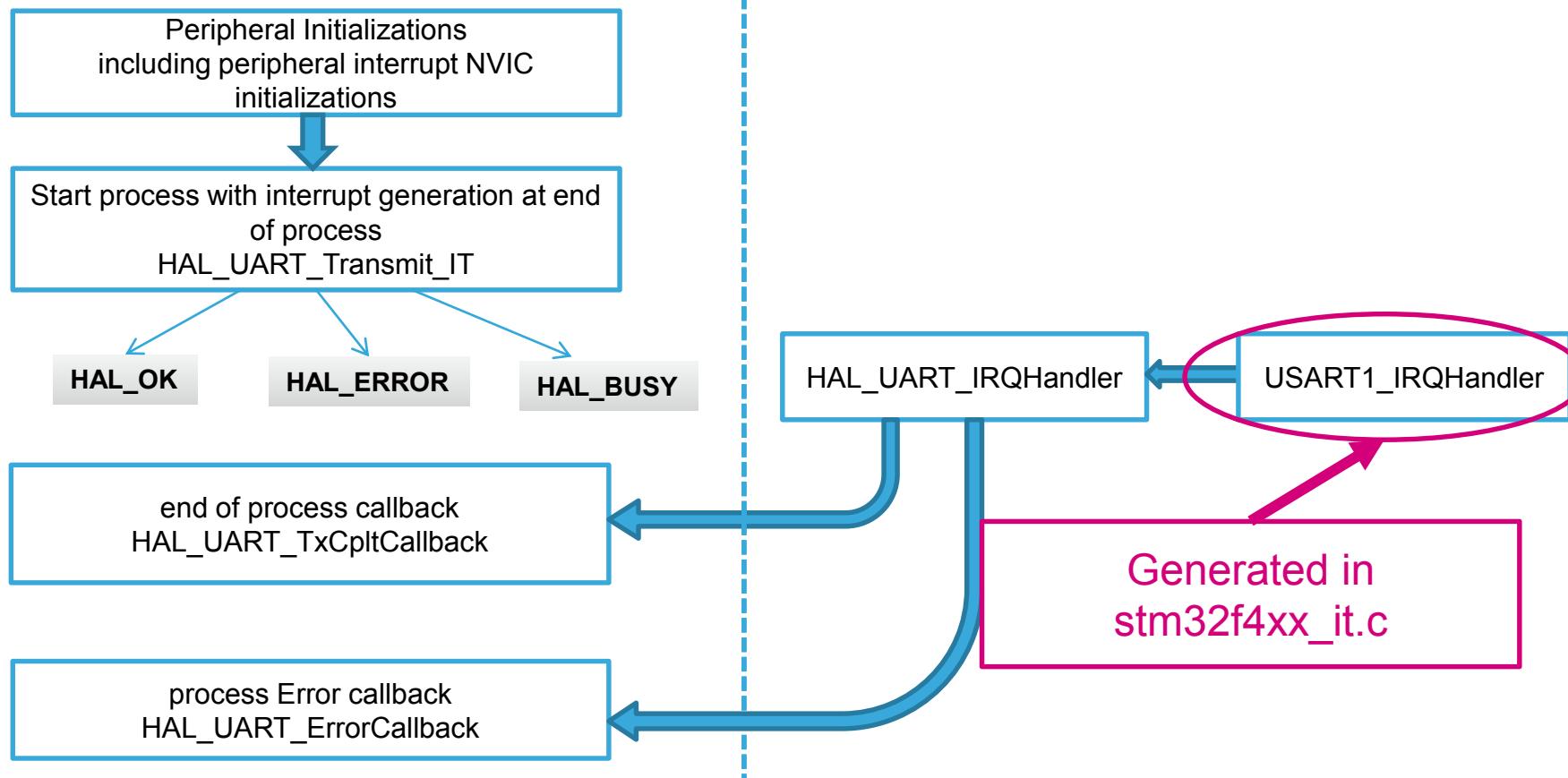
## HAL Library UART with IT receive flow



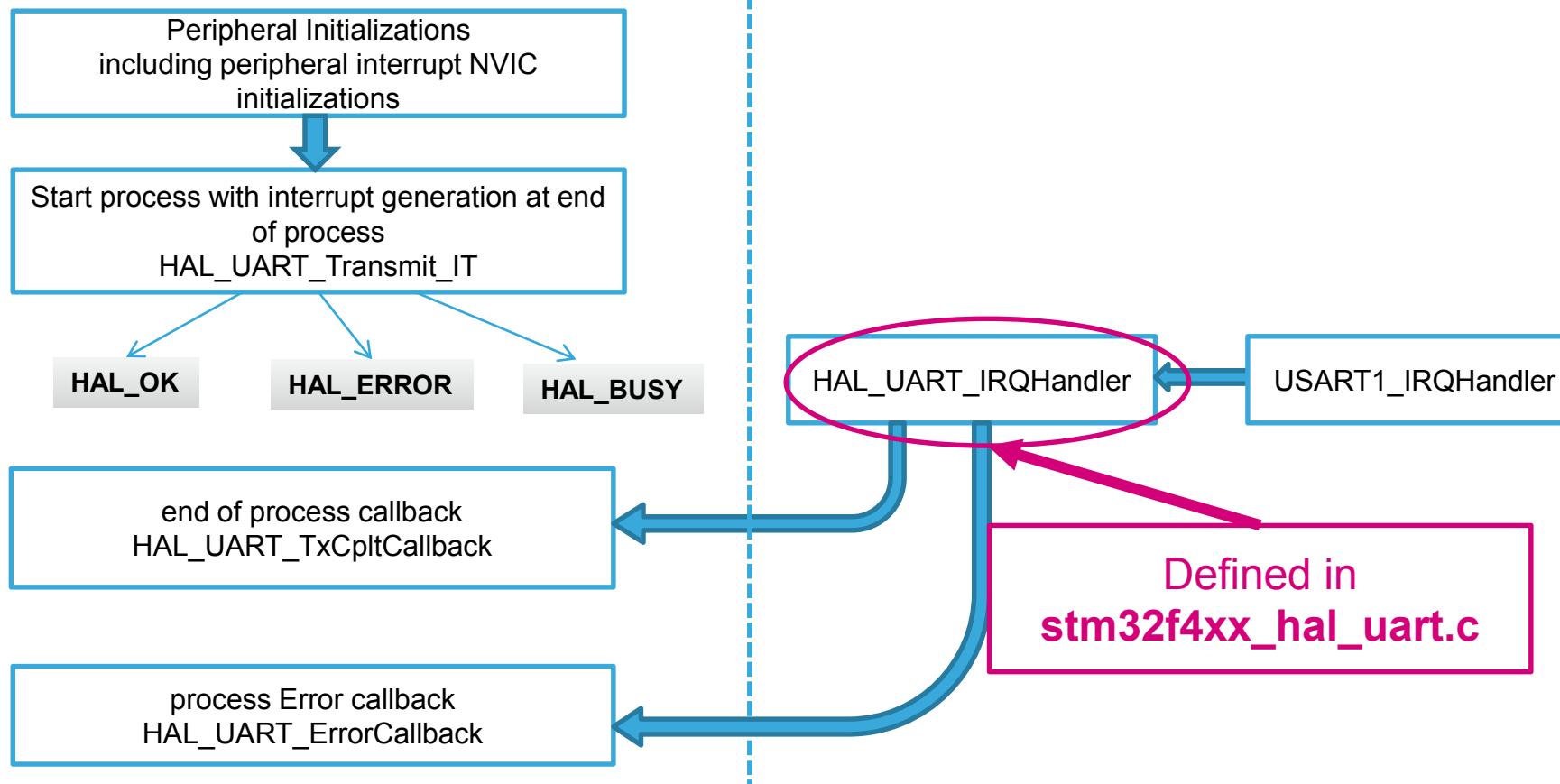
## HAL Library UART with IT receive flow



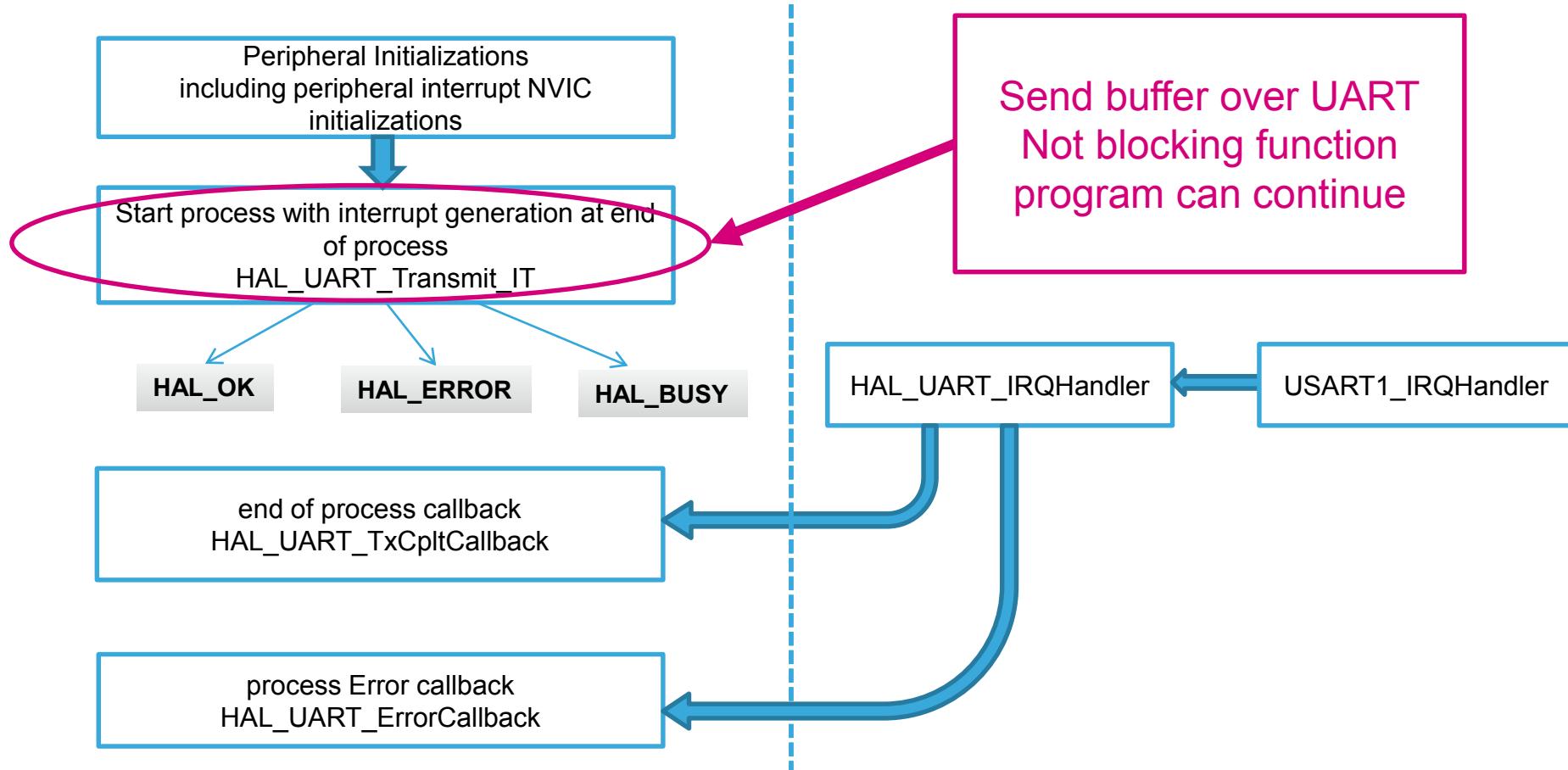
## HAL Library UART with IT receive flow



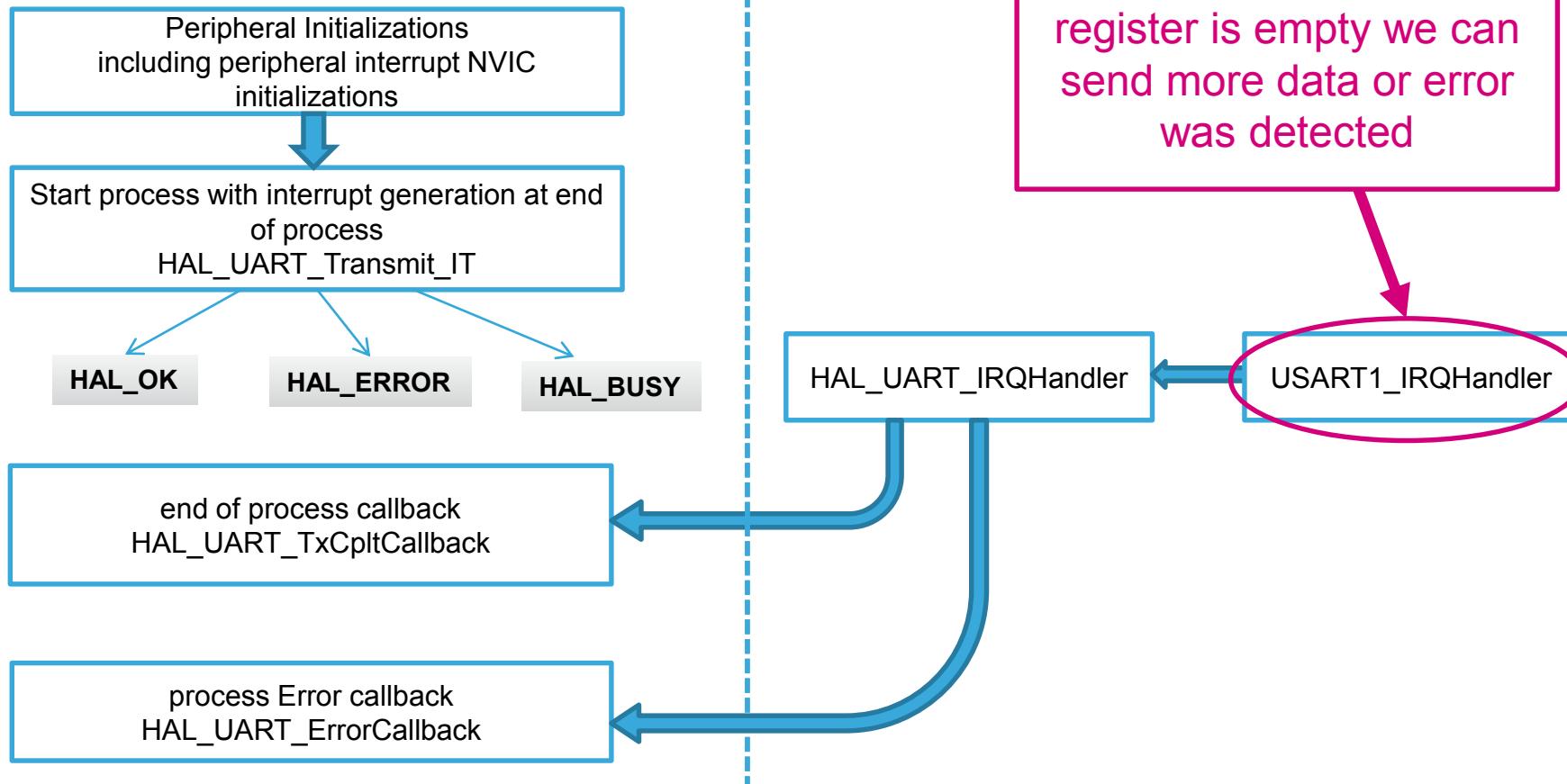
## HAL Library UART with IT receive flow



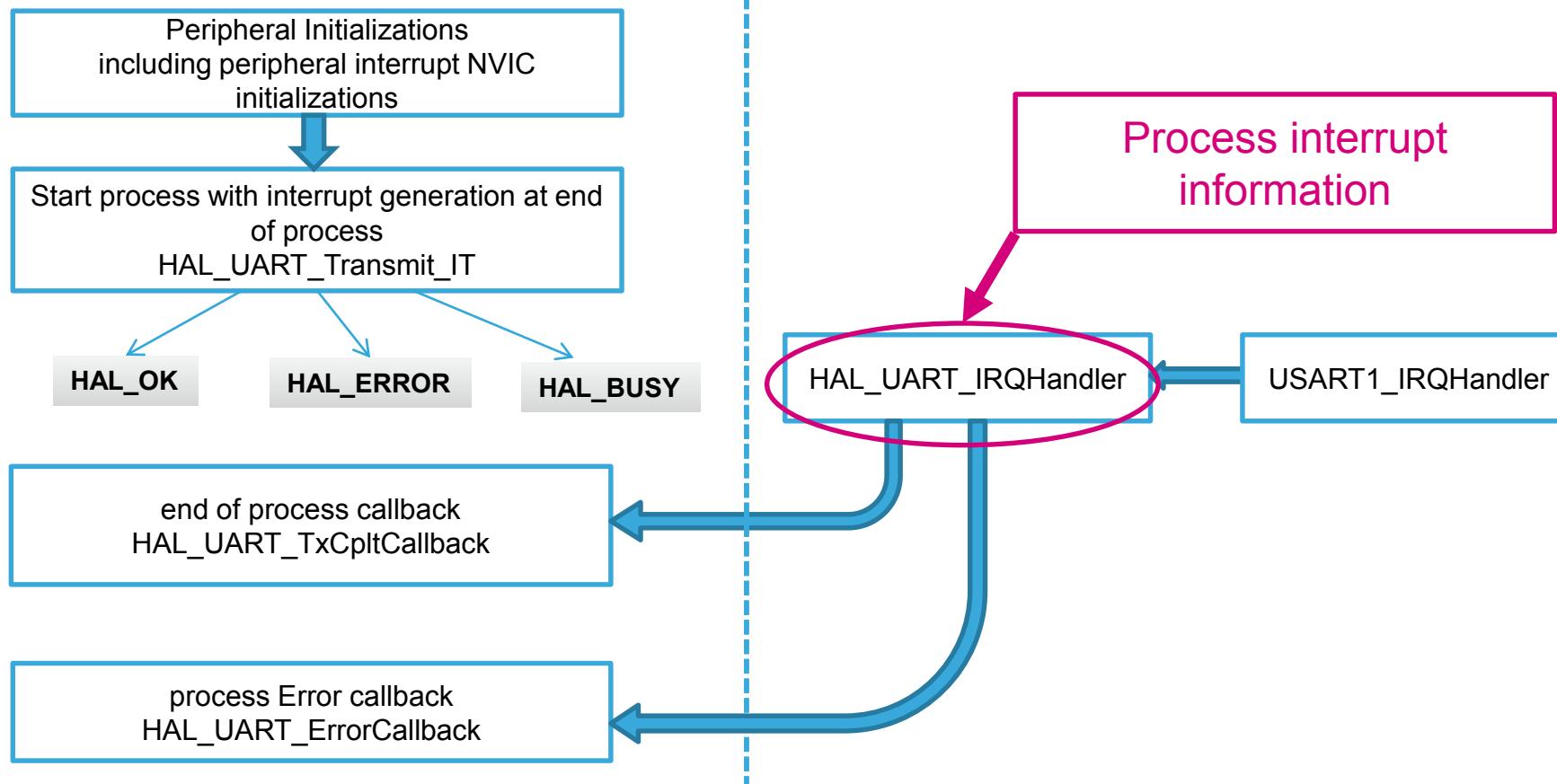
## HAL Library UART with IT receive flow



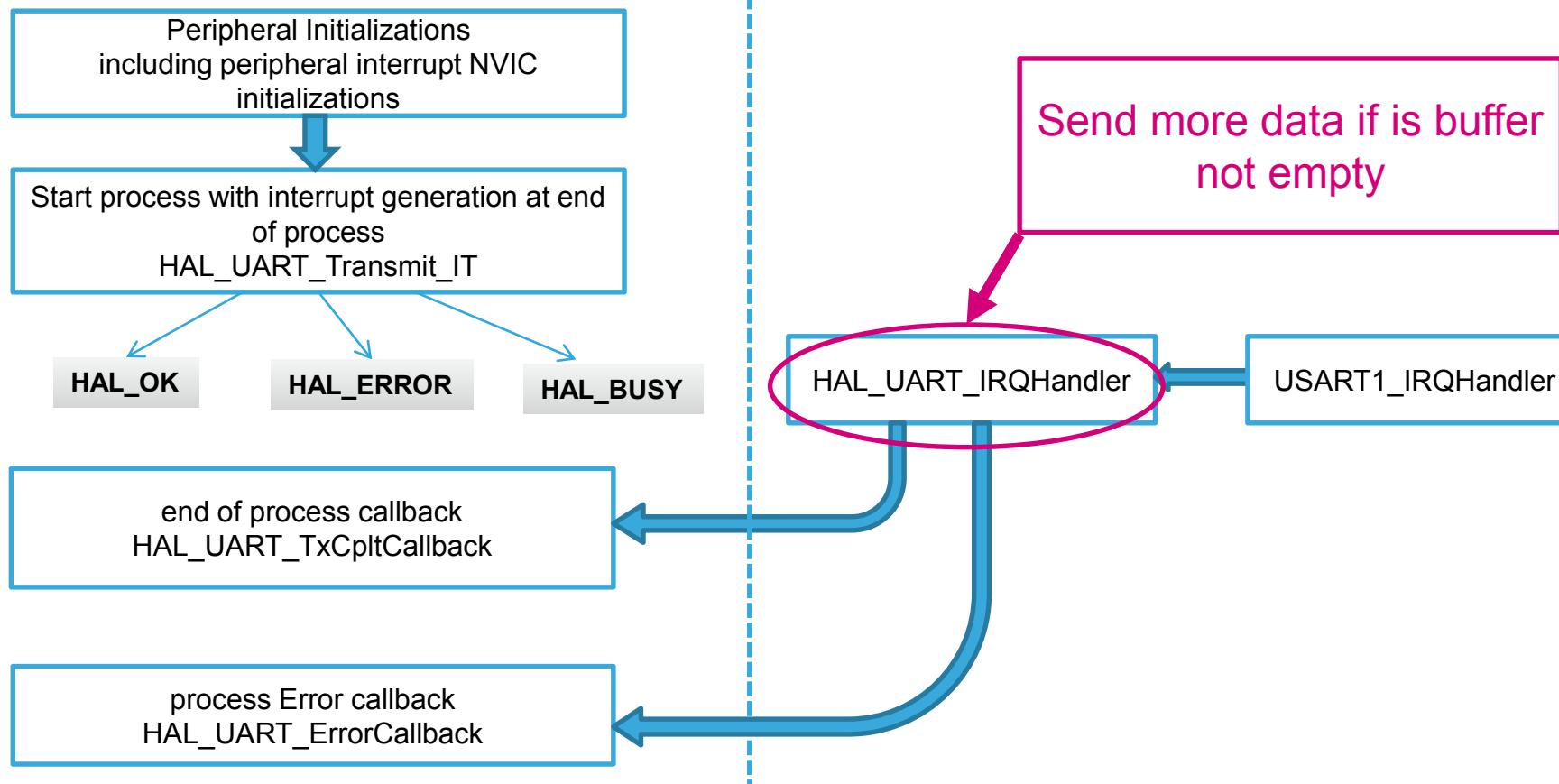
## HAL Library UART with IT receive flow



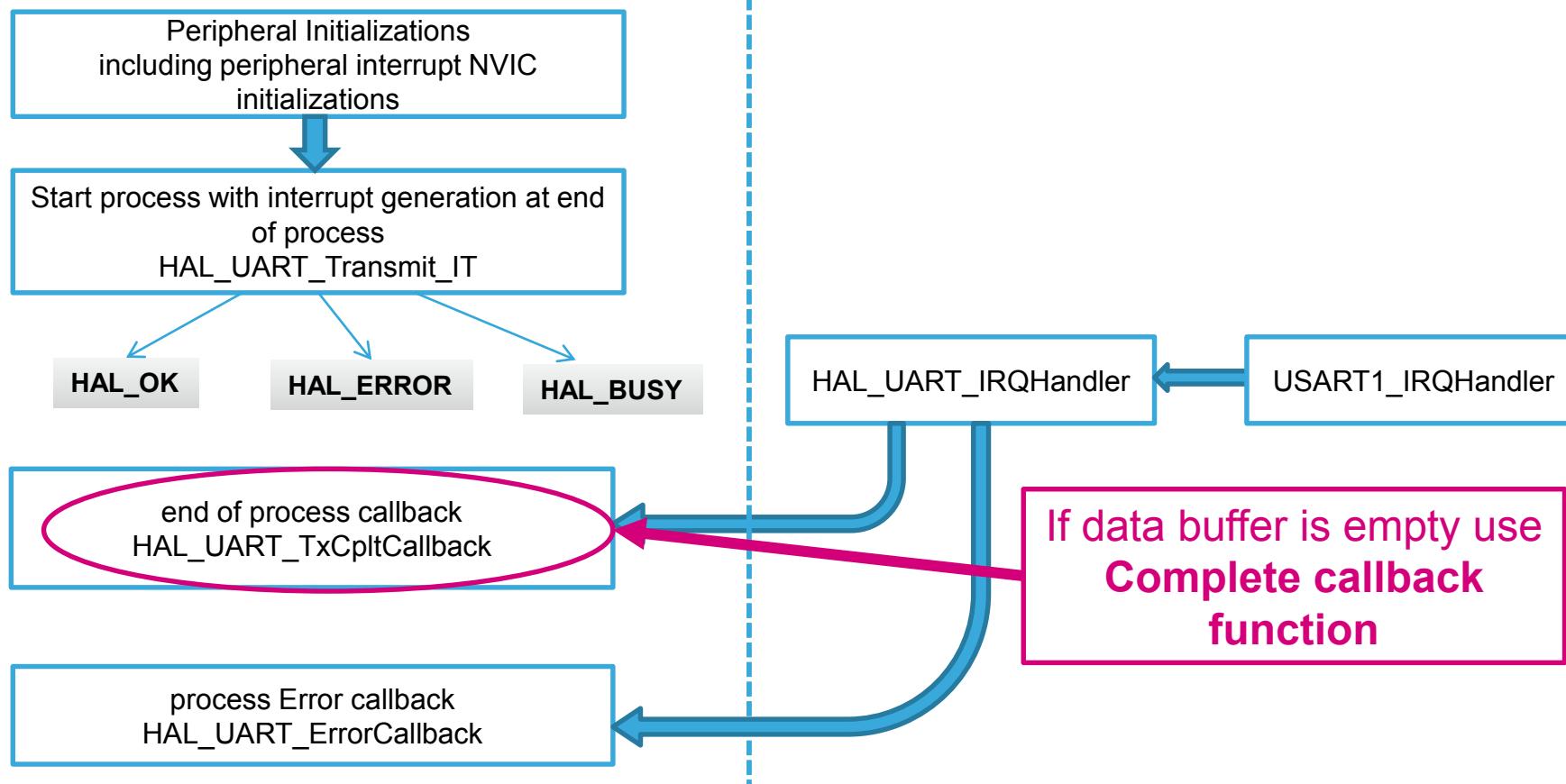
## HAL Library UART with IT receive flow



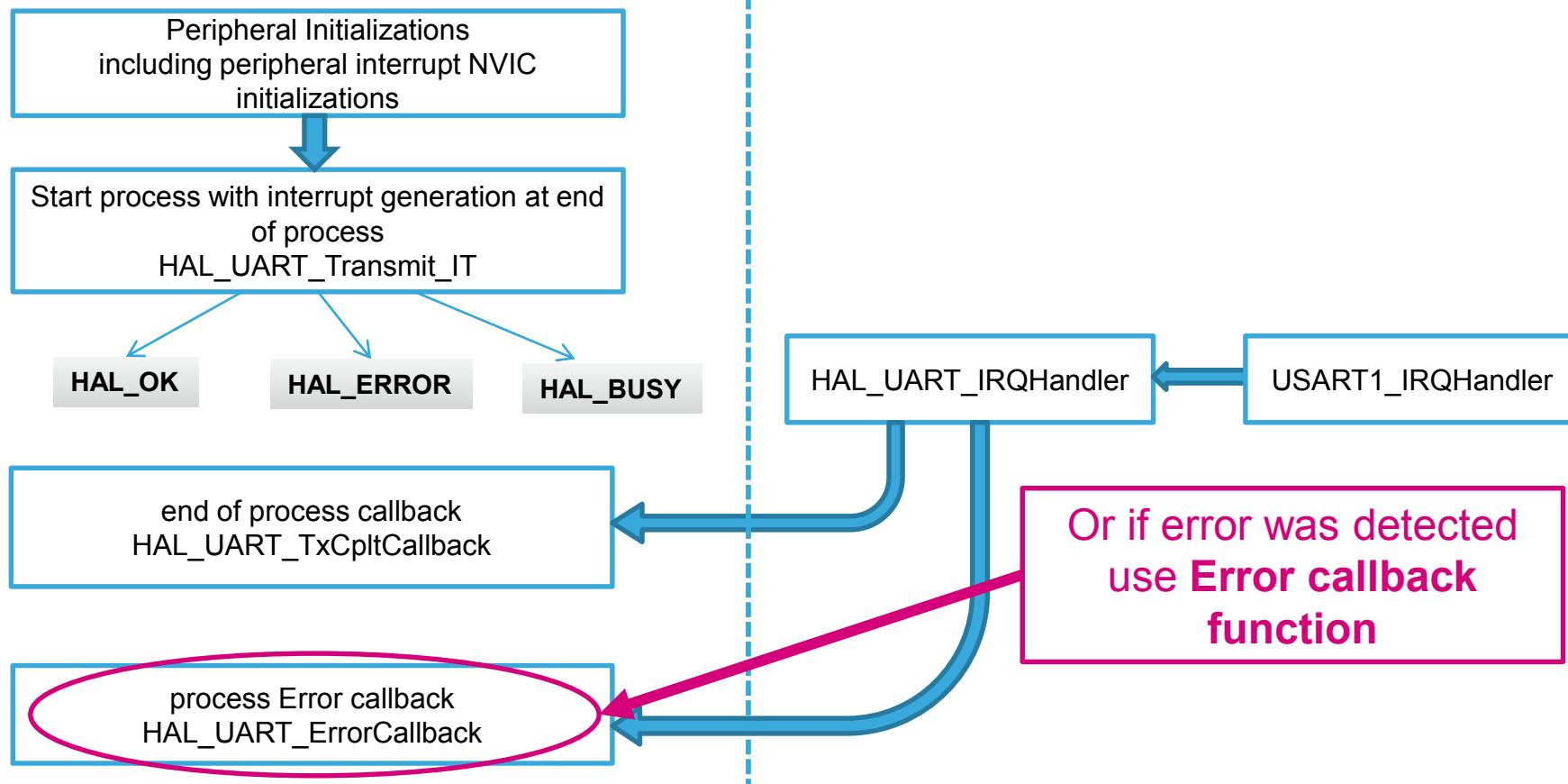
## HAL Library UART with IT receive flow



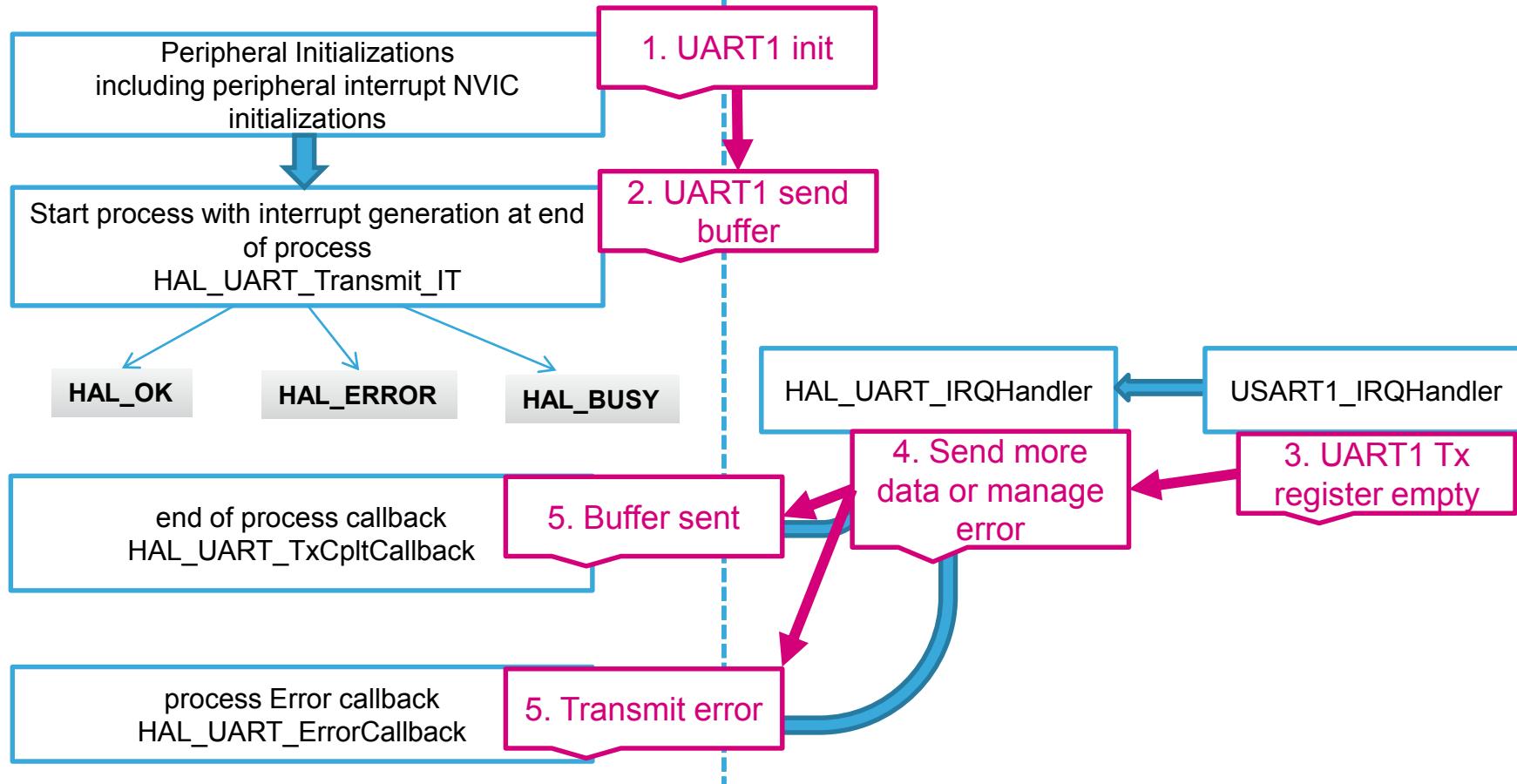
## HAL Library UART with IT receive flow



## HAL Library UART with IT receive flow



## HAL Library UART with IT receive flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For transmit use function
  - `HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);`
- For receive use function
  - `HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);`

- Buffer definition

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

- Sending and receiving methods

```
/* USER CODE BEGIN 2 */  
HAL_UART_Receive_IT(&huart1,rx_buff,10);  
HAL_UART_Transmit_IT(&huart1,tx_buff,10);  
/* USER CODE END 2 */
```

- Complete callback check
  - We can put breakpoints on NOPs to watch if we send or receive complete buffer

```
/* USER CODE BEGIN 4 */  
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  
{  
    __NOP(); //test if we reach this position  
}  
  
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)  
{  
    __NOP(); //test if we reach this position  
}  
/* USER CODE END 4 */
```



# UART DMA lab 11

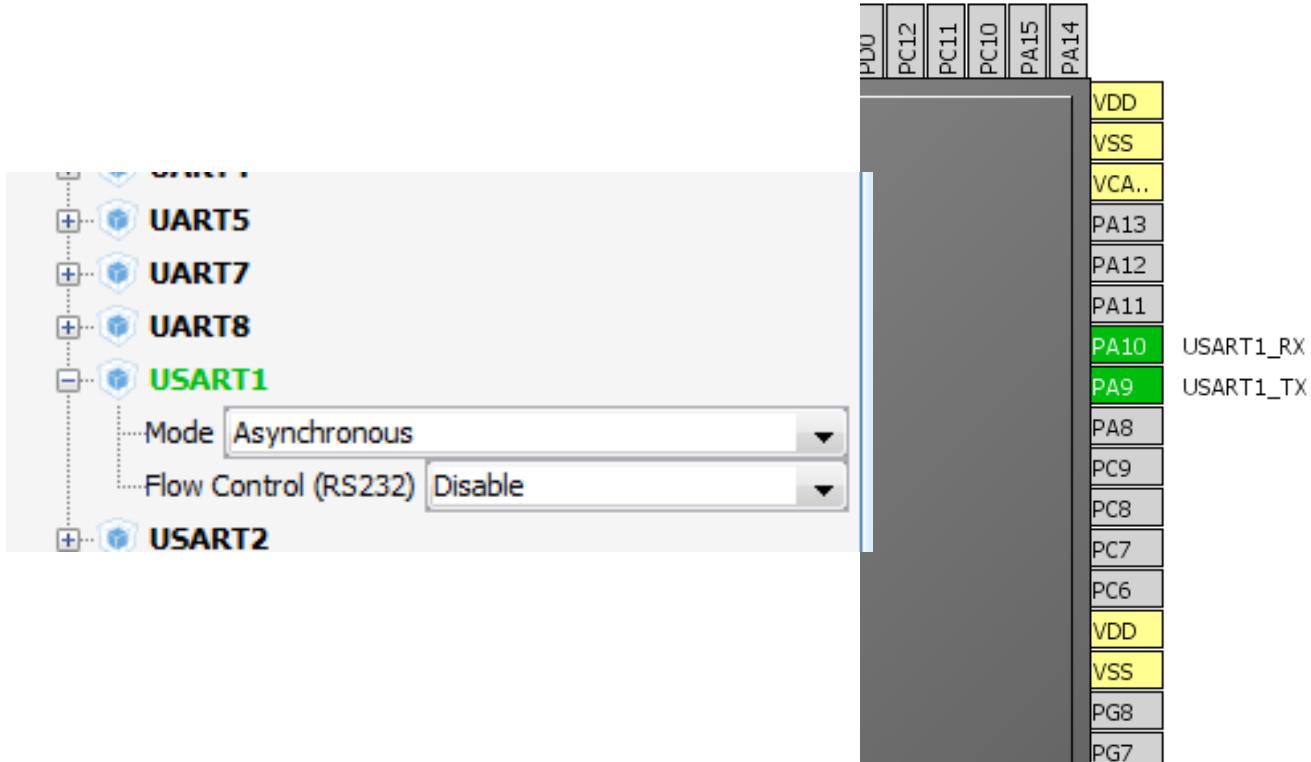
- Objective

- Learn how to setup UART with DMA in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Create simple loopback example with DMA

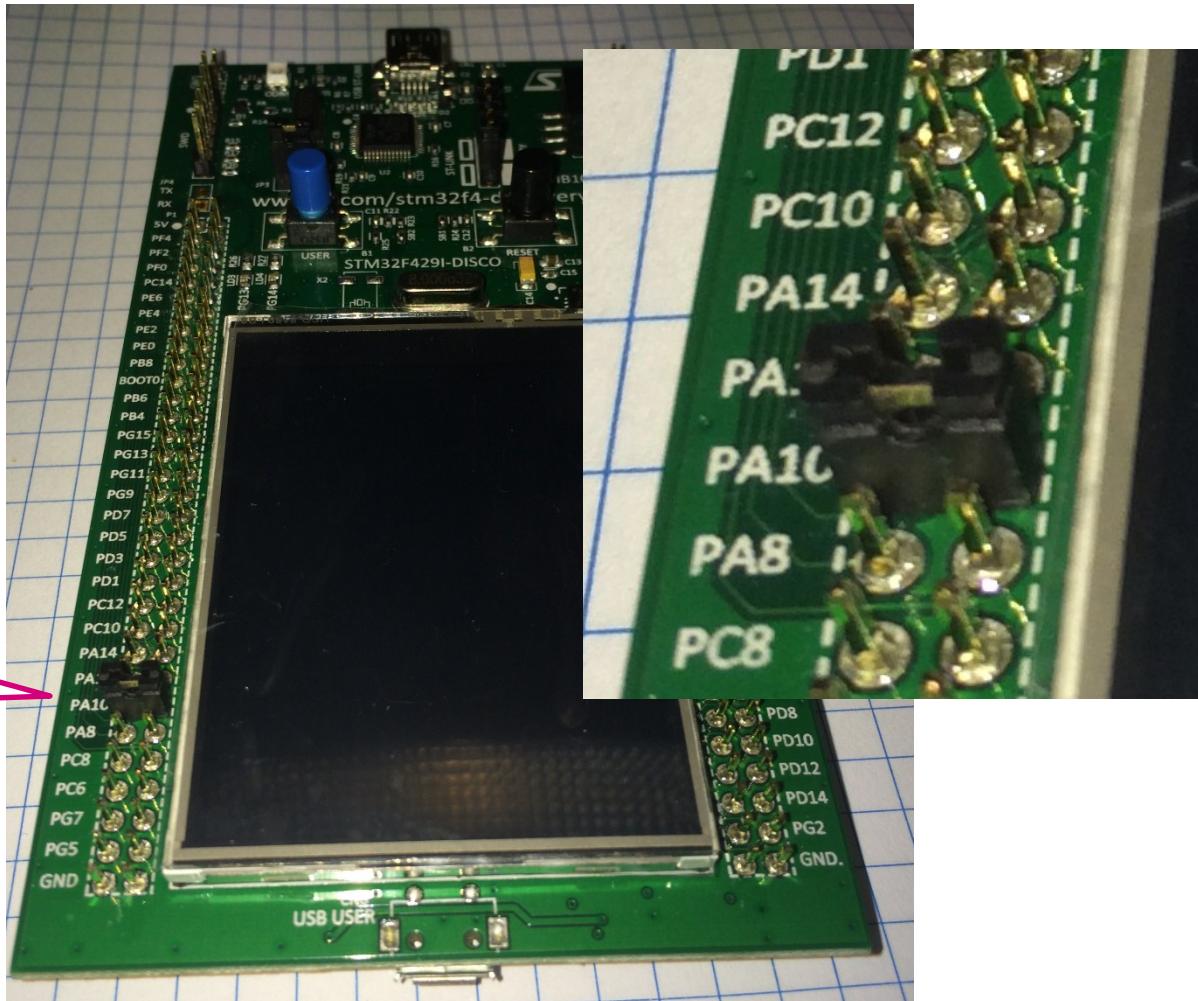
- Goal

- Configure UART in CubeMX and Generate Code
- Learn how to send and receive data over UART with DMA
- Verify the correct functionality

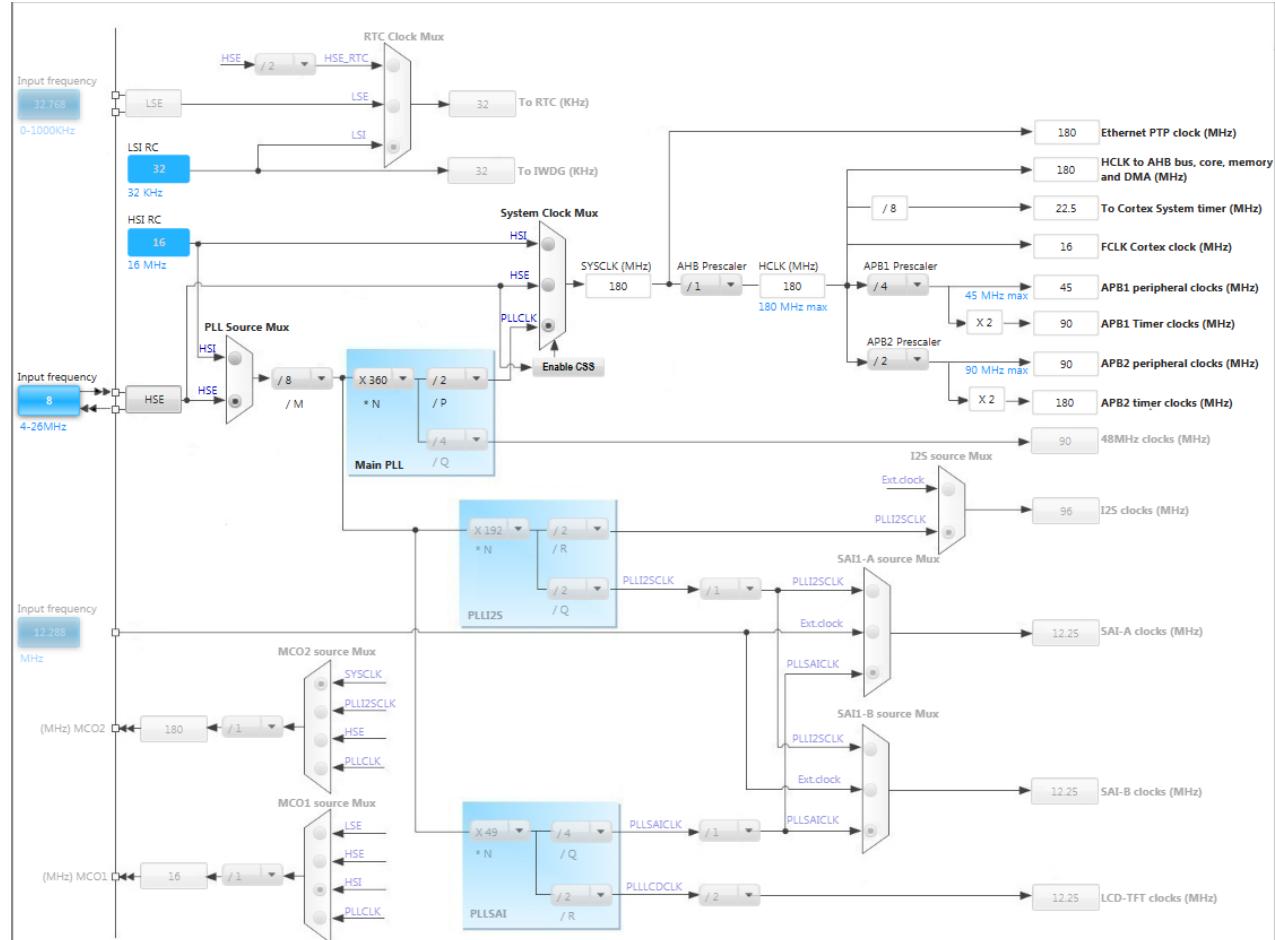
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Pin selection
  - It will be same as previous lab we use again PA9 and PA10



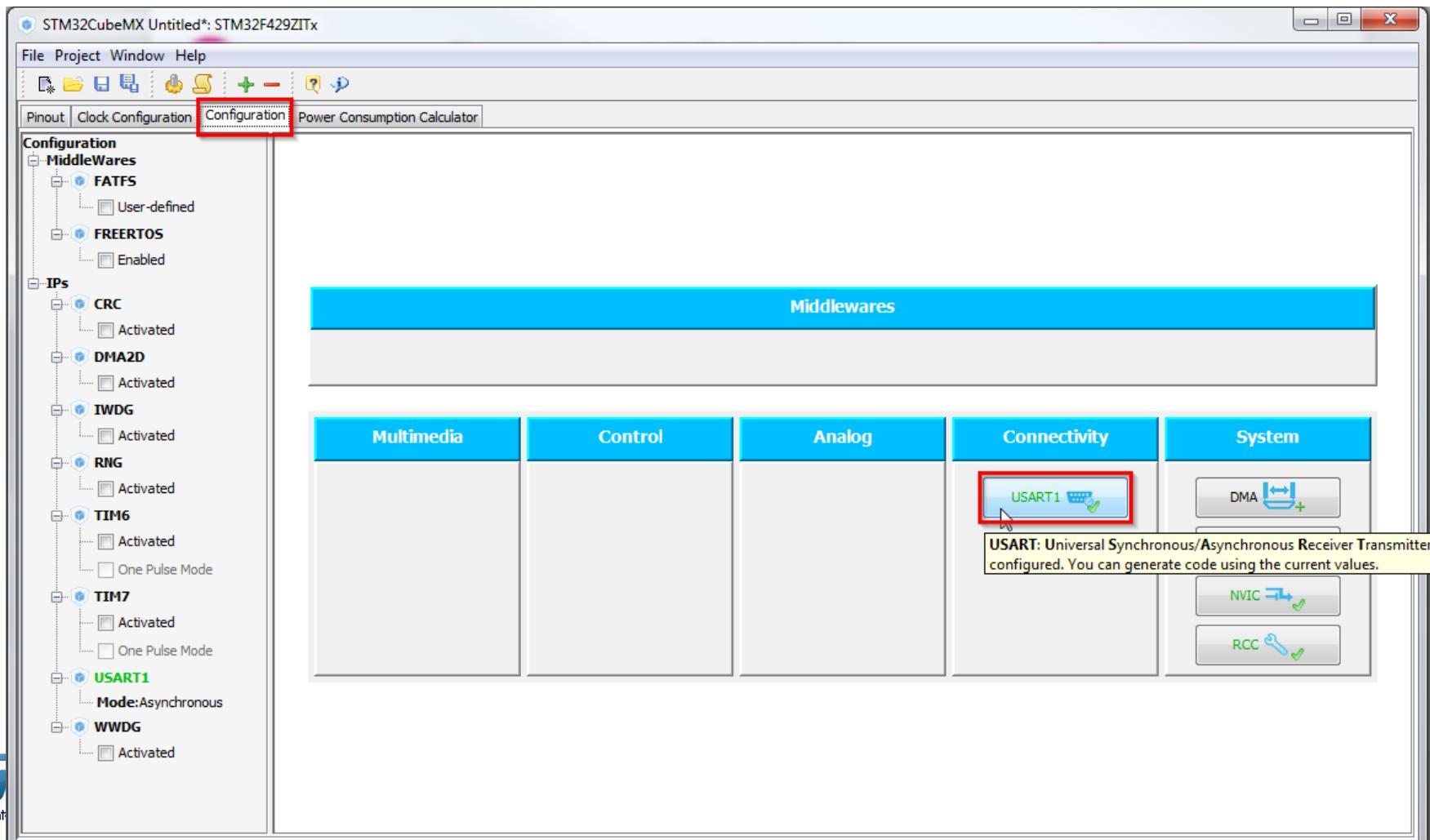
- Hardware preparation
  - We connect selected pins together by jumper, this help us to create loopback on UART



- In order to run on maximum frequency, setup clock system
- Details in lab 0

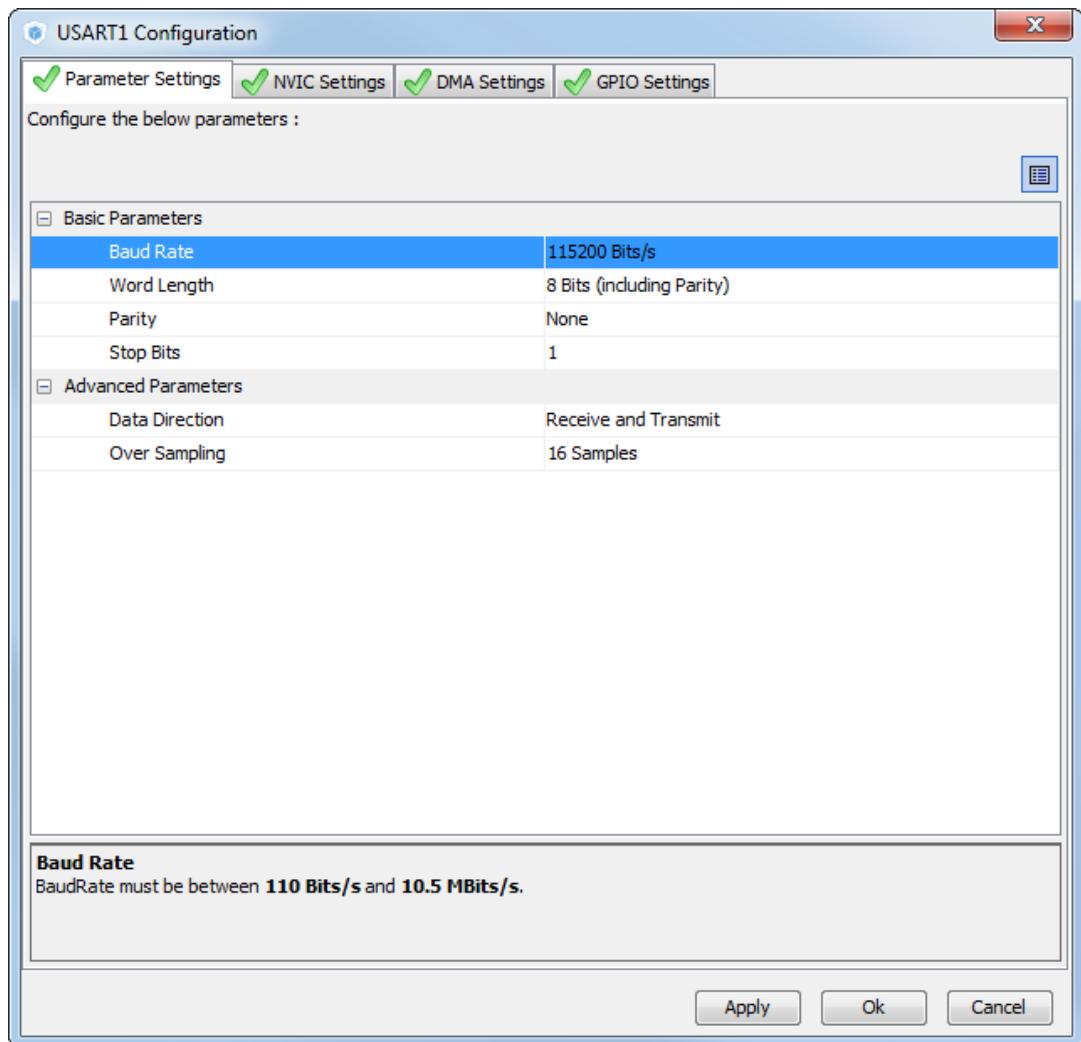


- CubeMX UART configuration
  - Tab>Configuration>Connectivity>USART1



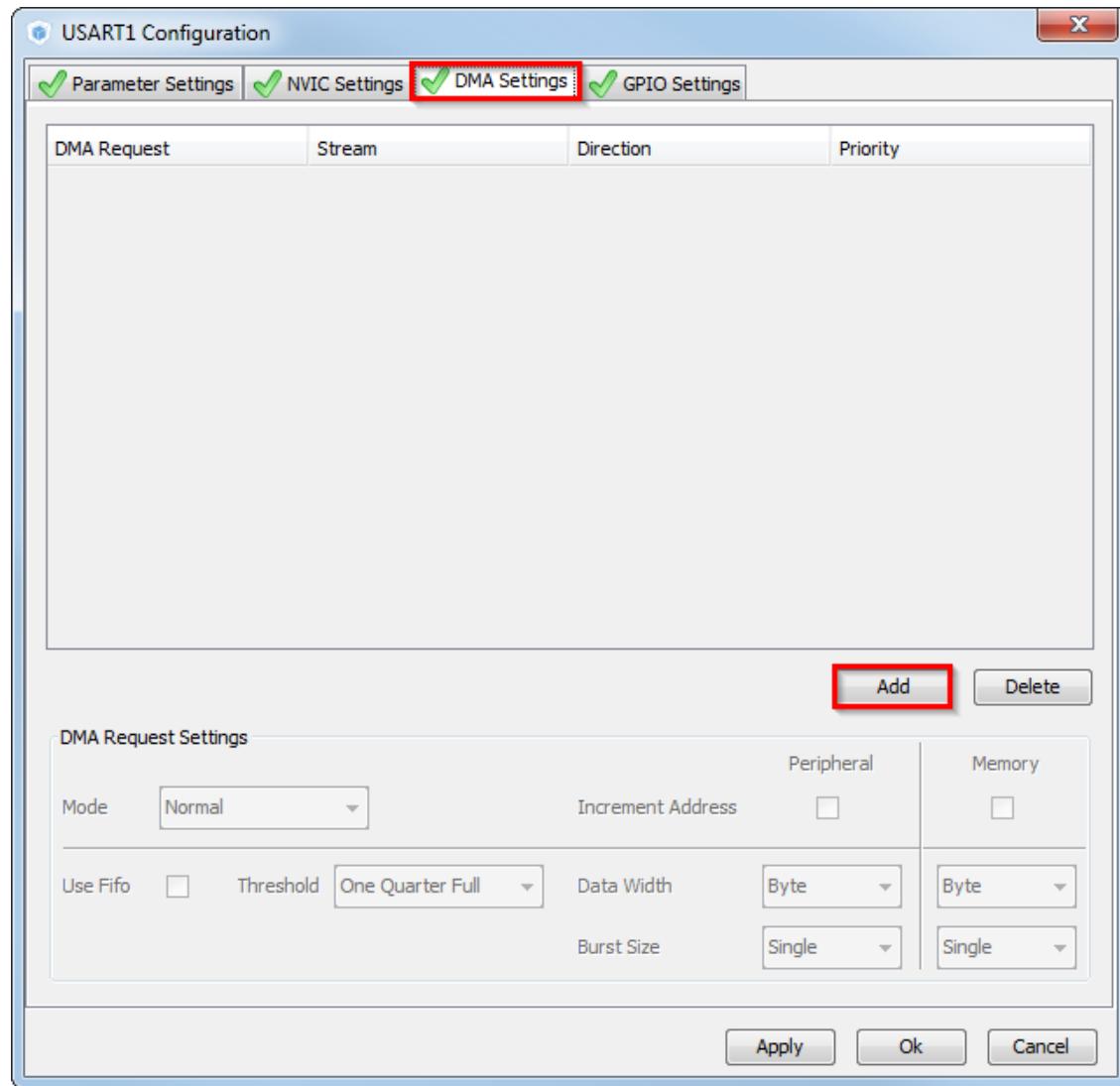
- CubeMX USART configuration check:

- BaudRate
- Word length
- Parity
- Stop bits
- Data direction
- Oversampling



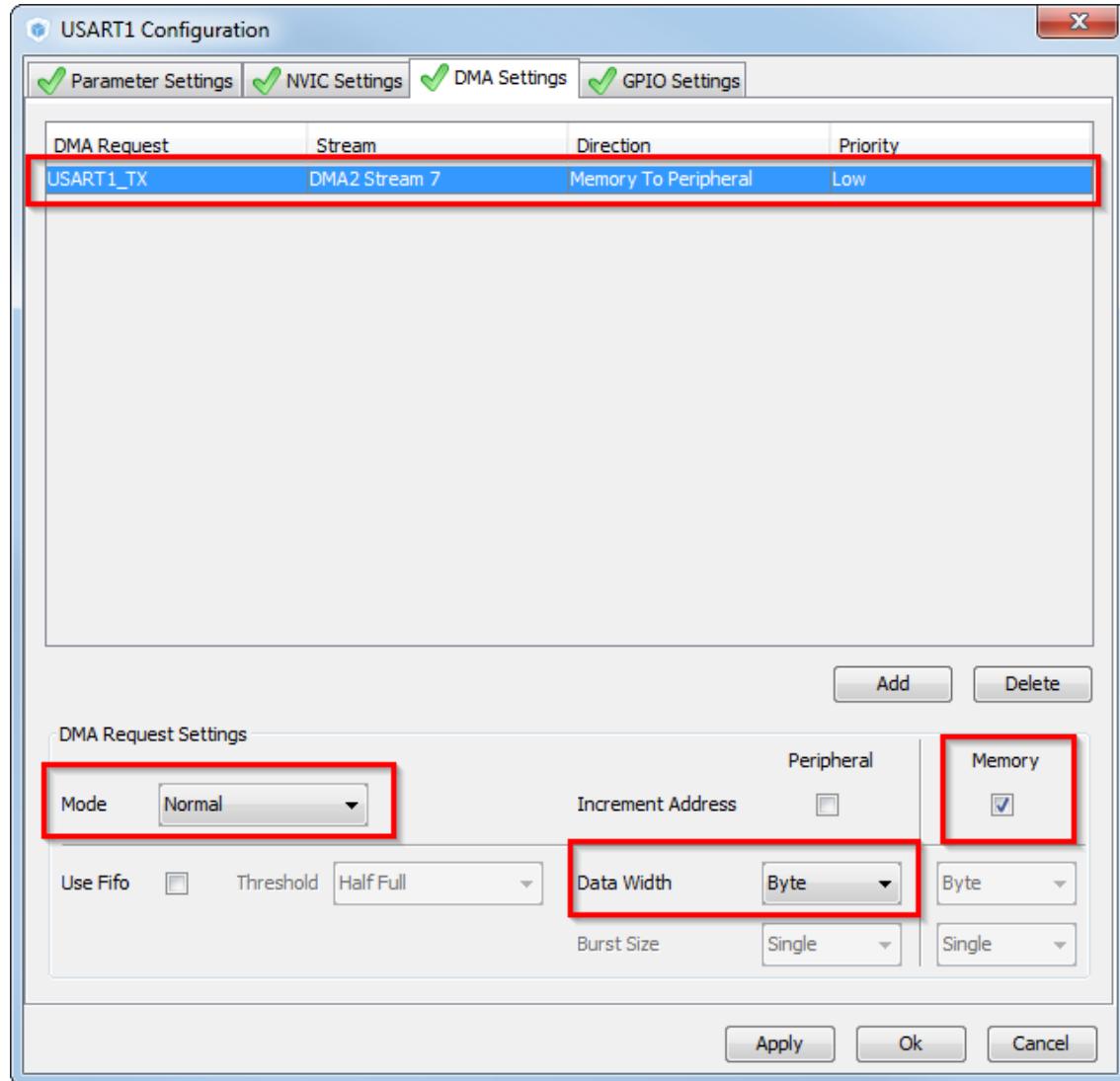
- CubeMX USART configuration DMA settings

- TAB>DMA Settings
- Button ADD



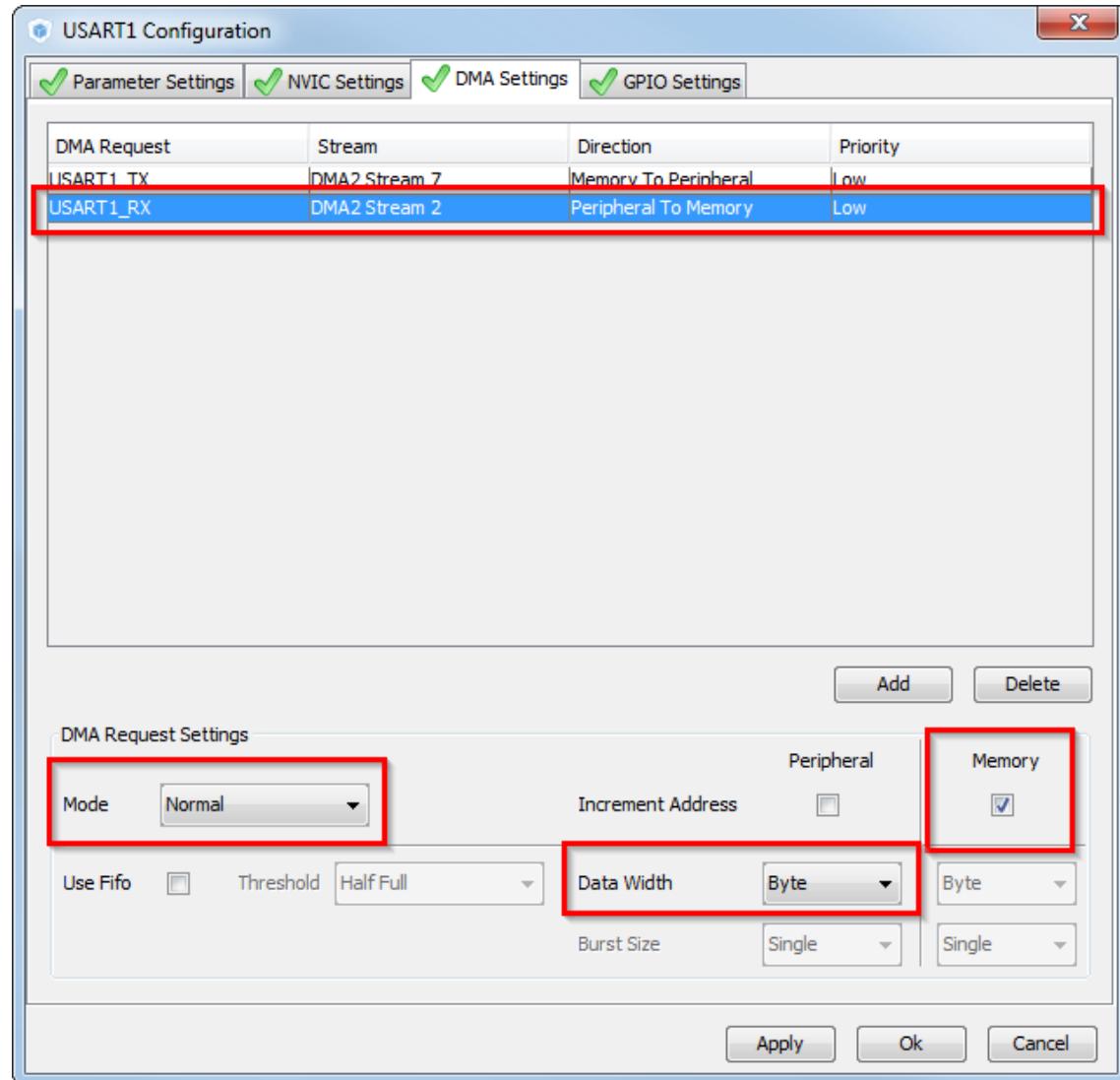
- CubeMX USART configuration DMA Tx settings

- Set USART1\_TX request
- Memory to peripheral direction
- Normal mode
- Byte data width
- Increment memory address



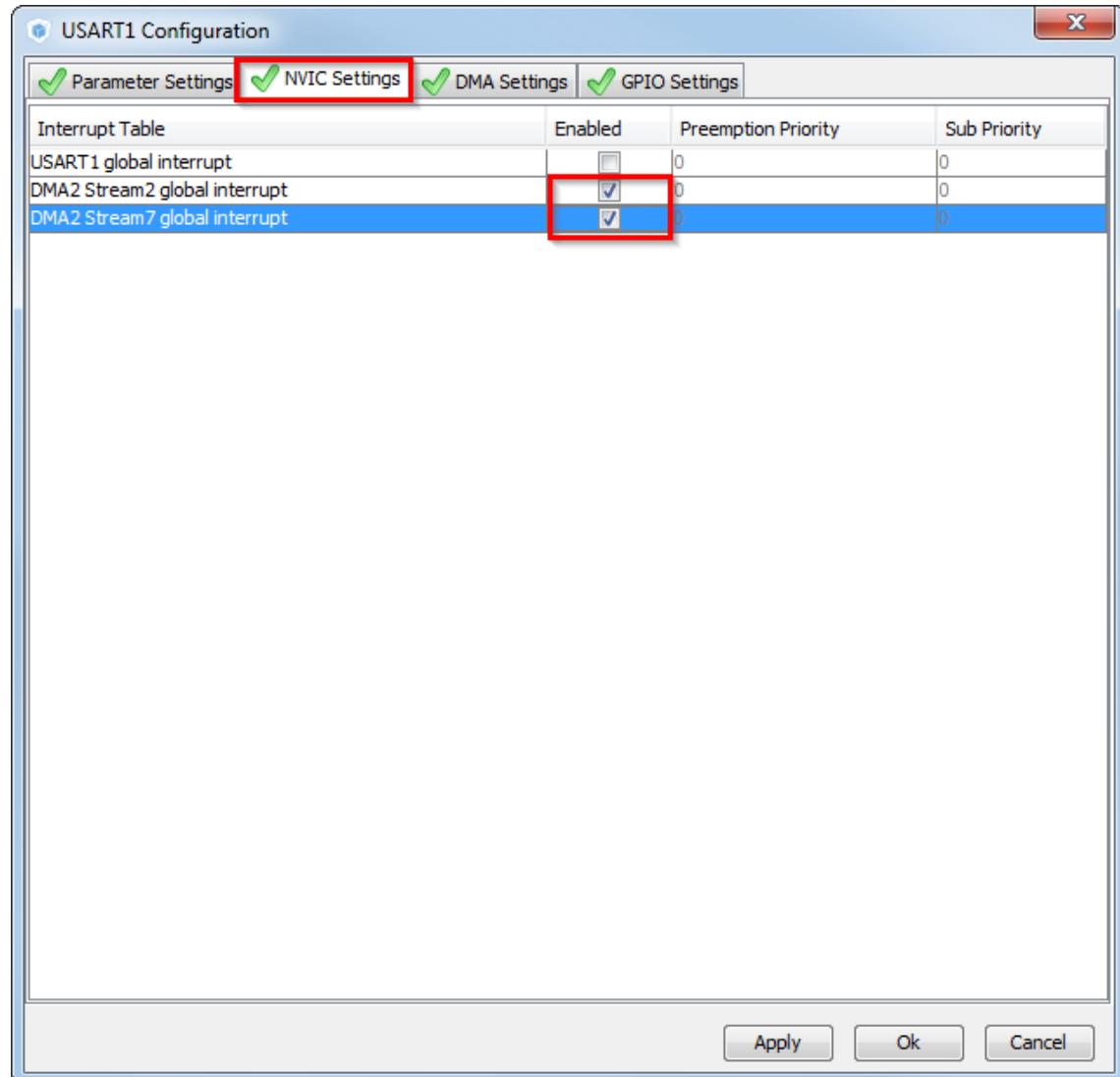
- CubeMX USART configuration DMA Rx settings

- Button ADD
- Set USART1\_RX request
- Peripheral to memory direction
- Normal mode
- Byte data width
- Increment memory address



- CubeMX USART configuration NVIC settings

- TAB>NVIC Settings
- Enable DMA2 interrupts for USART1
- Button OK

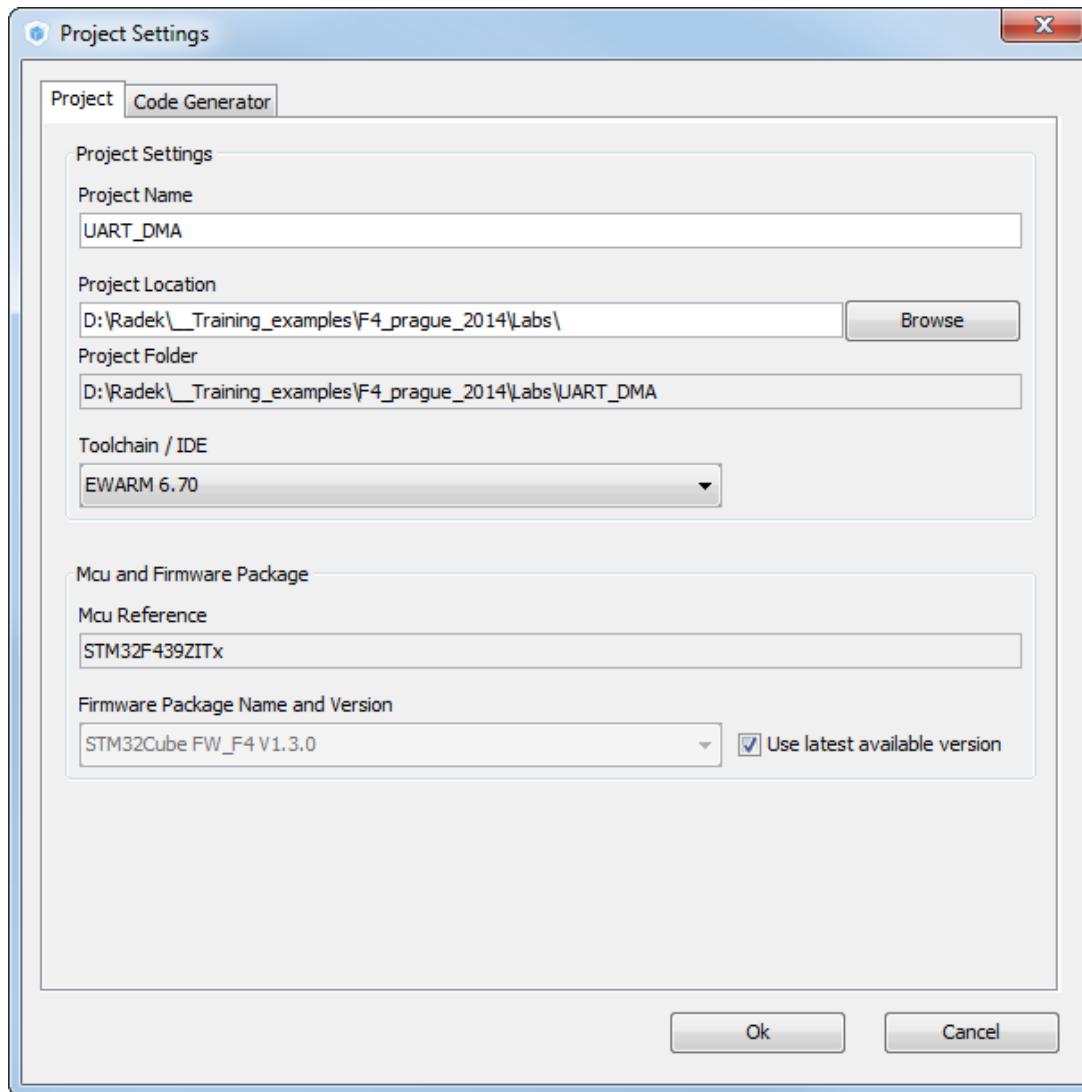


- Now we set the project details for generation

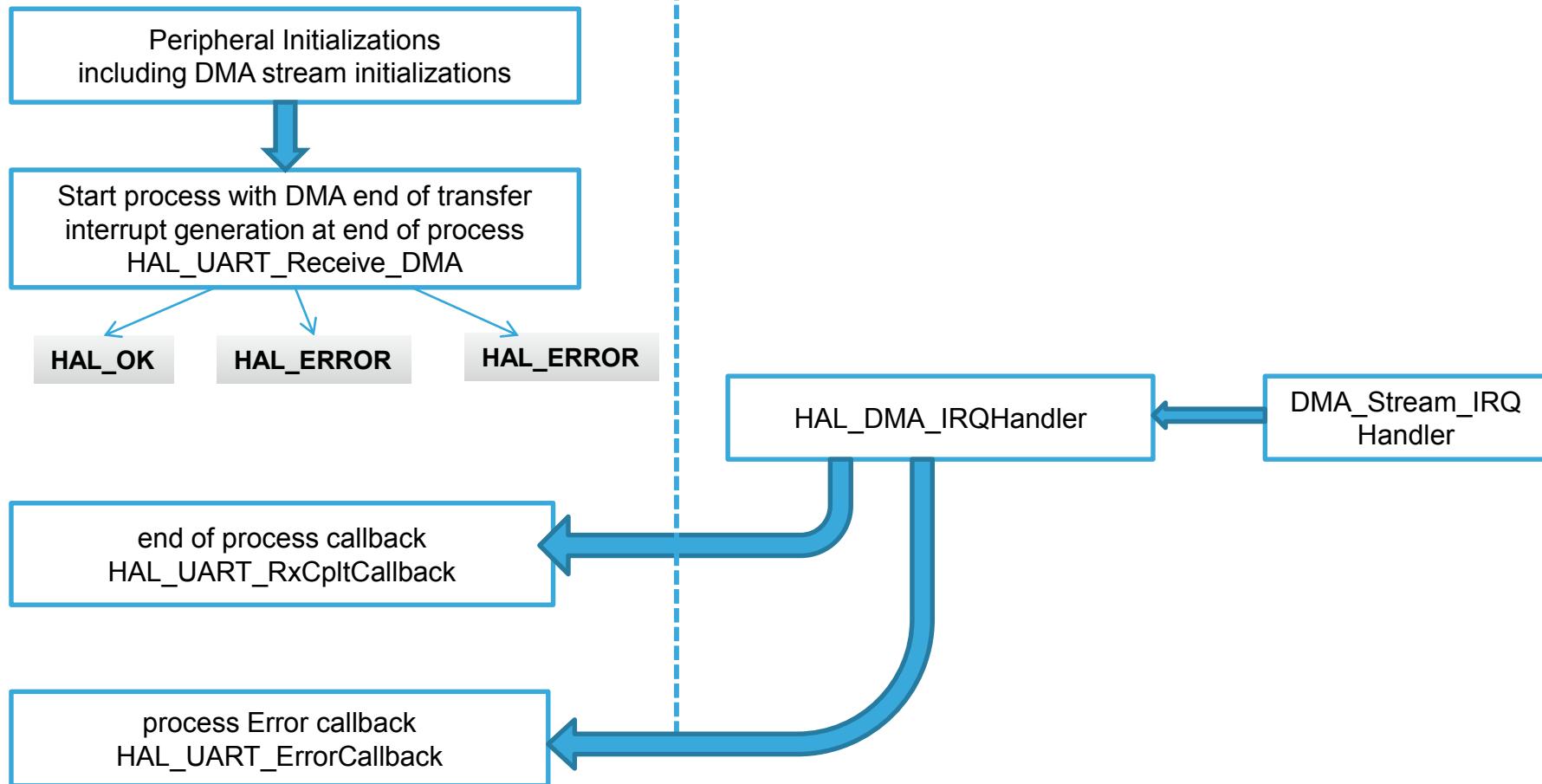
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

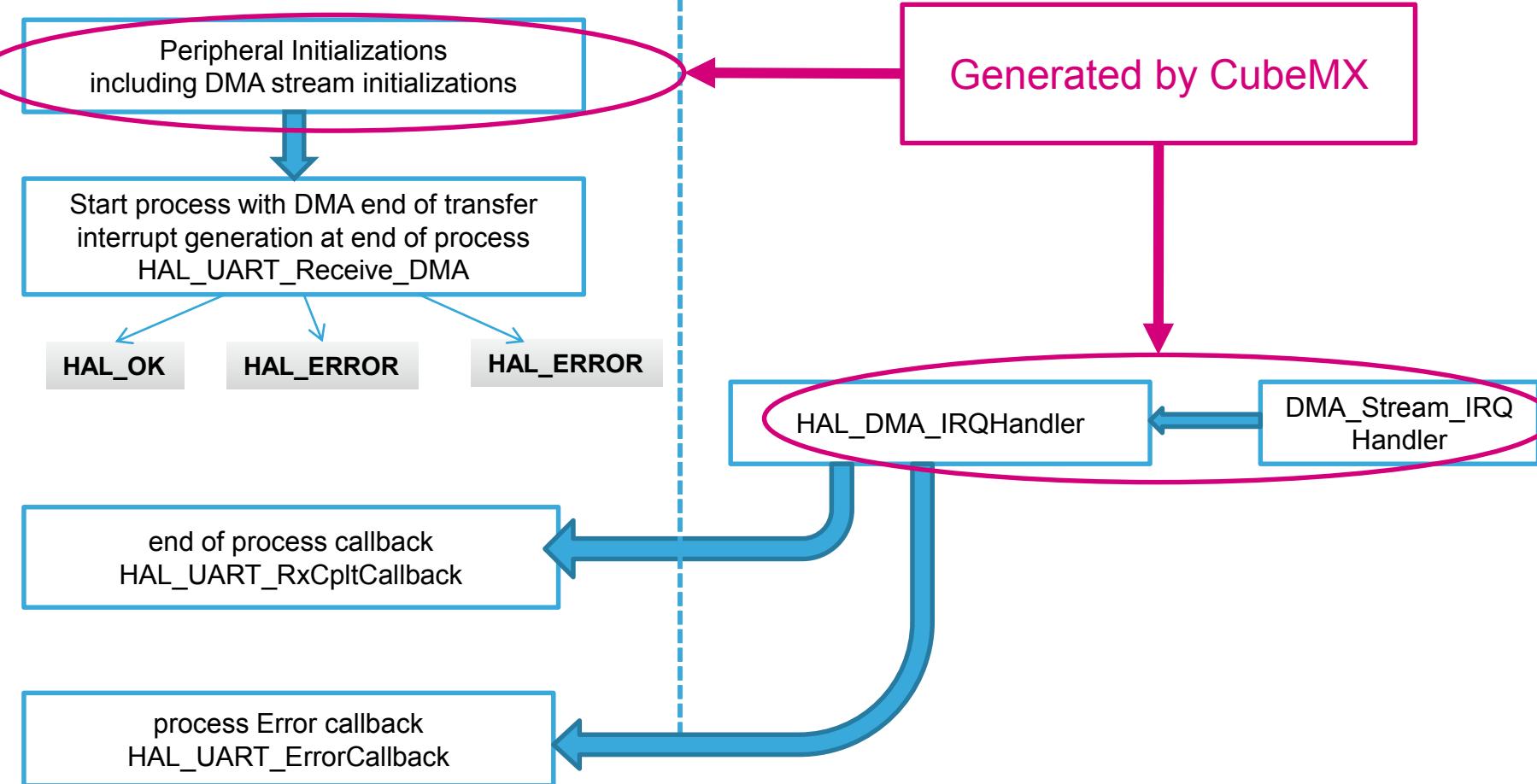
- Menu > Project > Generate Code



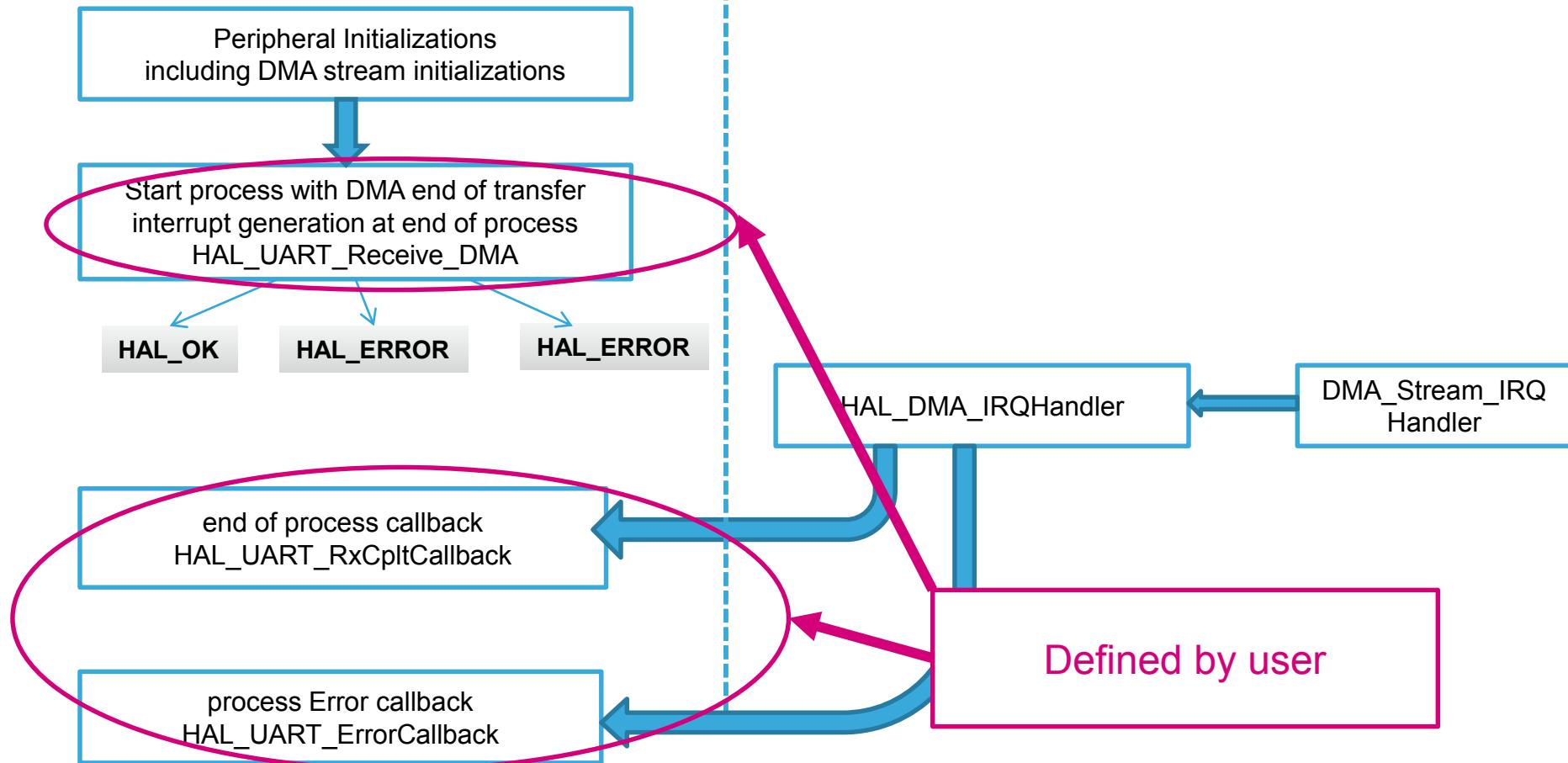
## HAL Library UART with DMA RX flow



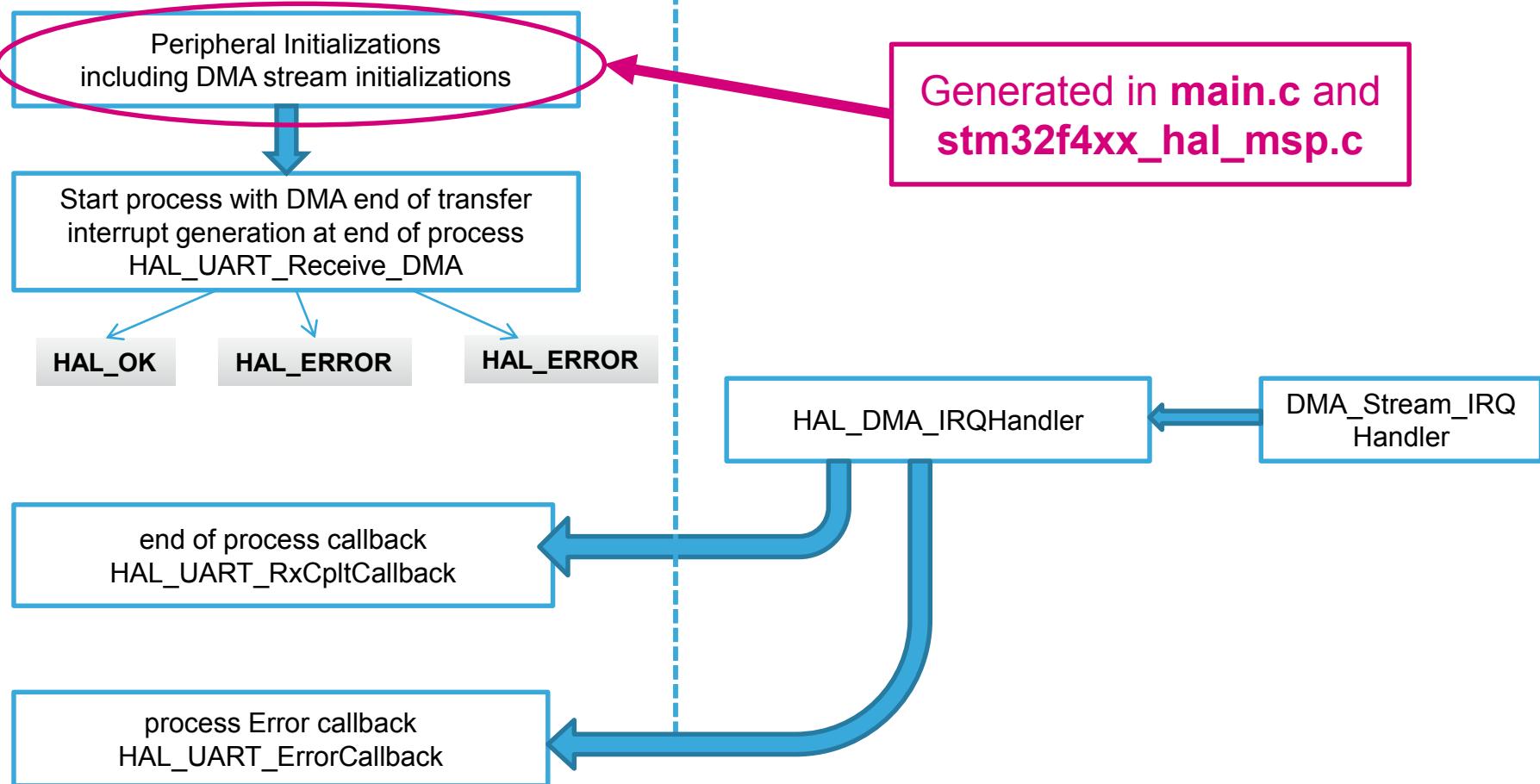
## HAL Library UART with DMA RX flow



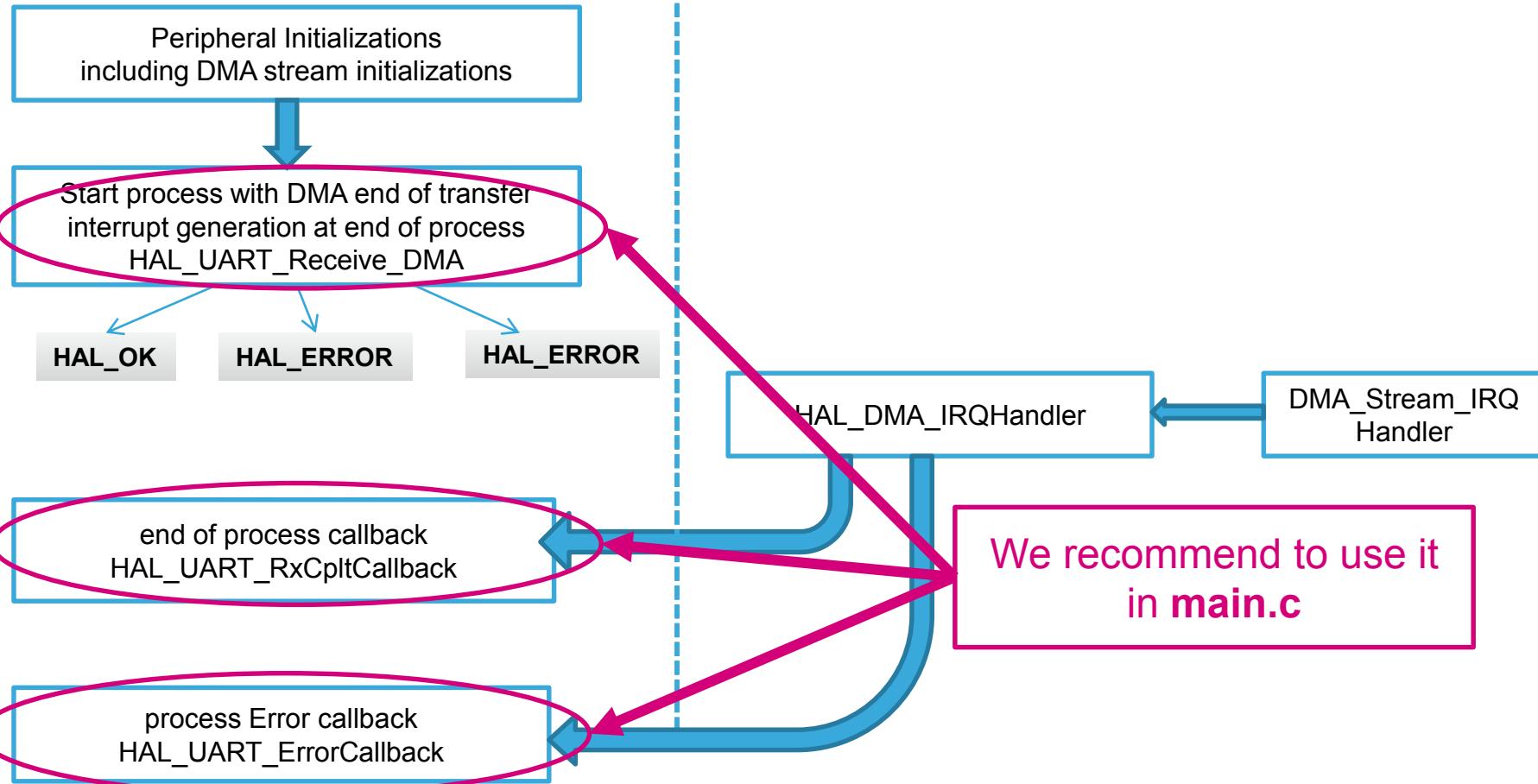
## HAL Library UART with DMA RX flow



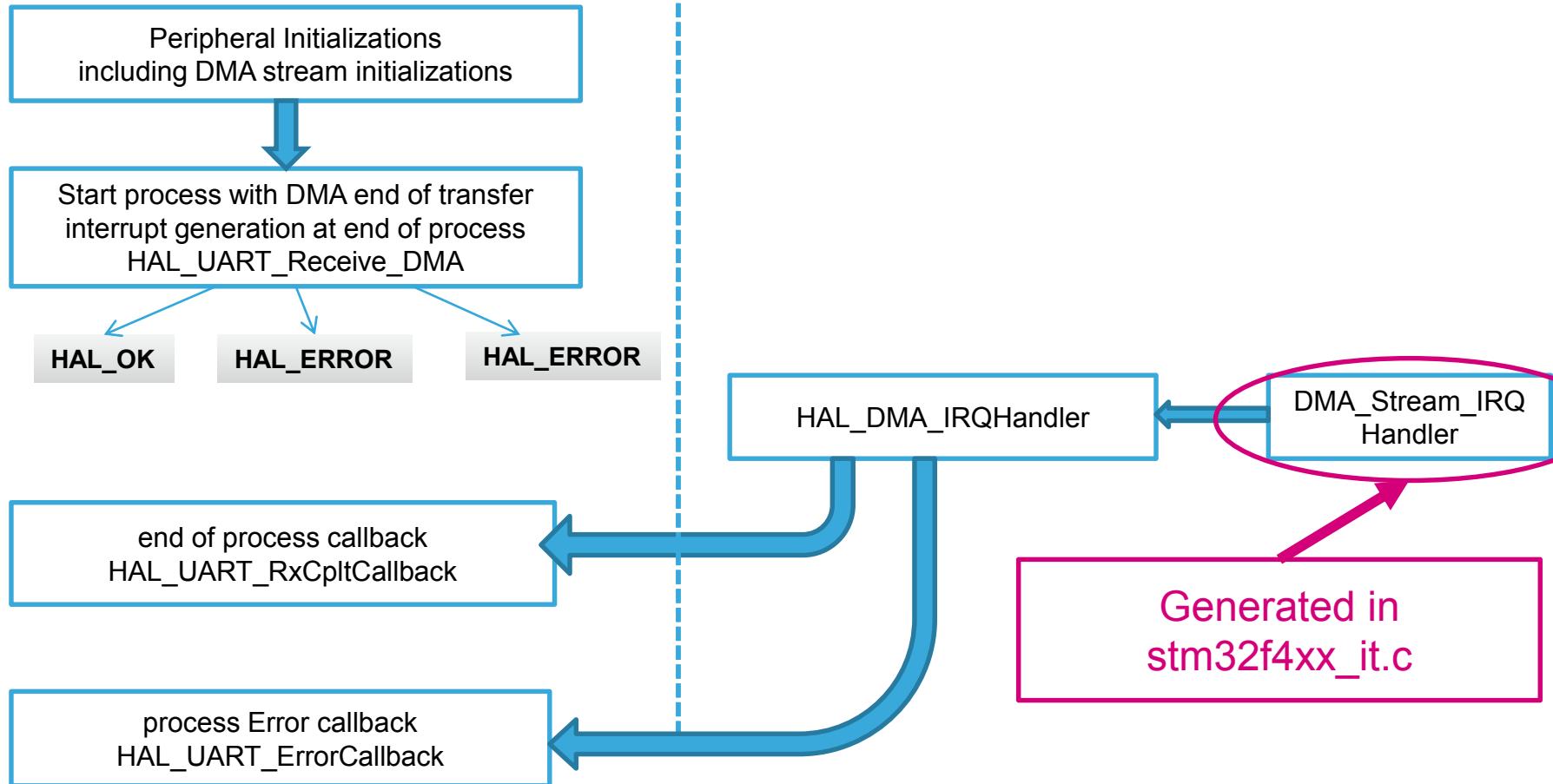
## HAL Library UART with DMA RX flow



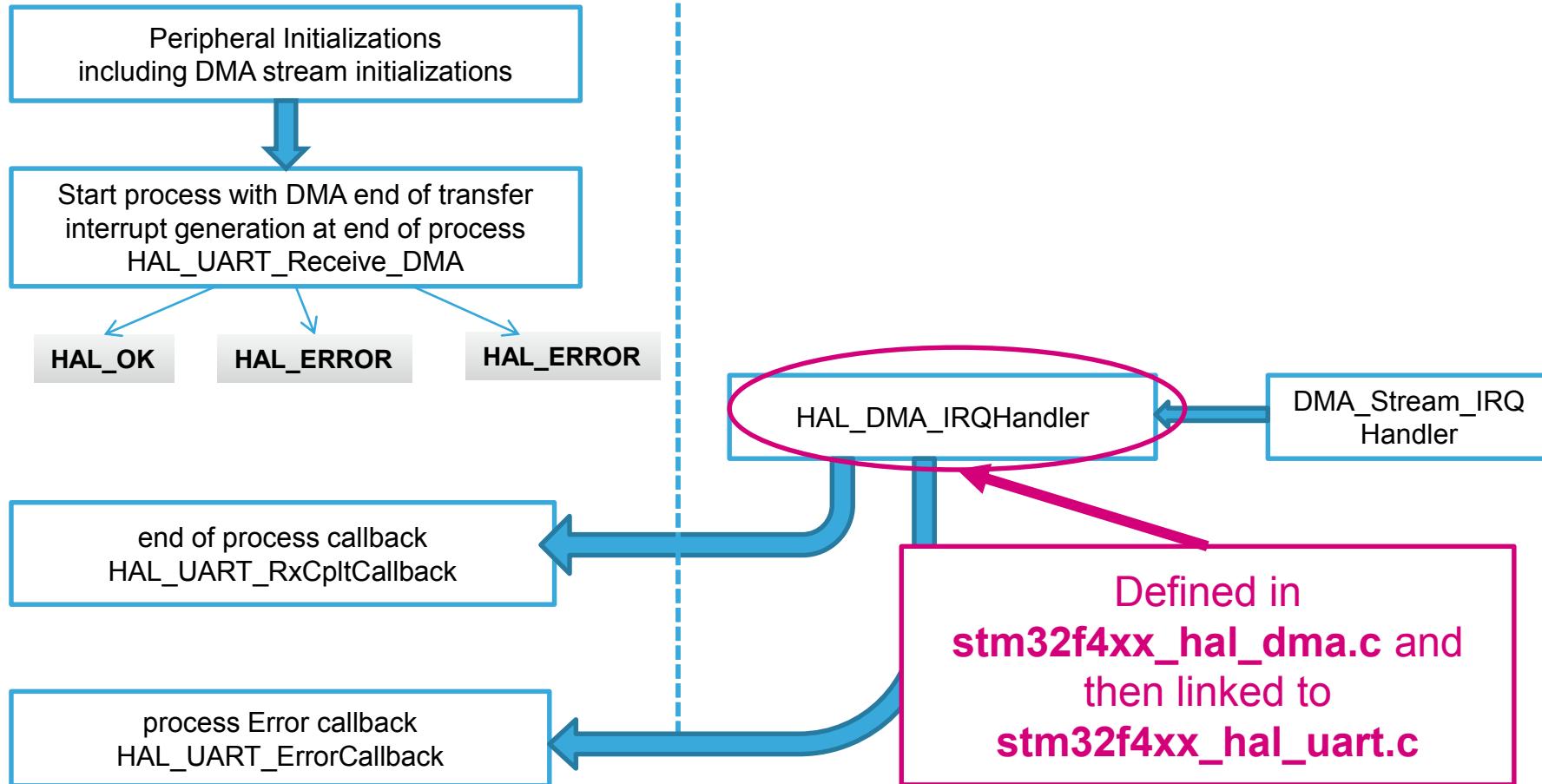
## HAL Library UART with DMA RX flow



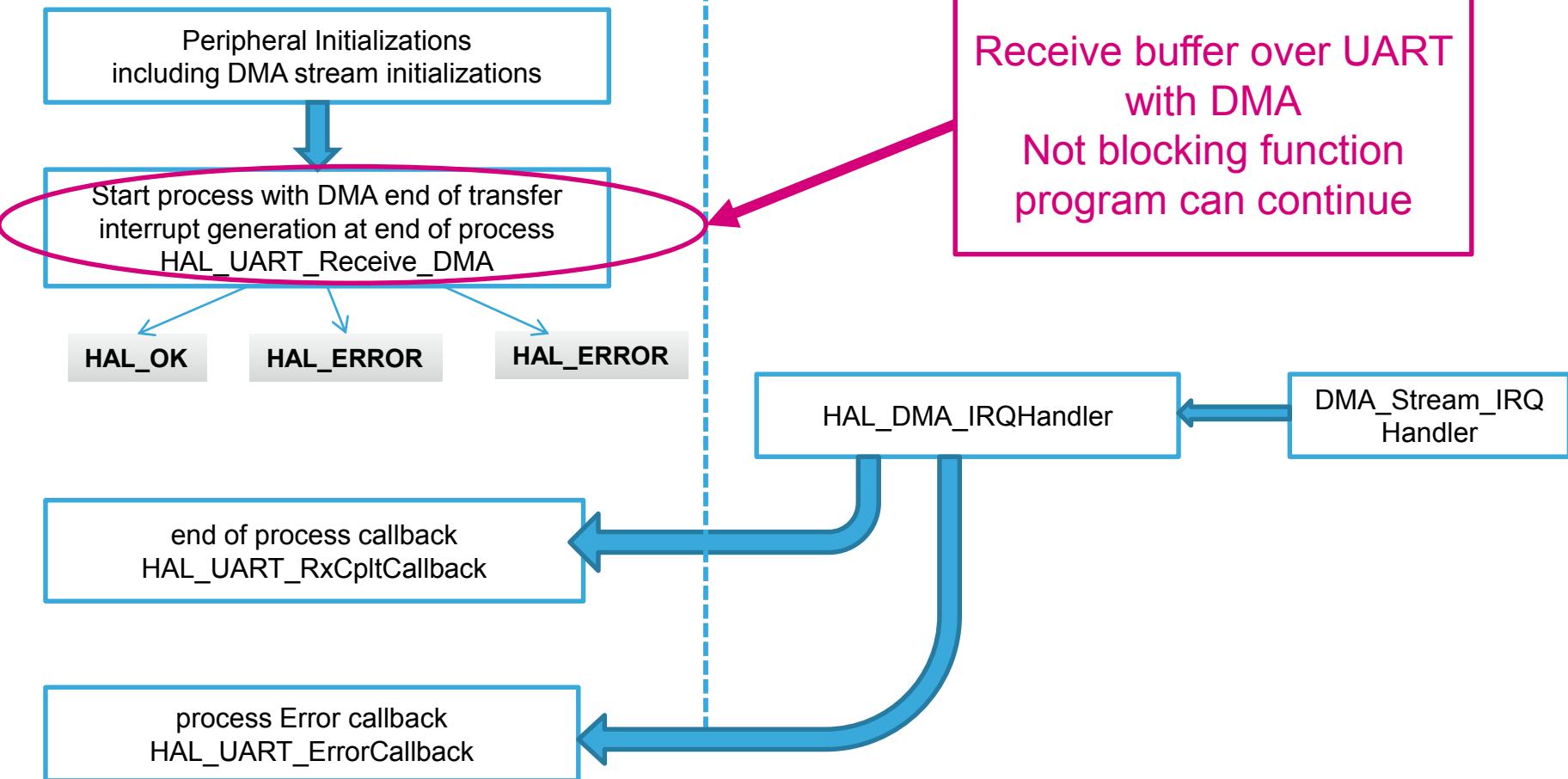
## HAL Library UART with DMA RX flow



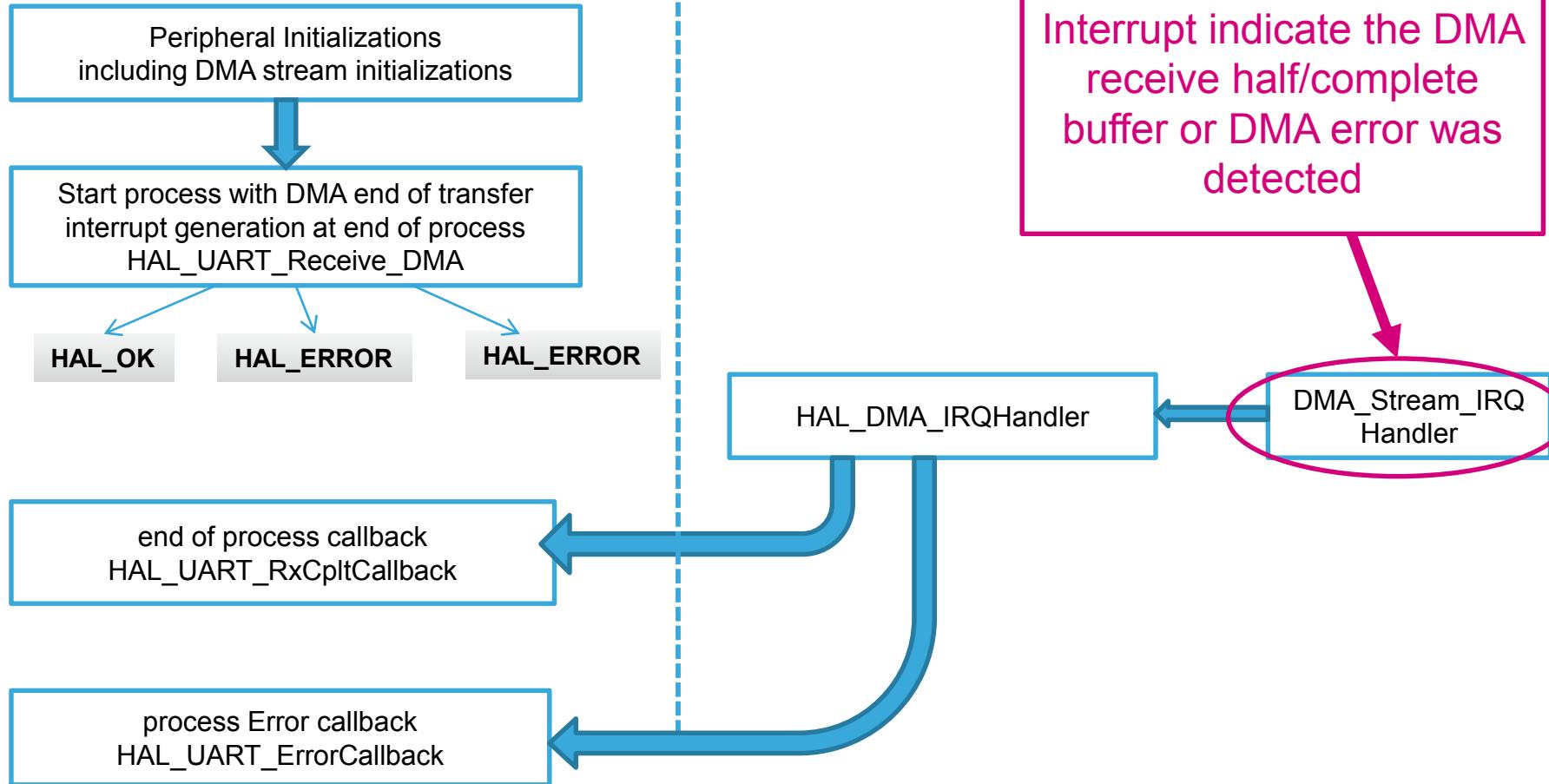
## HAL Library UART with DMA RX flow



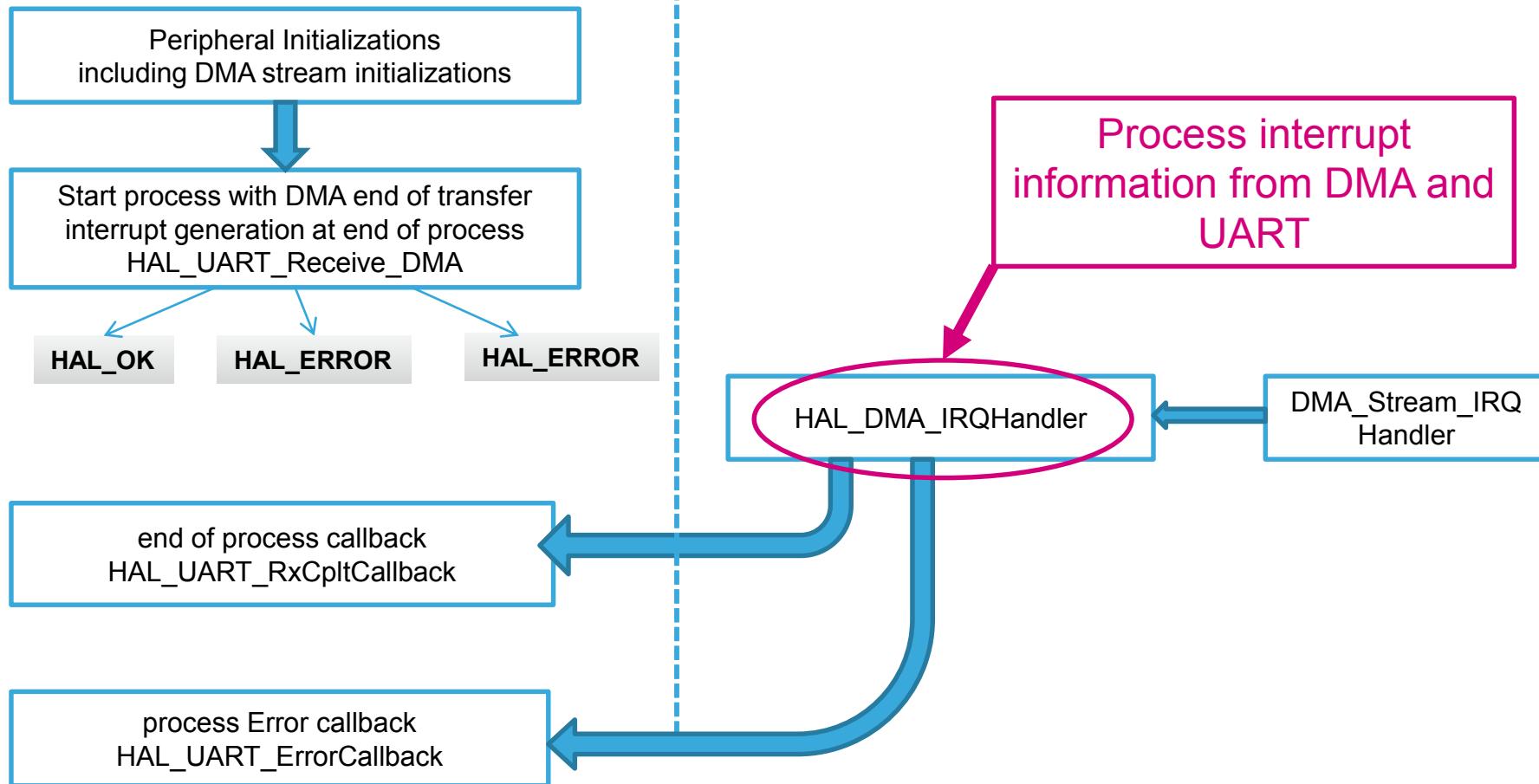
## HAL Library UART with DMA RX flow



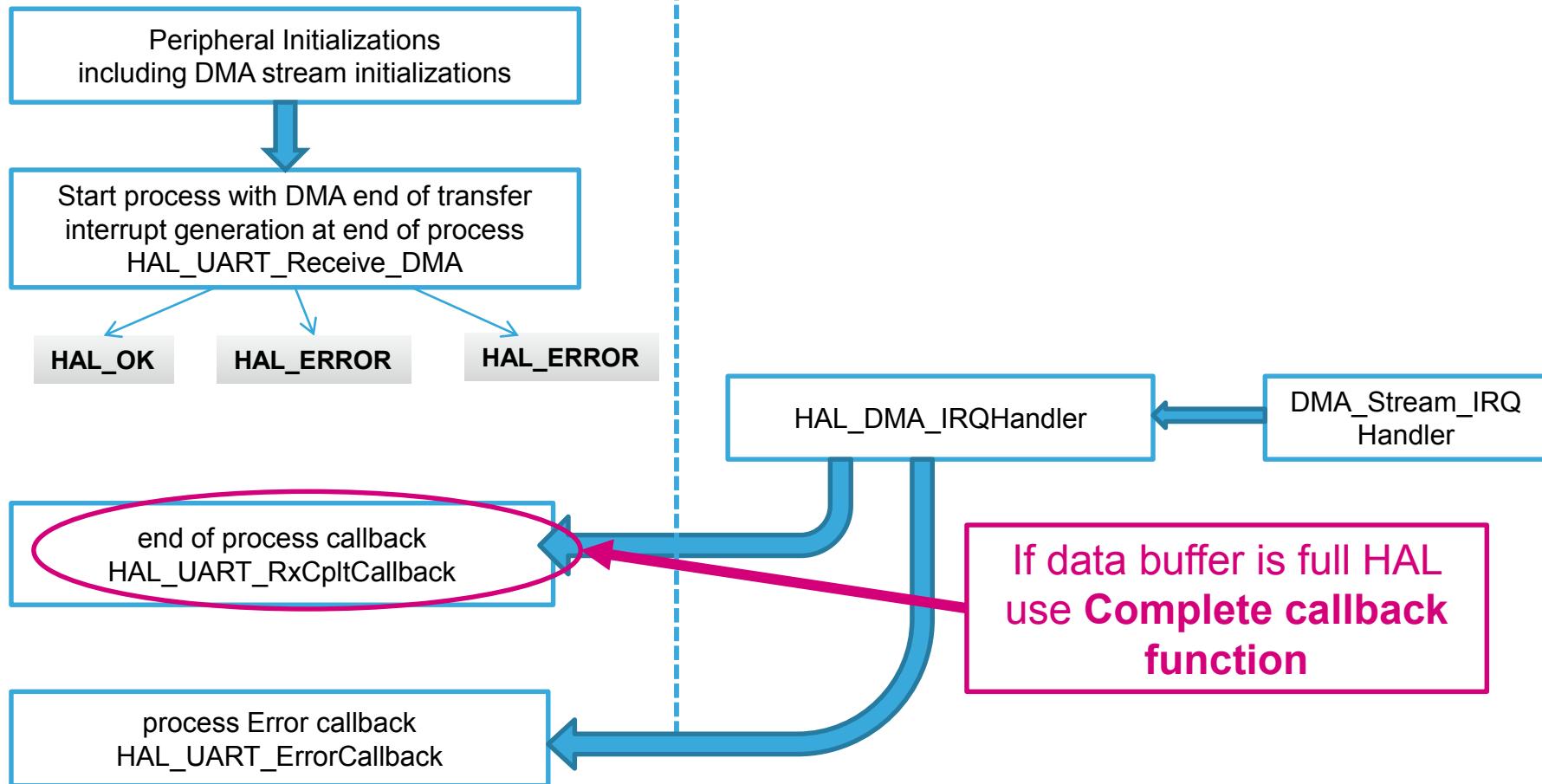
## HAL Library UART with DMA RX flow



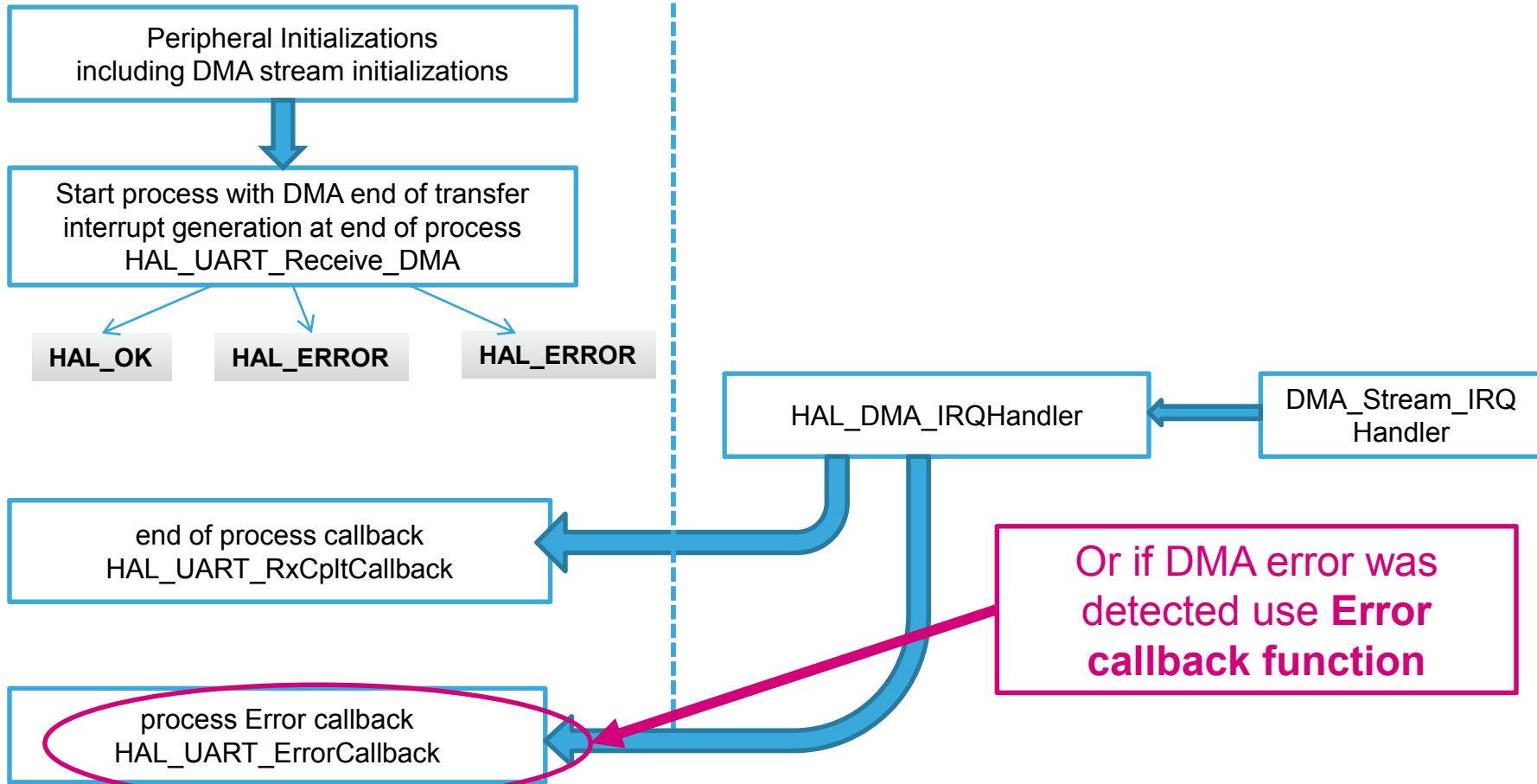
## HAL Library UART with DMA RX flow



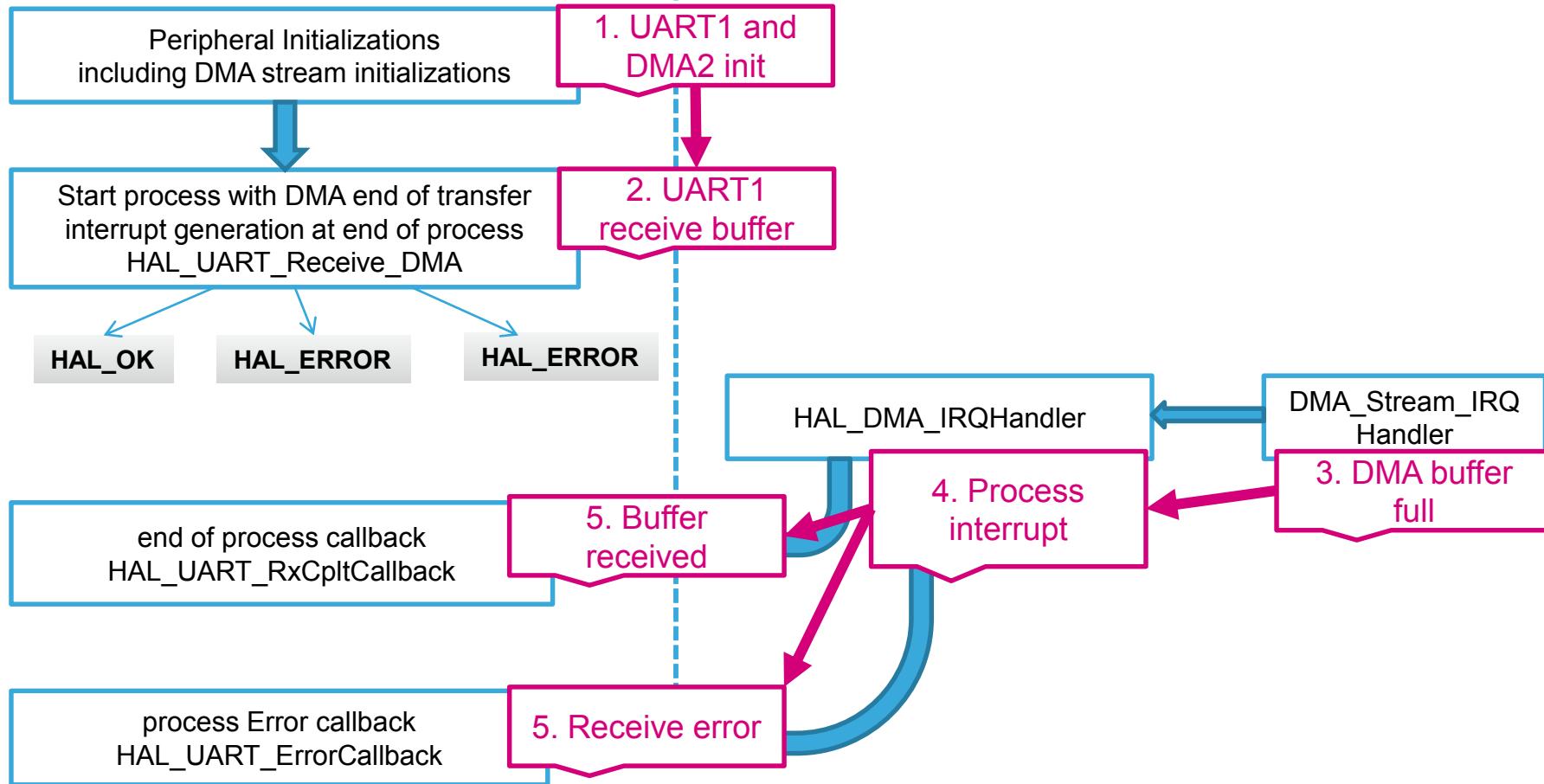
## HAL Library UART with DMA RX flow



## HAL Library UART with DMA RX flow



## HAL Library UART with DMA RX flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For transmit use function
  - `HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);`
- For receive use function
  - `HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);`

- Buffer definition

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

- Sending and receiving methods with DMA

```
/* USER CODE BEGIN 2 */  
HAL_UART_Receive_DMA(&huart1,rx_buff,10);  
HAL_UART_Transmit_DMA(&huart1,tx_buff,10);  
/* USER CODE END 2 */
```

- Complete callback check
  - We can put breakpoints on NOPs to watch if we receive complete buffer

```
/* USER CODE BEGIN 4 */  
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  
{  
    __NOP(); //check if we receive all data  
}  
/* USER CODE END 4 */
```



# SPI Poll lab 12

- Objective

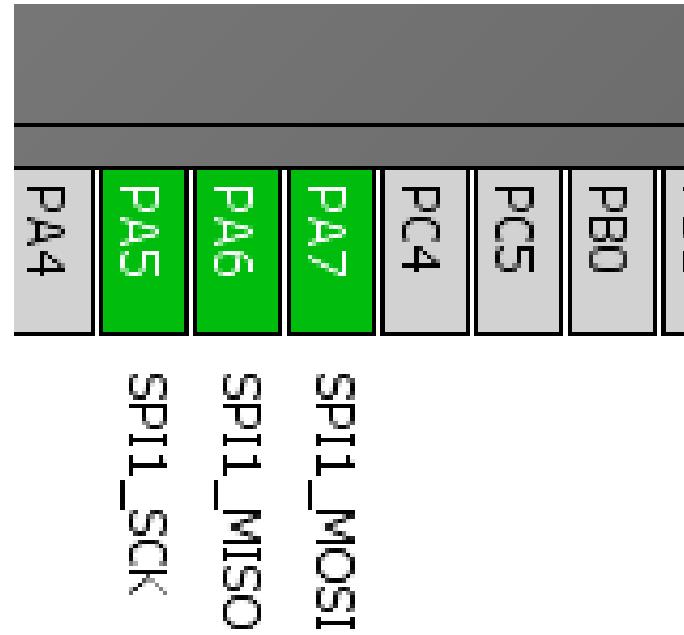
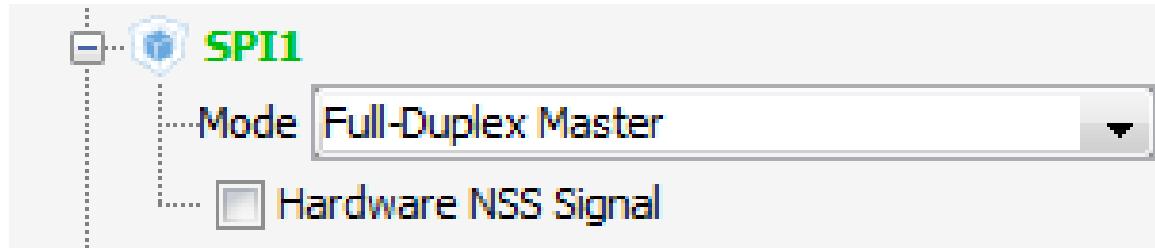
- Learn how to setup SPI in CubeMX
- How to Generate Code in CubeMX and use HAL functions

- Goal

- Configure SPI in CubeMX and Generate Code
- Learn how to send and receive data over SPI without interrupts
- Verify the correct functionality

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Pin selection
  - We are looking for free pins where is possible to create wire loopback connection

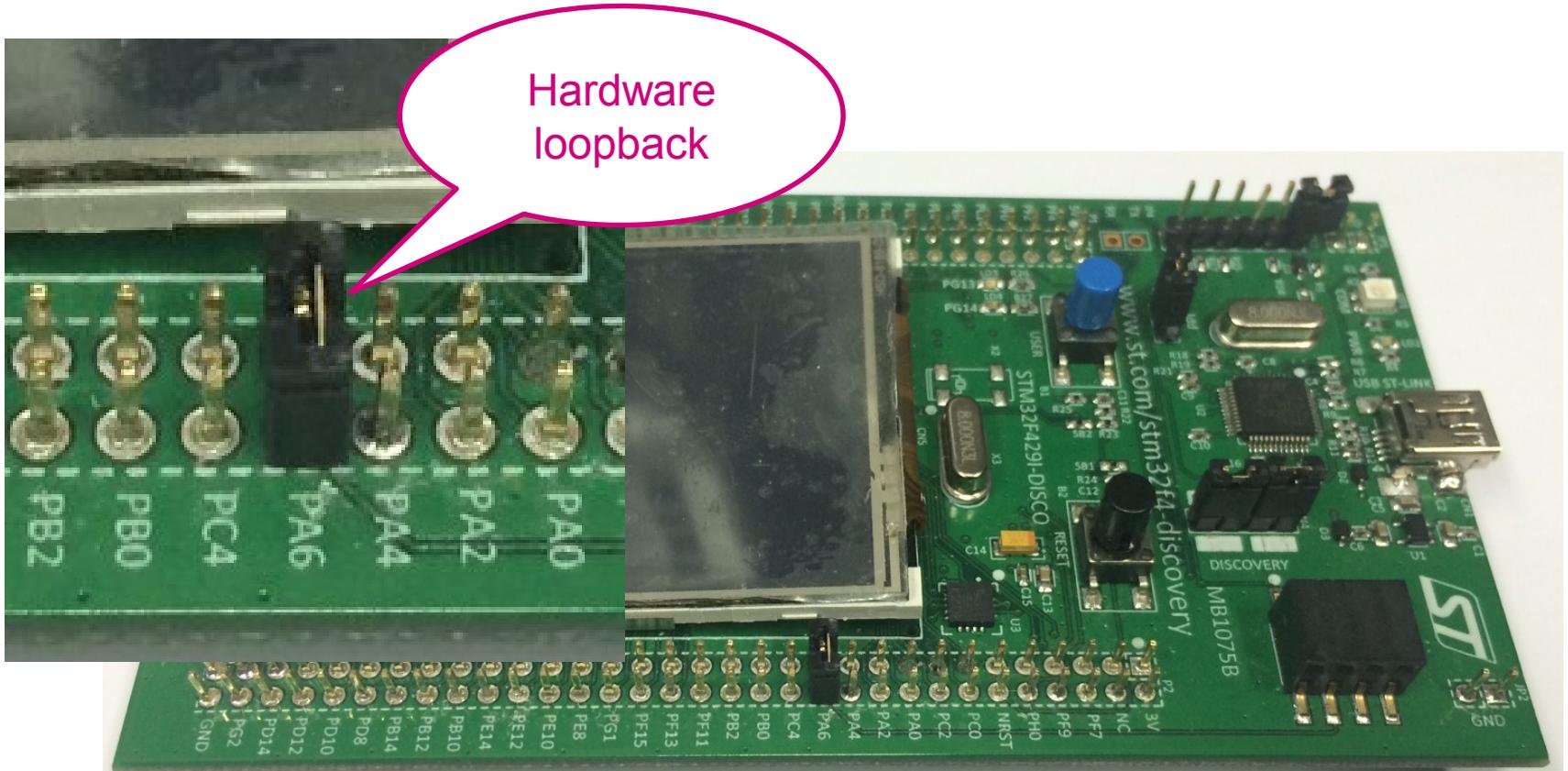
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX SPI selection
  - Select SPI1 Full-Duplex Master
  - Select PA5, PA6, PA7 for SPI1 if weren't selected



12

# Simple SPI communication

- Hardware preparation
    - Connect PA6 and PA7 together with jumper

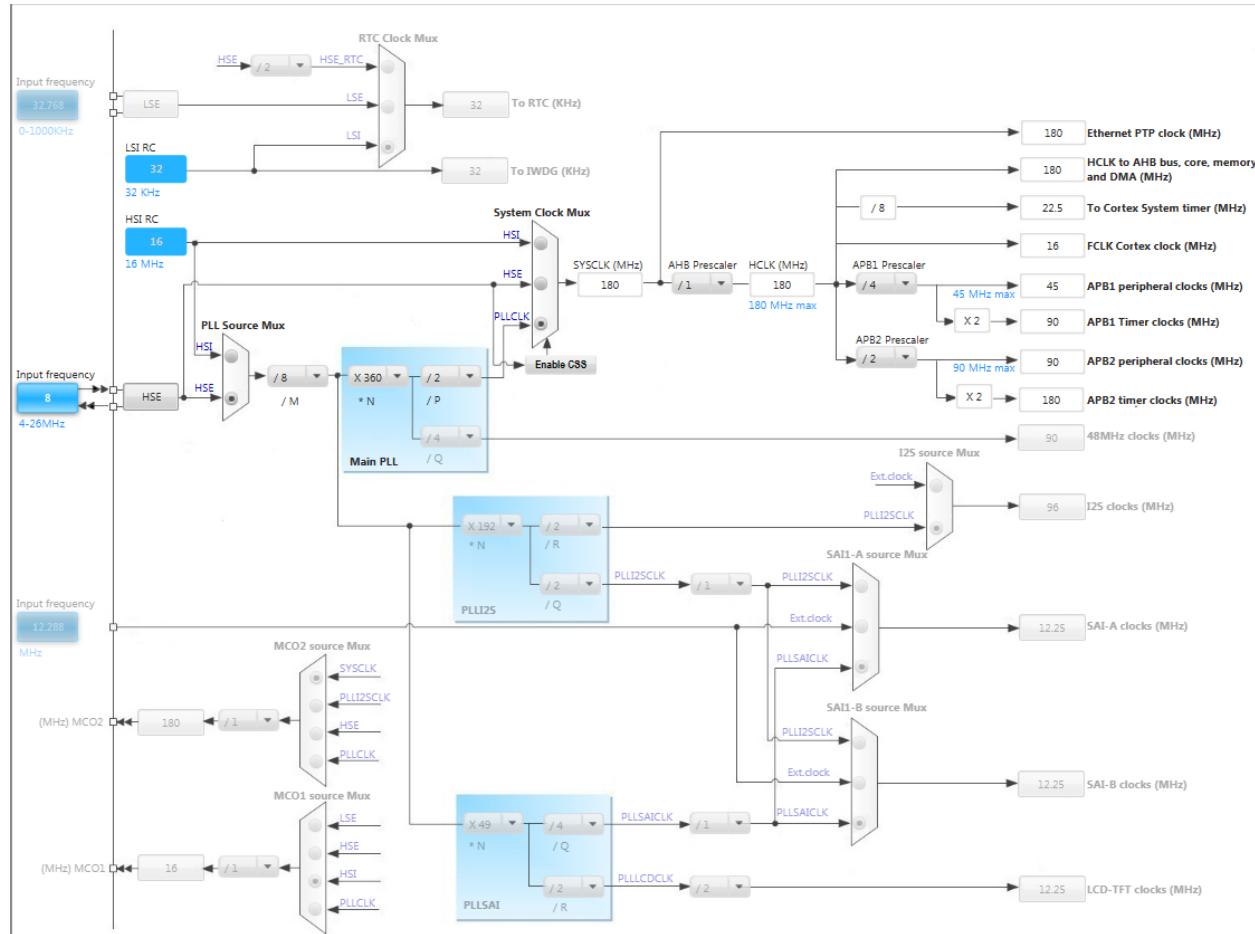


# 12

# Simple SPI communication

228

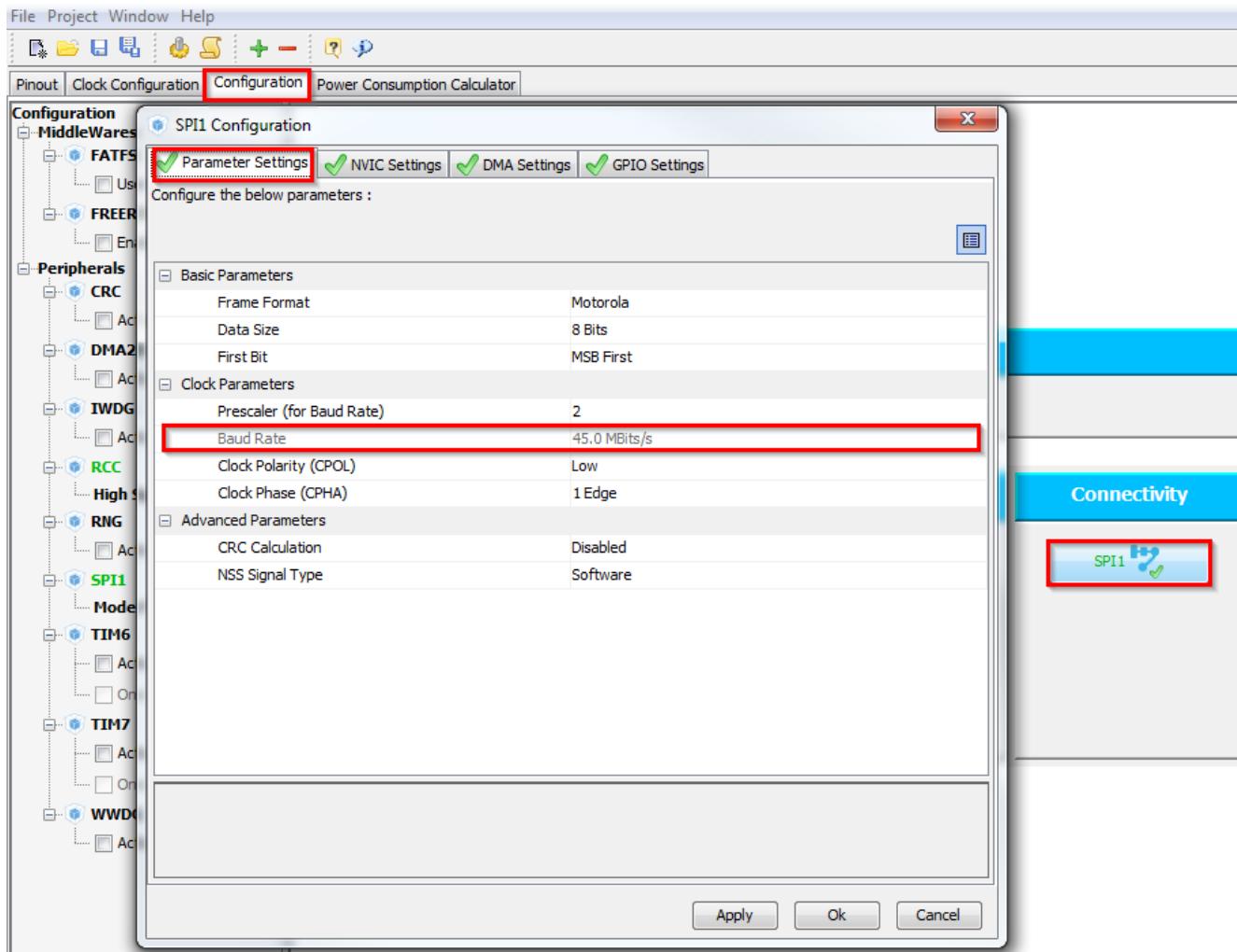
- In order to run on maximum frequency, setup clock system
- Details in lab 0



- CubeMX SPI configuration

- Tab>Configuration>Connectivity>SPI1
- Check the settings
- Button OK

- The CLK frequency with core on 180MHz is now 45MHz
- For this clock use HIGH GPIO speed

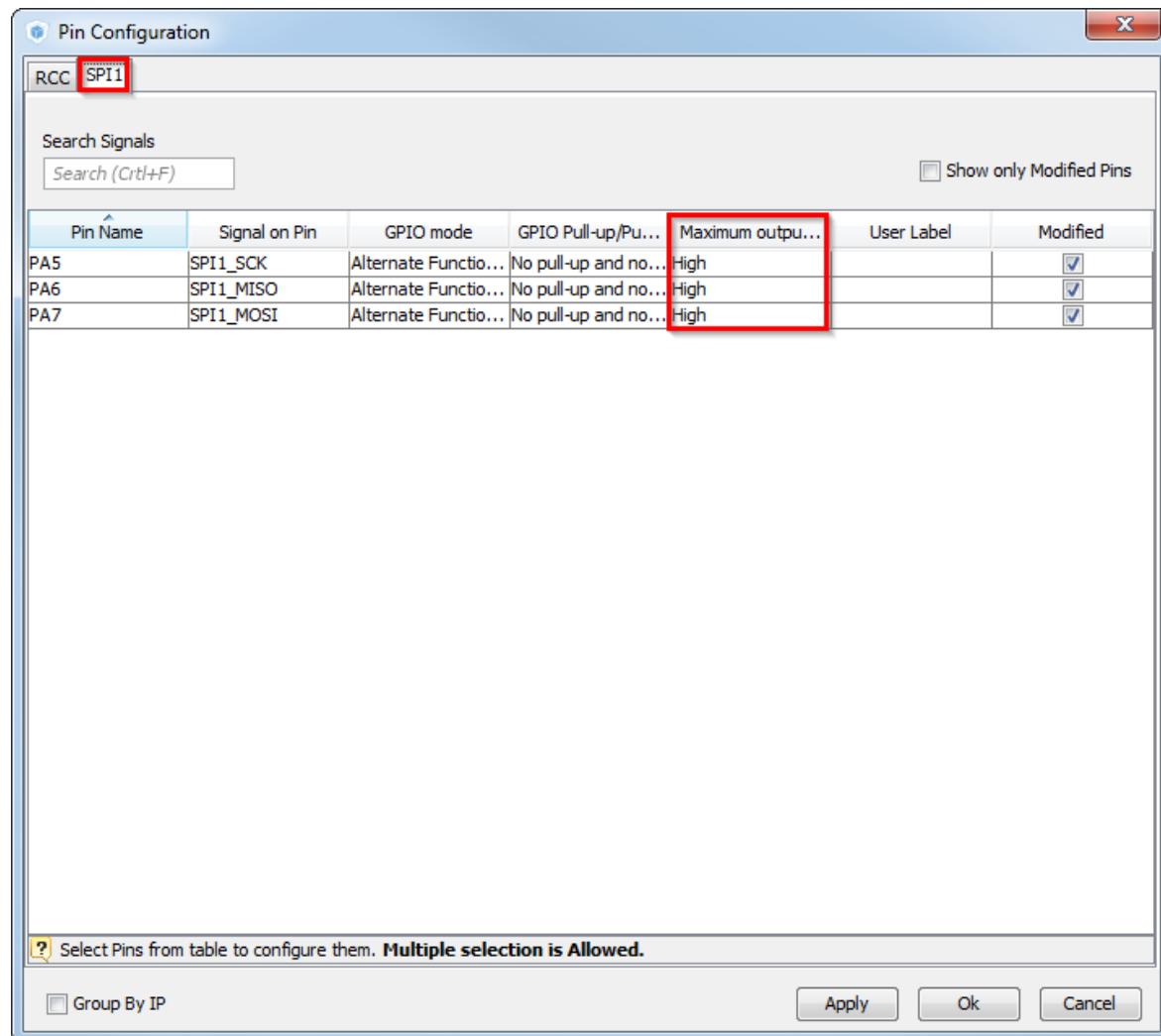


- CubeMX SPI – GPIO configuration

- The SPI CLK frequency with core on 180MHz is now 45MHz

- For this clock use HIGH GPIO speed

- Tab>Configuration>System>>GPIO
  - Tab>SPI1
  - Set High output speed
  - Button OK

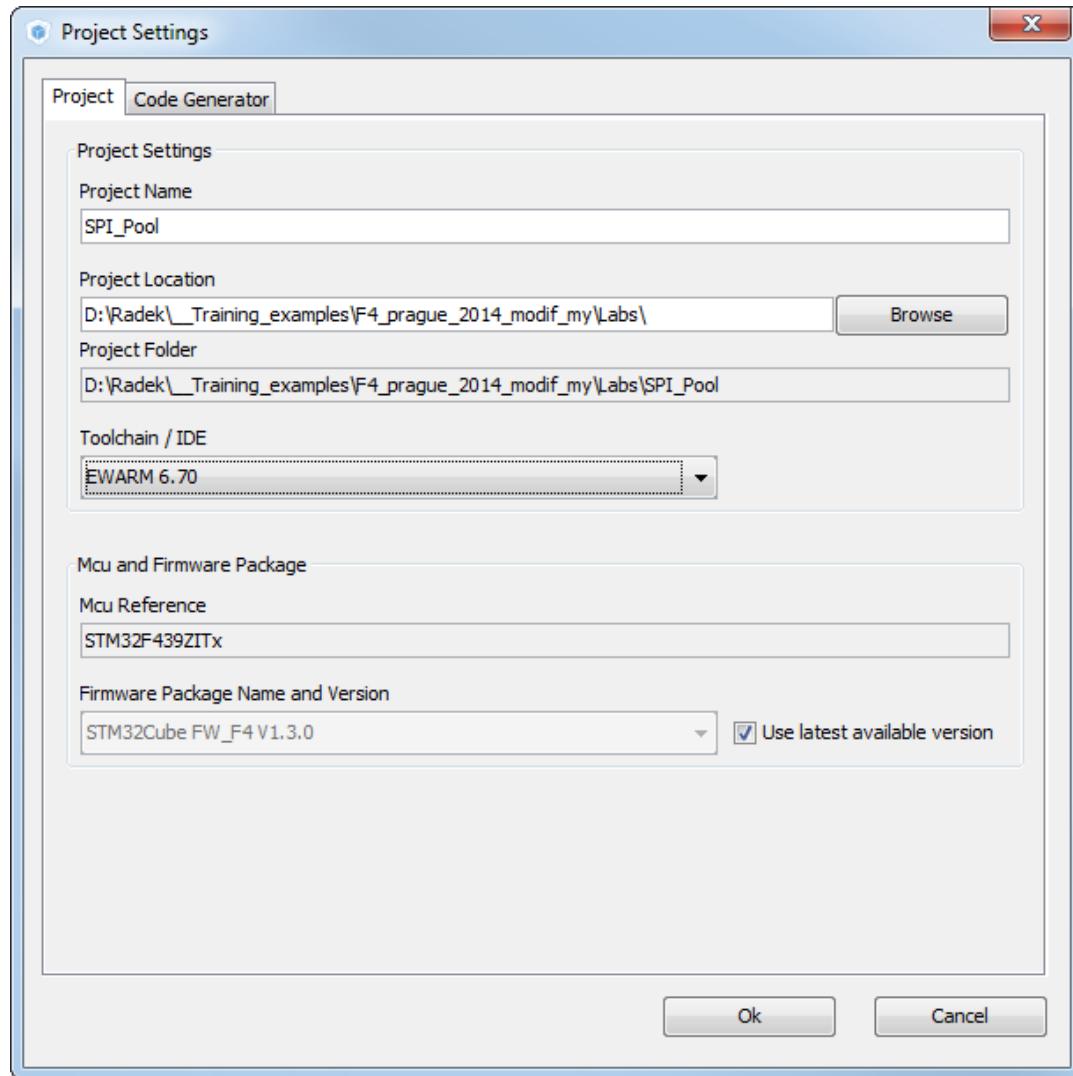


- Now we set the project details for generation

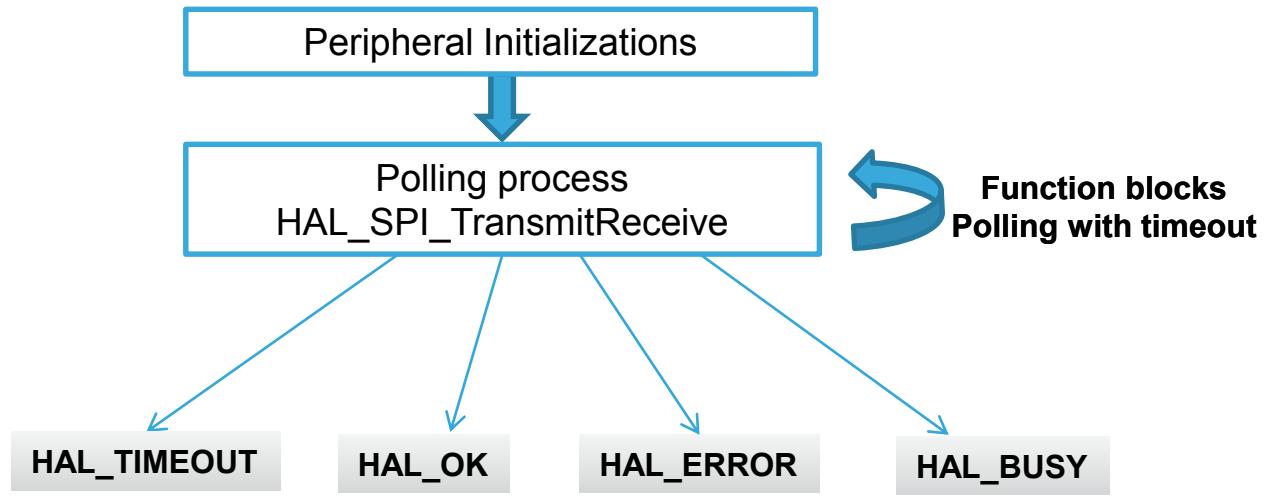
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



## HAL Library transmit receive flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 3 \*/* and */\* USER CODE END 3 \*/* tags
- For transmit and receive use function
  - `HAL_SPI_TransmitReceive(SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size, uint32_t Timeout)`

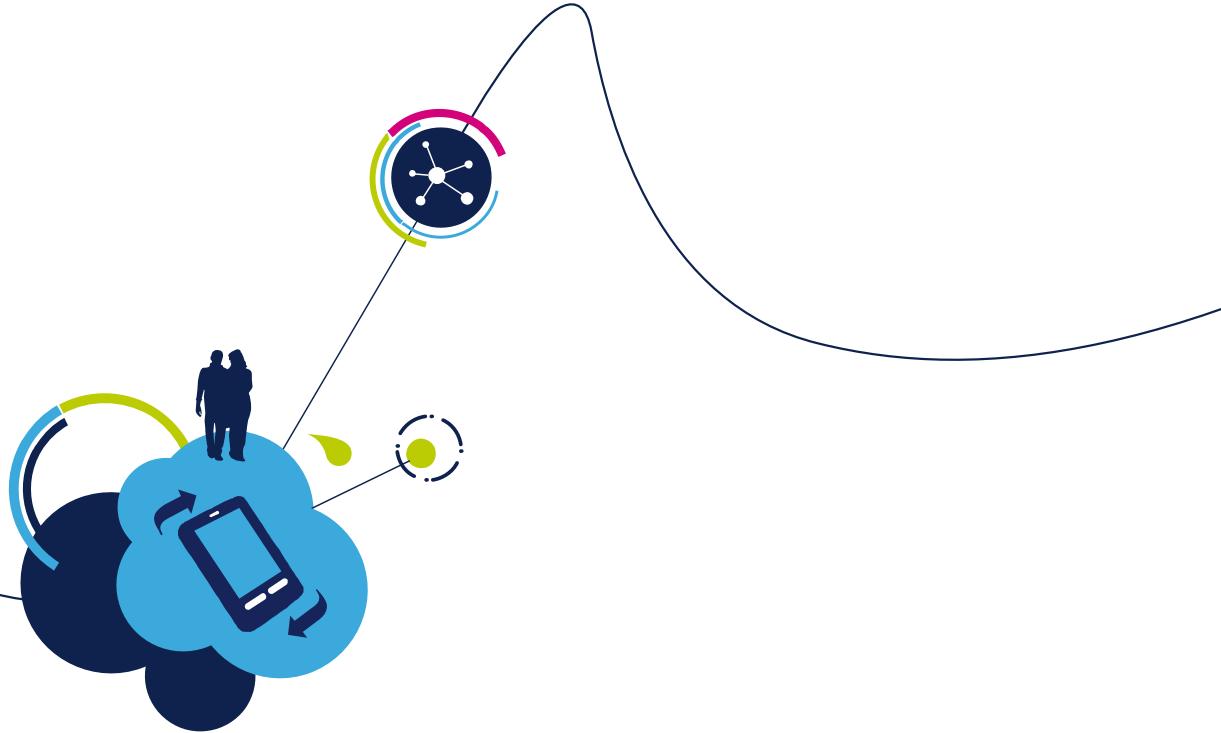
- Transmit receive solution

- Create data structure for data

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

- Call transmit receive function

```
/* USER CODE BEGIN 2 */  
HAL_SPI_TransmitReceive(&hspi1,tx_buffer,rx_buffer,10,100);  
/* USER CODE END 2 */
```



# SPI Interrupt lab 13

- Objective

- Learn how to setup SPI with interrupts in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Create simple loopback example with interrupts

- Goal

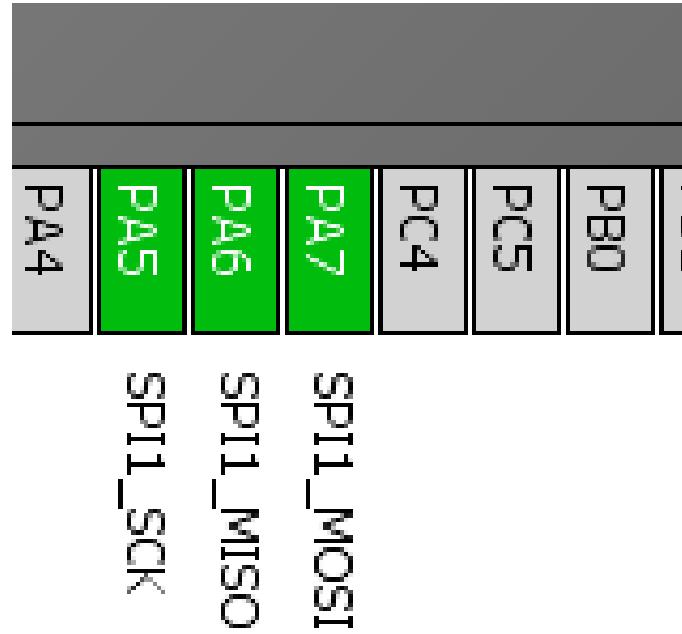
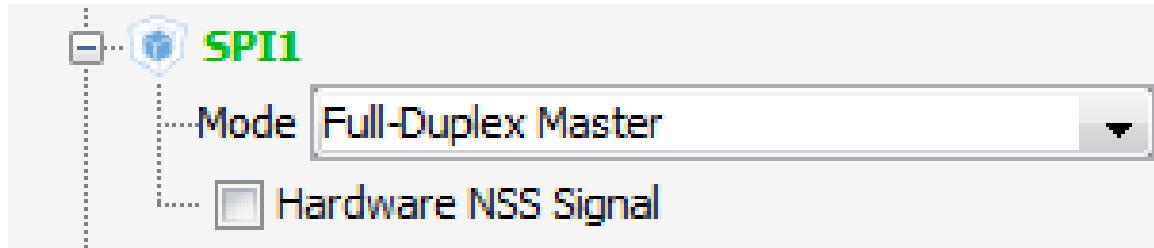
- Configure SPI in CubeMX and Generate Code
- Learn how to send and receive data over SPI with interrupts
- Verify the correct functionality

# 13

# Use SPI with interrupt

237

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX SPI selection
  - Select SPI1 Full-Duplex Master
  - Select PA5, PA6, PA7 for SPI1 if weren't selected

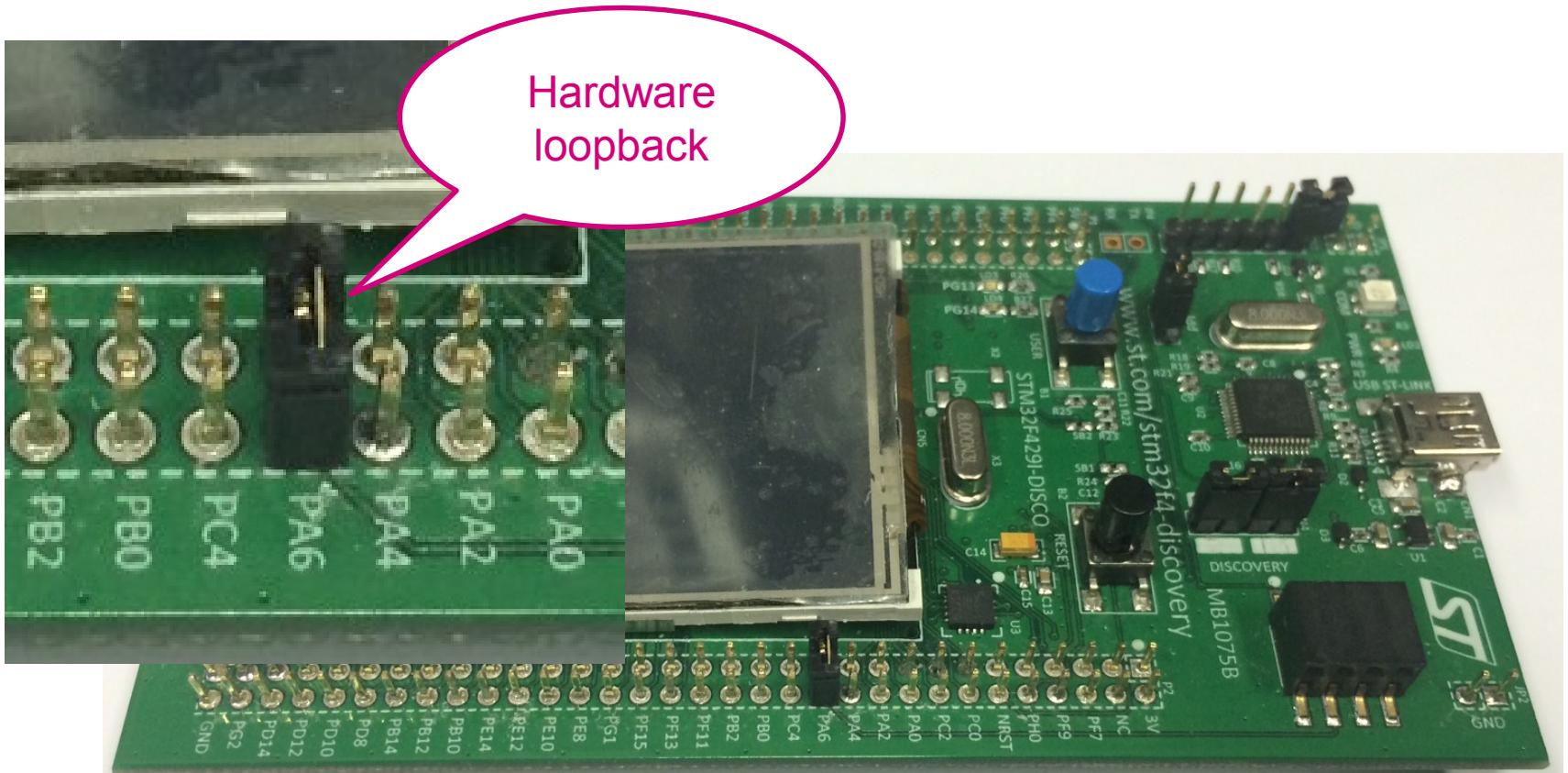


13

# Use SPI with interrupt

238

- Hardware preparation
    - Connect PA6 and PA7 together with jumper

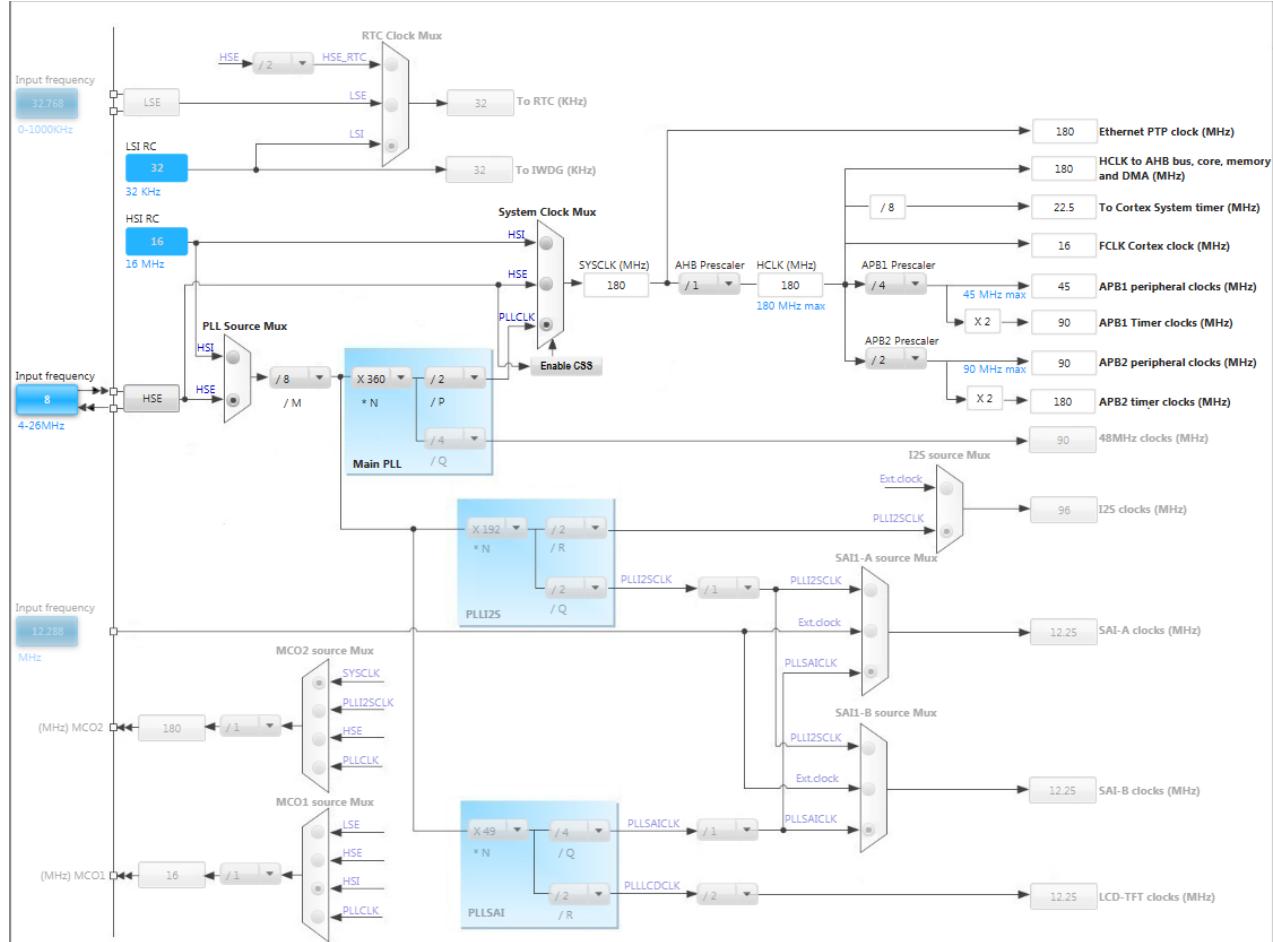


# 13

# Use SPI with interrupt

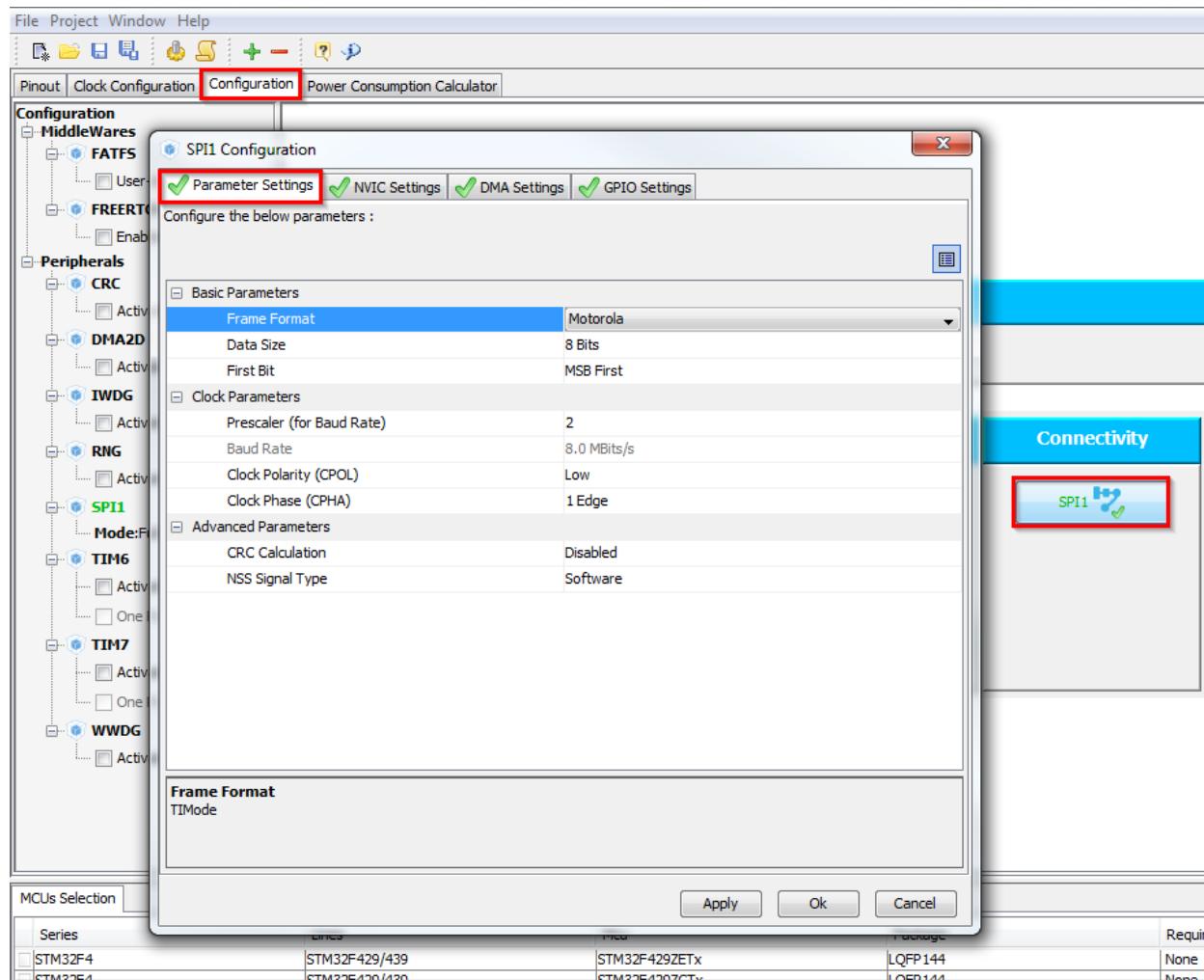
239

- In order to run on maximum frequency, setup clock system
- Details in lab 0



- CubeMX SPI configuration

- Tab>Configuration>Connectivity>SPI1
- Check the settings
- Button OK

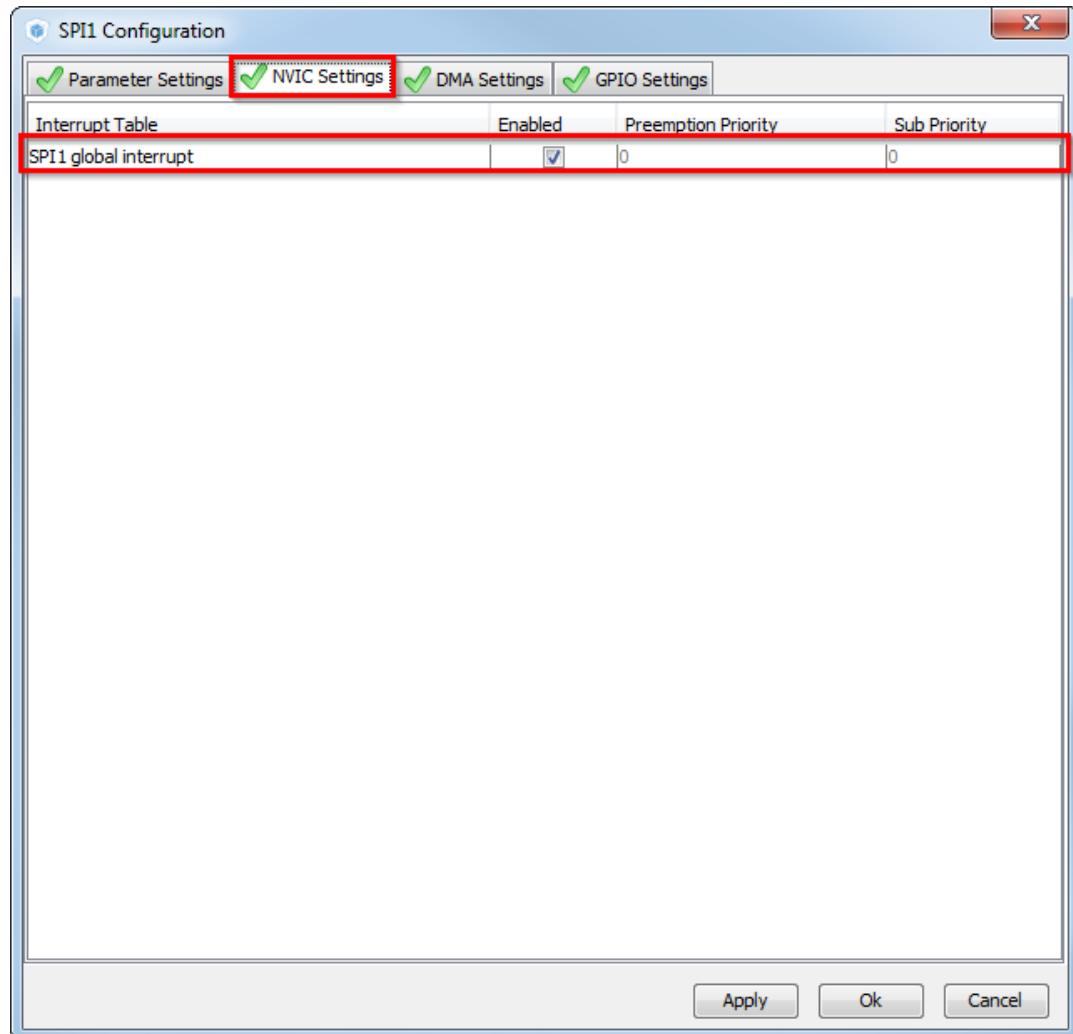


# 13

# Use SPI with interrupt

241

- CubeMX SPI configuration
  - TAB>NVIC Settings
  - Enable SPI interrupt
  - Button OK

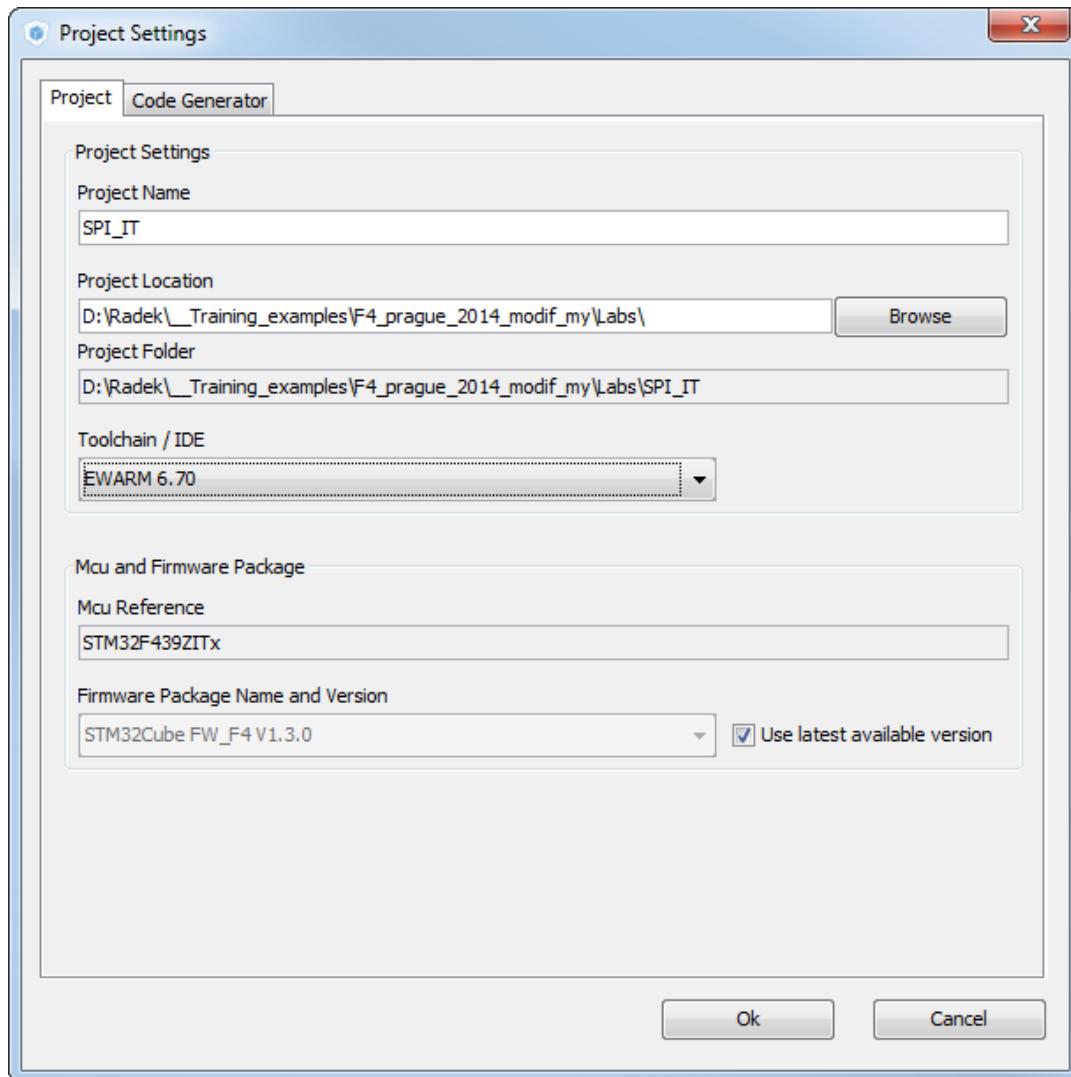


- Now we set the project details for generation

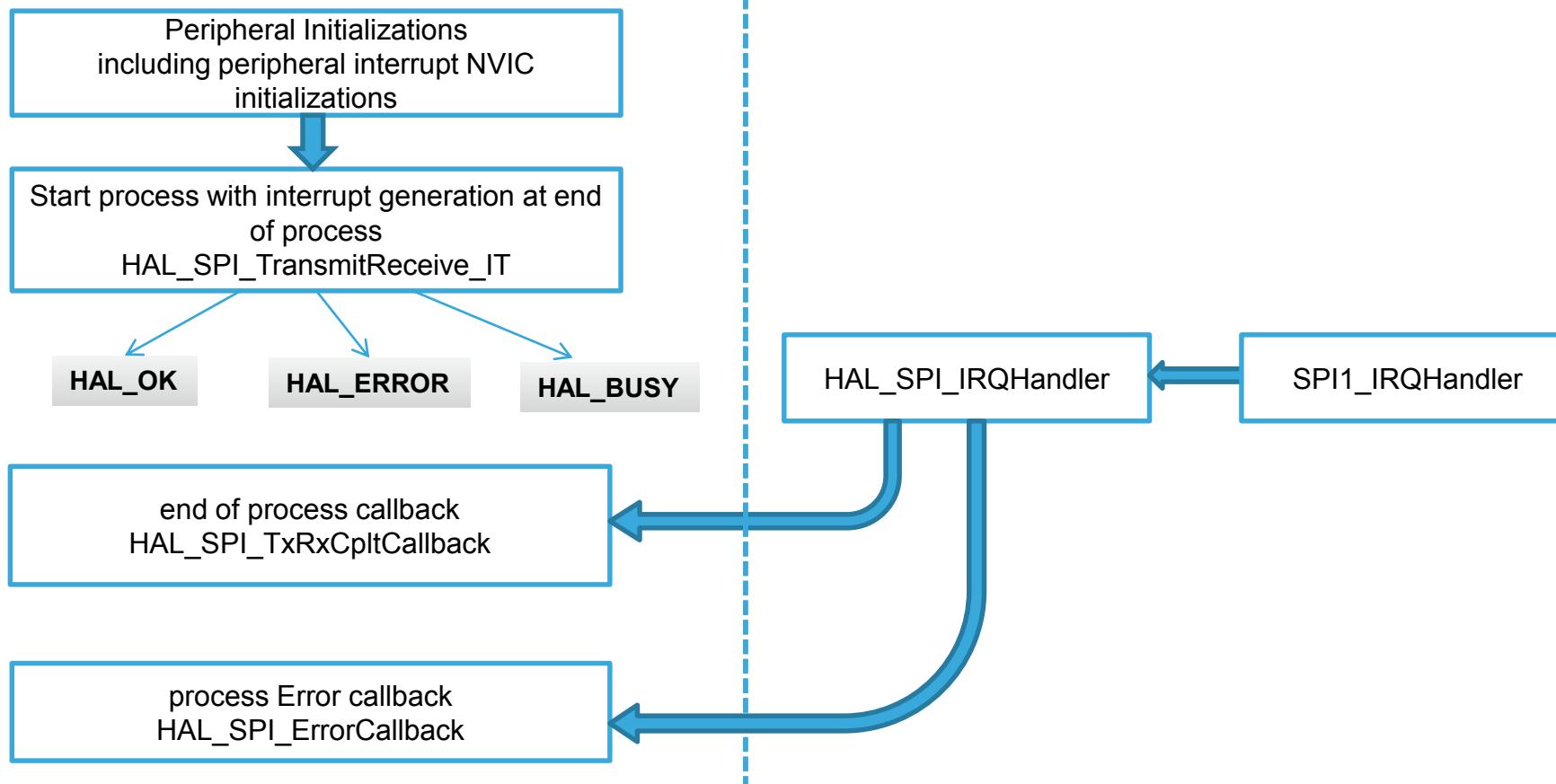
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



## HAL Library SPI with IT transmit receive flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For transmit use function
  - `HAL_SPI_TransmitReceive_IT(SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size)`

- Buffer definition

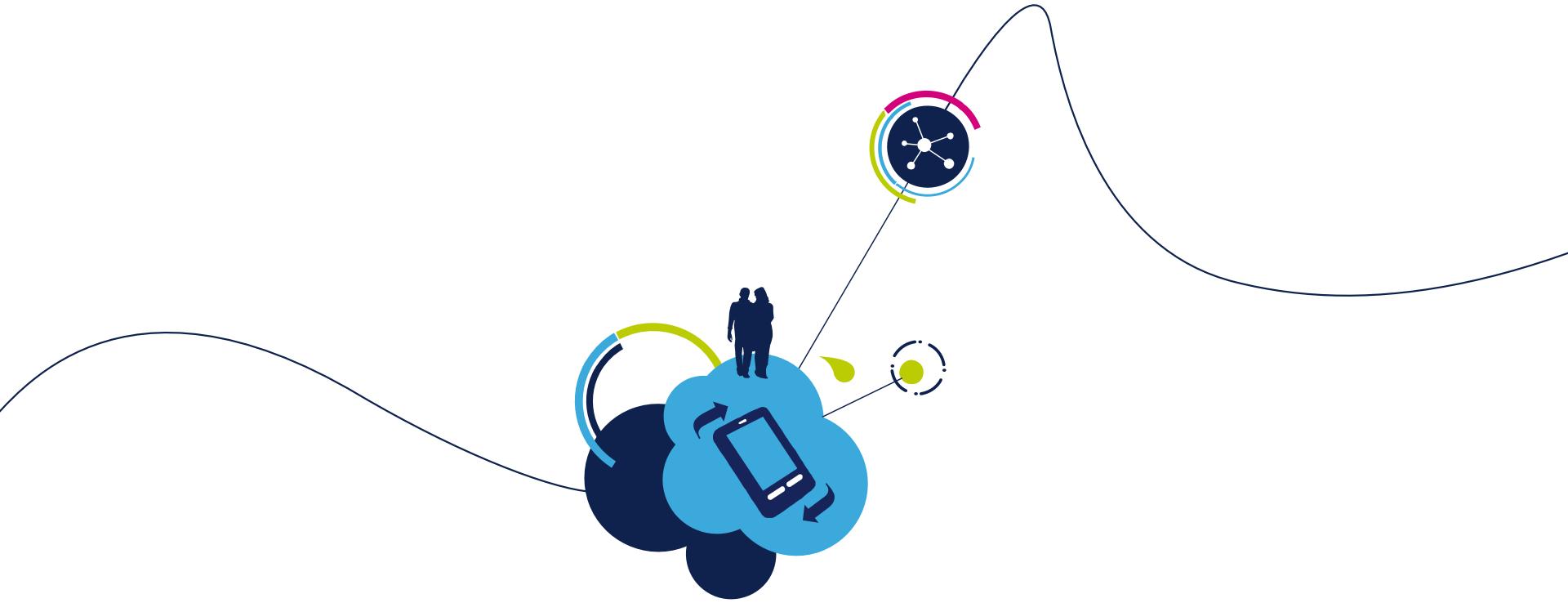
```
/* USER CODE BEGIN 0 */  
uint8_t tx_buff[]={0,1,2,3,4,5,6,7,8,9};  
uint8_t rx_buff[10];  
/* USER CODE END 0 */
```

- Sending and receiving methods

```
/* USER CODE BEGIN 2 */  
HAL_SPI_TransmitReceive_IT(&hspi1,tx_buff,rx_buff,10);  
/* USER CODE END 2 */
```

- Complete callback check
  - We can put breakpoints on NOPs to watch if we send or receive complete buffer

```
/* USER CODE BEGIN 4 */  
void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi)  
{  
    __NOP();  
}  
/* USER CODE END 4 */
```



# SPI DMA lab 14

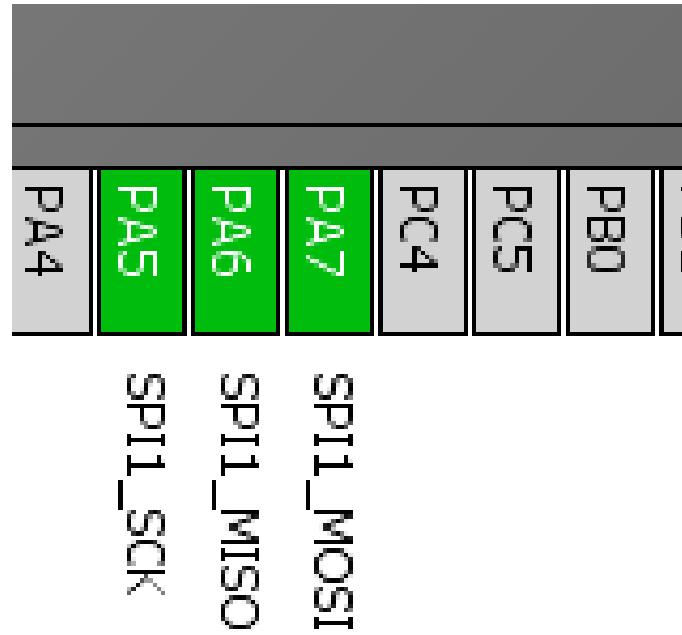
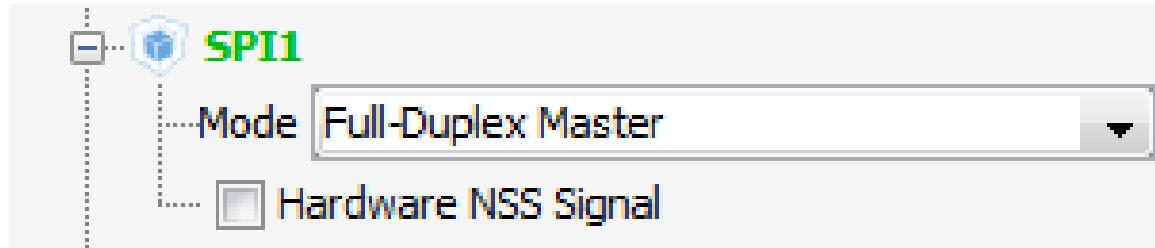
- Objective

- Learn how to setup SPI with DMA in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Create simple loopback example with DMA

- Goal

- Configure SPI in CubeMX and Generate Code
- Learn how to send and receive data over SPI with DMA
- Verify the correct functionality

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX SPI selection
  - Select SPI1 Full-Duplex Master
  - Select PA5, PA6, PA7 for SPI1 if weren't selected

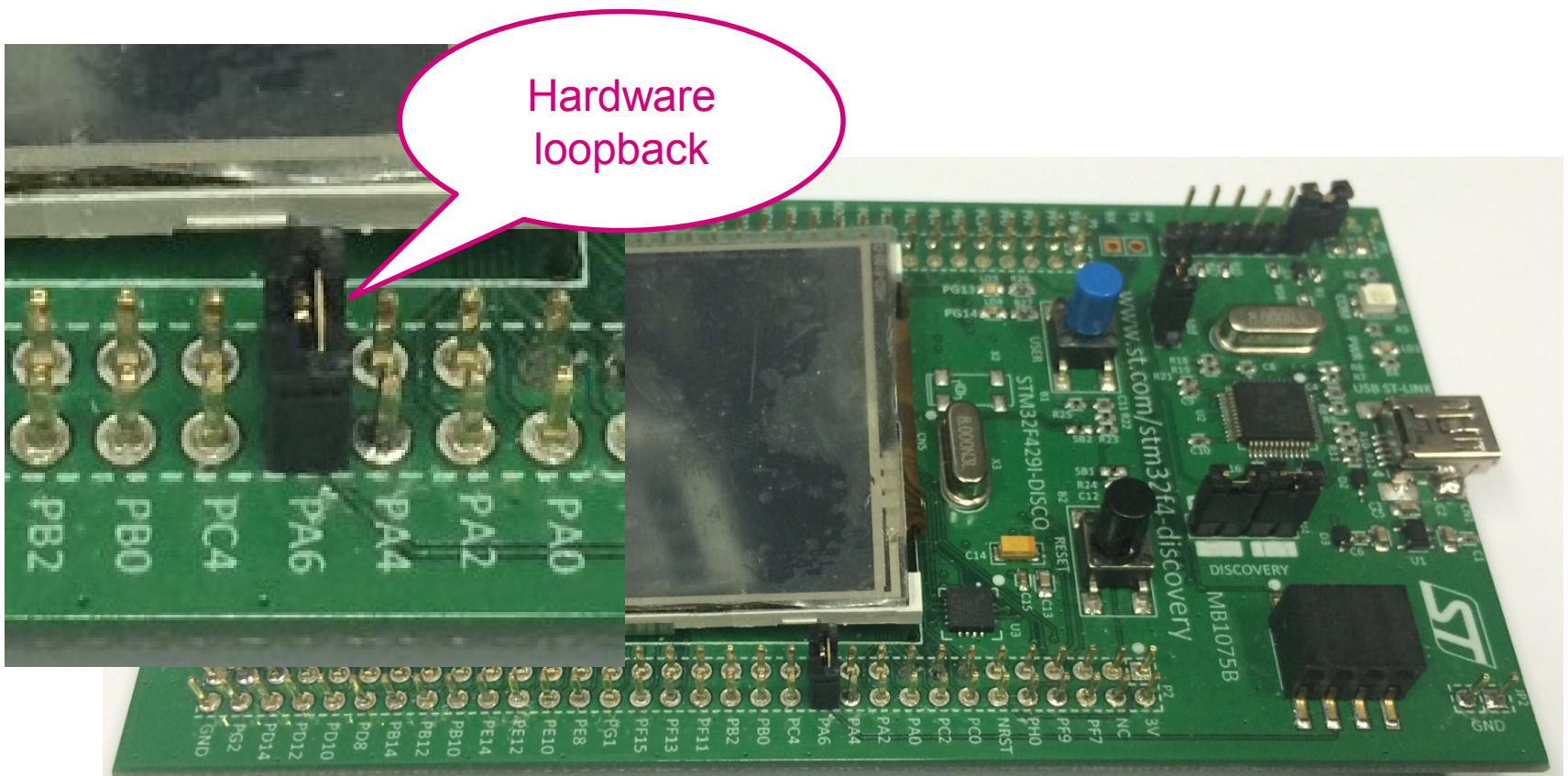


# 14

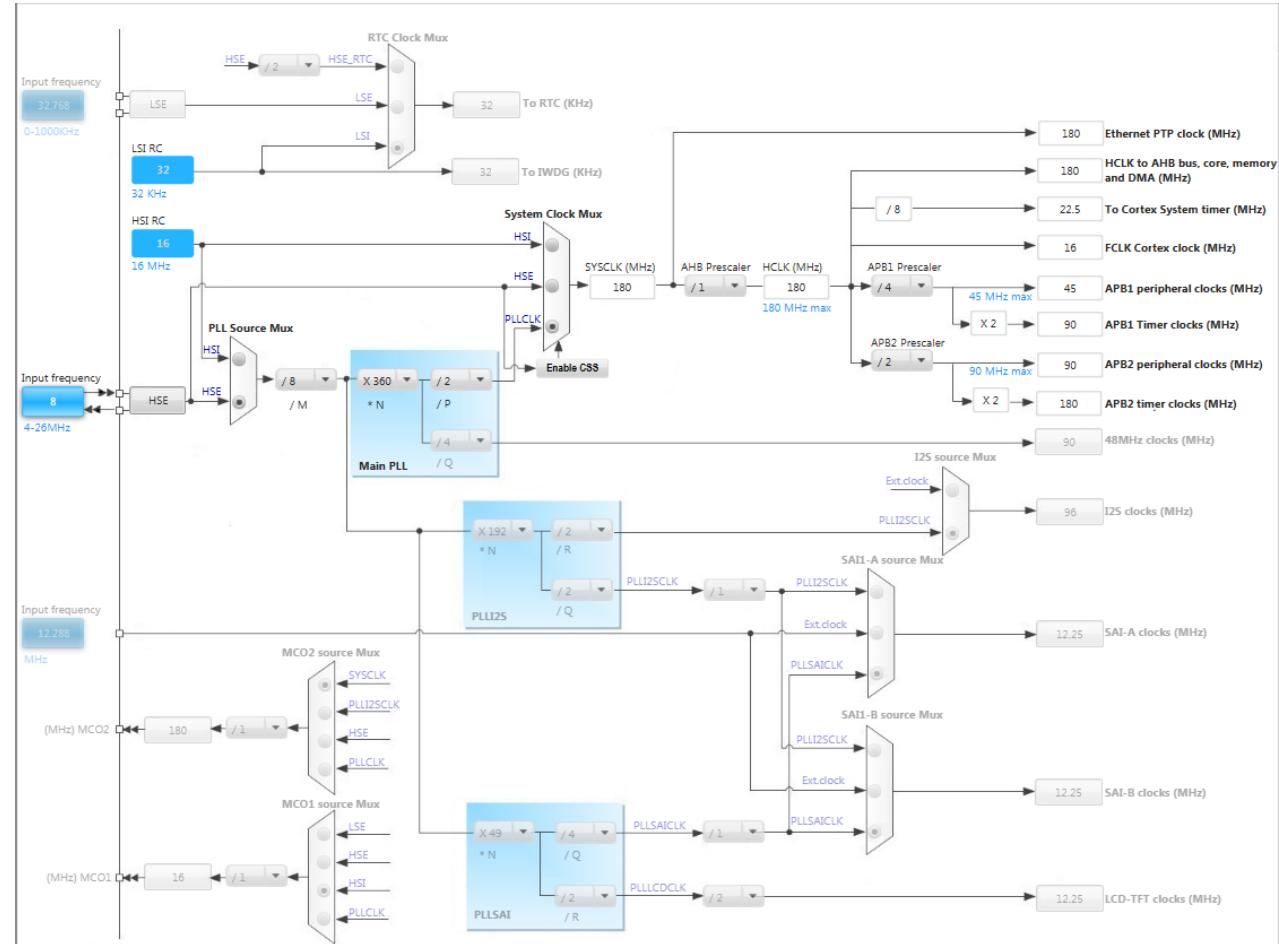
# Use SPI with DMA transfer

250

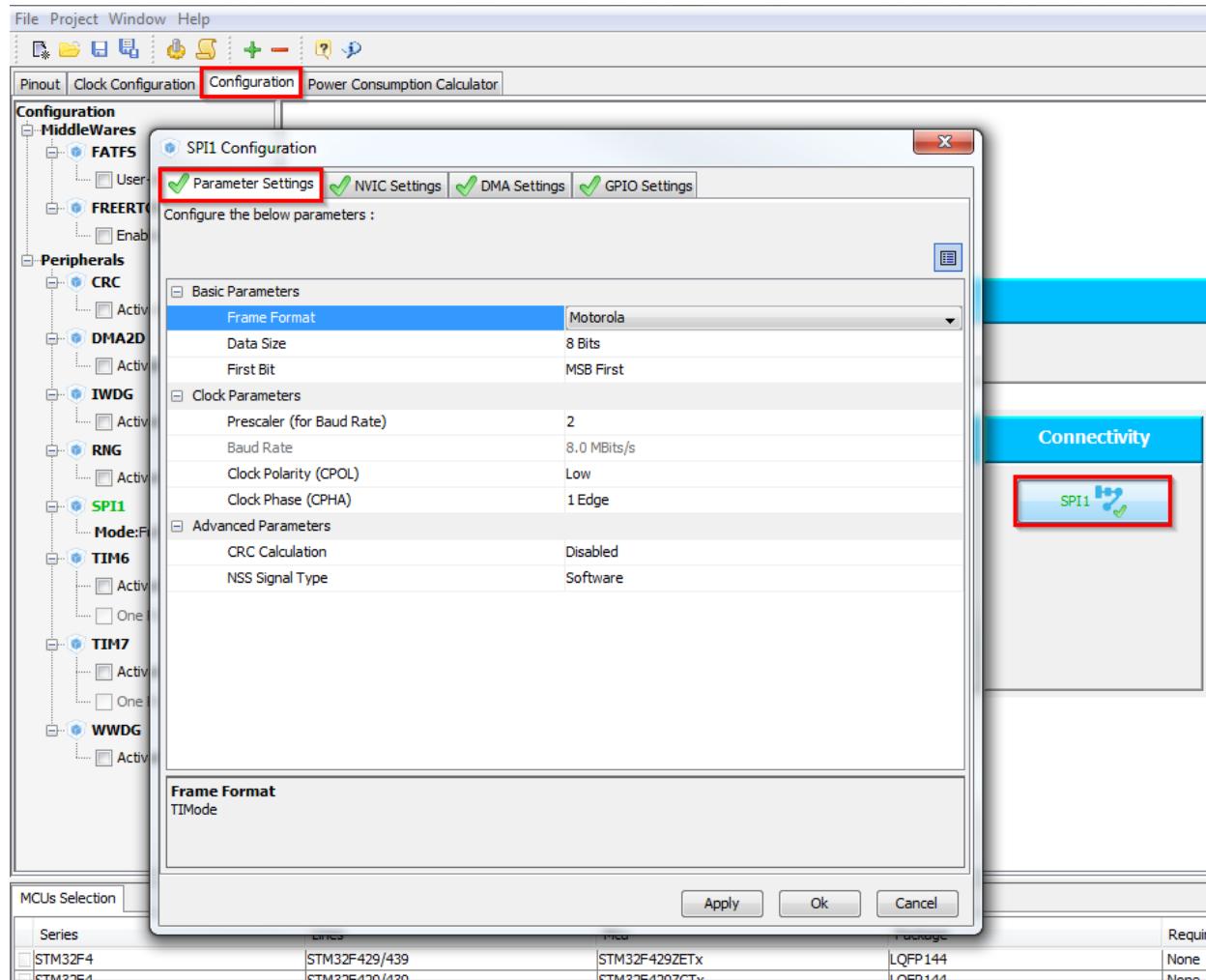
- Hardware preparation
  - Connect PA6 and PA7 together with jumper



- In order to run on maximum frequency, setup clock system
- Details in lab 0

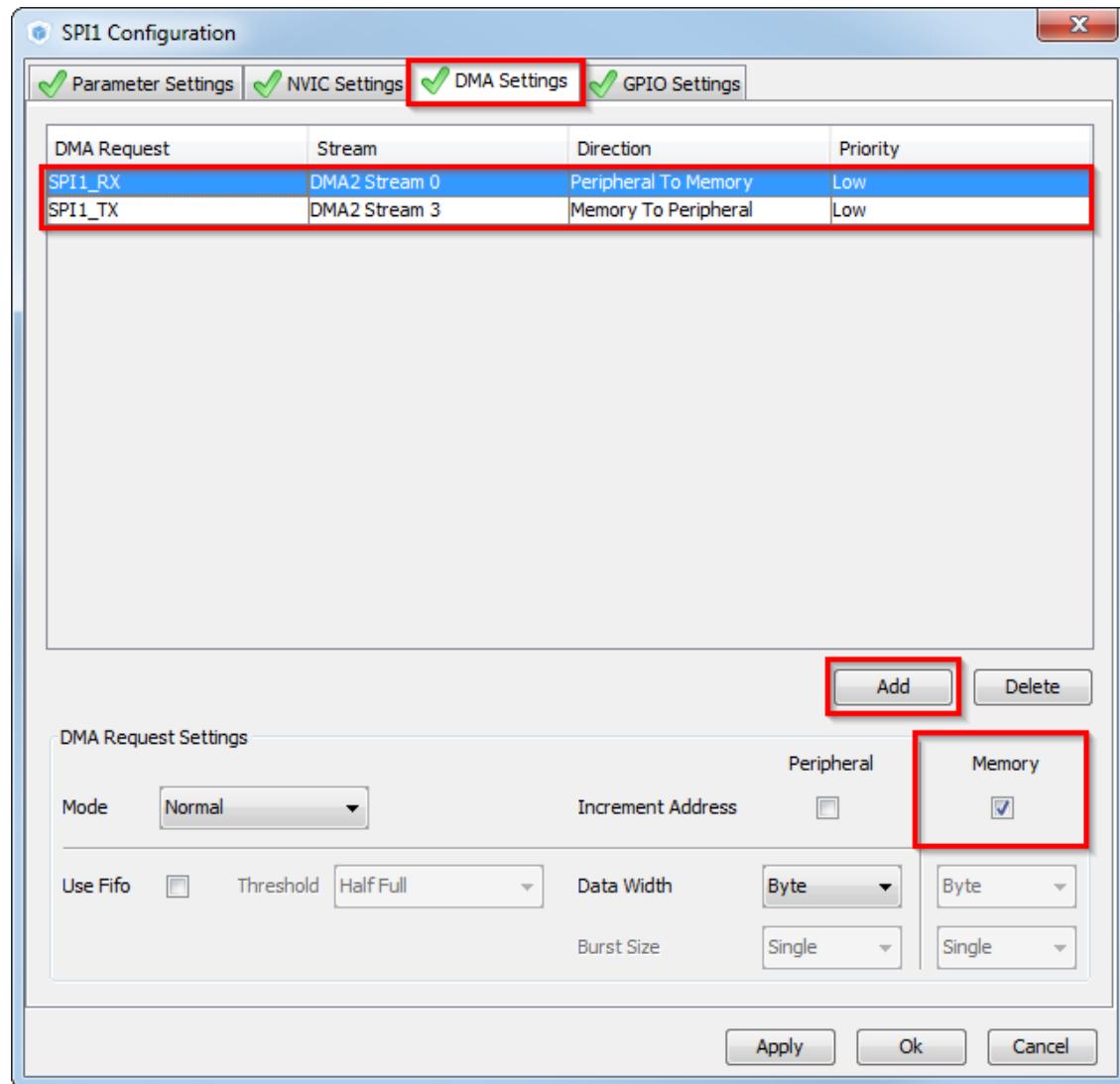


- CubeMX SPI configuration
  - Tab>Configuration>Connectivity>SPI1
  - Check the settings
  - Button OK



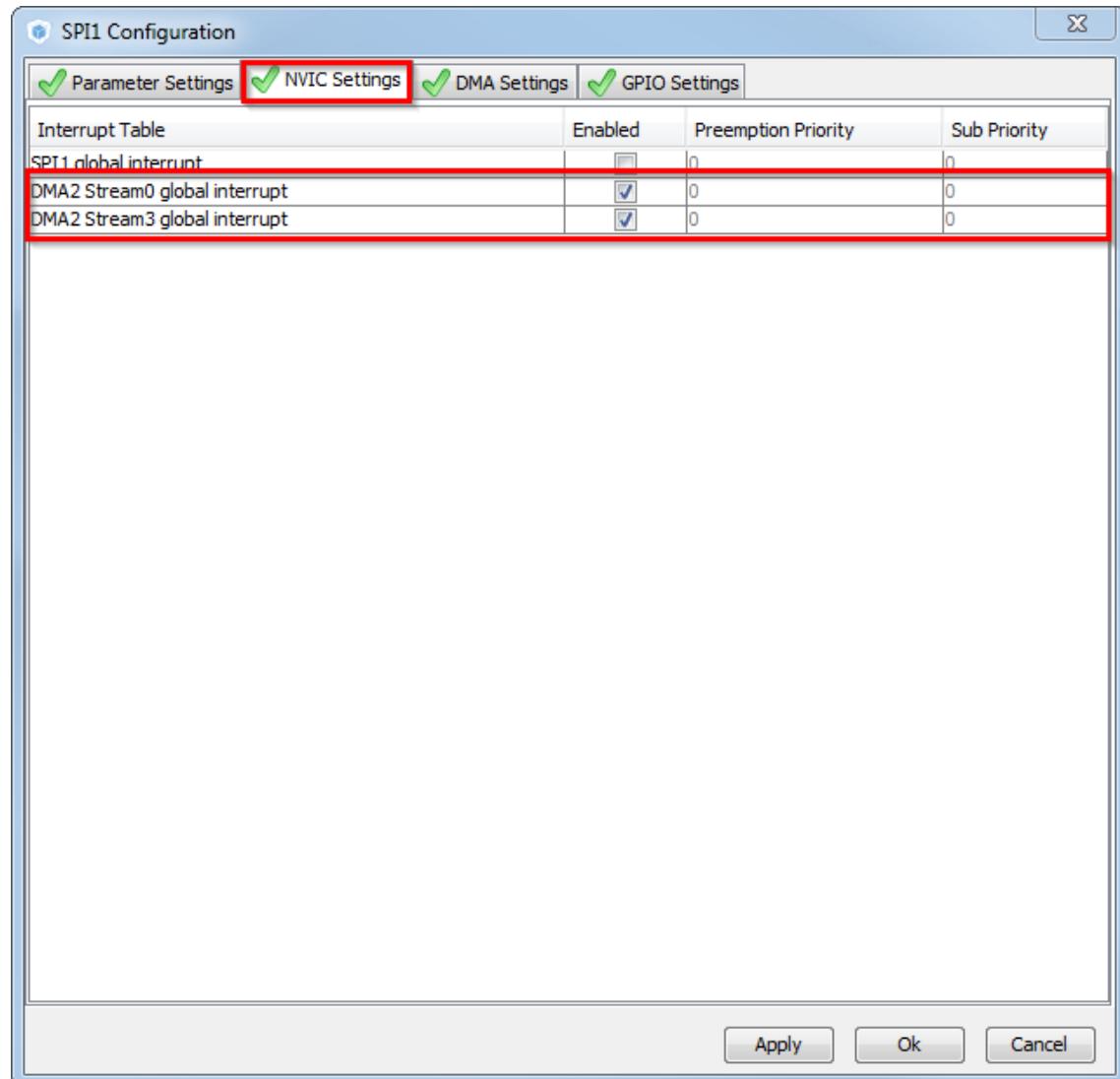
- CubeMX SPI configuration DMA settings

- TAB>DMA Settings
- Button ADD
- SPI1\_RX
- Memory increment
- Button ADD
- SPI1\_Tx
- Memory increment
- Button OK



- CubeMX SPI configuration NVIC settings

- TAB>NVIC Settings
- Enable DMA2 interrupts for SPI1
- Button OK

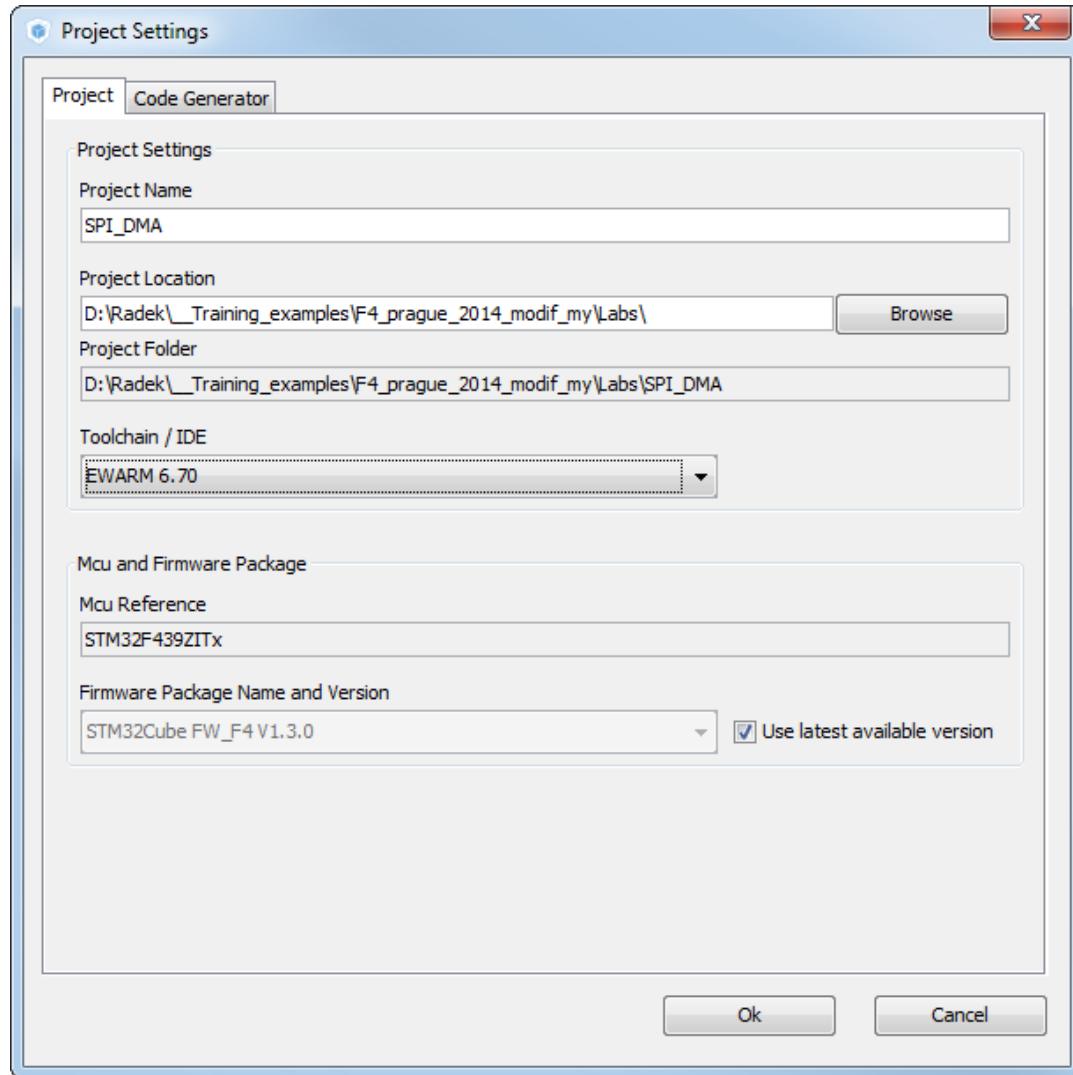


- Now we set the project details for generation

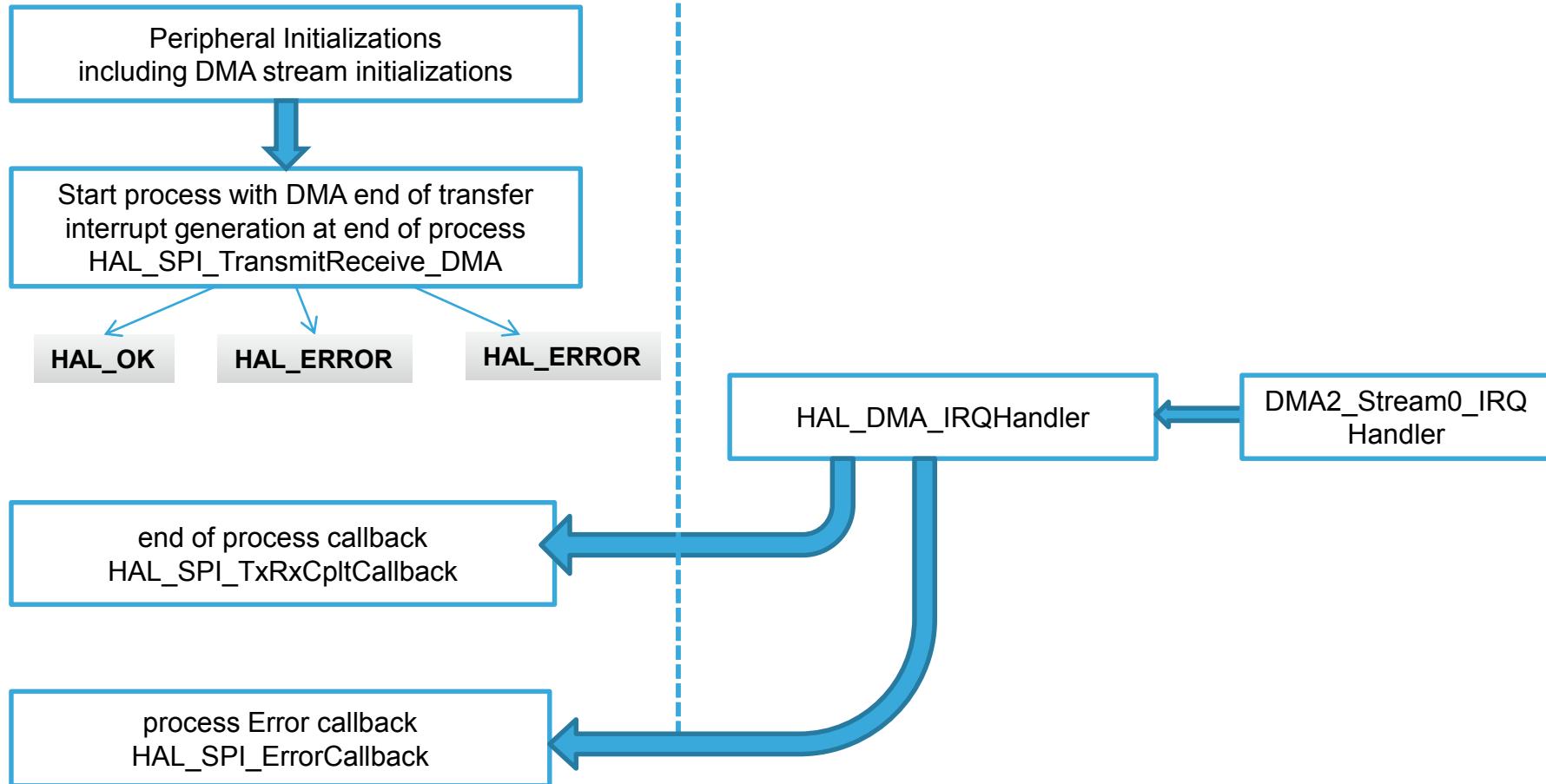
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



## HAL Library SPI with DMA TX RX flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For transmit use function
  - `HAL_SPI_TransmitReceive_DMA(SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size)`

- Buffer definition

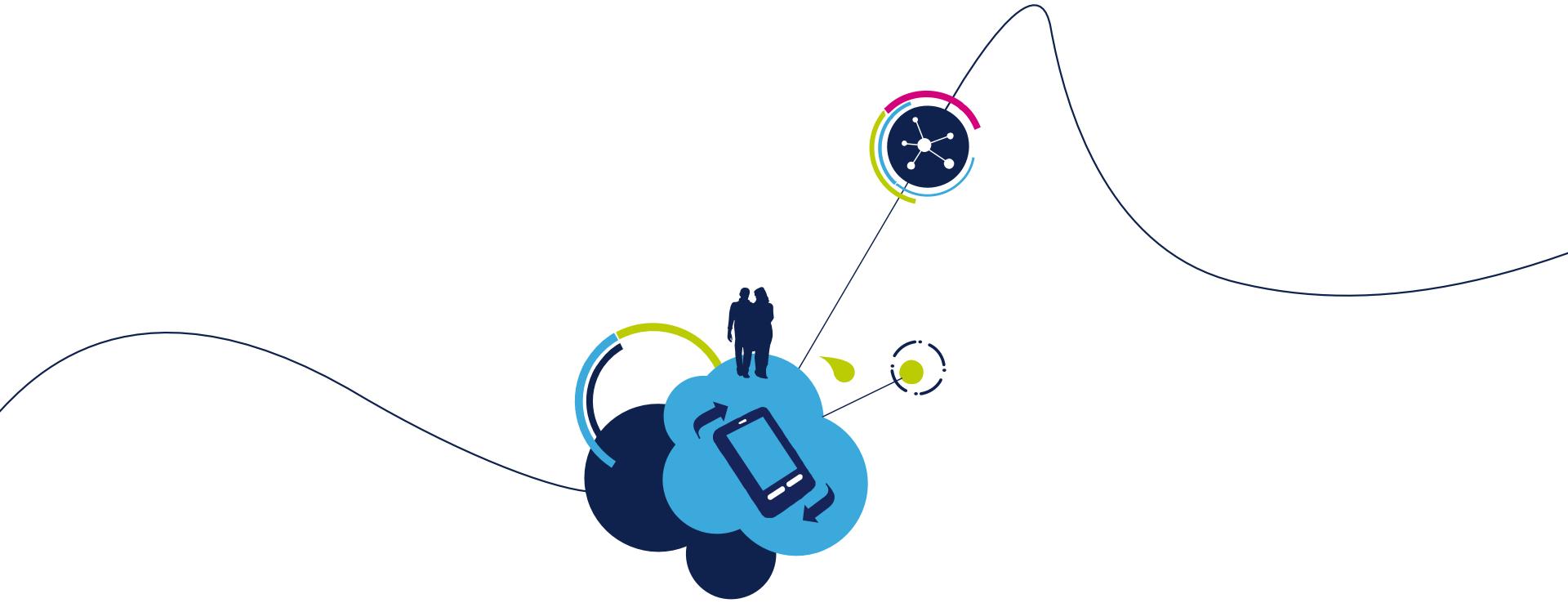
```
/* USER CODE BEGIN 0 */  
uint8_t tx_buff[]={0,1,2,3,4,5,6,7,8,9};  
uint8_t rx_buff[10];  
/* USER CODE END 0 */
```

- Sending and receiving methods

```
/* USER CODE BEGIN 2 */  
HAL_SPI_TransmitReceive_DMA(&hspi1,tx_buff,rx_buff,10);  
/* USER CODE END 2 */
```

- Complete callback check
  - We can put breakpoints on NOPs to watch if we send or receive complete buffer

```
/* USER CODE BEGIN 4 */  
void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi)  
{  
    __NOP();  
}  
/* USER CODE END 4 */
```



# TIM with interrupt lab 15

- Objective

- Learn how to setup TIM with Interrupt in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Indicate TIM interrupt with LED toggle

- Goal

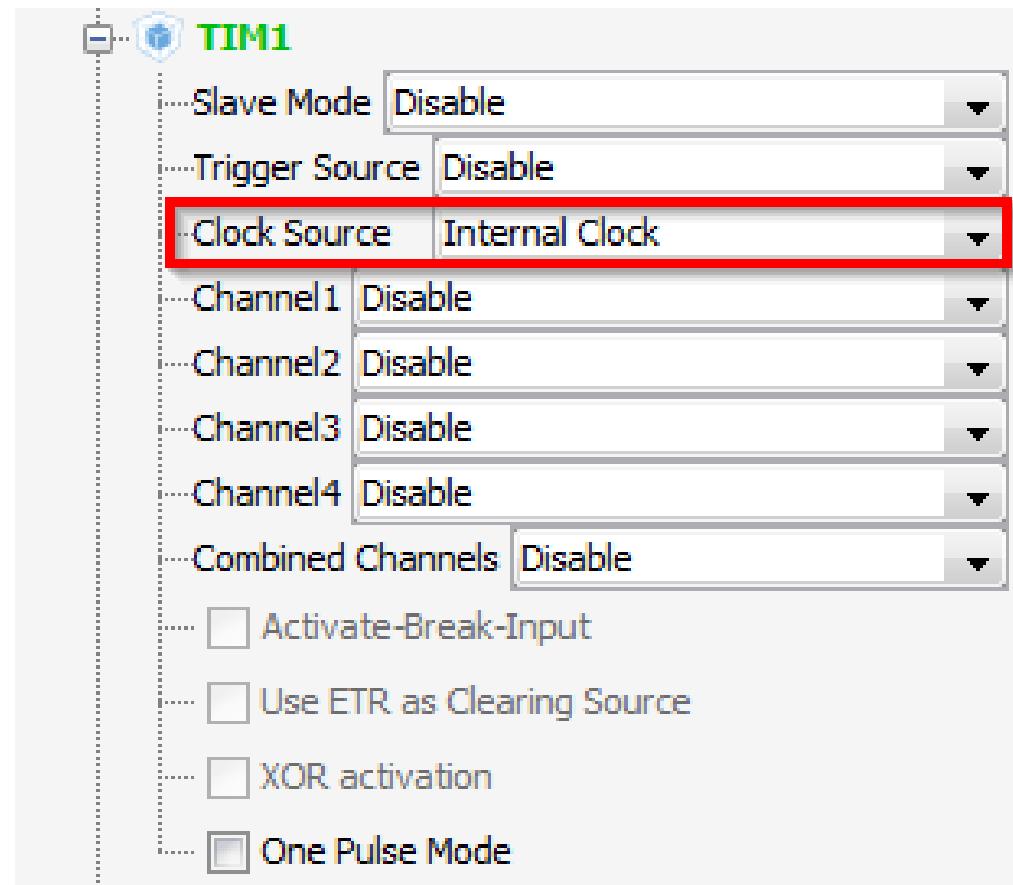
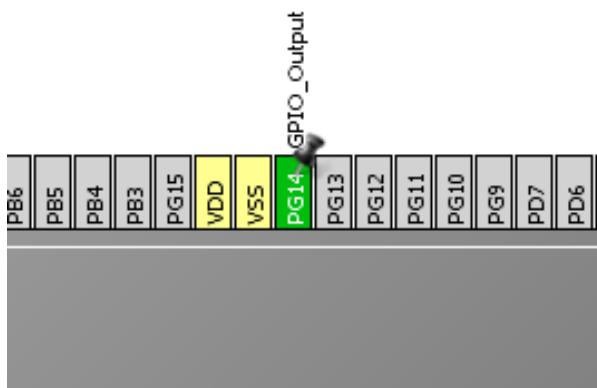
- Configure TIM in CubeMX and Generate Code
- Learn how start timer and handle interrupt
- Verify the correct functionality

- Create project in CubeMX

- Menu > File > New Project
- Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx

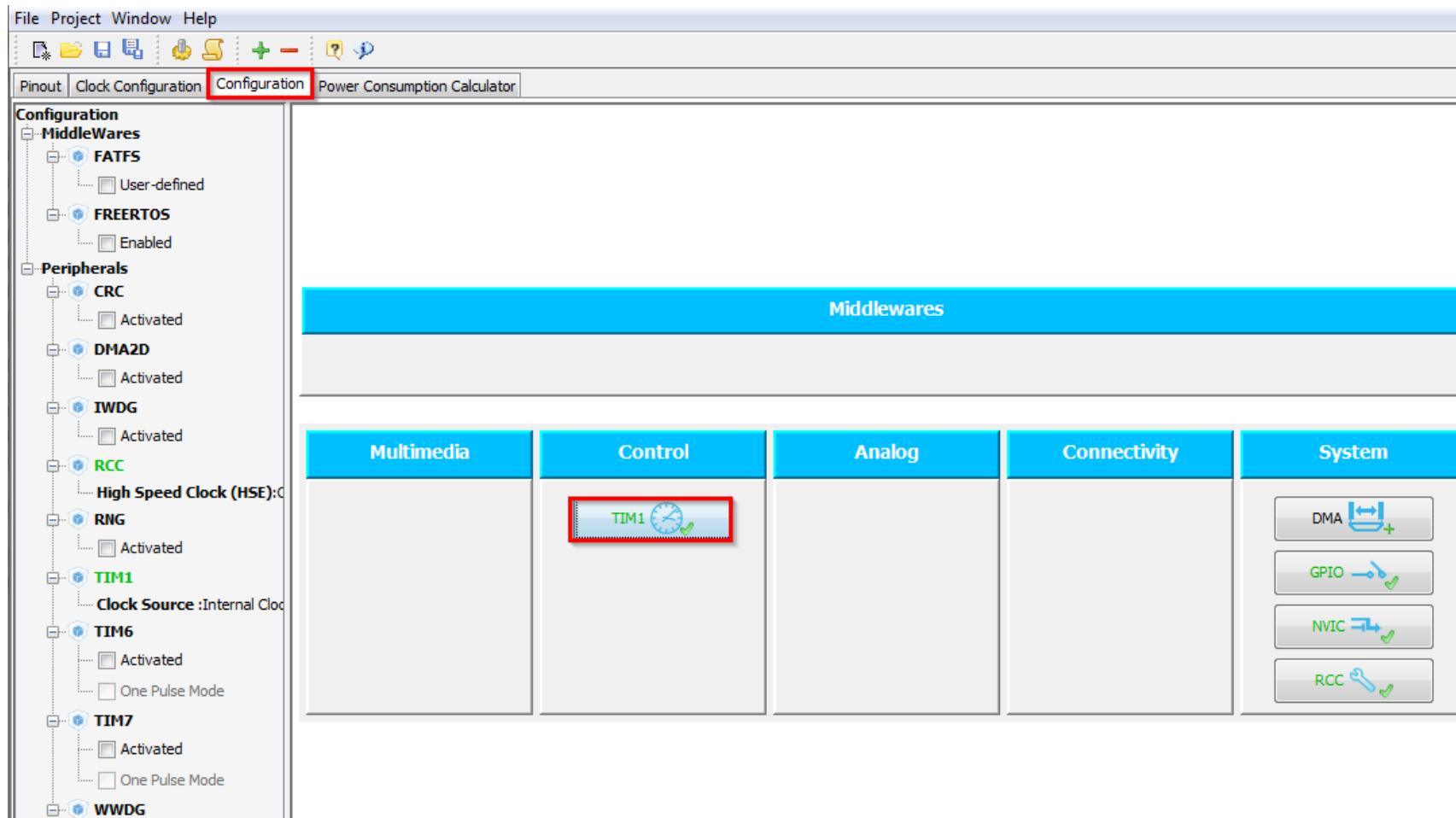
- CubeMX TIM selection

- Select TIM clock source Internal clock
- Enable GPIO for LED PG14

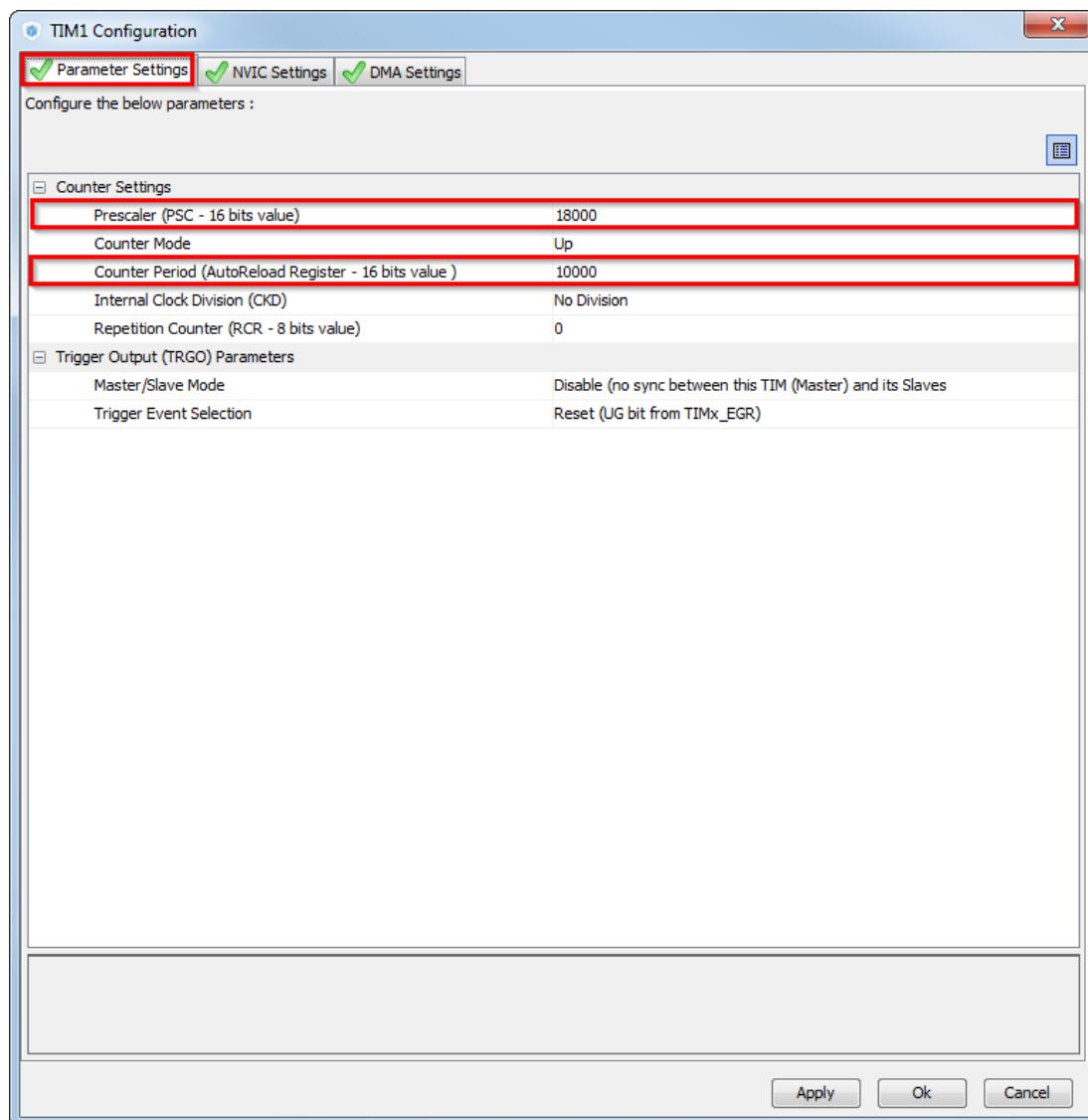


- CubeMX TIM configuration

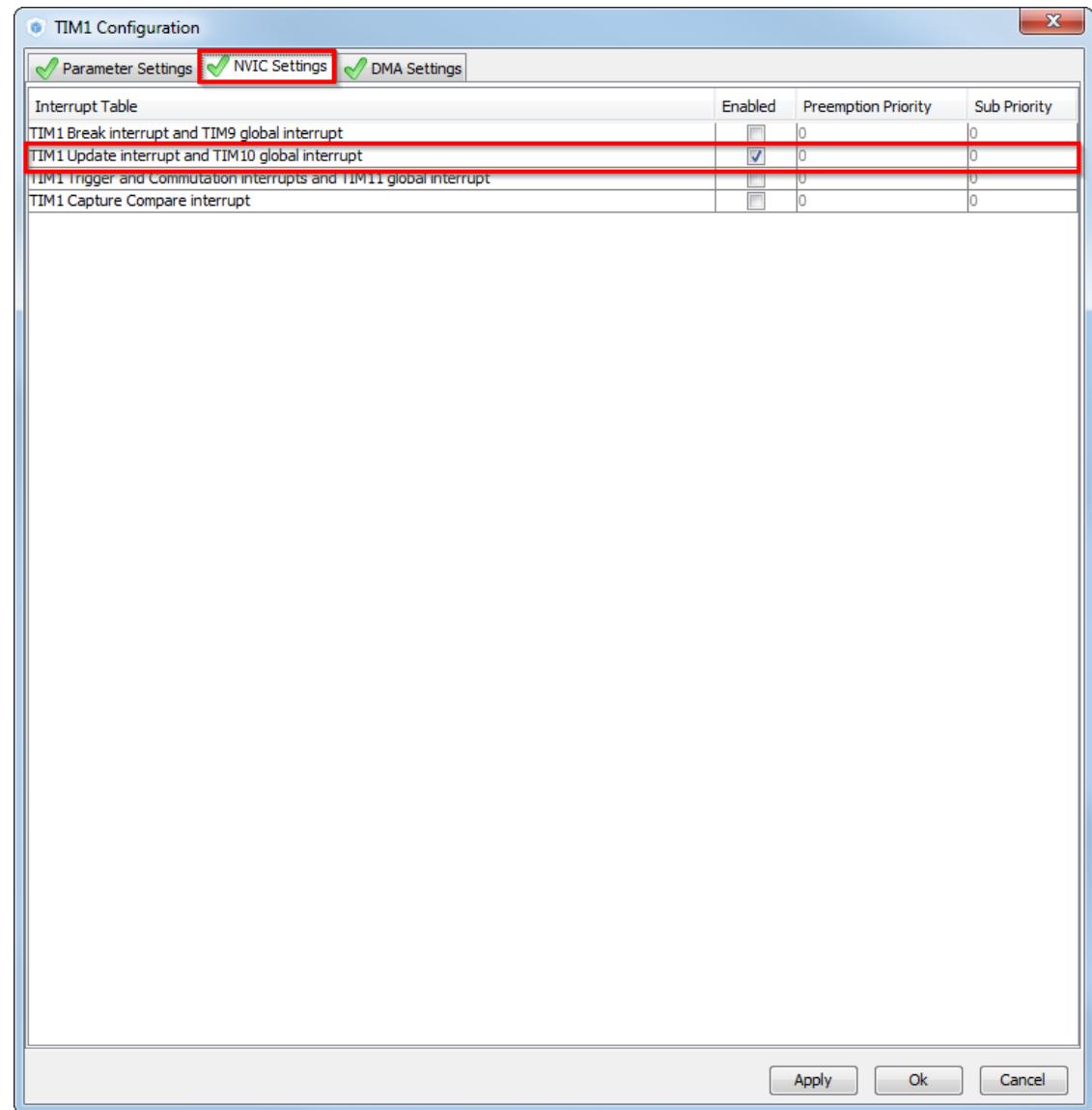
- Tab>Configuration>Control>TIM1
- Check the settings



- CubeMX TIM configuration
  - Tab>Parameter Settings
  - Prescaler to 18000
  - Counter period to 10000
  - Together with 180MHz TIMER1 clock we get period 1Hz



- CubeMX TIM configuration
  - Tab>NVIC Settings
  - Enable TIM1 Update interrupt
  - Button OK

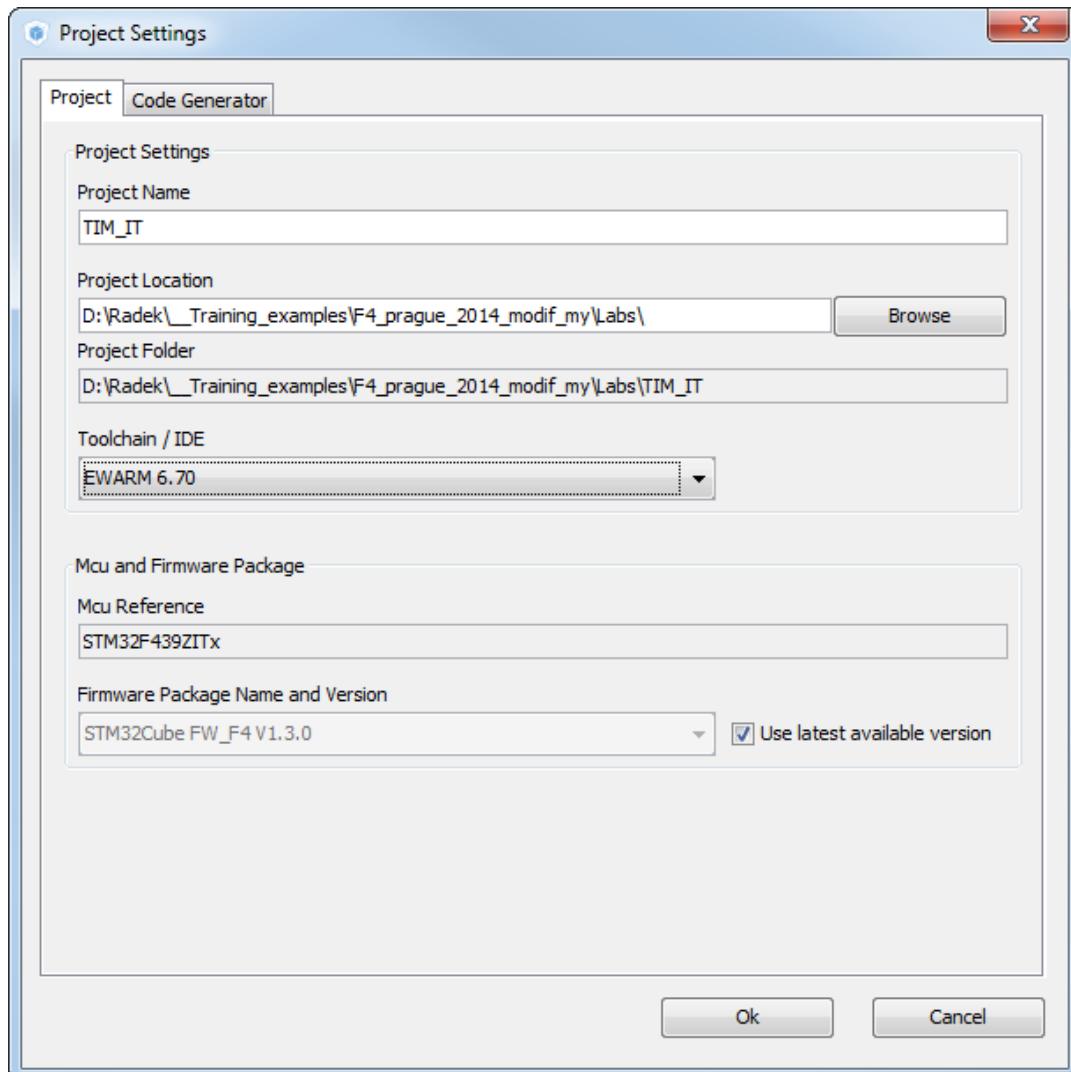


- Now we set the project details for generation

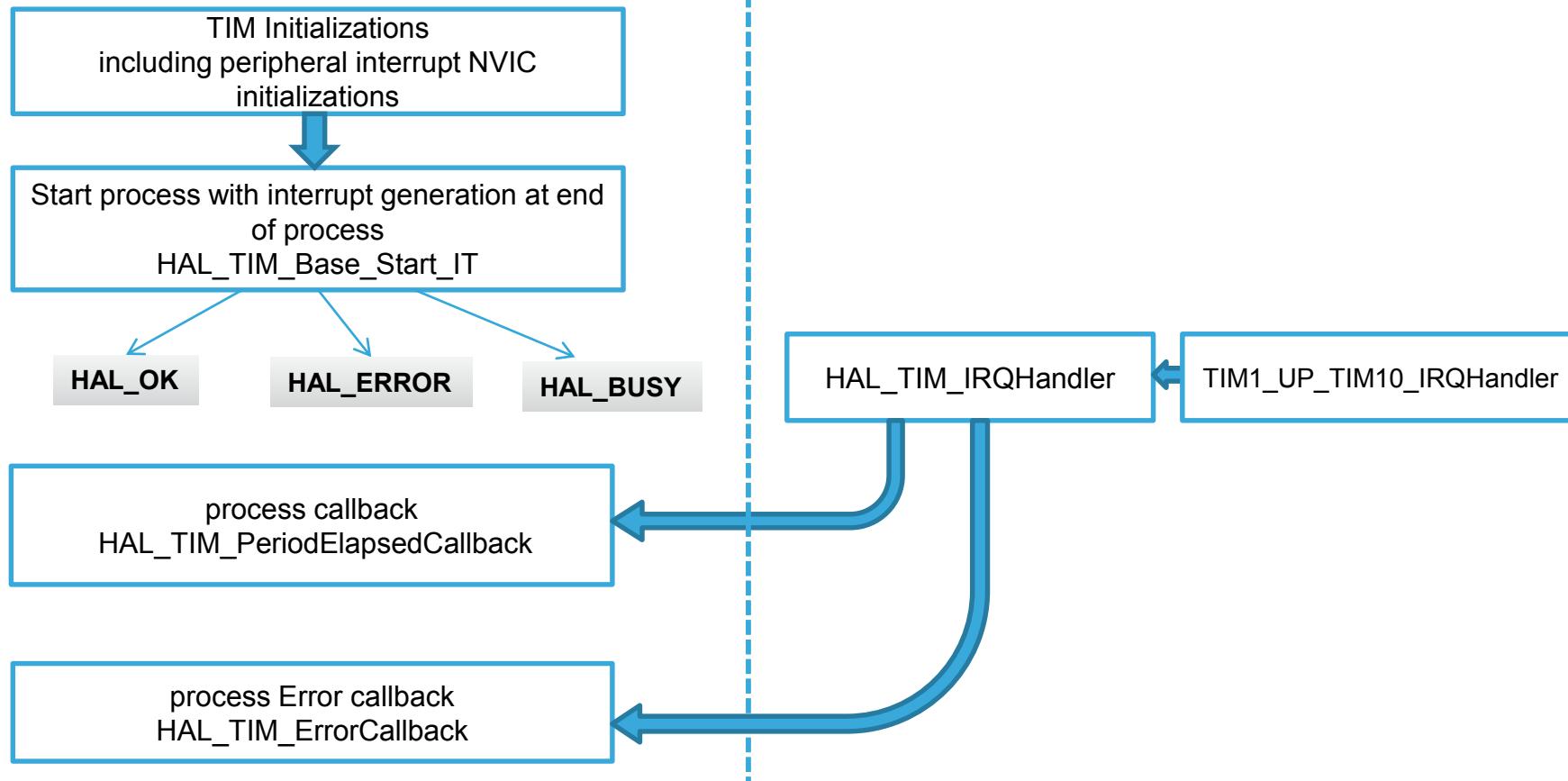
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



## HAL Library TIM with IT flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For TIM start use function
  - `HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)`
- TIM callback
  - `void TIM1_UP_TIM10_IRQHandler(void)`
- GPIO LED toggle
  - `HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

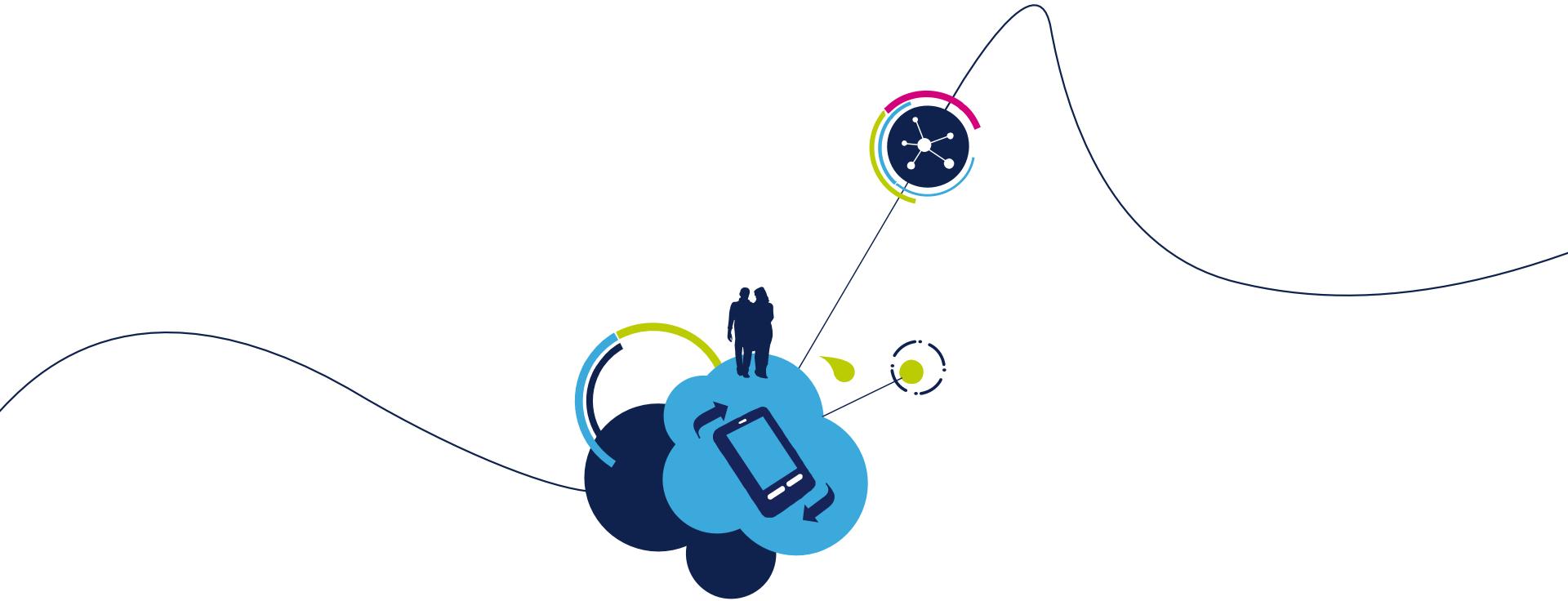
- Solution

- TIM start

```
/* USER CODE BEGIN 2 */  
HAL_TIM_Base_Start_IT(&htim1);  
/* USER CODE END 2 */
```

- Callback handling

```
/* USER CODE BEGIN 4 */  
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)  
{  
    HAL_GPIO_TogglePin(GPIOG,GPIO_PIN_14);  
}  
/* USER CODE END 4 */
```



# TIM with PWM output lab 16

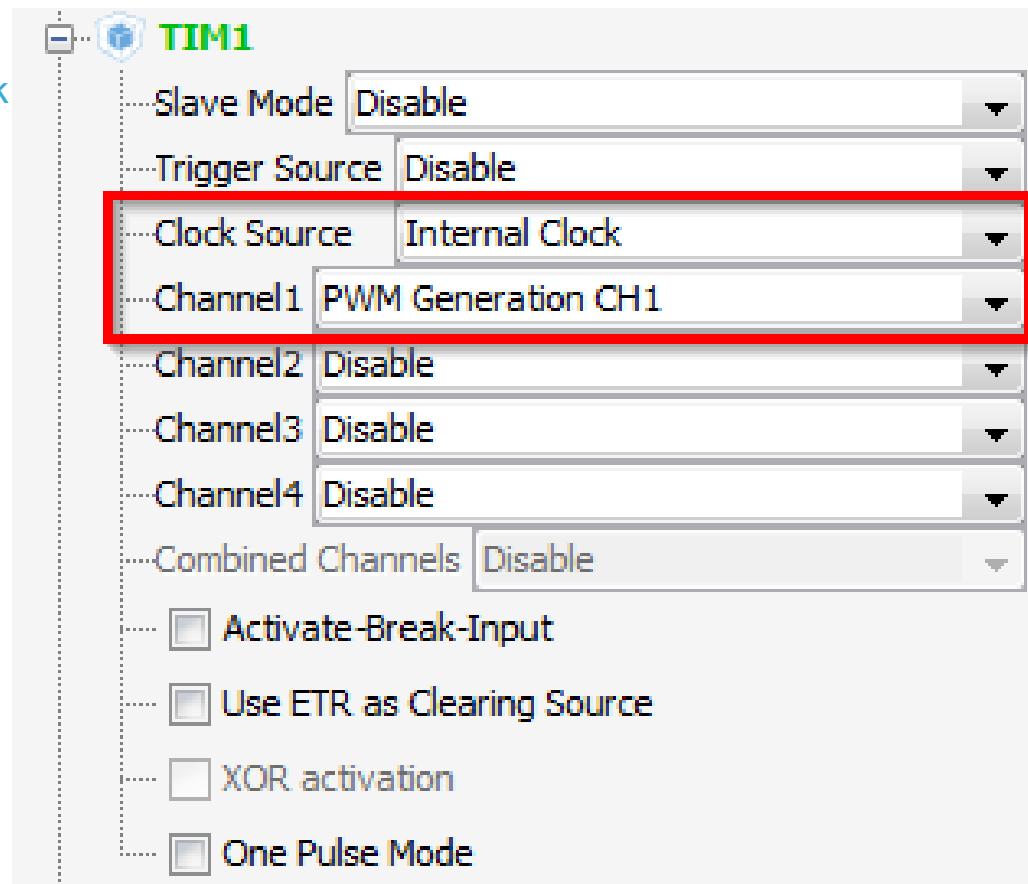
- Objective

- Learn how to setup TIM with PWM out in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Indicate TIM PWM on LED

- Goal

- Configure TIM in CubeMX and Generate Code
- Learn how start timer and set PWM out
- Verify the correct functionality with LED

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX TIM selection
  - Select TIM clock source - Internal clock
  - Set Channel1 to PWM generation

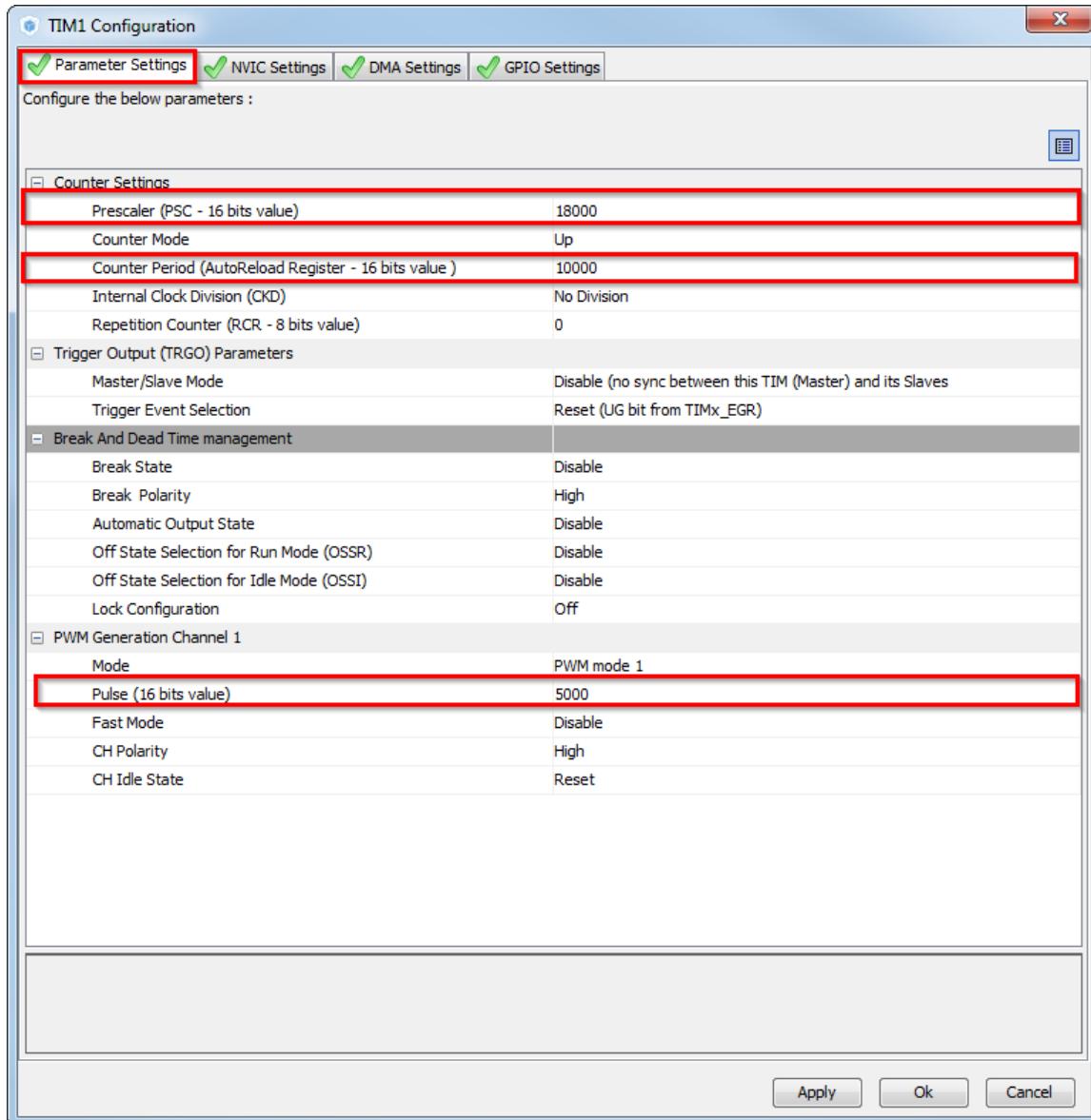


# 16

# Use TIM with PWM output

273

- CubeMX TIM configuration
  - TAB>Configuration  
>Control>TIM1
  - TAB>Parameter settings
  - Prescaler to 18000
  - Counter period to 10000
  - Together with 180MHz TIMER1 clock we get period 1Hz
  - PWM pulse to 5000 this give us 1Hz blinking frequency

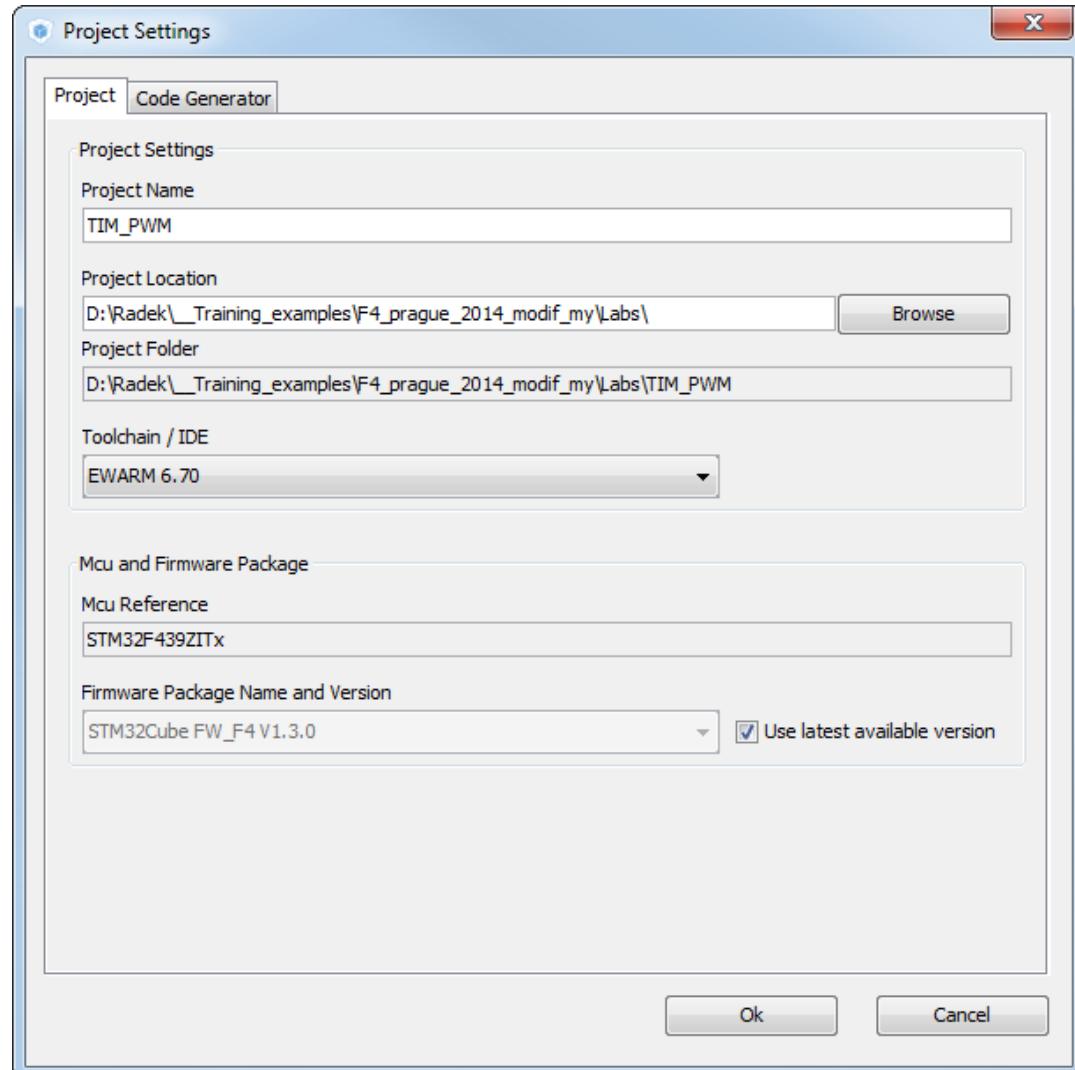


- Now we set the project details for generation

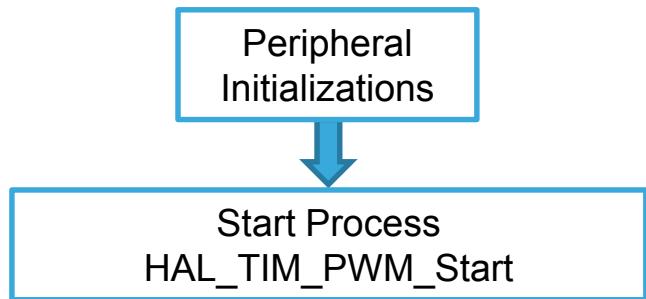
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



- Start process TIM with PWM(same for DMA, ADC)
  - Non blocking start process



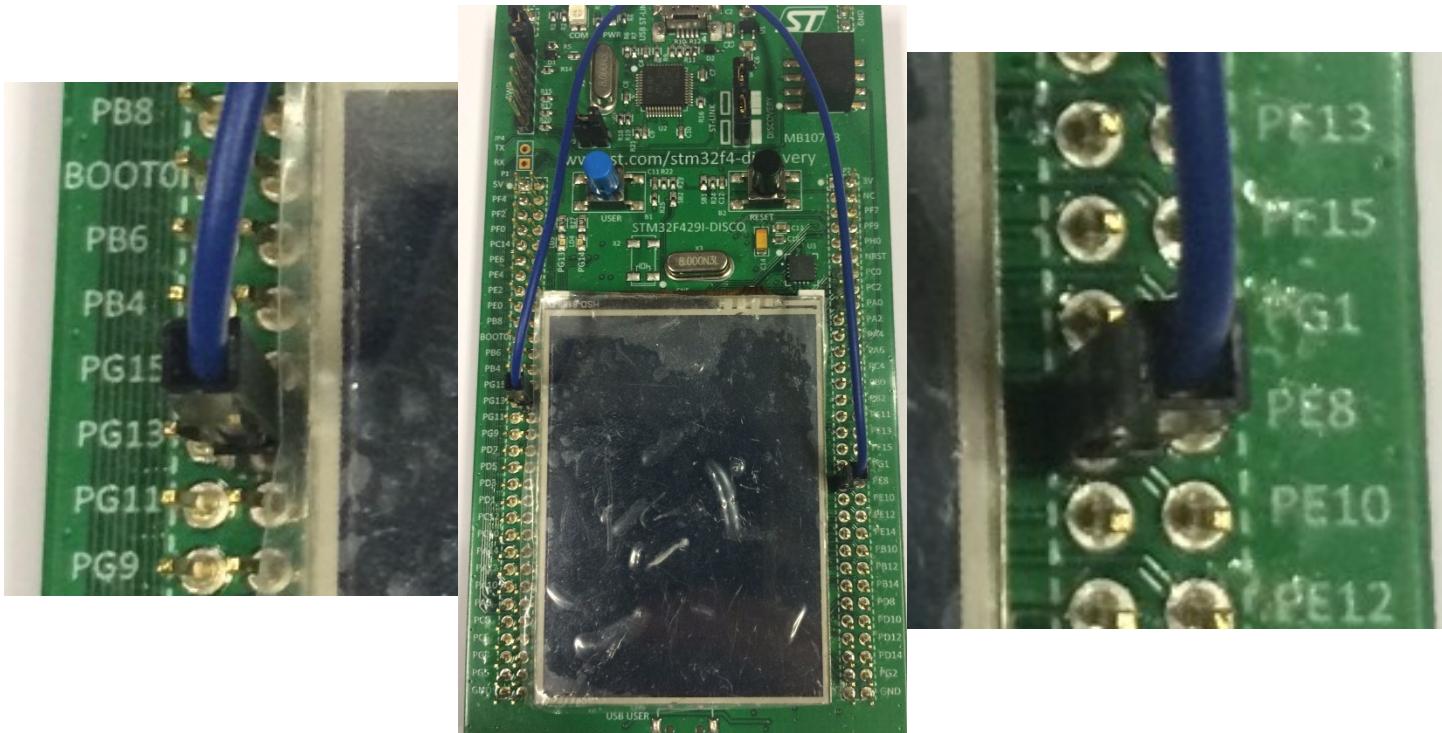
- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For TIM start use function
  - `HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel)`
- GPIO LED toggle
  - We wire the Channel1 PE9 with LED PG14

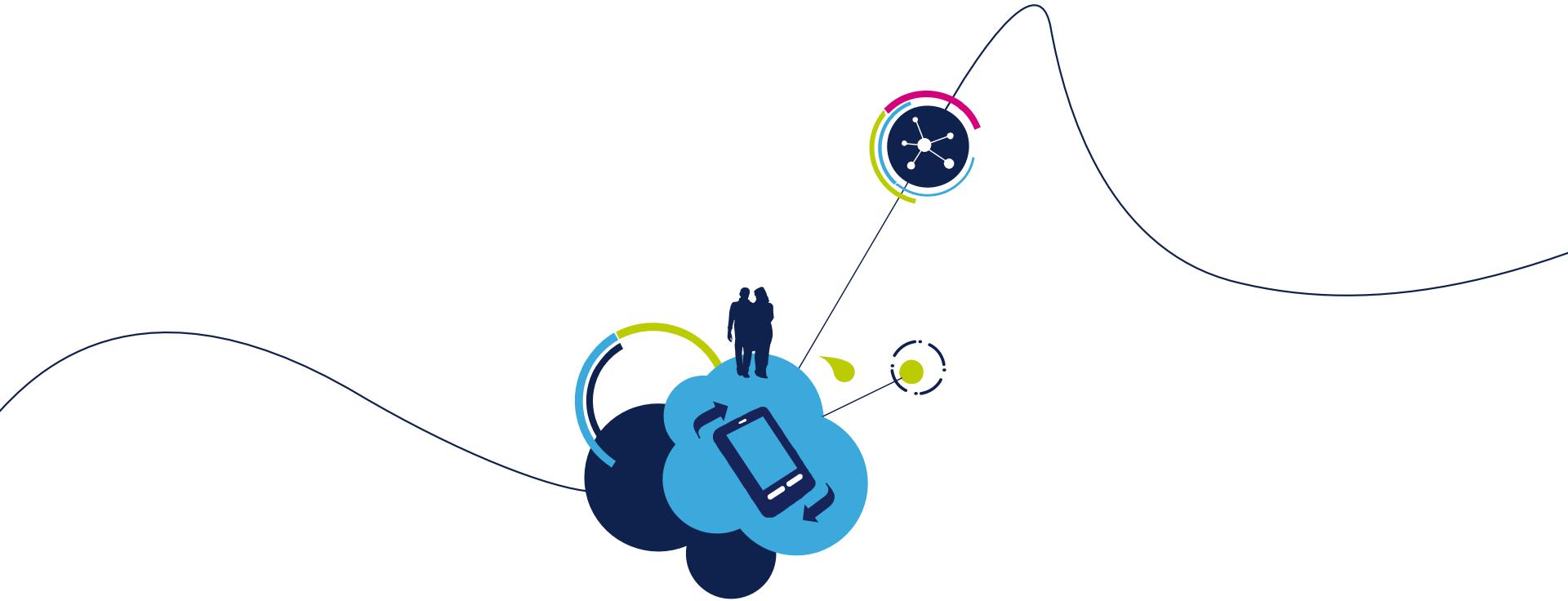
- Solution

- TIM PWM start

```
/* USER CODE BEGIN 2 */  
HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_1);  
/* USER CODE END 2 */
```

- TIM1 Channel 1 and LED connection





# TIM with DMA lab 17

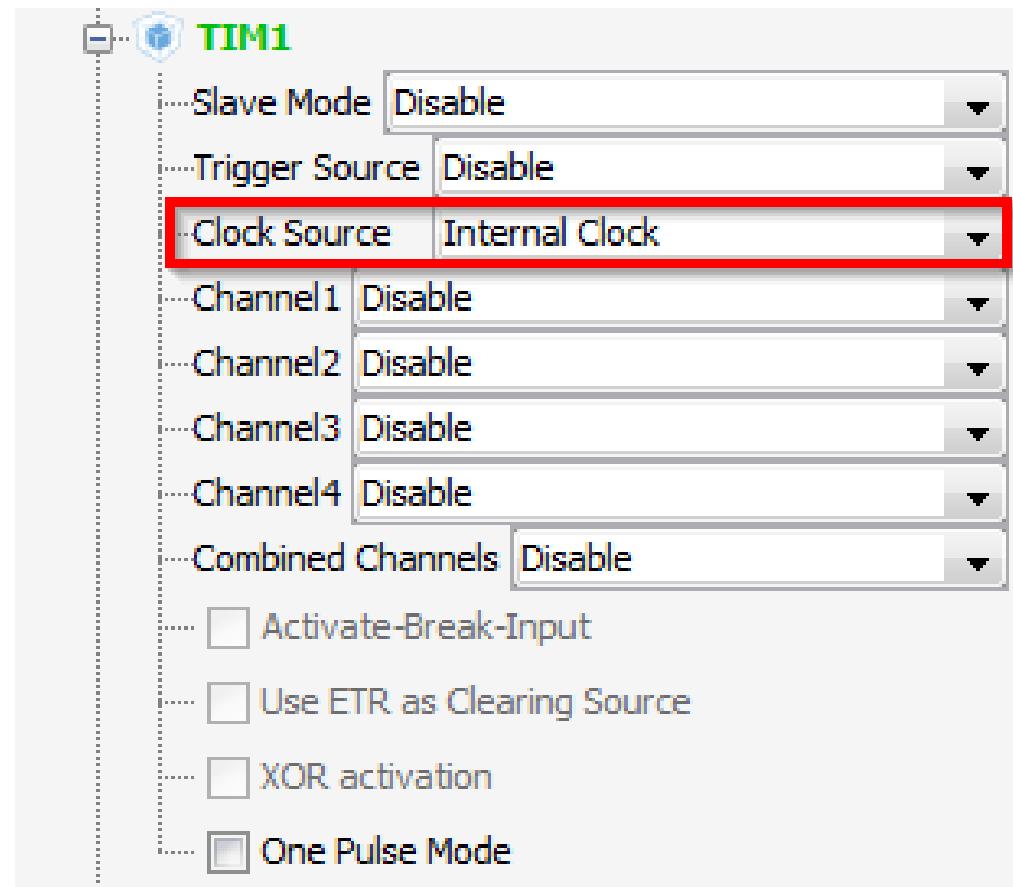
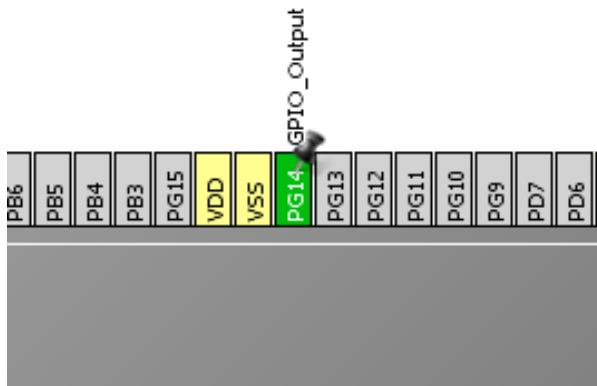
- Objective

- Learn how to setup TIM with DMA in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Indicate TIM DMA transfer with LED toggle

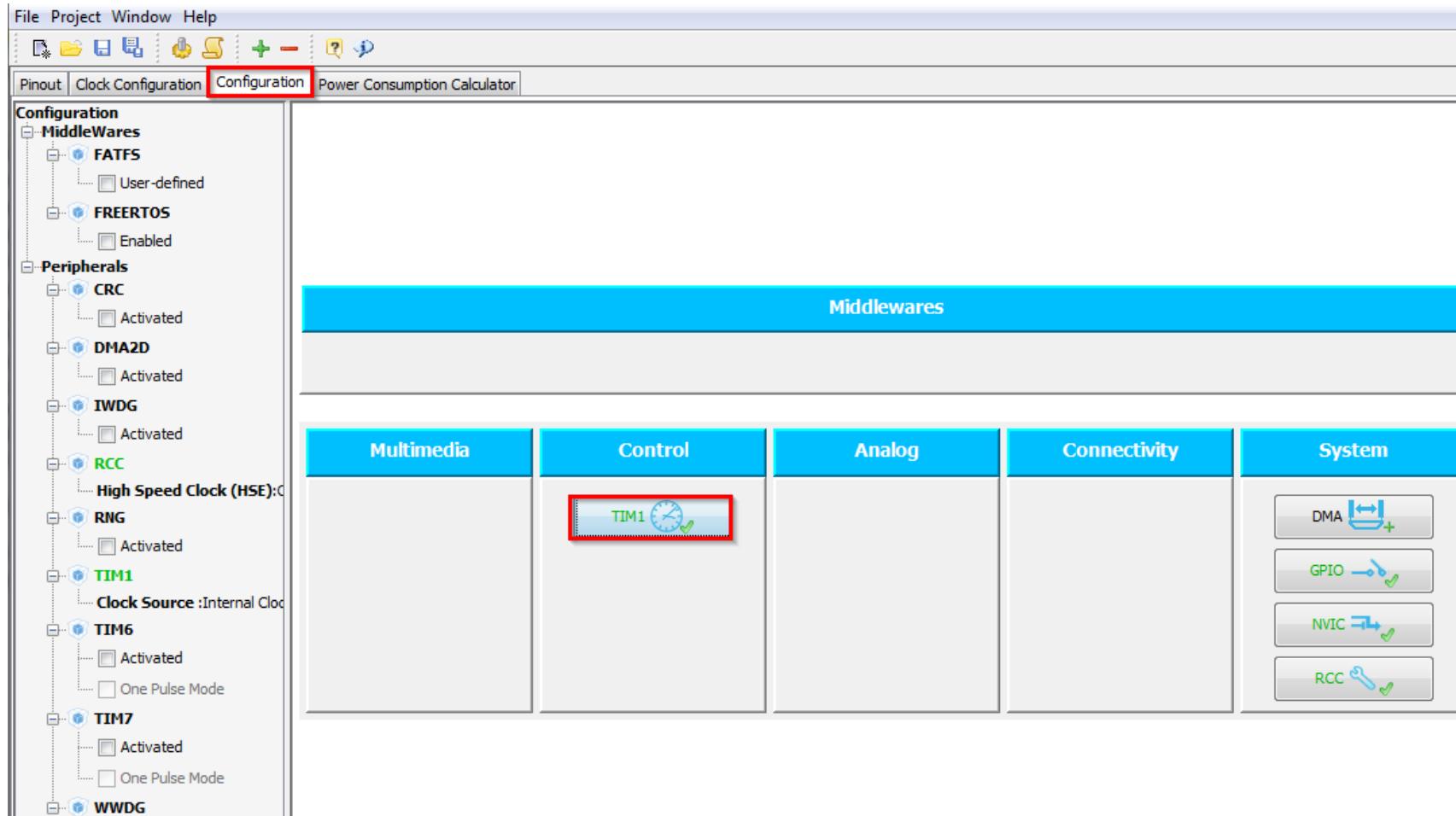
- Goal

- Configure TIM in CubeMX and Generate Code
- Learn how start timer and setup DMA
- Verify the correct functionality with DMA transfer into GPIO register

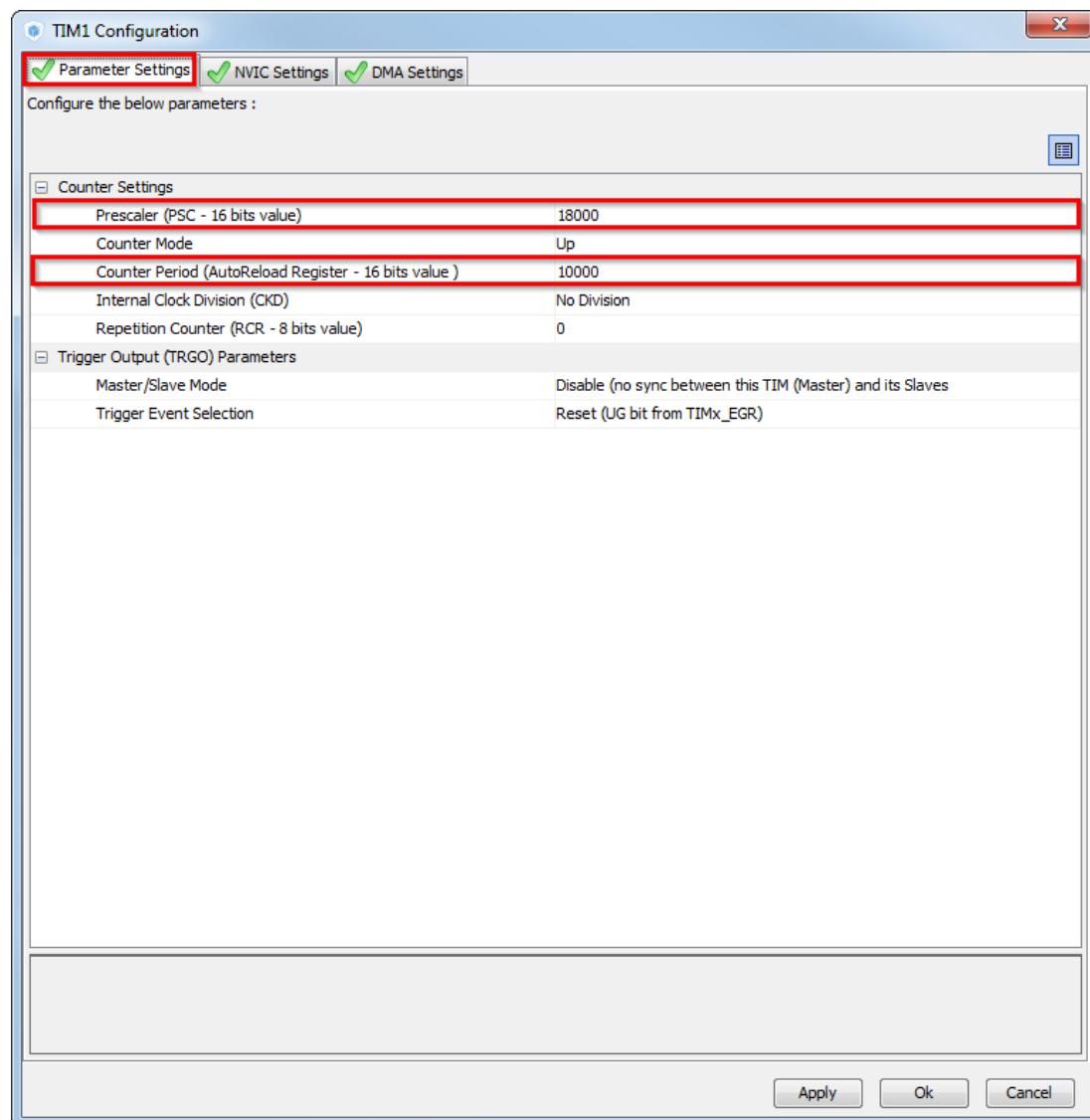
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX TIM selection
  - Select TIM clock source Internal clock
  - Enable GPIO for LED PG14



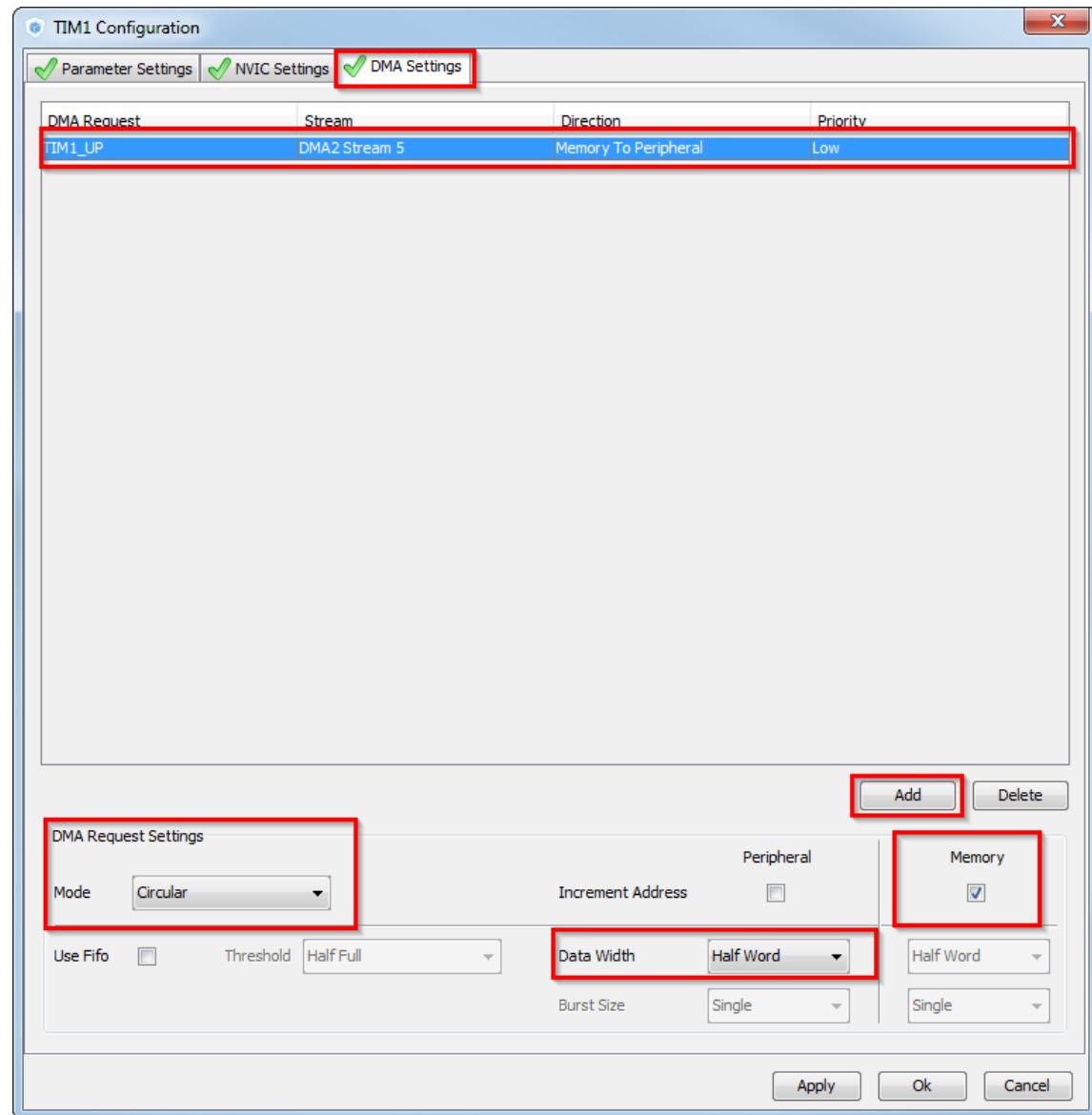
- CubeMX TIM configuration
  - Tab>Configuration>Control>TIM1
  - Check the settings



- CubeMX TIM configuration
  - Tab>Parameter Settings
  - Prescaler to 18000
  - Counter period to 10000
  - Together with 180MHz TIMER1 clock we get period 1Hz



- CubeMX TIM configuration
  - TAB>DMA Settings
  - Button ADD
  - Select TIM1\_UP DMA request
  - Memory to peripheral direction
  - Set Memory increment
  - Circular mode
  - Half word data width
  - Button OK

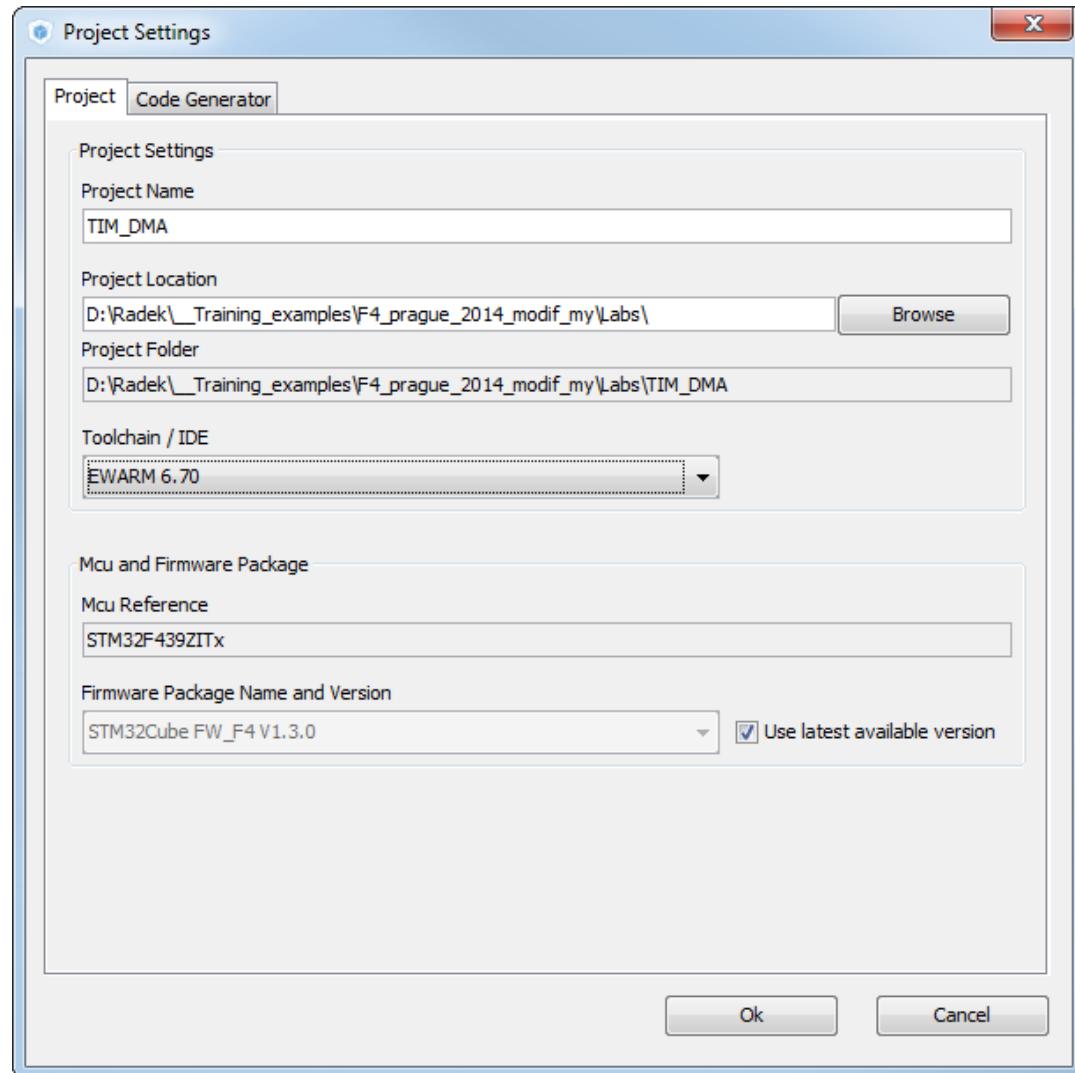


- Now we set the project details for generation

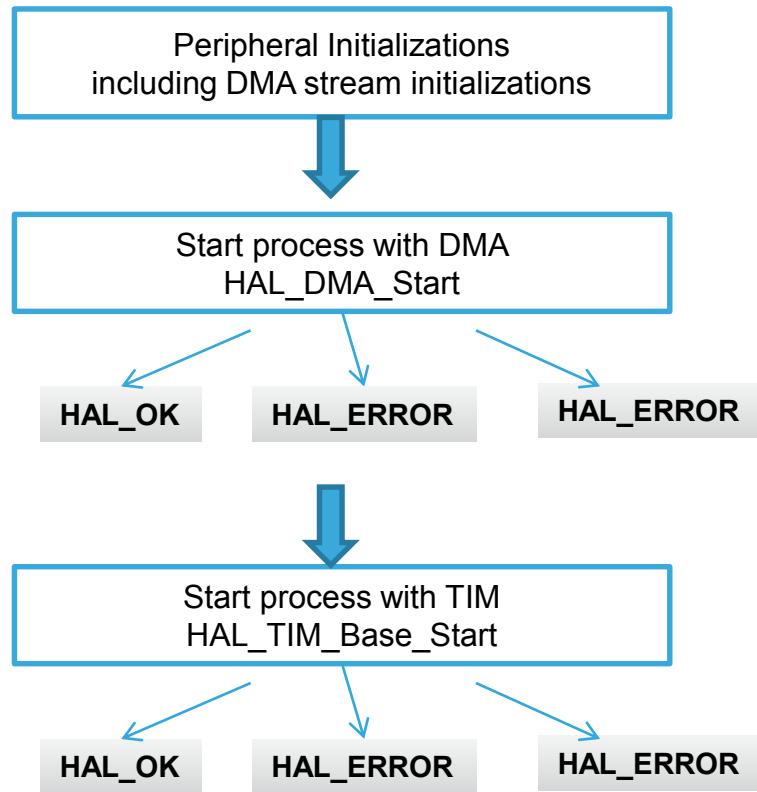
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



## HAL Library TIM with DMA flow



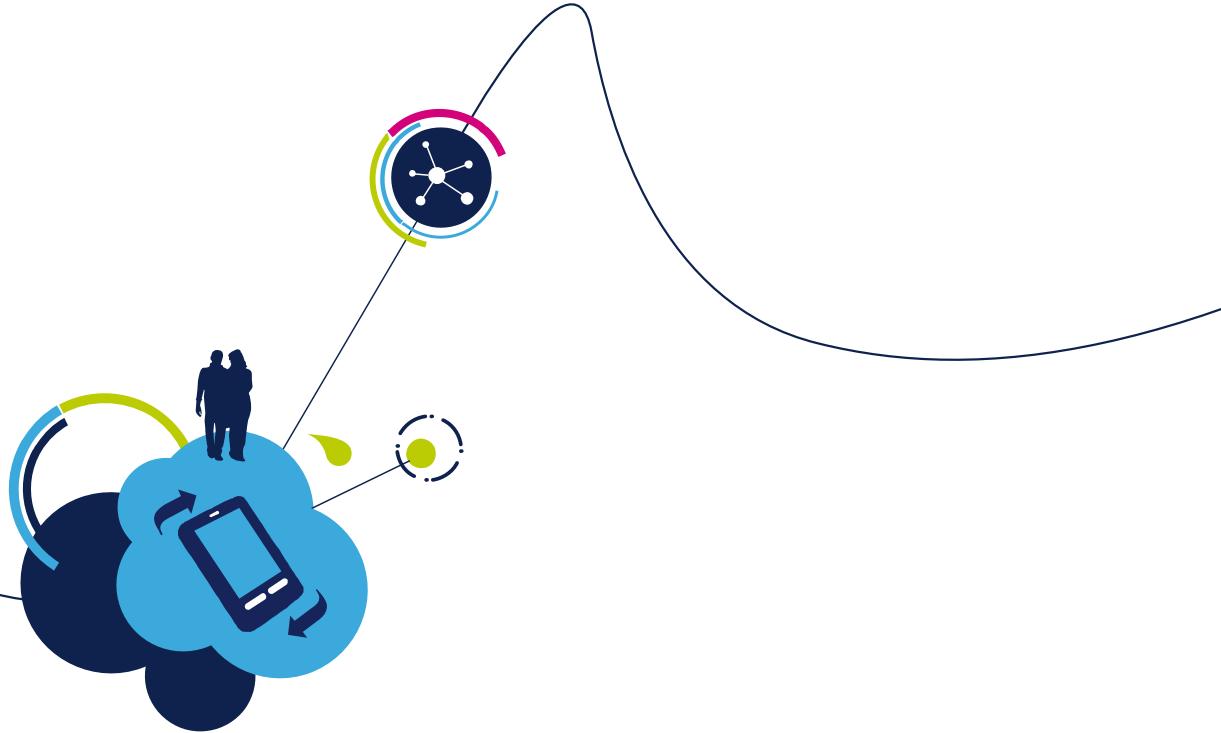
- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For TIM start use function
  - HAL\_TIM\_Base\_Start\_DMA(TIM\_HandleTypeDef \*htim, uint32\_t \*pData, uint16\_t Length)
- TIM1 trigger DMA transfer
  - \_\_HAL\_TIM\_ENABLE\_DMA
- DMA start function
  - HAL\_DMA\_Start(DMA\_HandleTypeDef \*hdma, uint32\_t SrcAddress, uint32\_t DstAddress, uint32\_t DataLength)
- GPIO LED register address
  - (uint32\_t)(&GPIOG->ODR)

- Variable data definition

```
/* USER CODE BEGIN PV */  
uint16_t data[]={GPIO_PIN_14,0x0000};  
/* USER CODE END PV */
```

- DMA and TIM start

```
/* 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 */
```



# TIM as counter lab 18

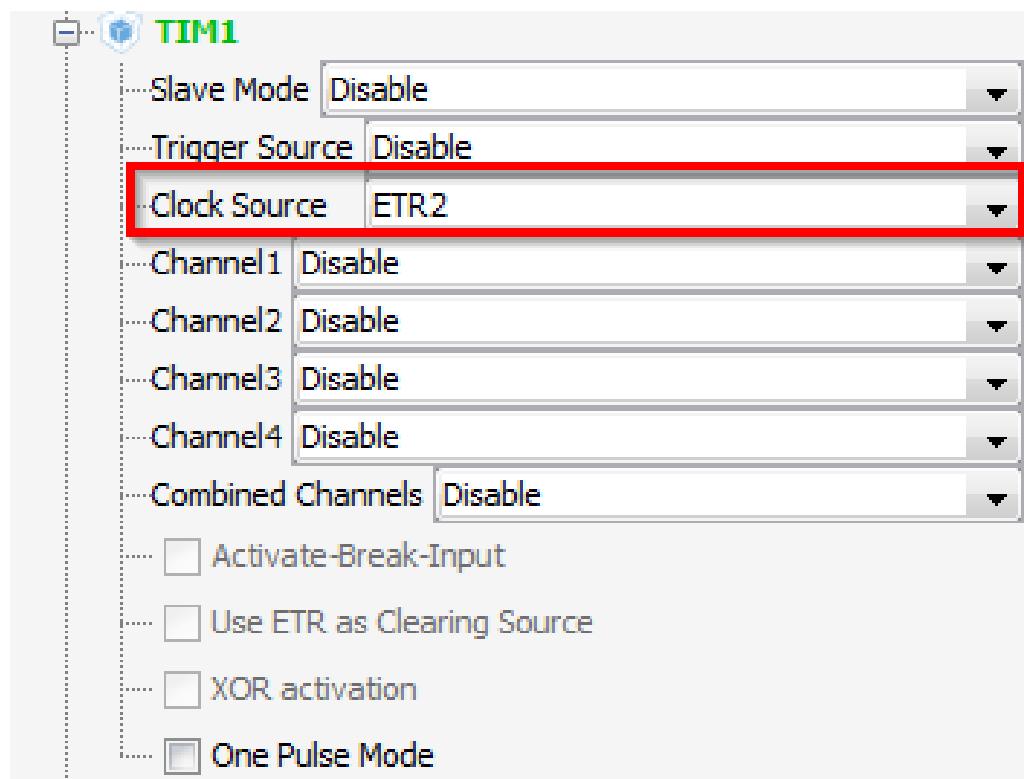
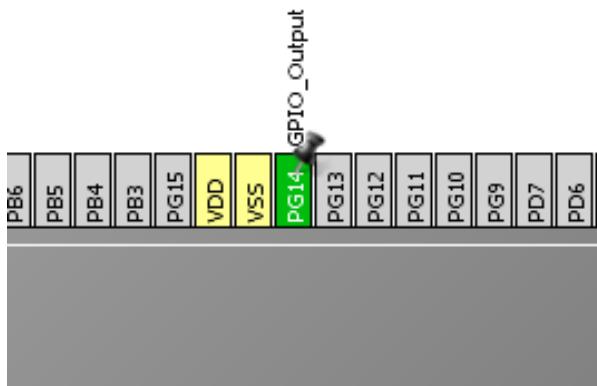
- Objective

- Learn how to setup TIM as counter in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Indicate TIM count 5 button press with LED toggle

- Goal

- Configure TIM as counter in CubeMX and Generate Code
- Learn how start timer and handle interrupt
- Verify the correct functionality with LED toggle after 5 button press

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX TIM selection
  - Select TIM clock source ETR2
  - Enable GPIO for LED PG14

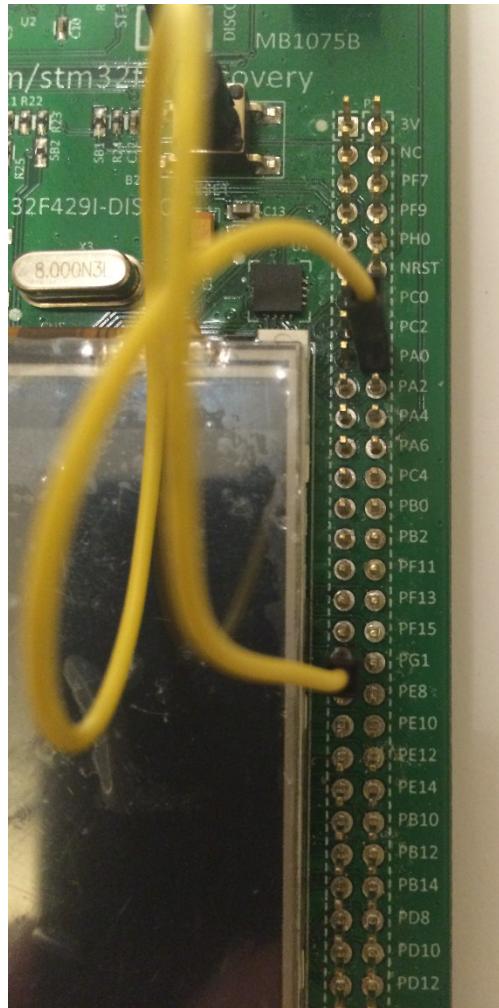


# 18

# Use TIM as pulse counter

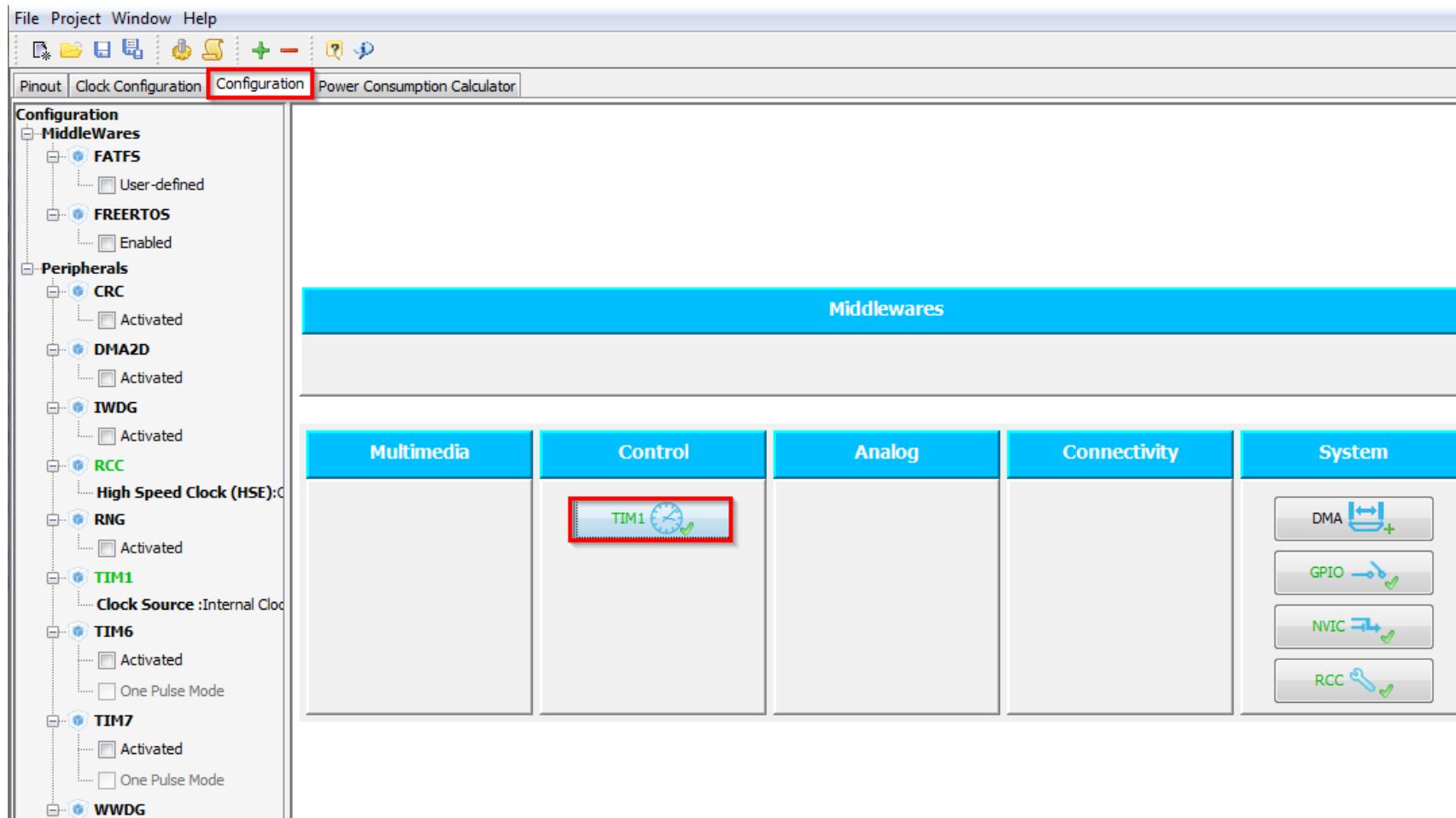
291

- Hard ware setting
  - Connect Button PA0 and ETR pin PE7 with wire together

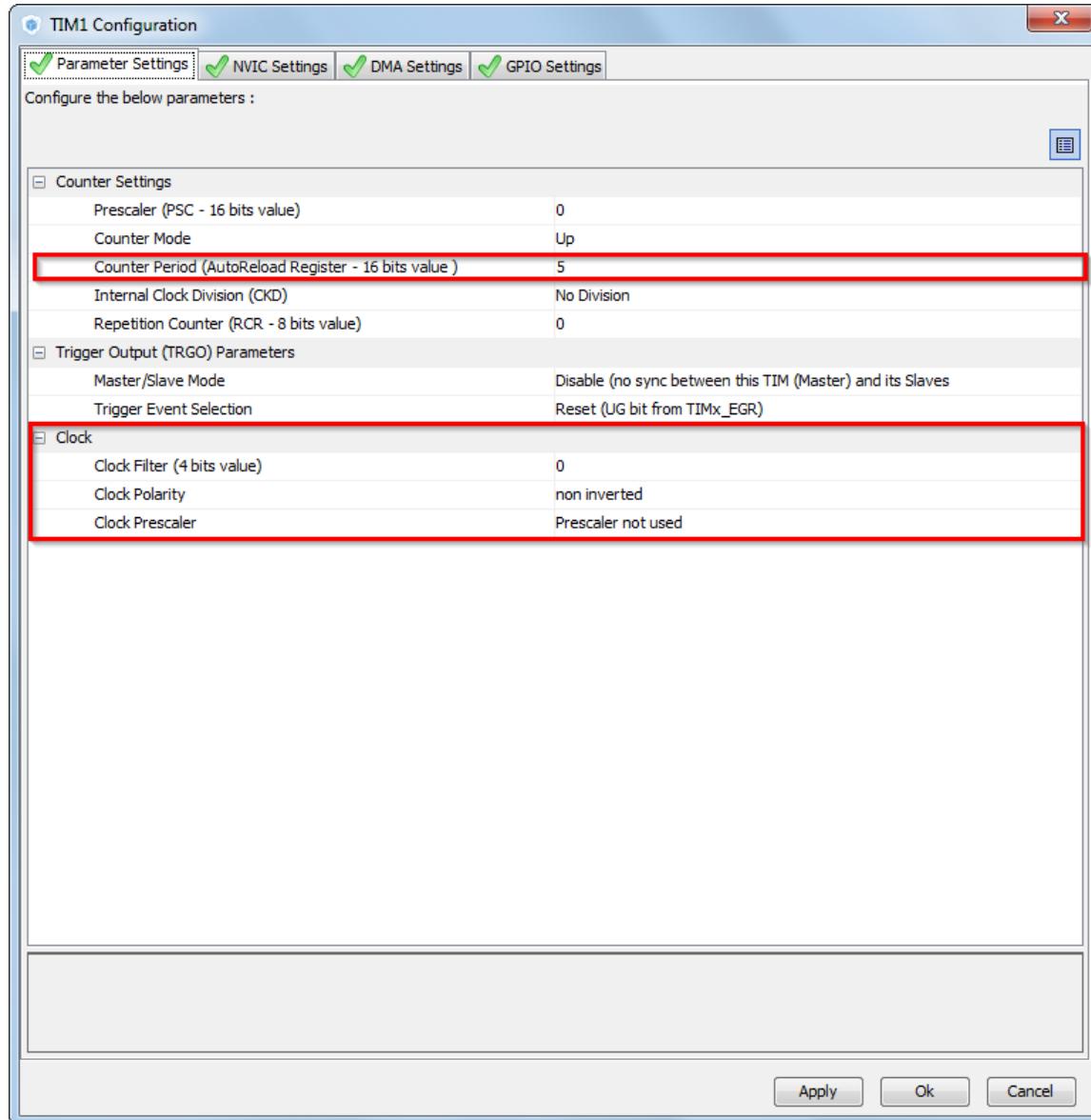


- CubeMX TIM configuration

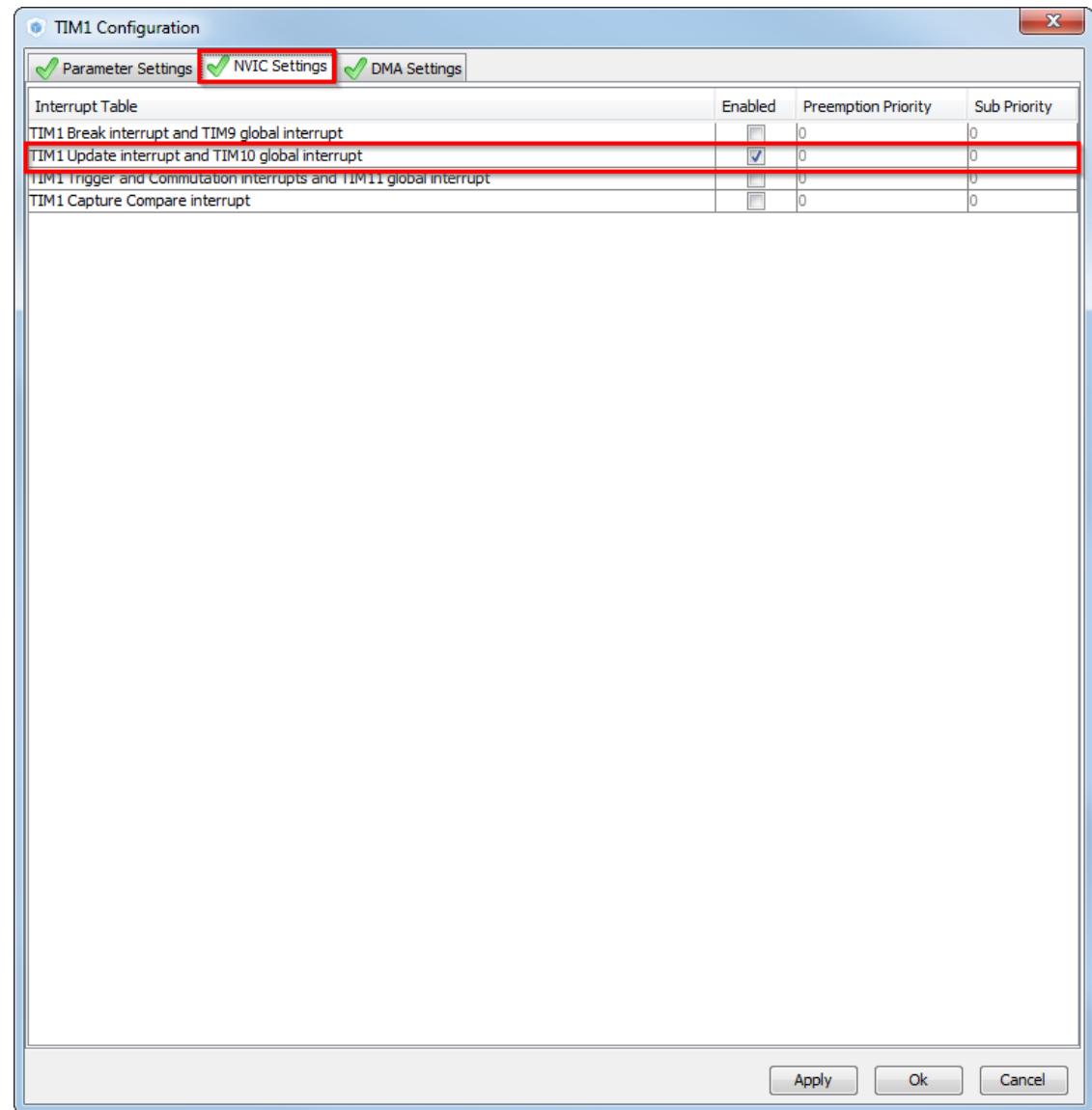
- Tab>Configuration>Control>TIM1
- Check the settings



- CubeMX TIM configuration
  - Tab>Parameter Settings
  - Counter set to 5, 5 button press
  - Clock set the ETR pin filter and edge reaction



- CubeMX TIM configuration
  - Tab>NVIC Settings
  - Enable TIM1 Update interrupt
  - Button OK

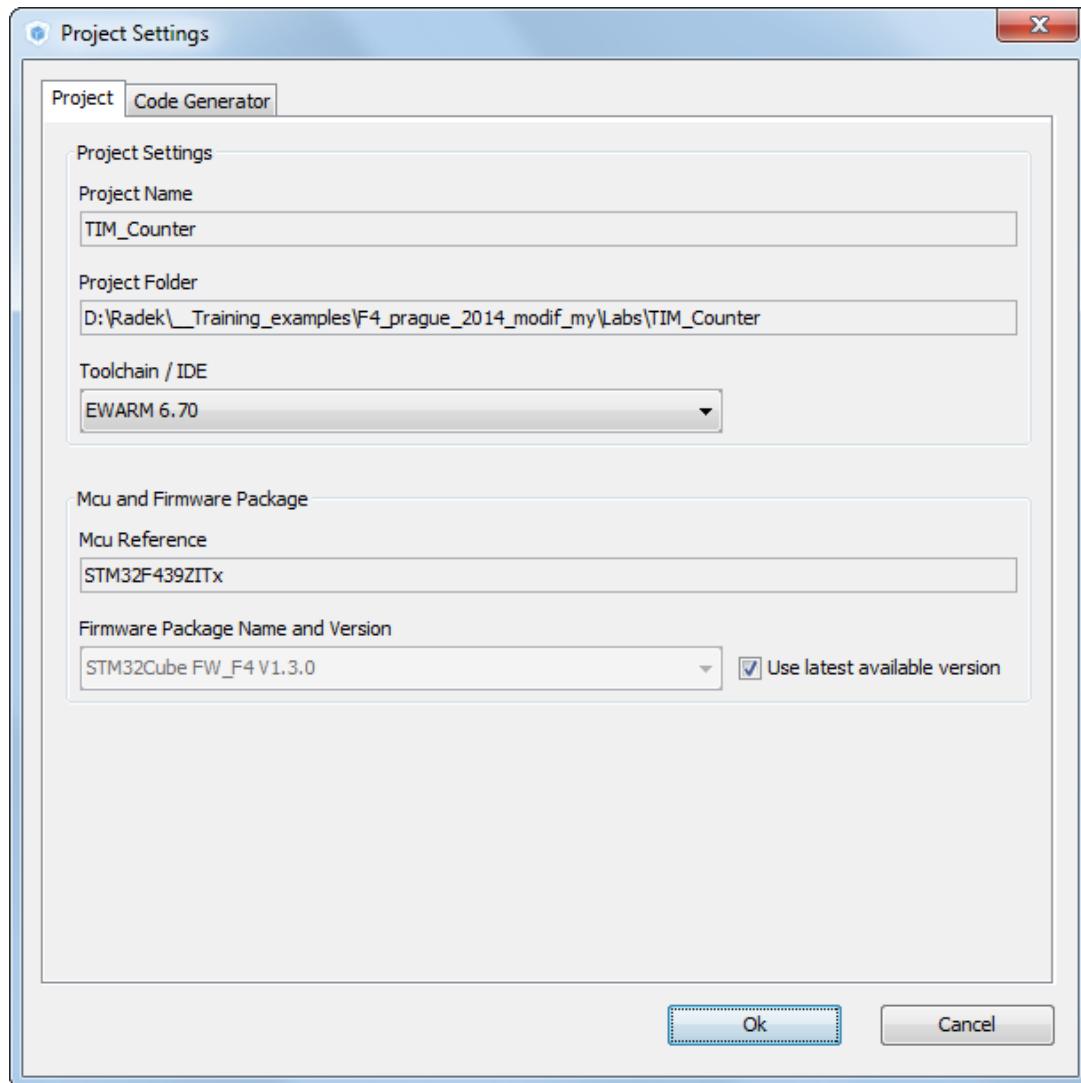


- Now we set the project details for generation

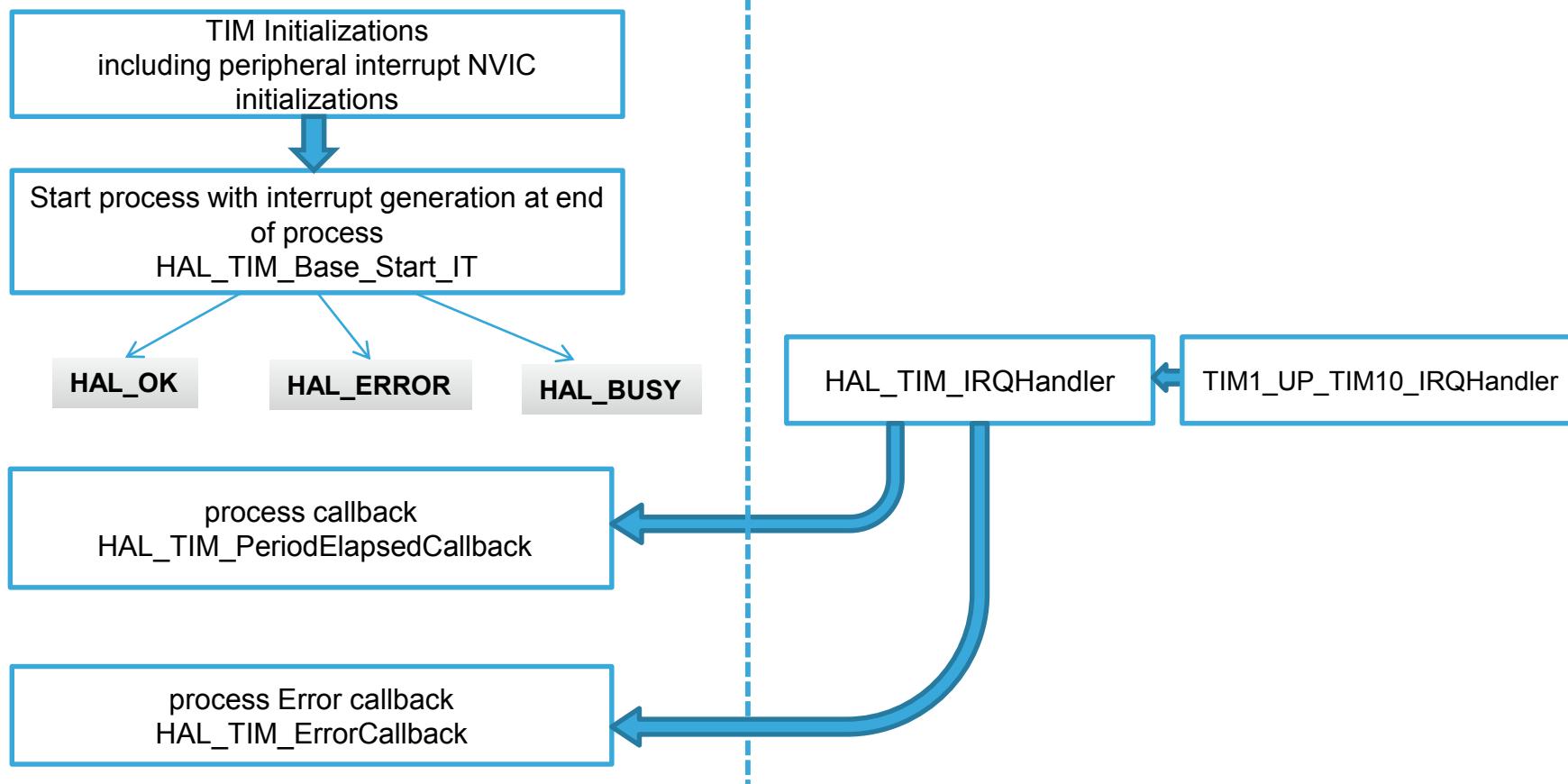
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



## HAL Library TIM with IT flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
- For TIM start use function
  - `HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)`
- TIM callback
  - `void TIM1_UP_TIM10_IRQHandler(void)`
- GPIO LED toggle
  - `HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

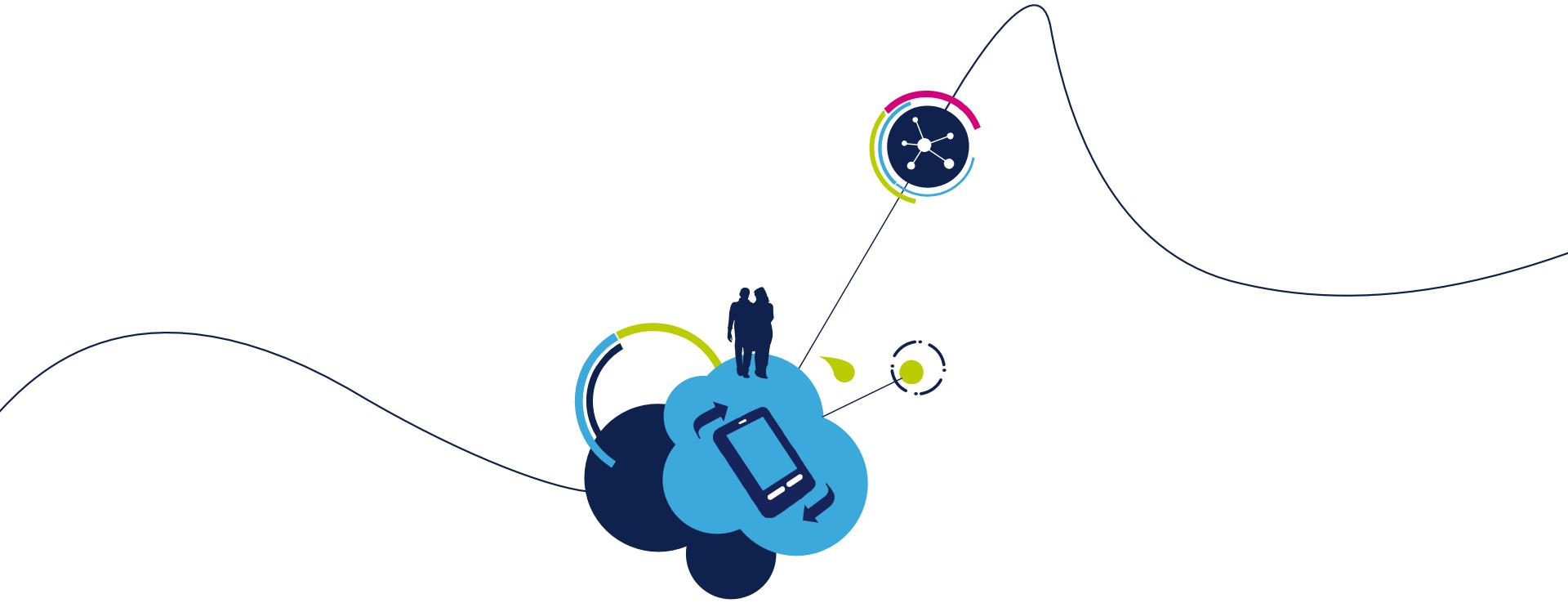
- Solution

- TIM start

```
/* USER CODE BEGIN 2 */  
HAL_TIM_Base_Start_IT(&htim1);  
/* USER CODE END 2 */
```

- Callback handling

```
/* USER CODE BEGIN 4 */  
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)  
{  
    HAL_GPIO_TogglePin(GPIOG,GPIO_PIN_14);  
}  
/* USER CODE END 4 */
```



# DAC wave generator lab 19

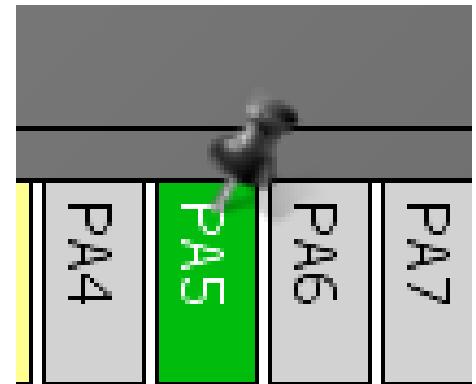
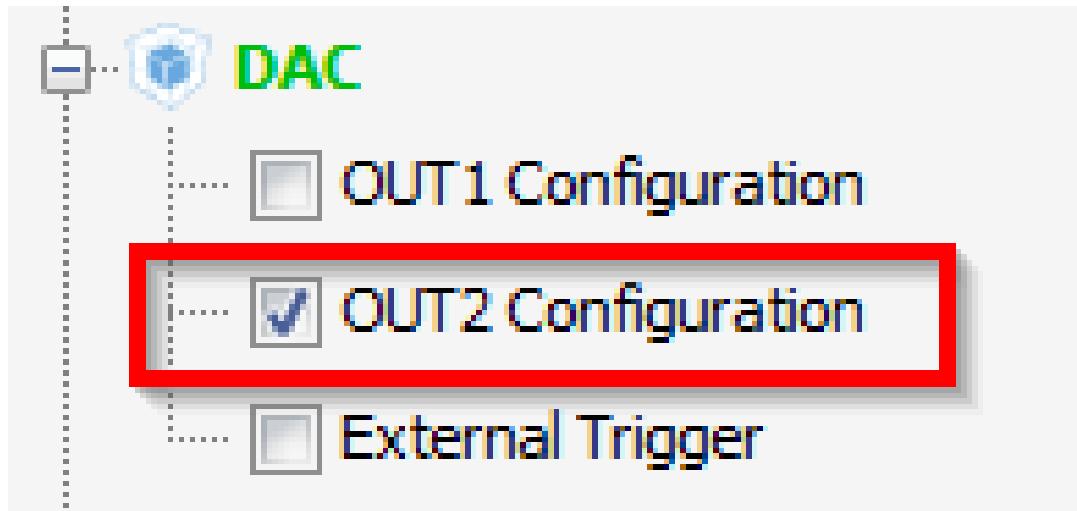
- Objective

- Learn how to setup DAC as wave generator in CubeMX
  - How to Generate Code in CubeMX and use HAL functions

- Goal

- Configure DAC as wave generator in CubeMX and Generate Code
  - Learn how start it in project

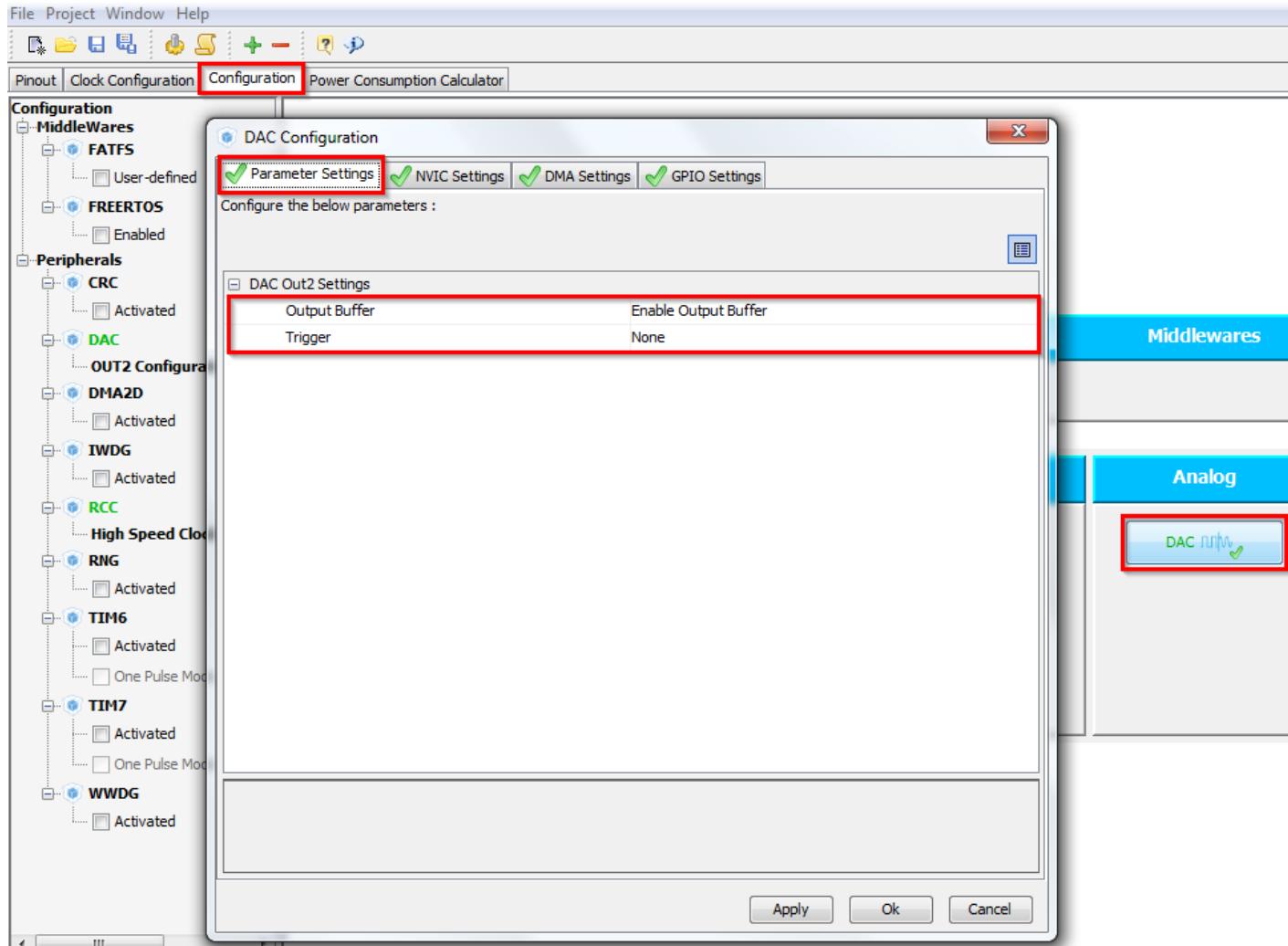
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX DAC selection
  - Select DAC OUT2



DAC\_OUT2

- CubeMX DAC configuration

- TAB>Configuration>Analog>DAC>Parametr Settings
- Enable Output buffer
- Button OK

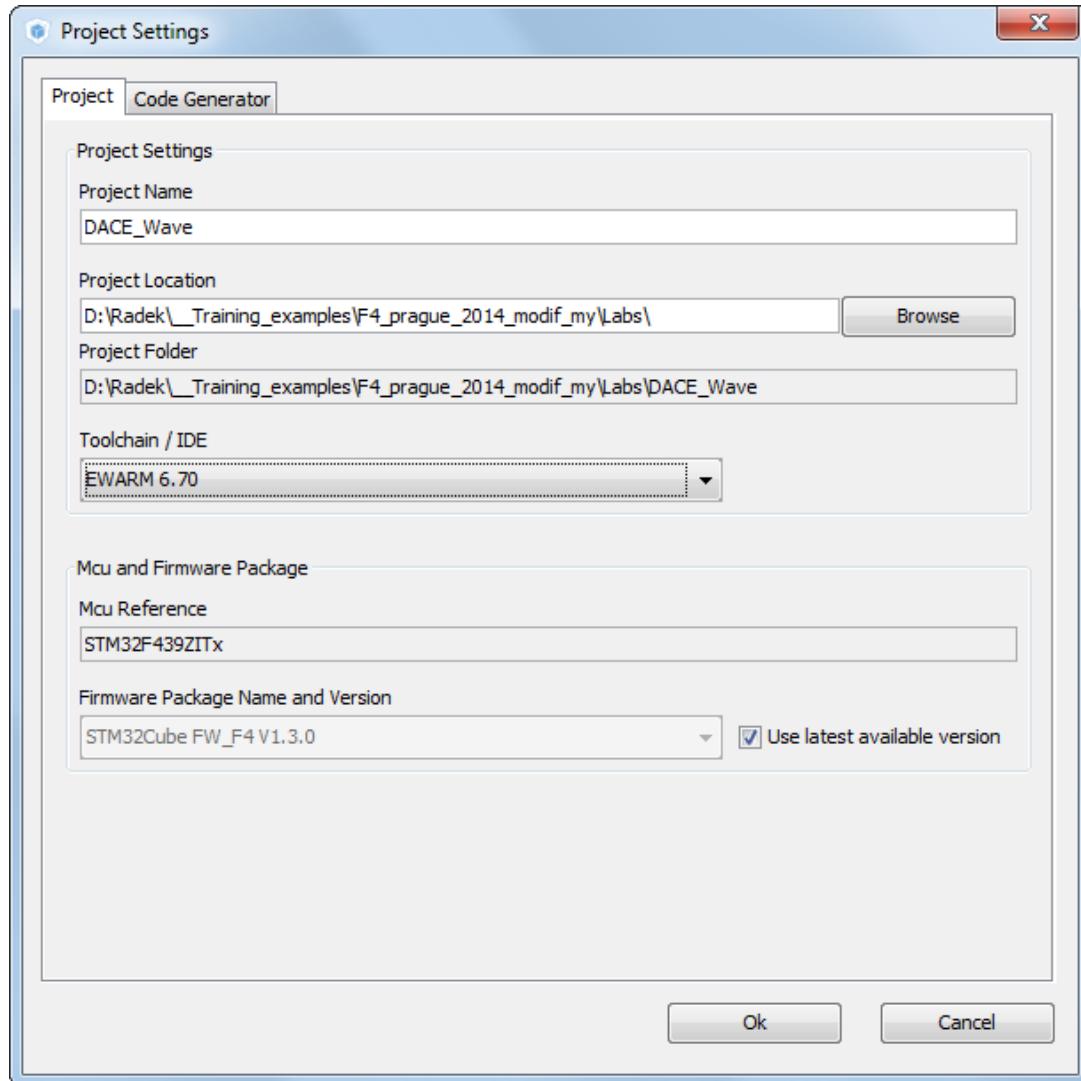


- Now we set the project details for generation

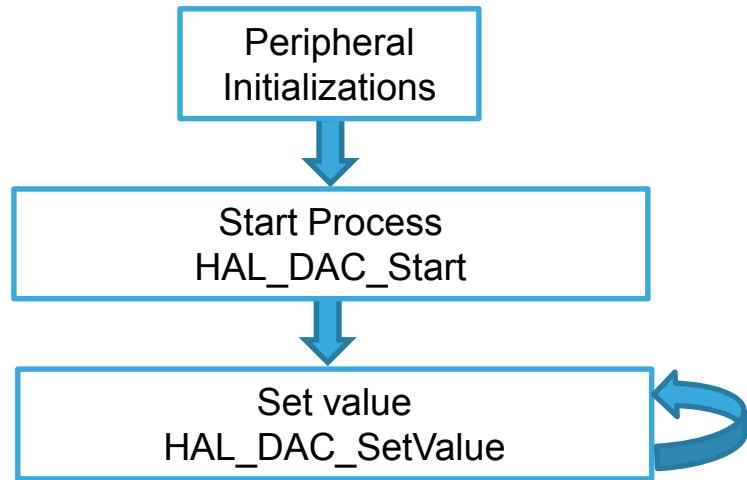
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



- Start process DAC generation (same for DMA, ADC)
  - Non blocking start process



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
  - and */\* USER CODE BEGIN 3 \*/* and */\* USER CODE END 3 \*/* tags
- For DAC start use function
  - `HAL_DAC_Start(DAC_HandleTypeDef* hdac, uint32_t Channel)`
- DAC set DAC value
  - `HAL_DAC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)`

- Solution
  - DAC setup and start

```
/* USER CODE BEGIN 2 */  
HAL_DAC_Start(&hdac,DAC_CHANNEL_2);  
/* USER CODE END 2 */
```

- Create the wave

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);  
    if(value_dac>4095)  
    {  
        value_dac=0;  
    }  
    HAL_Delay(1);  
  
}  
/* USER CODE END 3 */
```



# ADC Poll lab 20

- Objective

- Use the DAC part from previous lab
- Learn how to setup ADC in CubeMX
- How to Generate Code in CubeMX and use HAL functions

- Goal

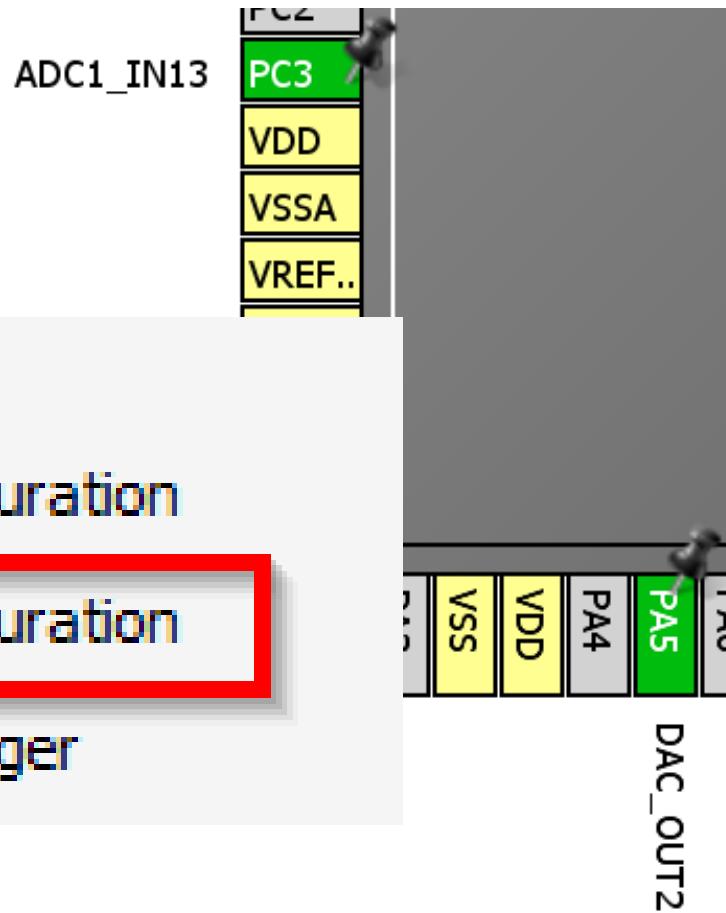
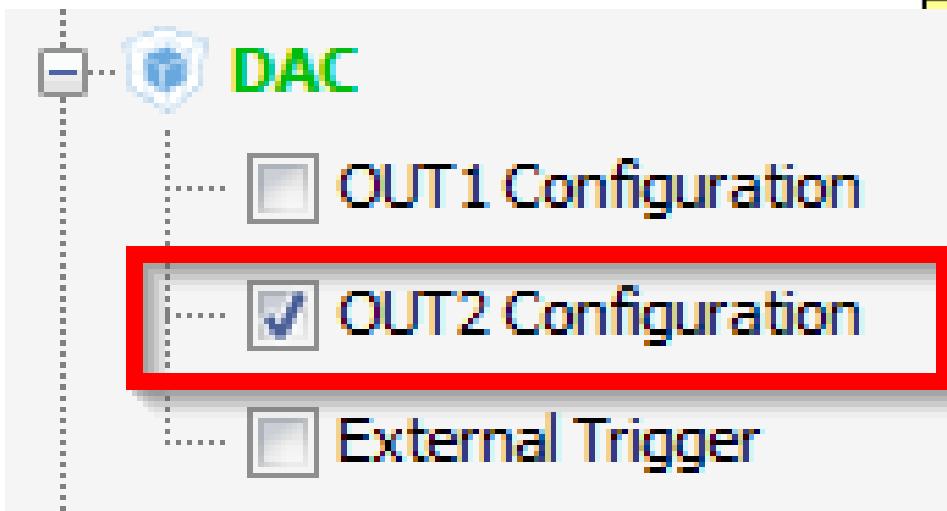
- Configure ADC in poll in CubeMX and Generate Code
- Learn how to start ADC and measure the DAC
- Verify the measured wave in STMStudio  
(<http://www.st.com/web/en/catalog/tools/PF251373> require JAVA)

- Create project in CubeMX

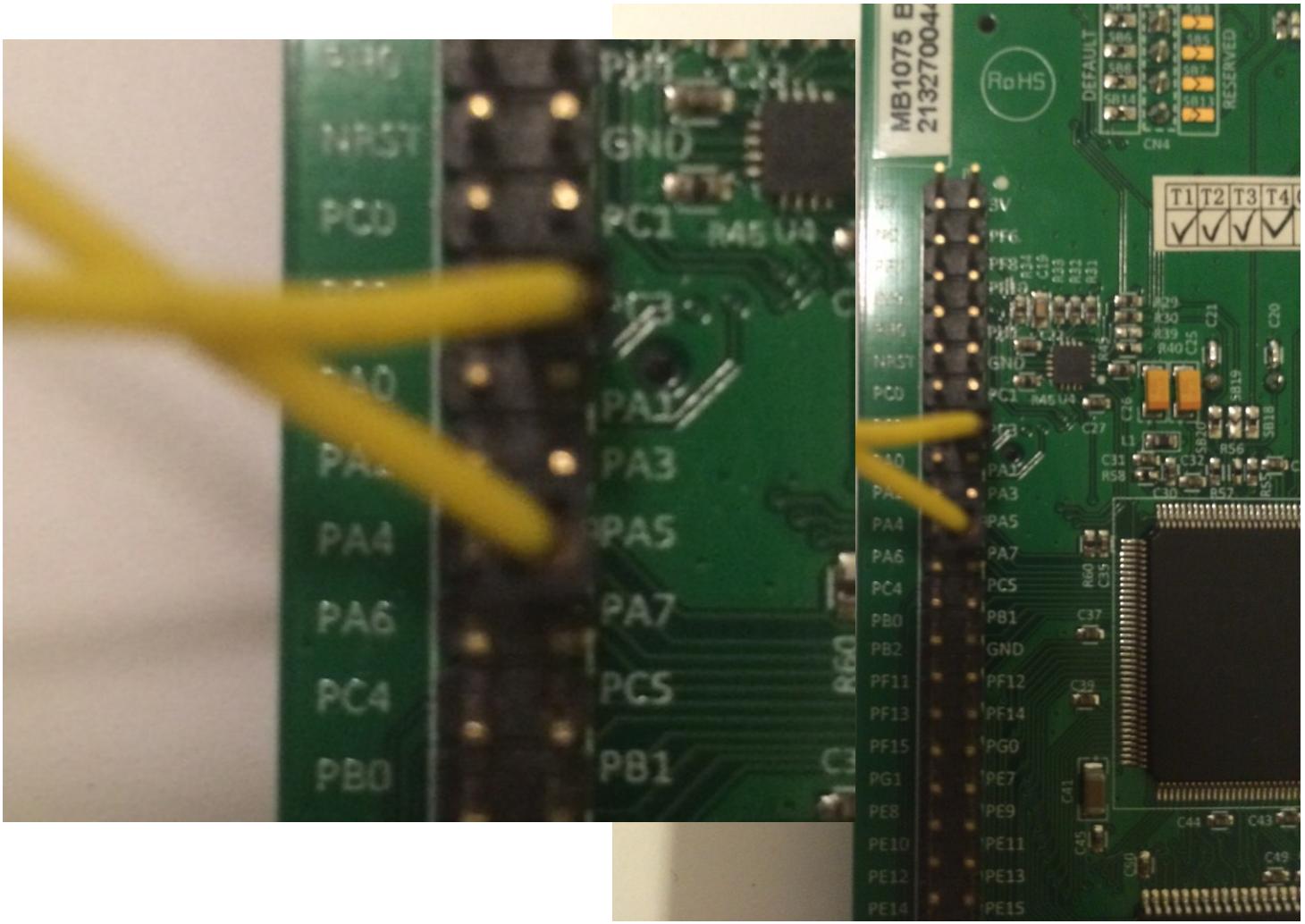
- Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx

- CubeMX DAC selection

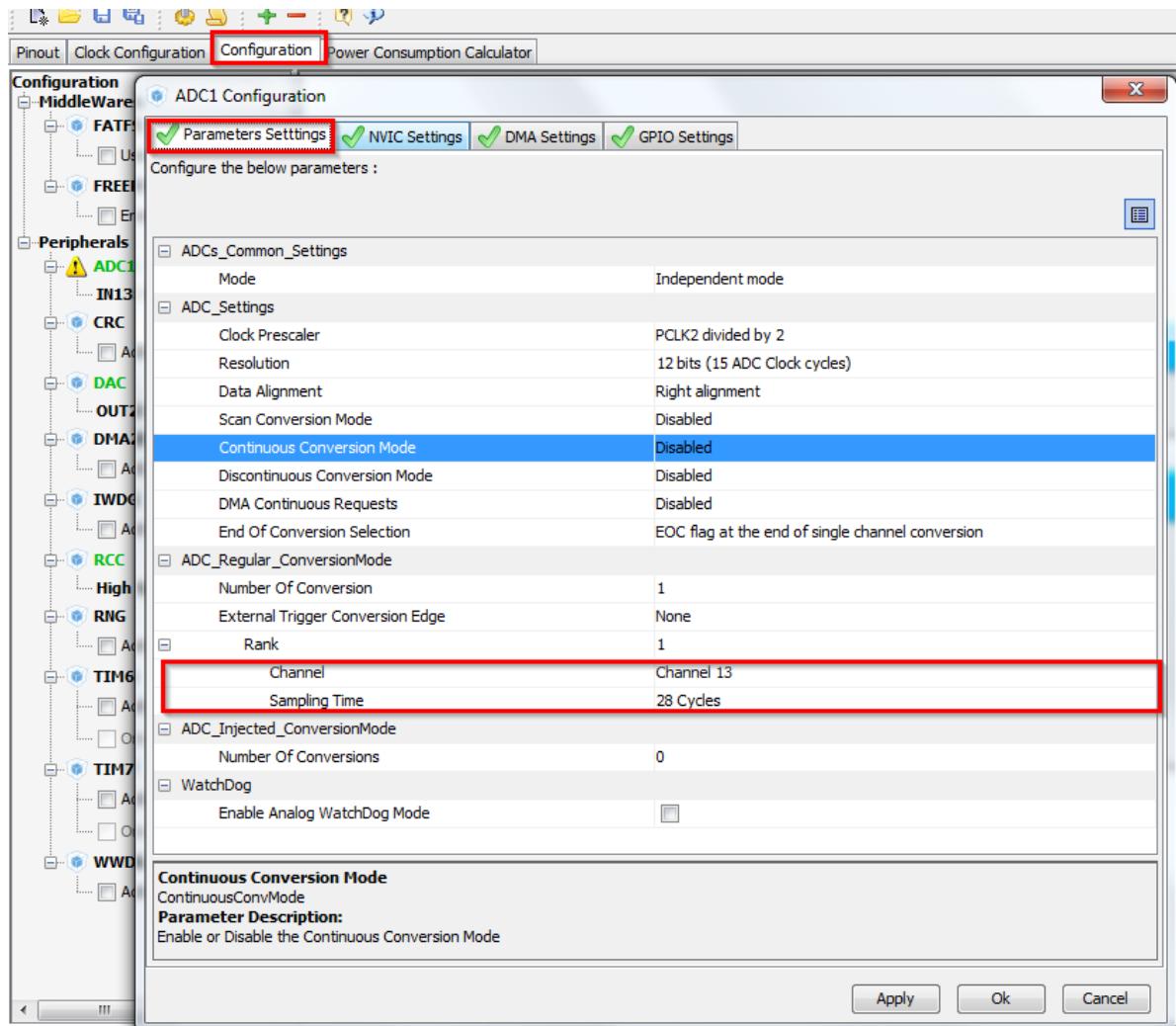
- Select DAC OUT2
  - Select ADC IN13



- Hardware connection
  - Connect DAC out2 PA5 and ADC1 IN13 PC3 together



- CubeMX ADC configuration
  - TAB>Configuration>Analog>ADC1>Parametr Settings
  - Set ADC1
  - Set sampling time for CH13
  - Button OK
- DAC from previous example

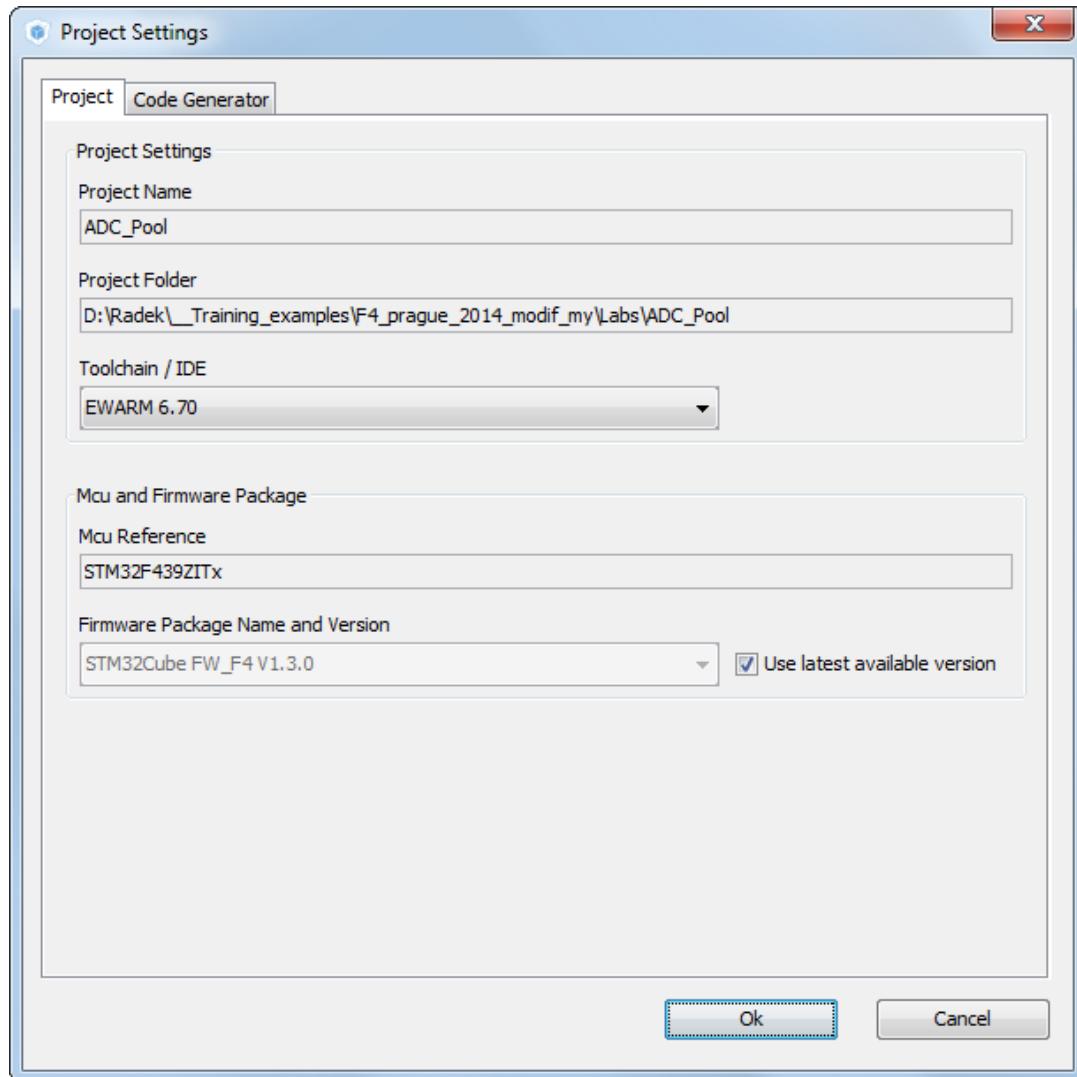


- Now we set the project details for generation

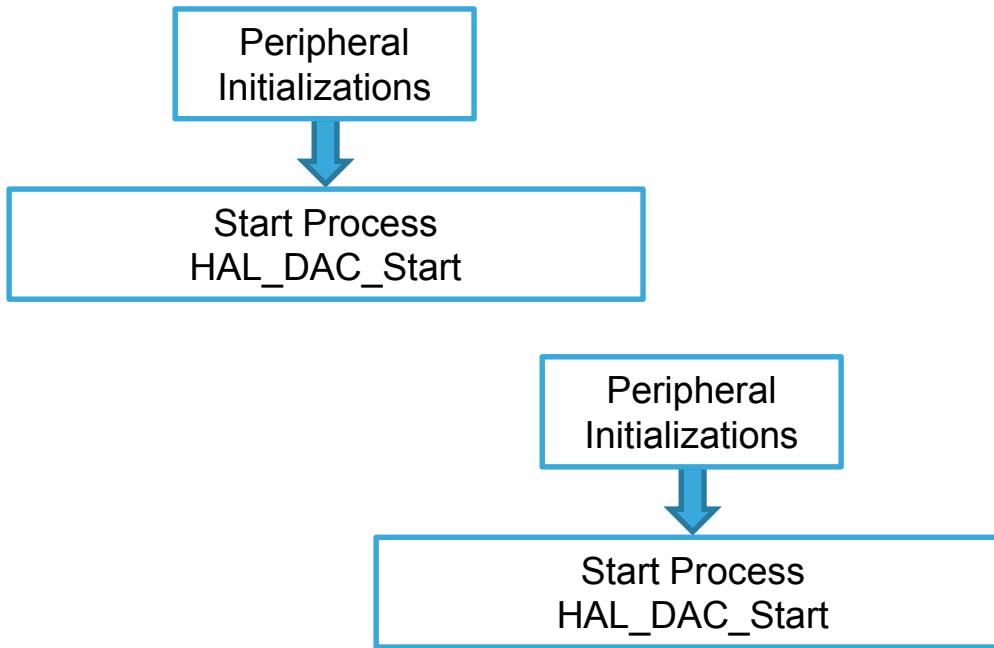
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



- Start process ADC(same for DMA, DAC, TIM)
  - Non blocking start process



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
  - and */\* USER CODE BEGIN 3 \*/* and */\* USER CODE END 3 \*/* tags
- For DAC start use function
  - `HAL_DAC_Start(DAC_HandleTypeDef* hdac, uint32_t Channel)`
  - `HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout)`
  - `HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)`
- DAC functions
  - `HAL_DAC_Start(DAC_HandleTypeDef* hdac, uint32_t Channel)`
  - `HAL_DAC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)`

- Solution

- Variables

```
/* USER CODE BEGIN PV */  
uint32_t value_adc;  
uint32_t value_dac=0;  
/* USER CODE END PV */
```

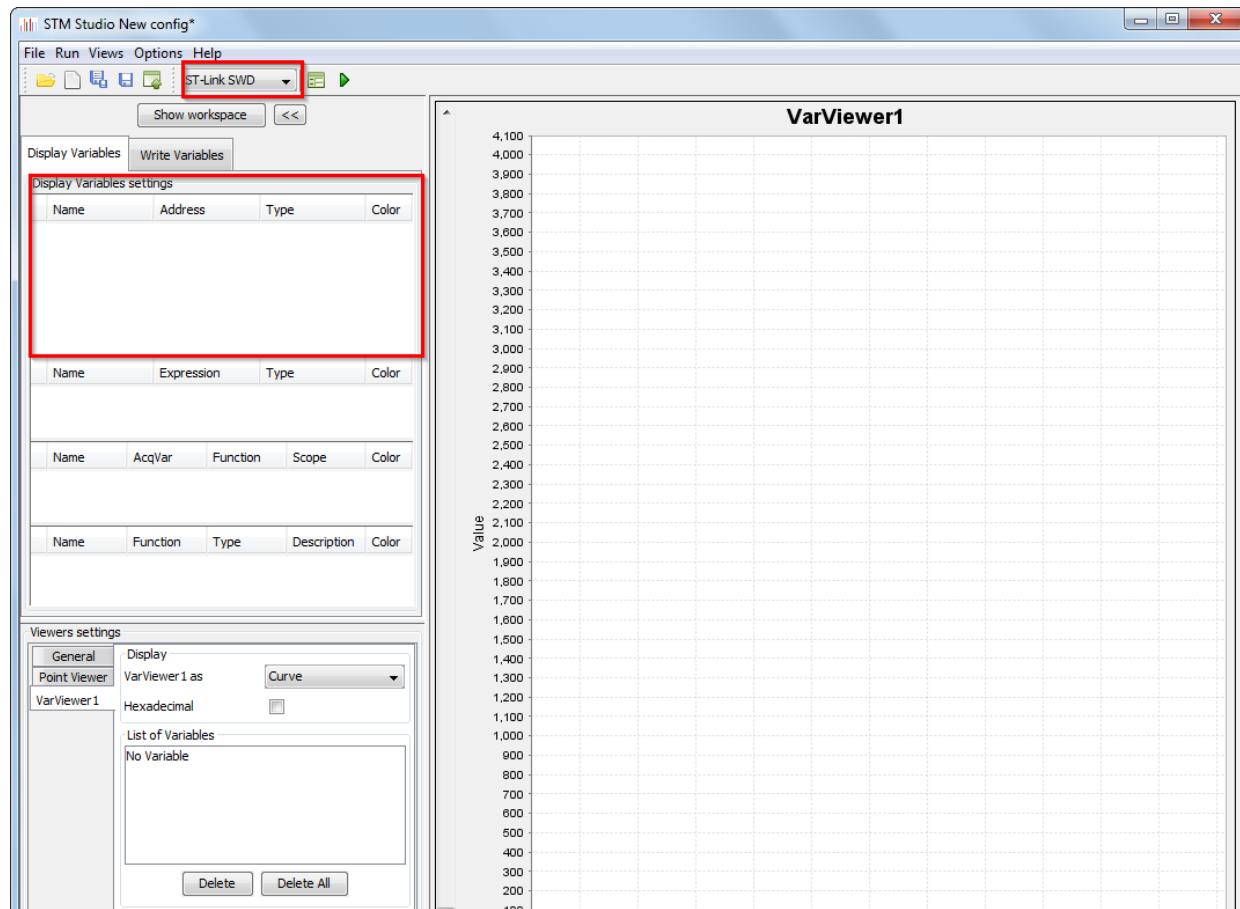
- DAC setup and start

```
/* USER CODE BEGIN 2 */  
HAL_DAC_Start(&hdac,DAC_CHANNEL_2);  
HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);  
/* USER CODE END 2 */
```

- Solution
  - Main loop with DAC set and ADC set

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_ADC_Start(&hadc1);  
    HAL_ADC_PollForConversion(&hadc1,10);  
    value_adc=HAL_ADC_GetValue(&hadc1);  
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);  
    value_dac++;  
    if(value_dac>4095){  
        value_dac=0;  
    }  
    HAL_Delay(1);  
}  
/* USER CODE END 3 */
```

- Test the functionality
  - We need the address of variable `value_adc`
  - This can be found usually in debug mode in watch, my address is `0x2000005C` (depends on compiler and optimizations)
- Start the STMStudio
  - Set the ST Link SWD
  - Right click into Display variable settings
  - Select NEW



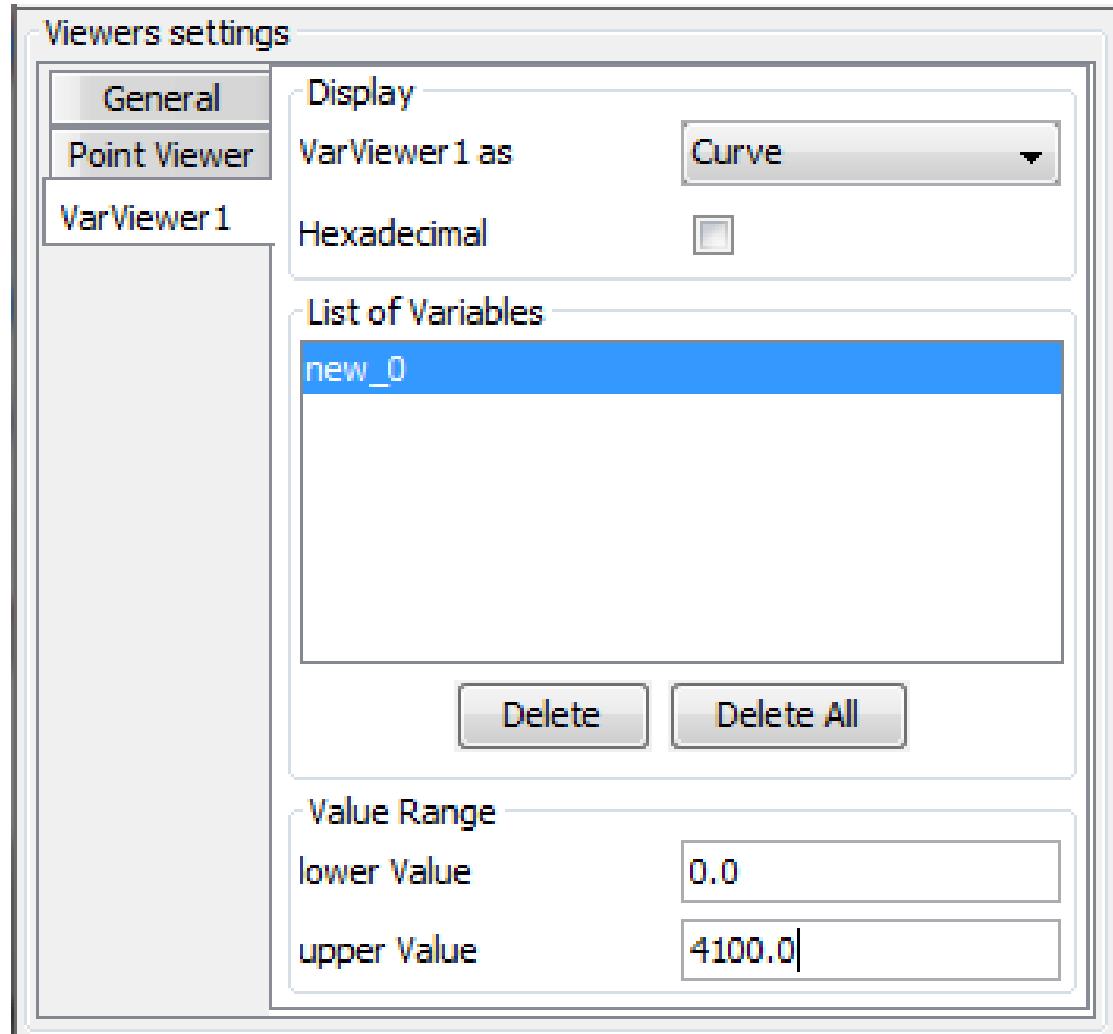
- STM studio settings

- Set value\_adc address my 0x2000005C
- Set 16bit unsigned val
- Right click on this line
- Select Send To VarViewer1

Display Variables settings				
Name	Address	Type	Color	
D new_0	0x2000005C	unsigned 16-bit	blue	[...]

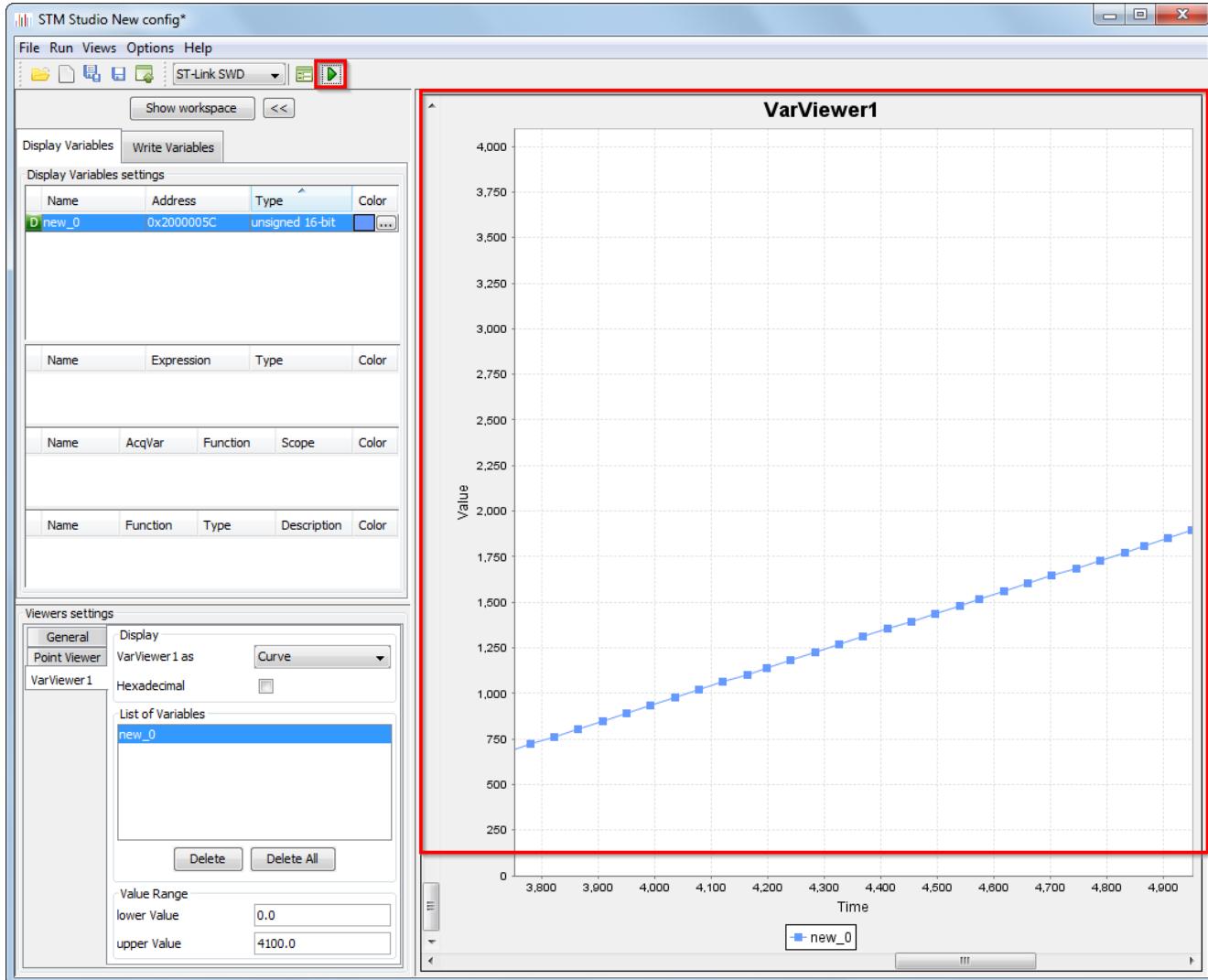
Display Variables settings				
Name	Address	Type	Color	
D new_0	0x2000005C	unsigned 16-bit	blue	[...]
Delete				
New				
Send To				
VarViewer1				
Import ...				
Update				
Name	Expression	Type	Color	

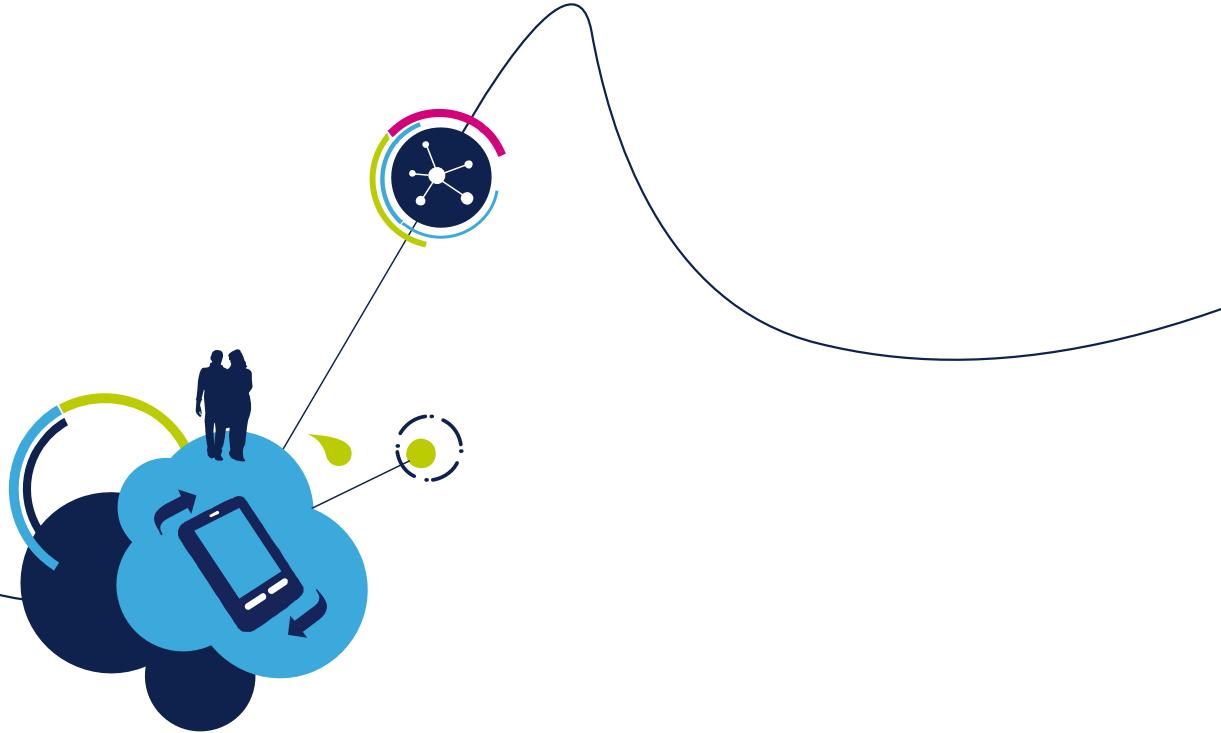
- STM studio settings
  - Viewers settings is on bottom
  - Set the correct upper value to 4096(12bit)



- STM studio settings

- Now press green play button
- And you will see content of value\_adc





# ADC Interrupt lab 21

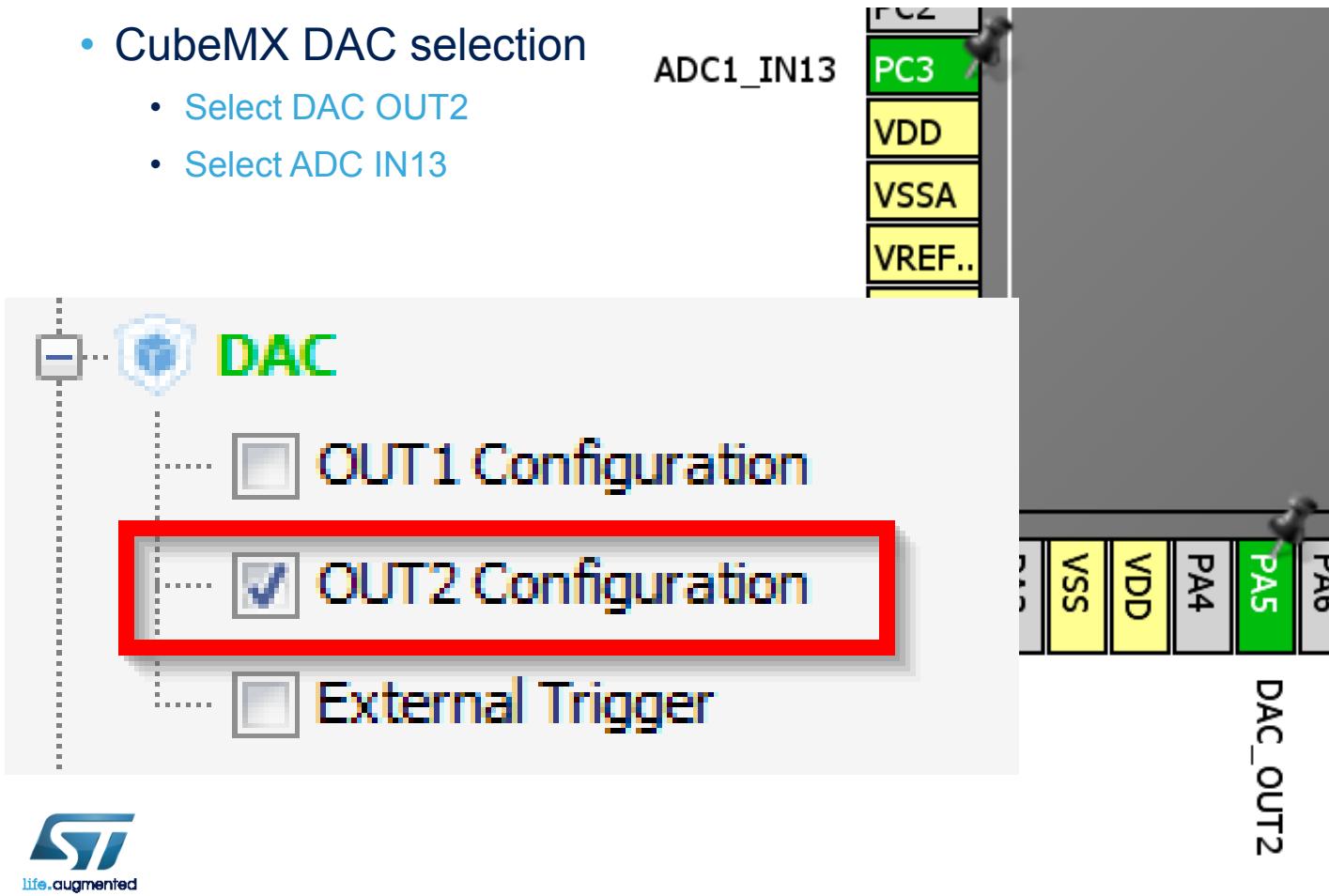
- Objective

- Use the DAC part from previous lab
- Learn how to setup ADC with interrupt in CubeMX
- How to Generate Code in CubeMX and use HAL functions

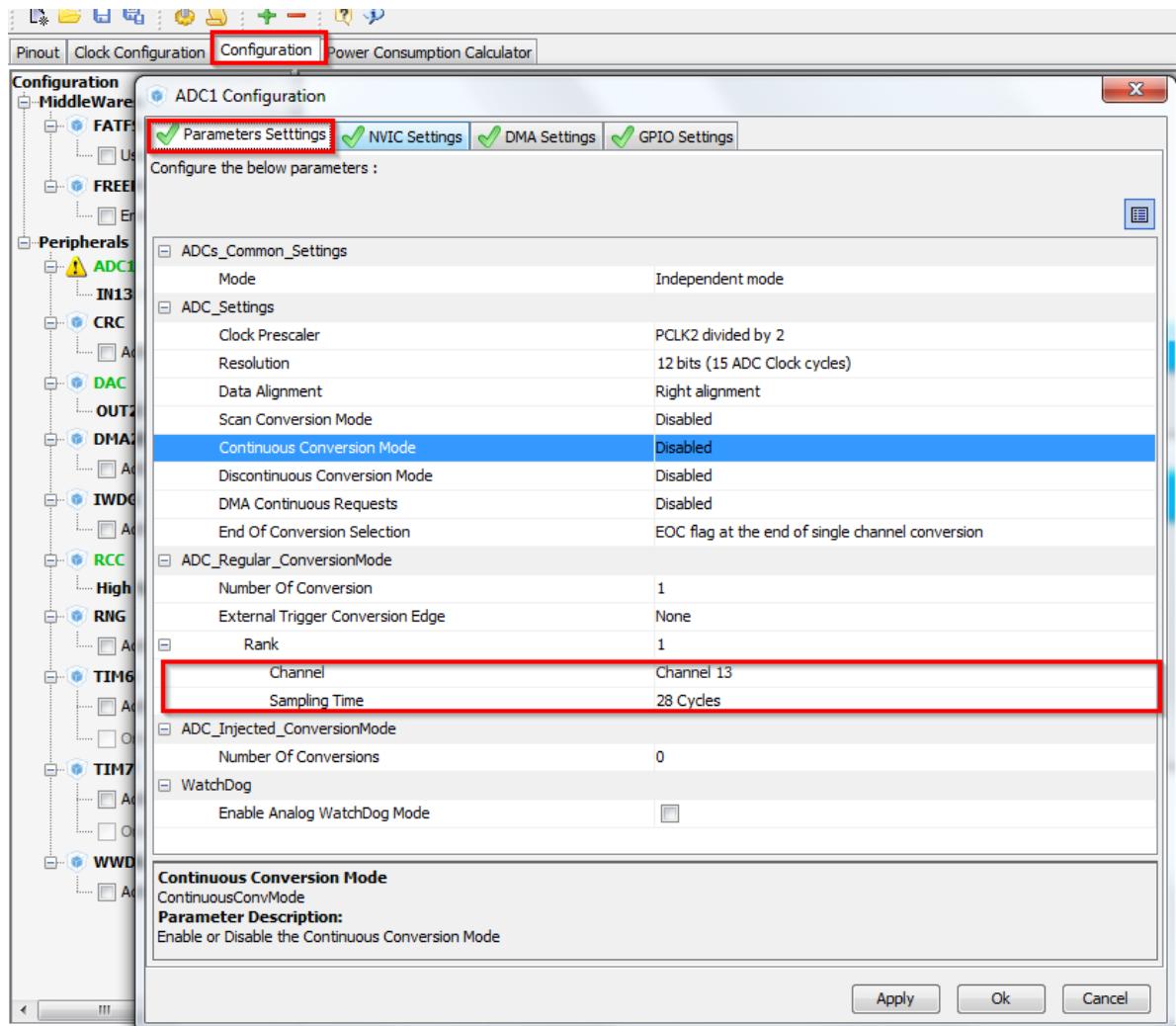
- Goal

- Configure ADC in interrupt in CubeMX and Generate Code
- Learn how to start ADC and measure the DAC
- Verify the measured wave in STMStudio  
(<http://www.st.com/web/en/catalog/tools/PF251373> require JAVA)

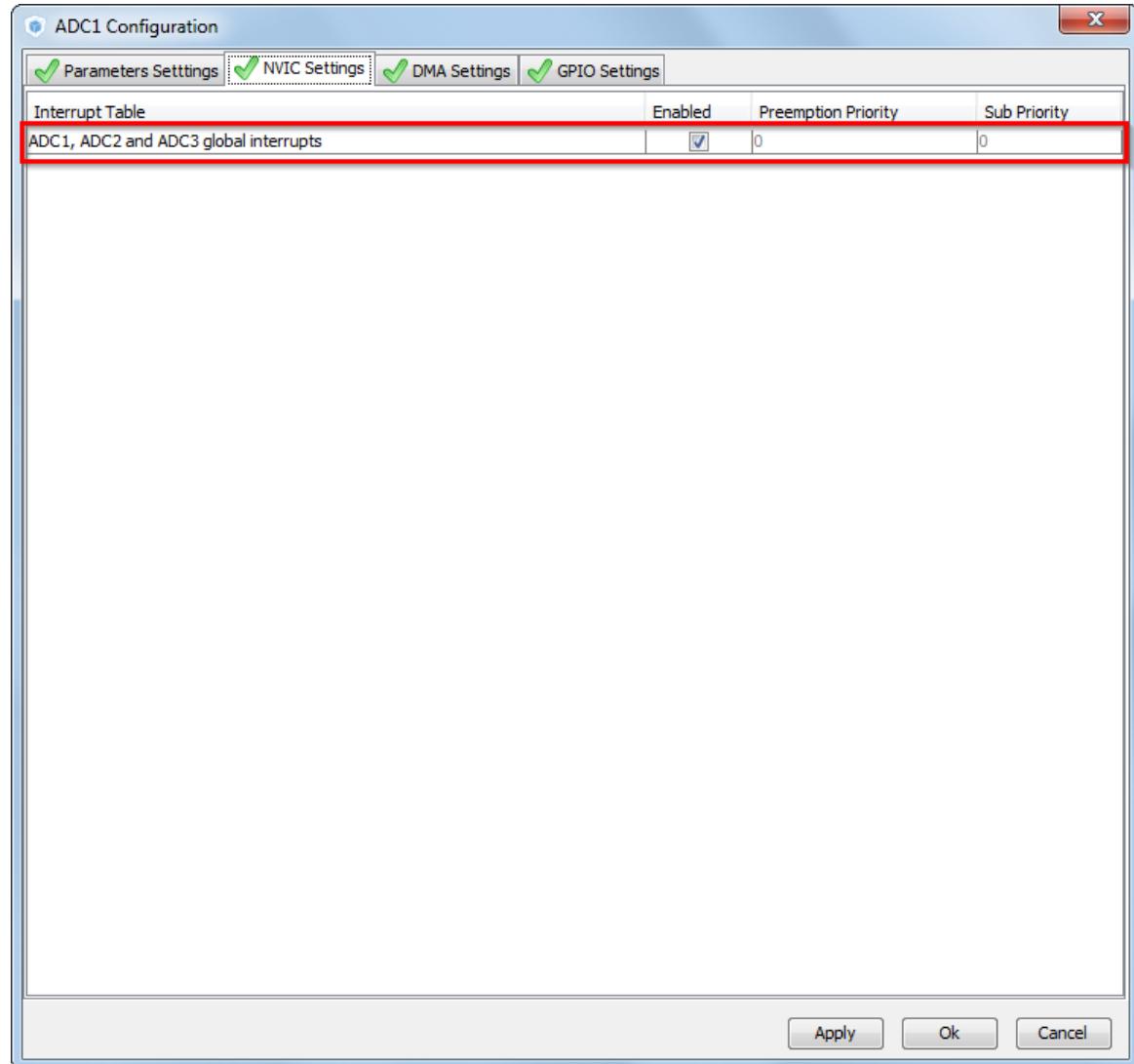
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX DAC selection
  - Select DAC OUT2
  - Select ADC IN13



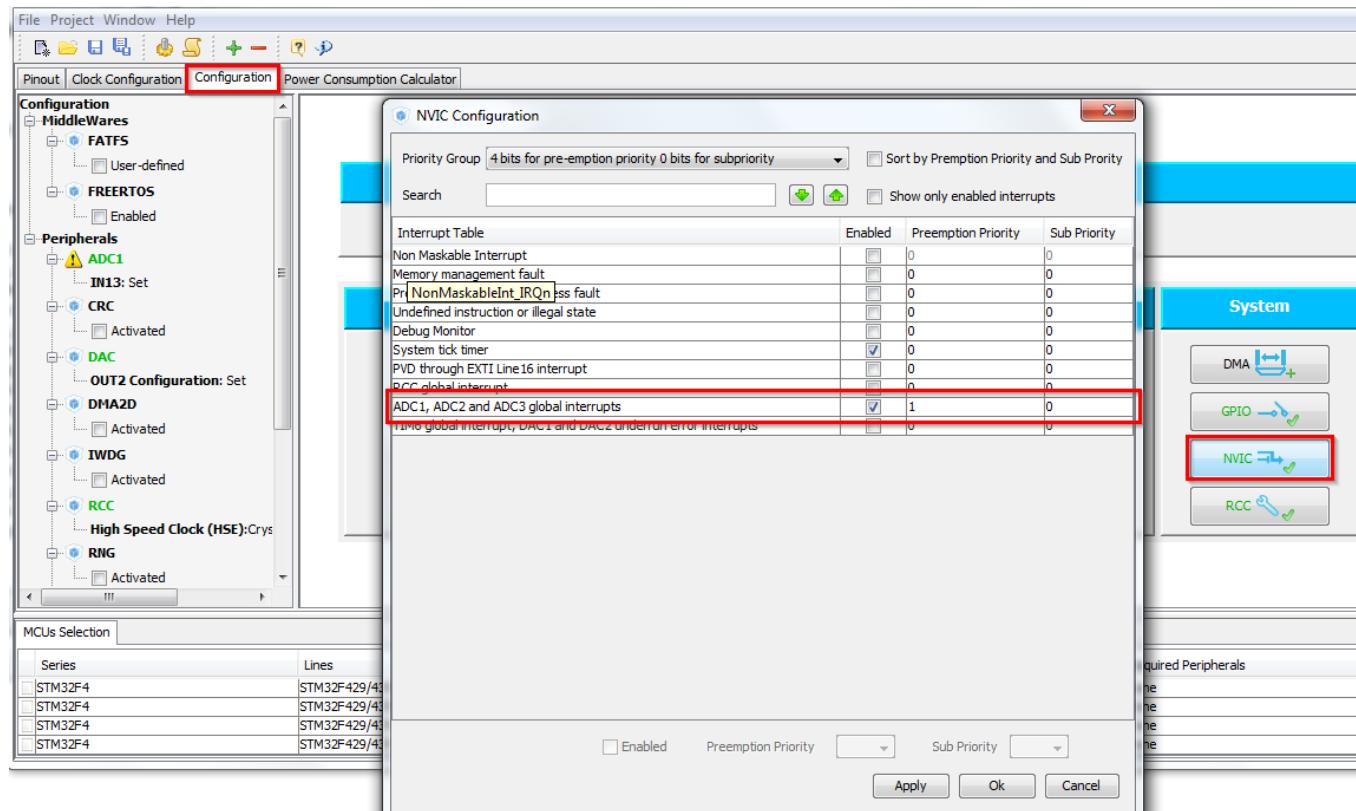
- CubeMX ADC configuration
  - TAB>Configuration>Analog>ADC1>Parametr Settings
  - Set ADC1
  - Set sampling time for CH13
  - Button OK
- DAC from previous example



- CubeMX ADC configuration
  - TAB>NVIC settings
  - Enable ADC1 interrupt
  - Button OK



- CubeMX NVIC configuration
  - Because we want use the Systick for delay in interrupt The ADC interrupt priority must be changed
  - TAB>Configuration>System>NVIC
  - Change ADC1 preemption priority to 1

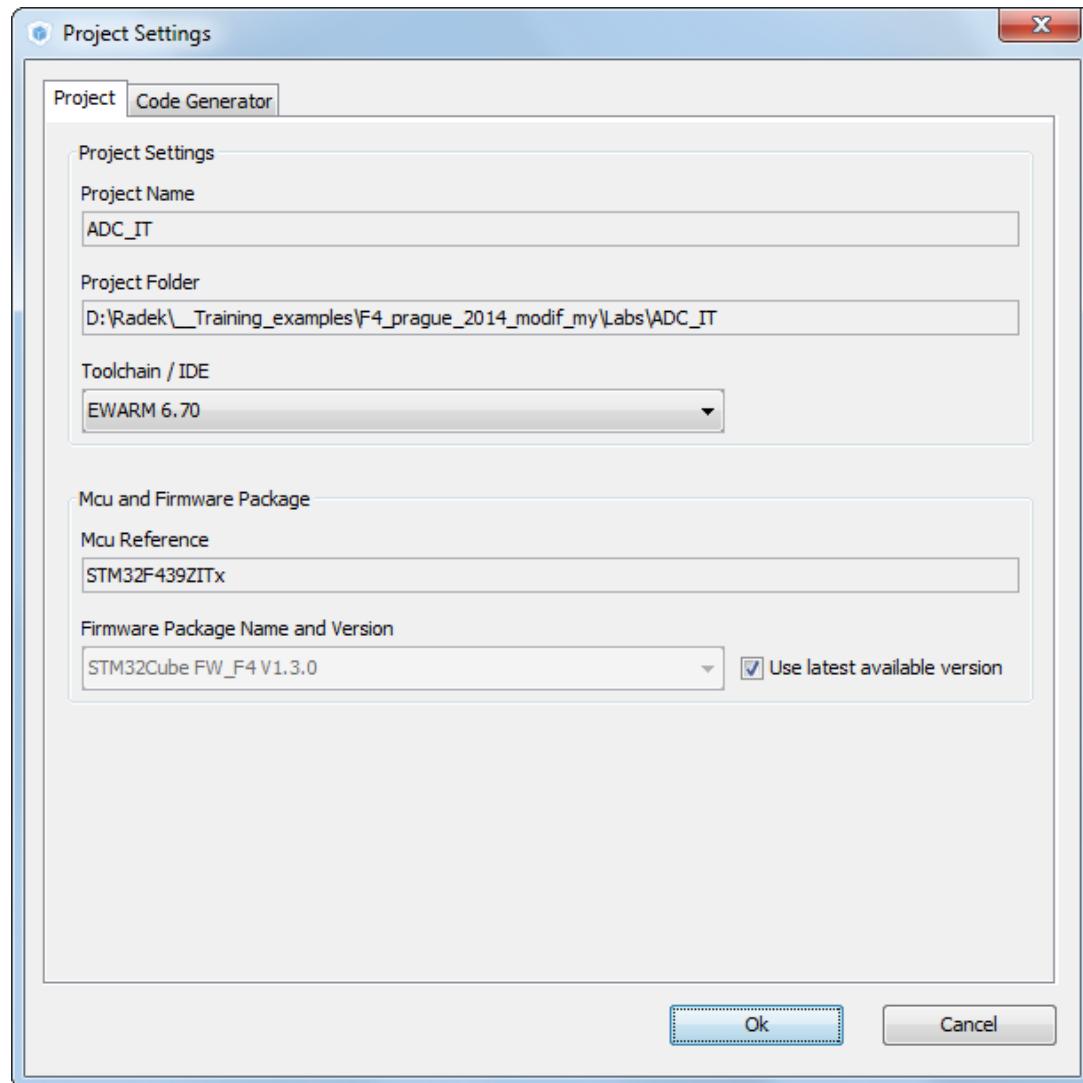


- Now we set the project details for generation

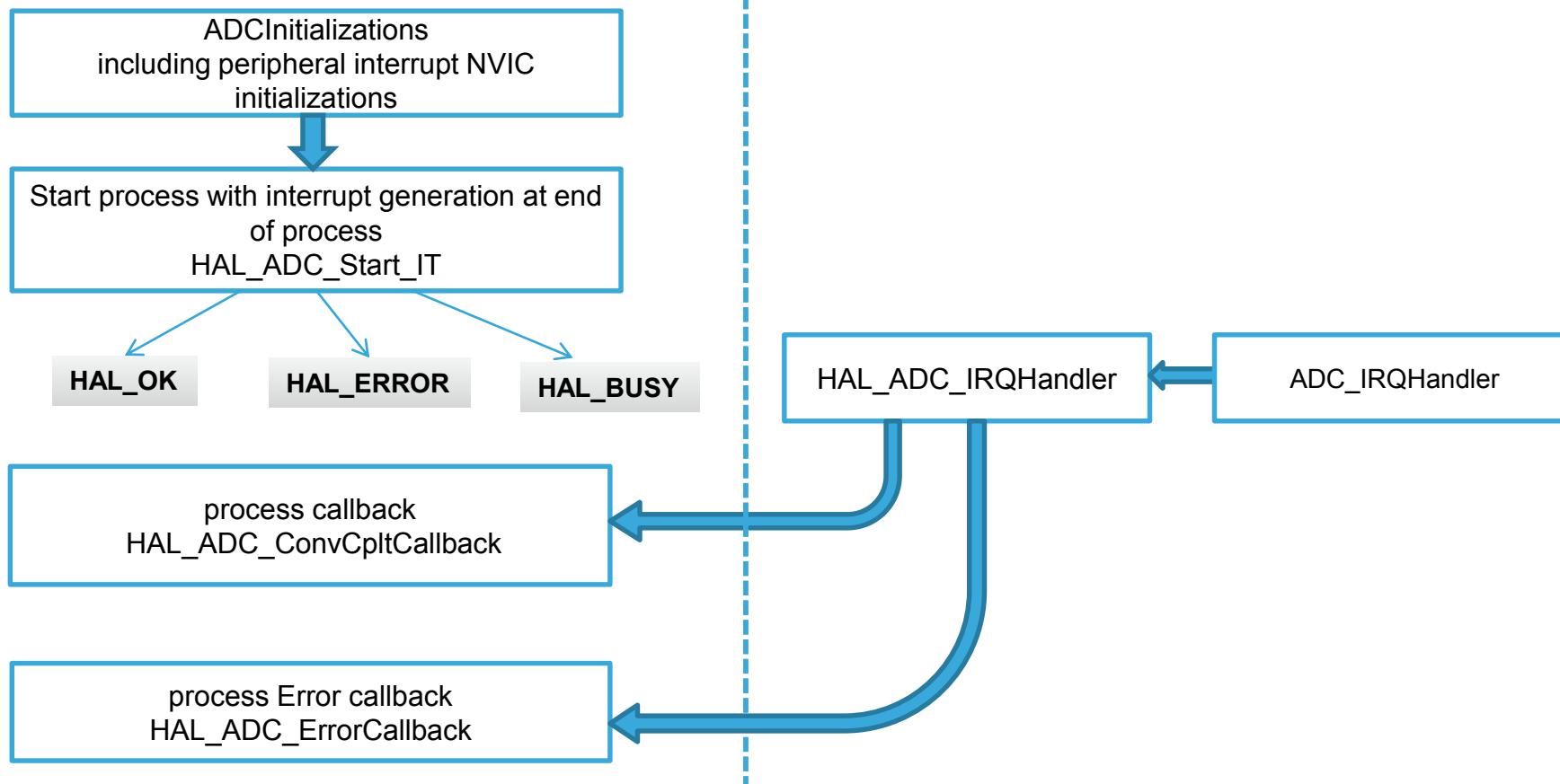
- Menu > Project > Project Settings
- Set the project name
- Project location
- Type of toolchain

- Now we can Generate Code

- Menu > Project > Generate Code



## HAL Library ADC with IT flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
  - and */\* USER CODE BEGIN 4 \*/* and */\* USER CODE END 4 \*/* tags
- For DAC start use function
  - `HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc, uint32_t Channel)`
  - `HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)`
- ADC complete callback function
  - `HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)`
- DAC functions
  - `HAL_DAC_Start(DAC_HandleTypeDef* hdac, uint32_t Channel)`
  - `HAL_DAC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)`

- Solution

- Variables

```
/* USER CODE BEGIN PV */  
uint32_t value_adc;  
uint32_t value_dac=0;  
/* USER CODE END PV */
```

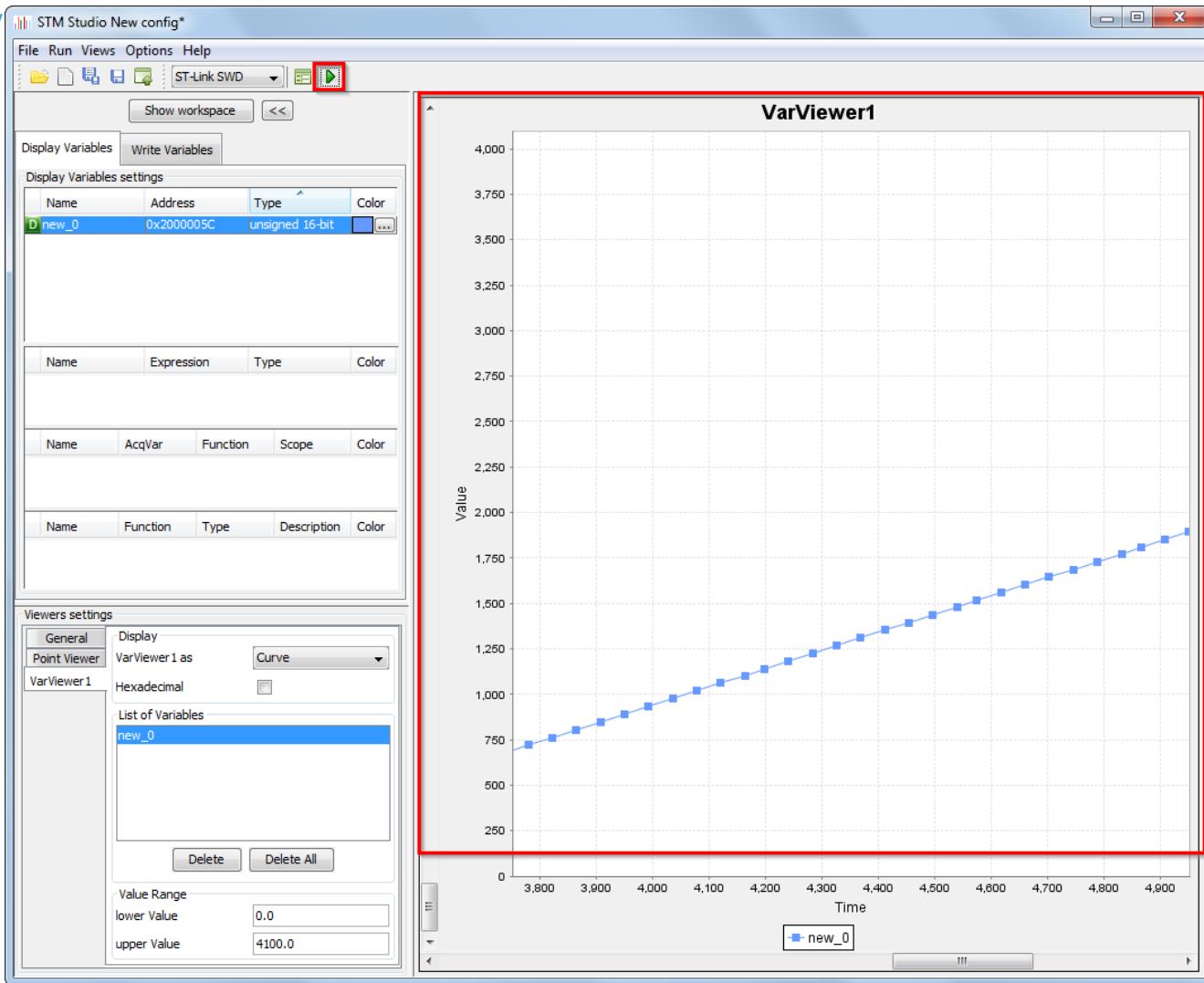
- DAC setup and start ADC/DAC

```
/* USER CODE BEGIN 2 */  
HAL_DAC_Start(&hdac,DAC_CHANNEL_2);  
HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);  
HAL_ADC_Start_IT(&hadc1);  
/* USER CODE END 2 */
```

- Solution
  - ADC complete callback routine

```
/* USER CODE BEGIN 4 */  
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)  
{  
    value_adc=HAL_ADC_GetValue(&hadc1);  
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);  
    value_dac++;  
    if(value_dac>4095){  
        value_dac=0;  
    }  
    HAL_Delay(1);  
    HAL_ADC_Start_IT(&hadc1);  
}  
/* USER CODE END 4 */
```

- STM studio settings
  - Check functionality again with STMstudio





# ADC with DMA lab 22

- Objective

- Use the DAC part from previous lab
- Learn how to setup ADC with DMA in CubeMX
- How to Generate Code in CubeMX and use HAL functions

- Goal

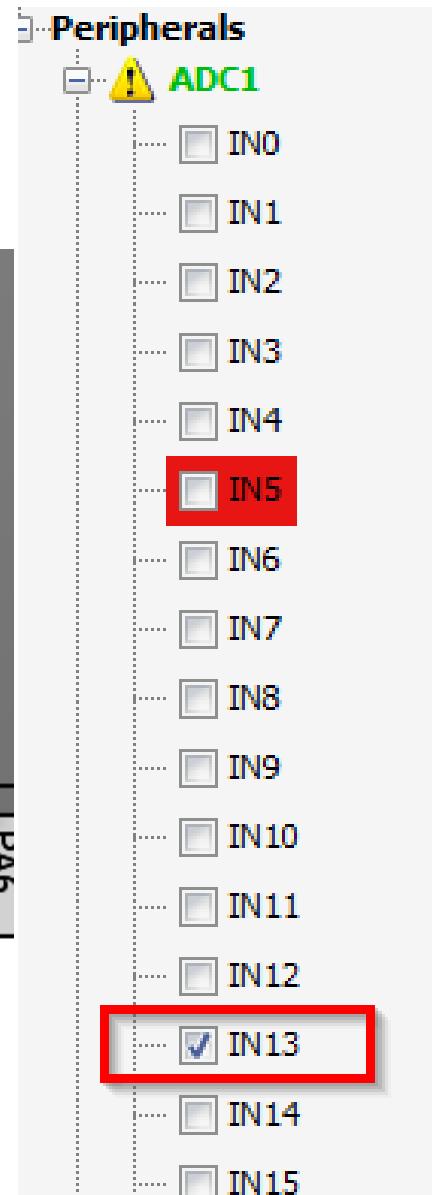
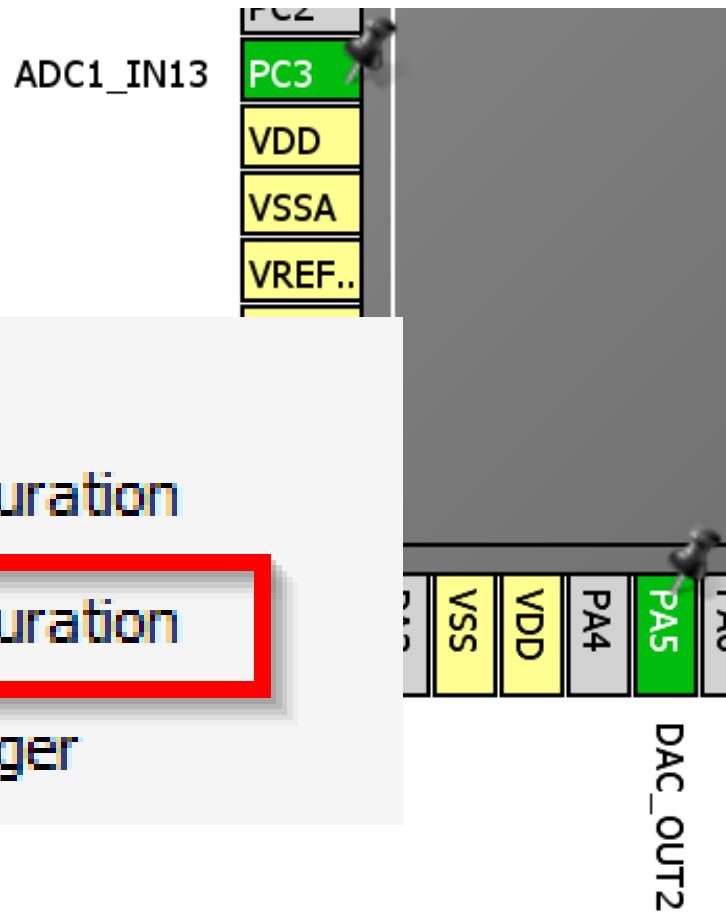
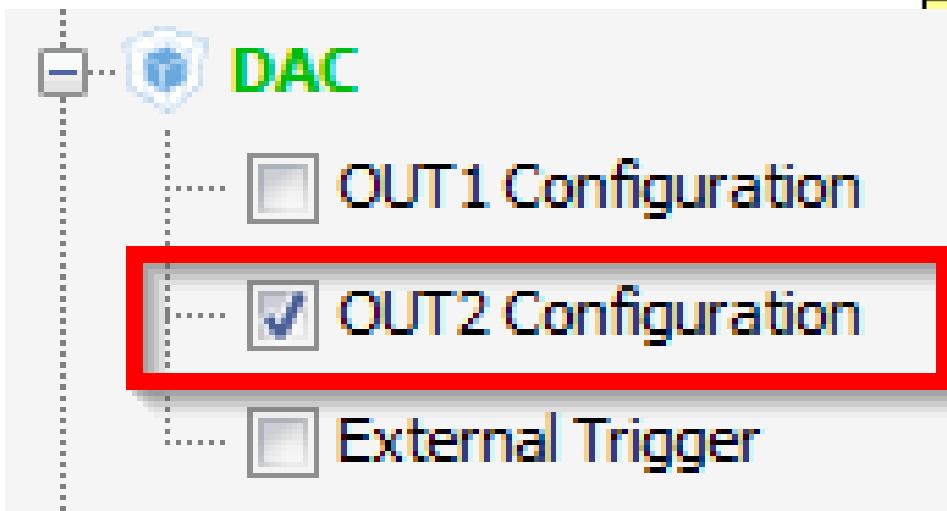
- Configure ADC in DMA in CubeMX and Generate Code
- Learn how to start ADC and measure the DAC
- Verify the measured wave in STMStudio  
(<http://www.st.com/web/en/catalog/tools/PF251373> require JAVA)

- Create project in CubeMX

- Menu > File > New Project
- Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx

- CubeMX DAC selection

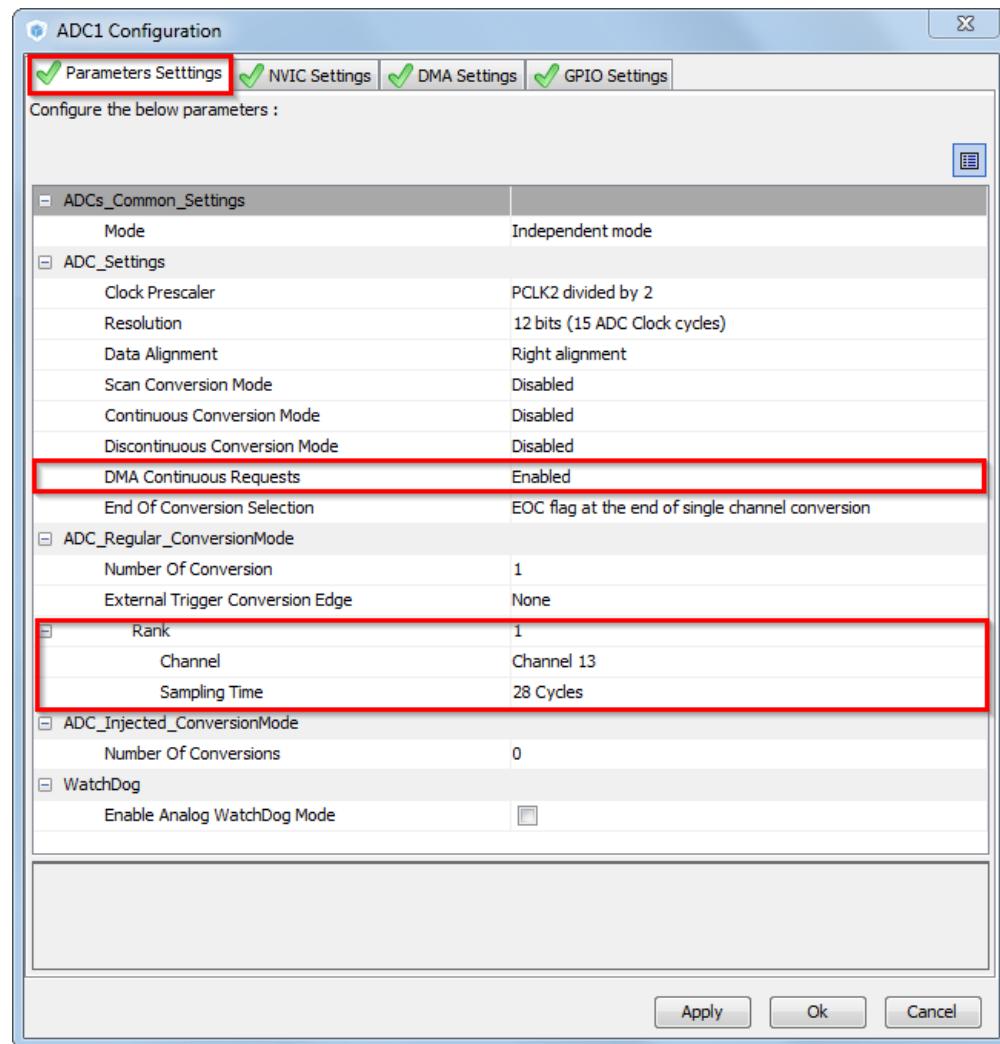
- Select DAC OUT2
- Select ADC IN13



- CubeMX ADC configuration

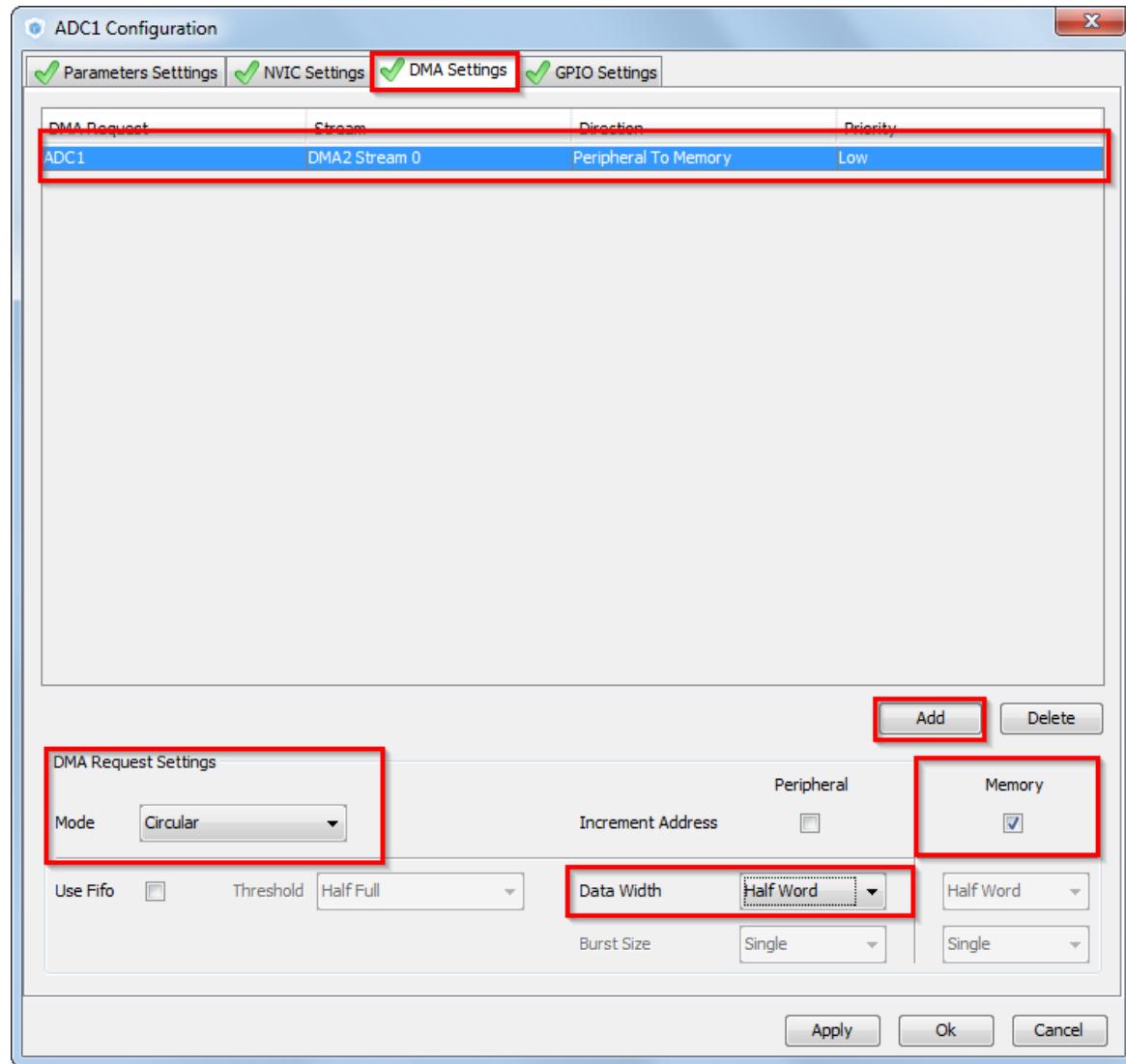
- TAB>Configuration>Analog>ADC1>Parameter Settings
- Set ADC1
- Set sampling time for CH13
- DMA Continuous requests
- Button OK

- DAC from previous example

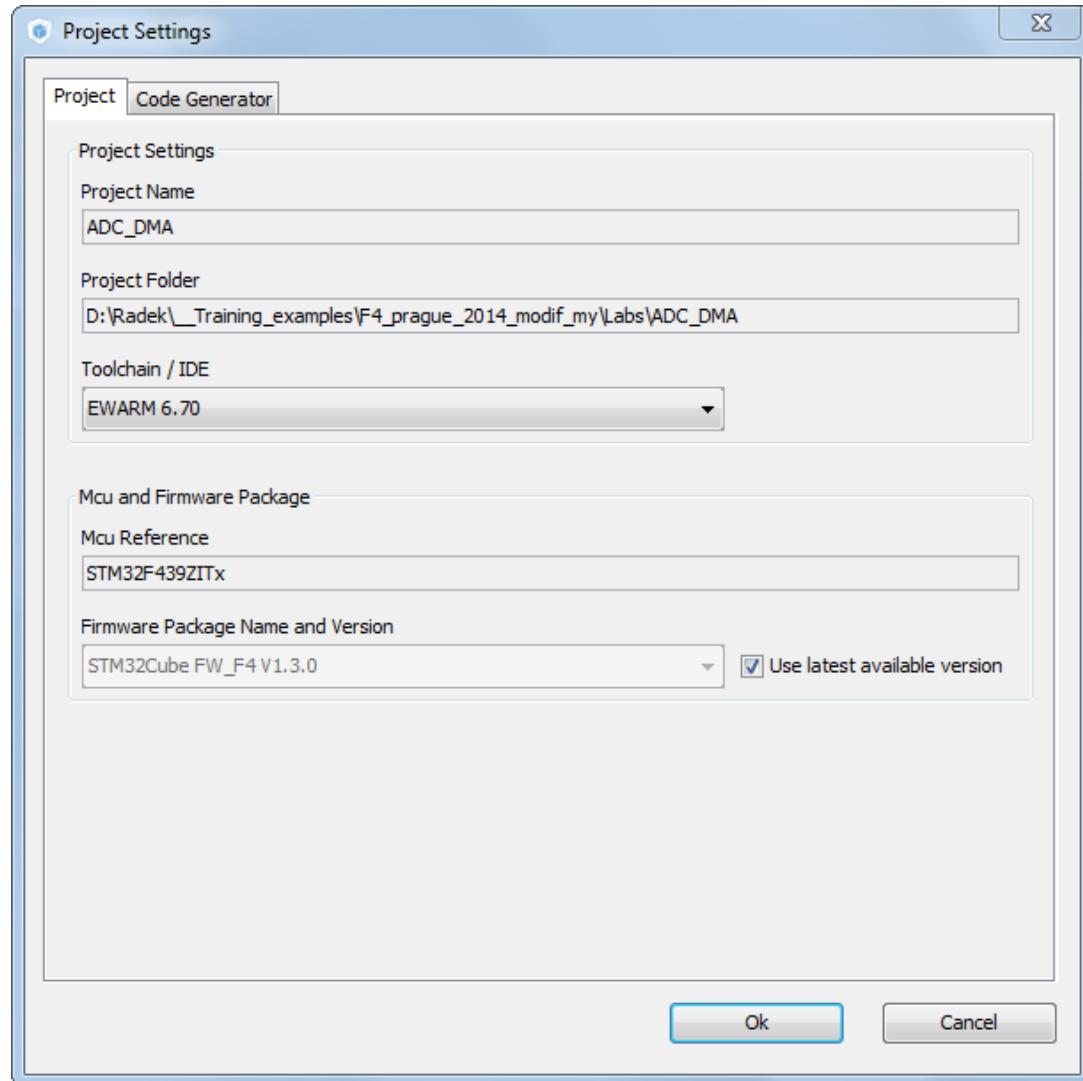


- CubeMX ADC configuration

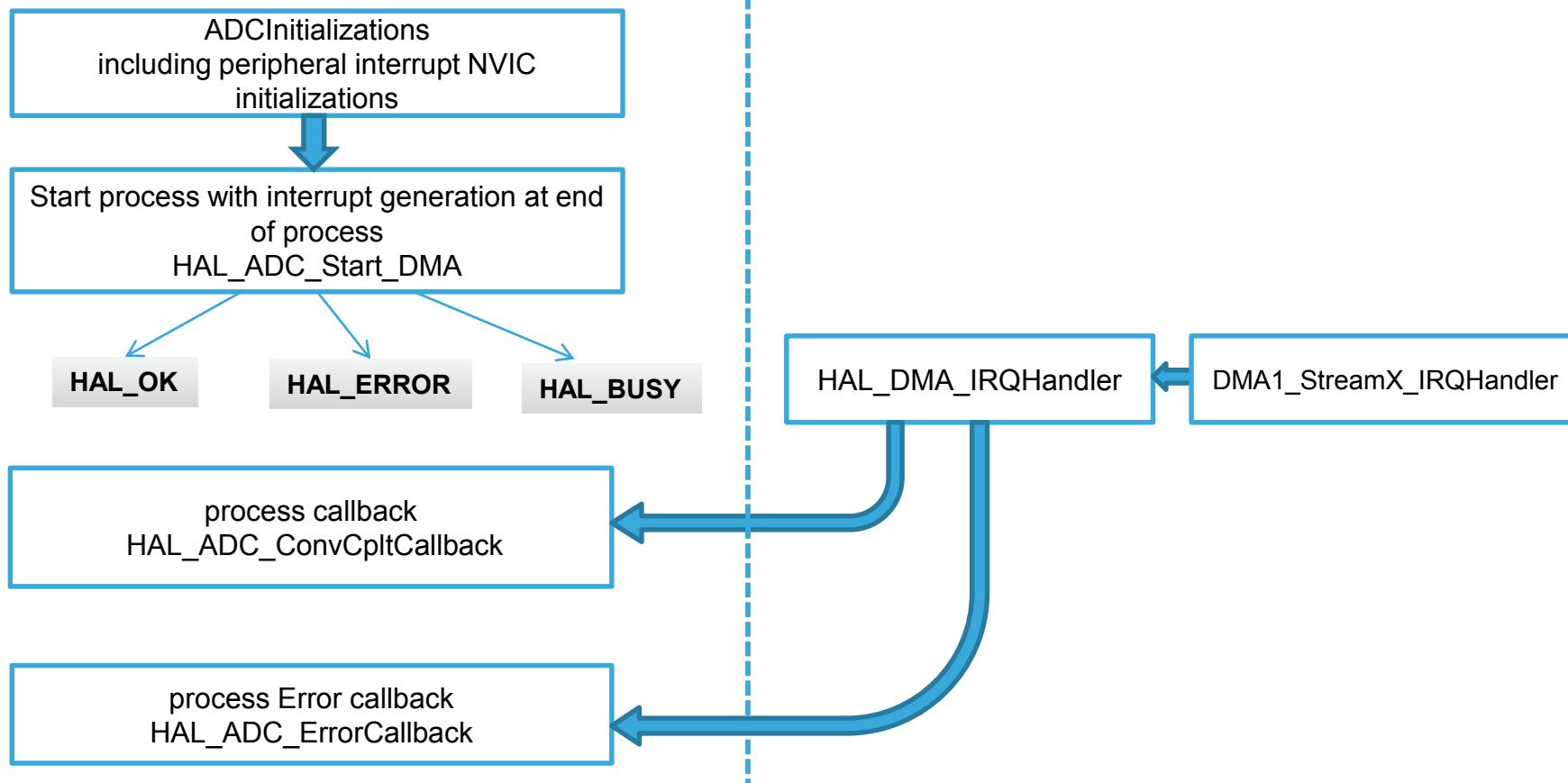
- TAB>DMA Settings
- Button ADD
- DMA request ADC1
- Peripheral to memory direction
- Circular mode
- Memory increment
- Half word data width
- Button OK



- 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 Library ADC with DMA flow



- Open the project in our IDE
  - The functions we want to put into main.c
  - Between */\* USER CODE BEGIN 2 \*/* and */\* USER CODE END 2 \*/* tags
  - and */\* USER CODE BEGIN 3 \*/* and */\* USER CODE END 3 \*/* tags
- For DAC start use function
  - `HAL_ADC_Start_DMA(ADC_HandleTypeDef* hadc, uint32_t* pData, uint32_t Length)`
- DAC functions
  - `HAL_DAC_Start(DAC_HandleTypeDef* hdac, uint32_t Channel)`
  - `HAL_DAC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)`

- Solution

- Variables

```
/* USER CODE BEGIN PV */  
uint32_t value_adc;  
uint32_t value_dac=0;  
/* USER CODE END PV */
```

- DAC setup and start ADC/DAC

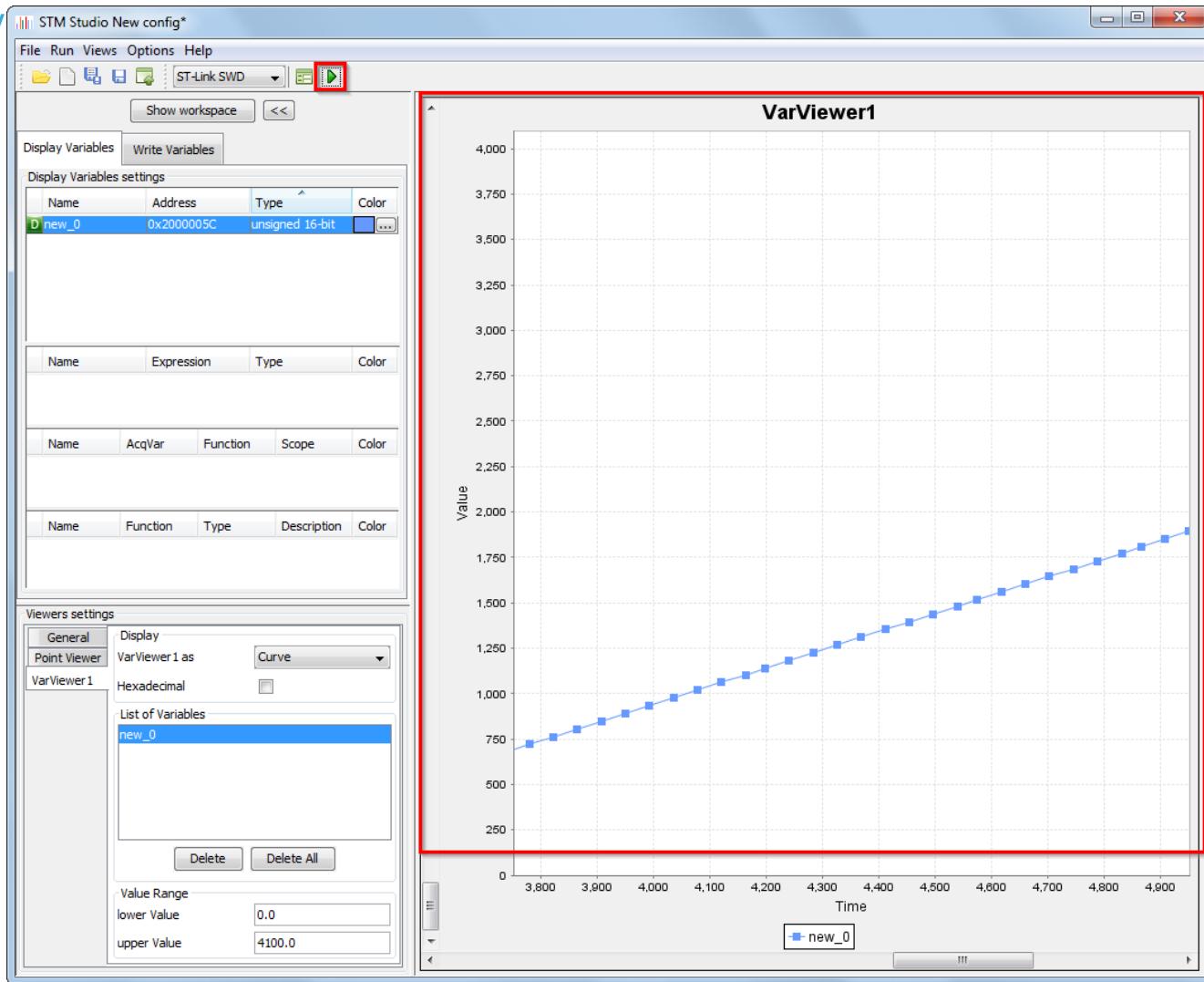
```
/* USER CODE BEGIN 2 */  
HAL_DAC_Start(&hdac,DAC_CHANNEL_2);  
HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);  
HAL_ADC_Start_DMA(&hadc1,(uint32_t*)&value_adc,1);  
/* USER CODE END 2 */
```

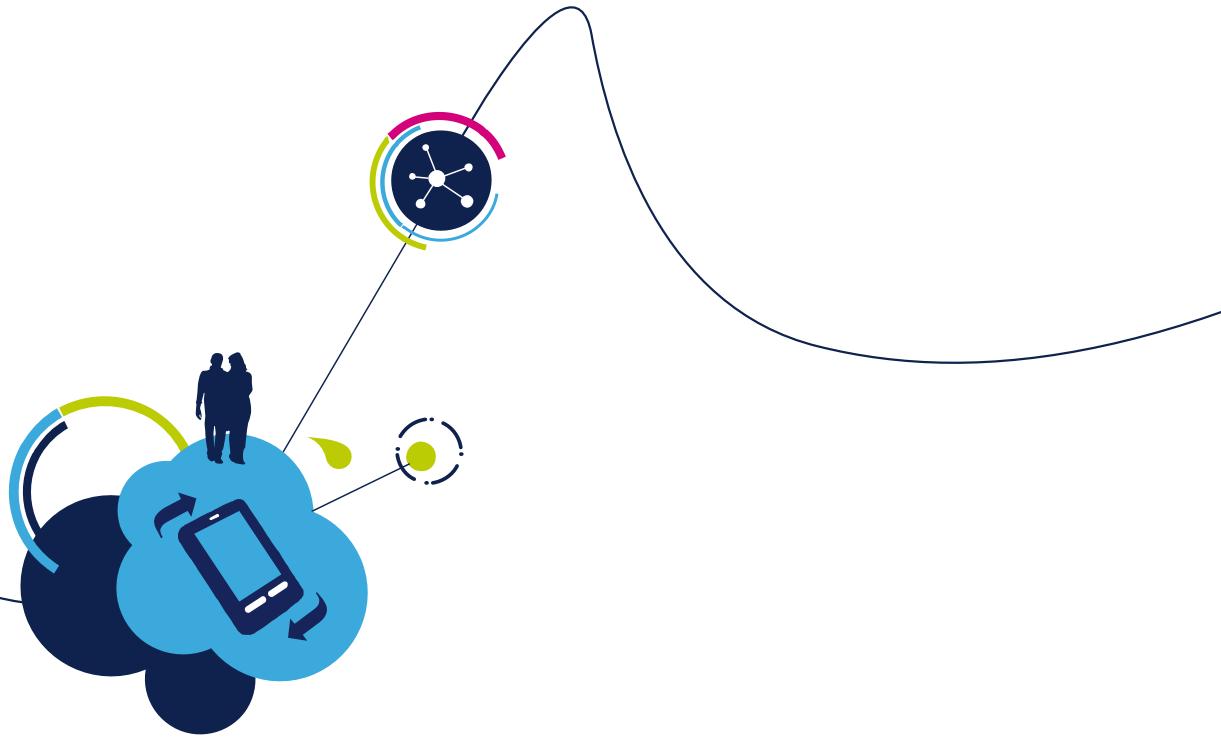
- Solution

- ADC main routine

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);  
    value_dac++;  
    if(value_dac>4095){  
        value_dac=0;  
    }  
    HAL_Delay(5);  
    HAL_ADC_Start(&hadc1);  
    HAL_Delay(5);  
}  
/* USER CODE END 3 */
```

- STM studio settings
  - Check functionality again with STMstudio





# WWDG lab 23

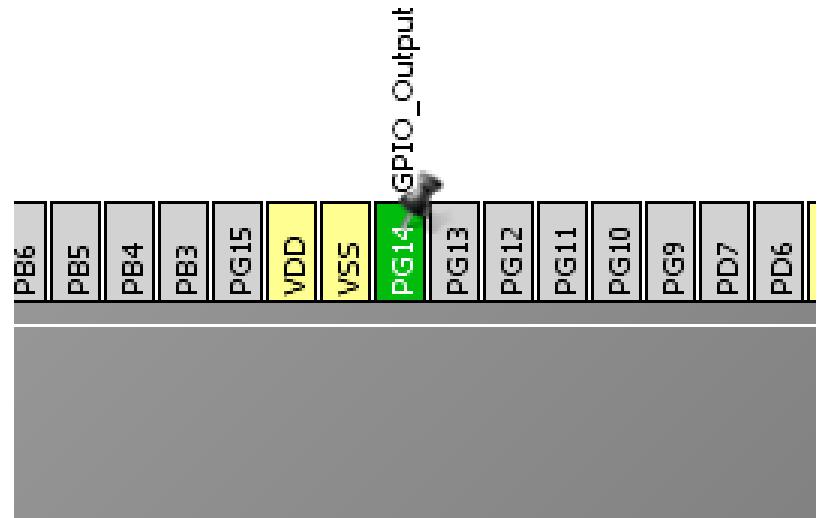
- Objective

- Learn how to setup WWDG in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Create simple application to test WWDG

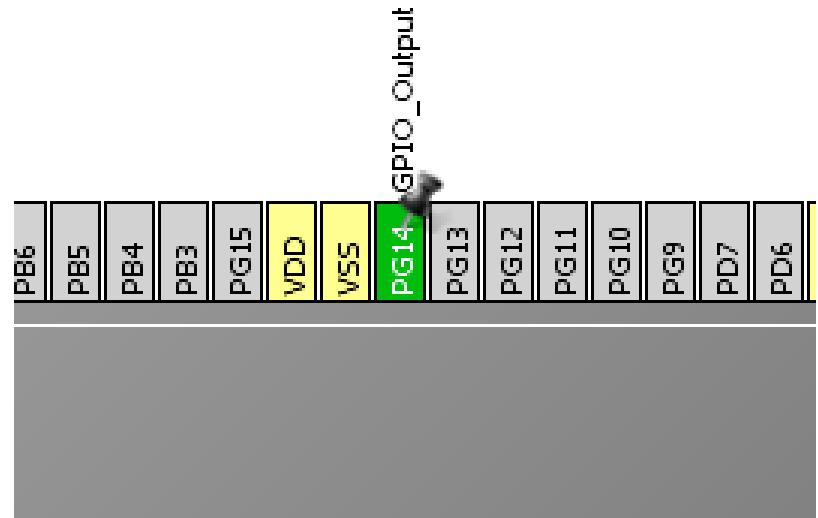
- Goal

- Configure WDGIN in CubeMX and Generate Code
- Learn how to start WWDG
- WWDG indication via LED

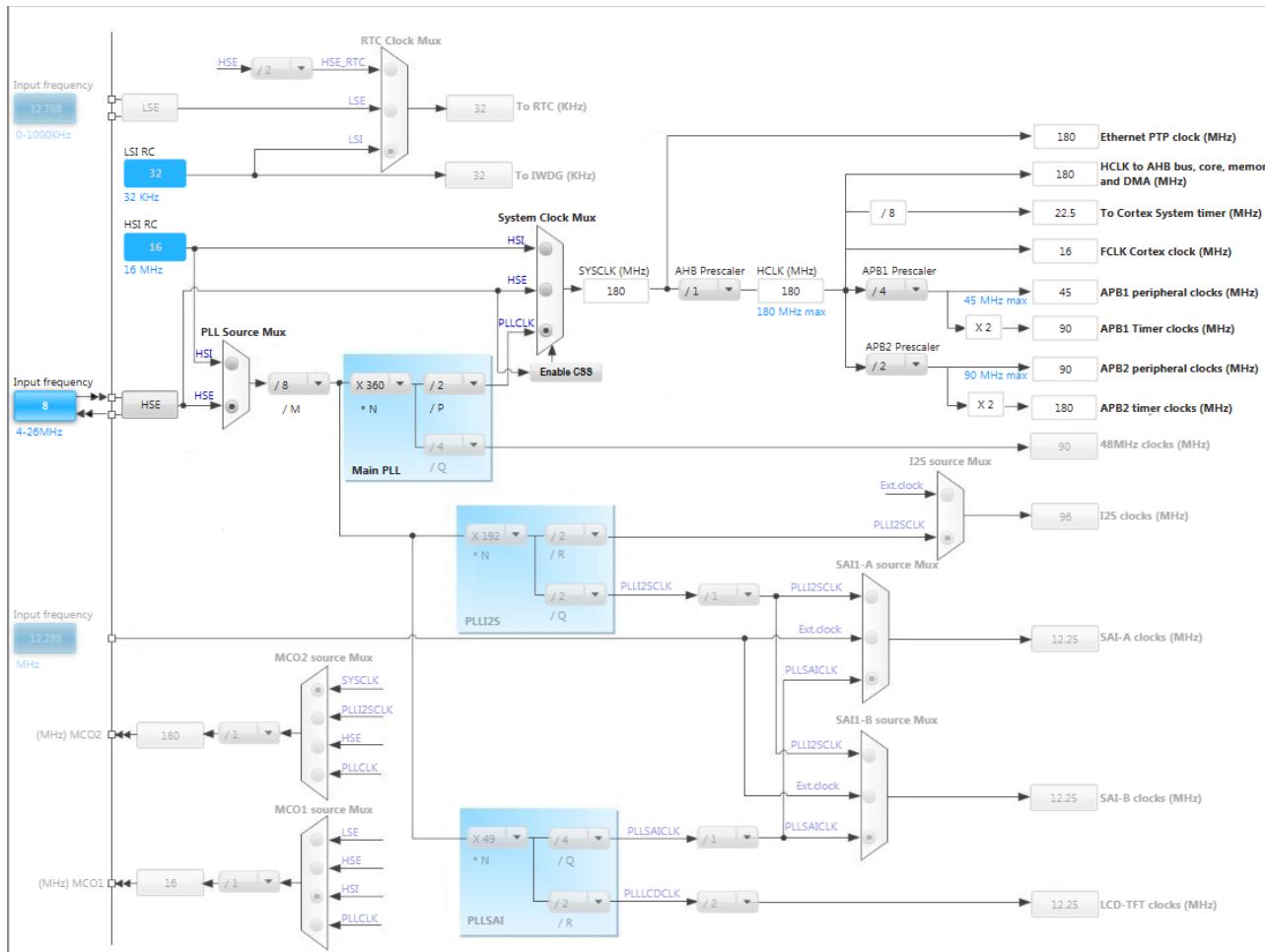
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX WWDG selection
  - Select WWDG
  - Configure PG14 for LED indication



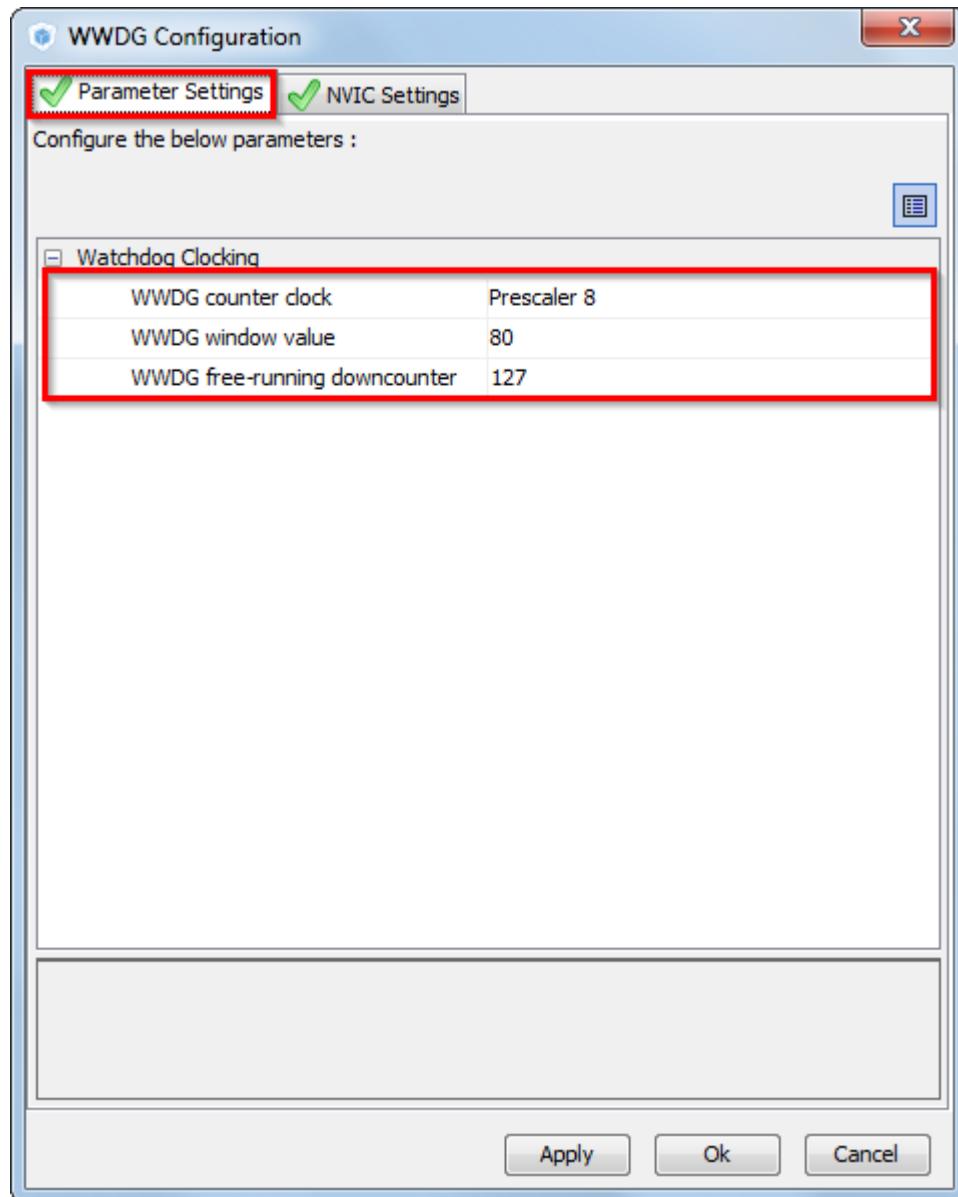
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX WWDG selection
  - Select WWDG
  - Configure PG14 for LED indication



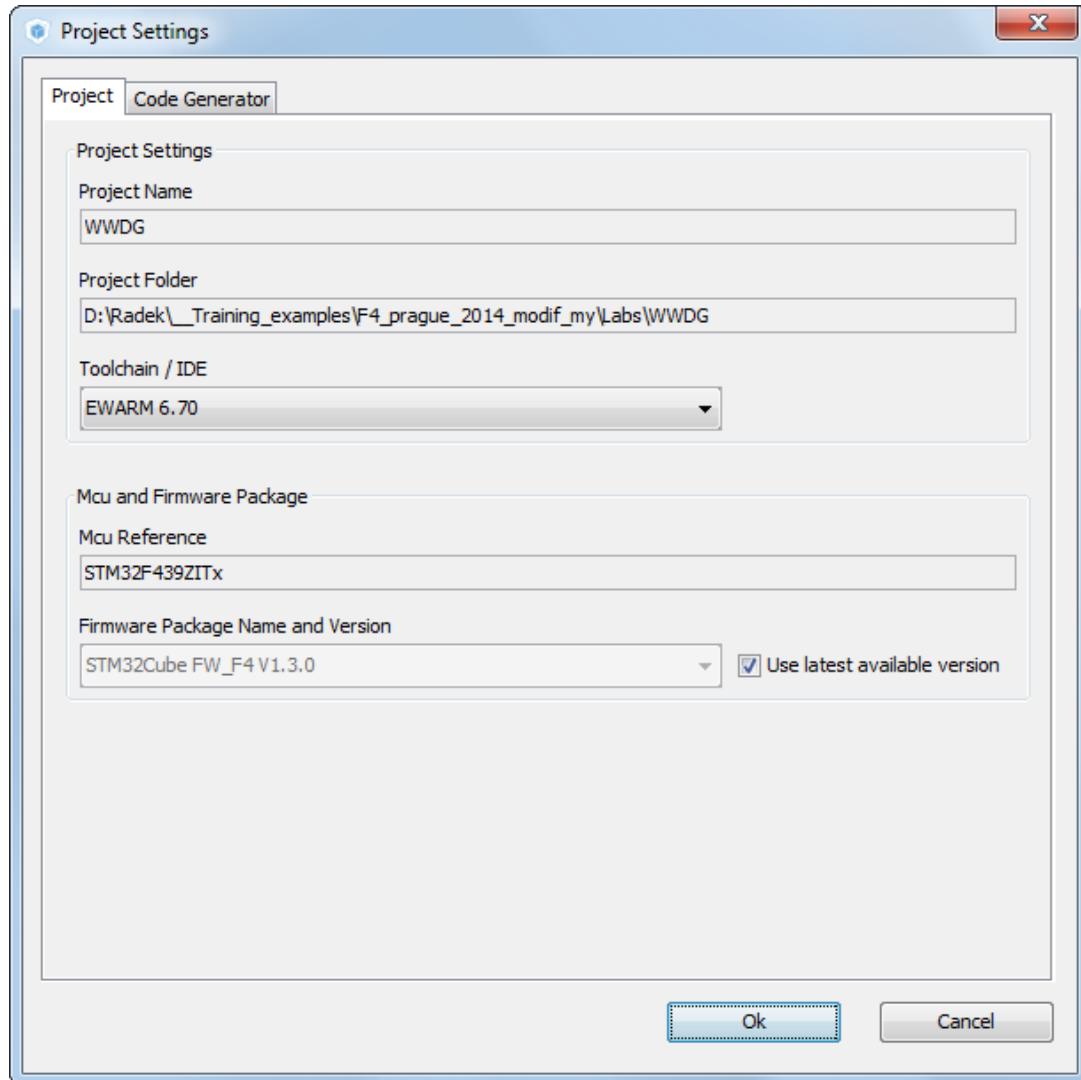
- In order to run on maximum frequency, setup clock system
- Details in lab 0



- CubeMX ADC configuration
  - TAB>Configuration>System>>WWDG>Parameter Settings
  - Set prescaller to 8
  - WWDG window to 80
  - And free running counter to 127
  - Button OK



- 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



- How calculate the window APB1<sub>freq</sub>=45MHz, prescaler 8

$$t_{wwdg_{min}} = f_{APB1} * 4096 * N_{WWDG\_PRESCALLER} * (N_{REFRESH} - N_{WINDOW}) =$$

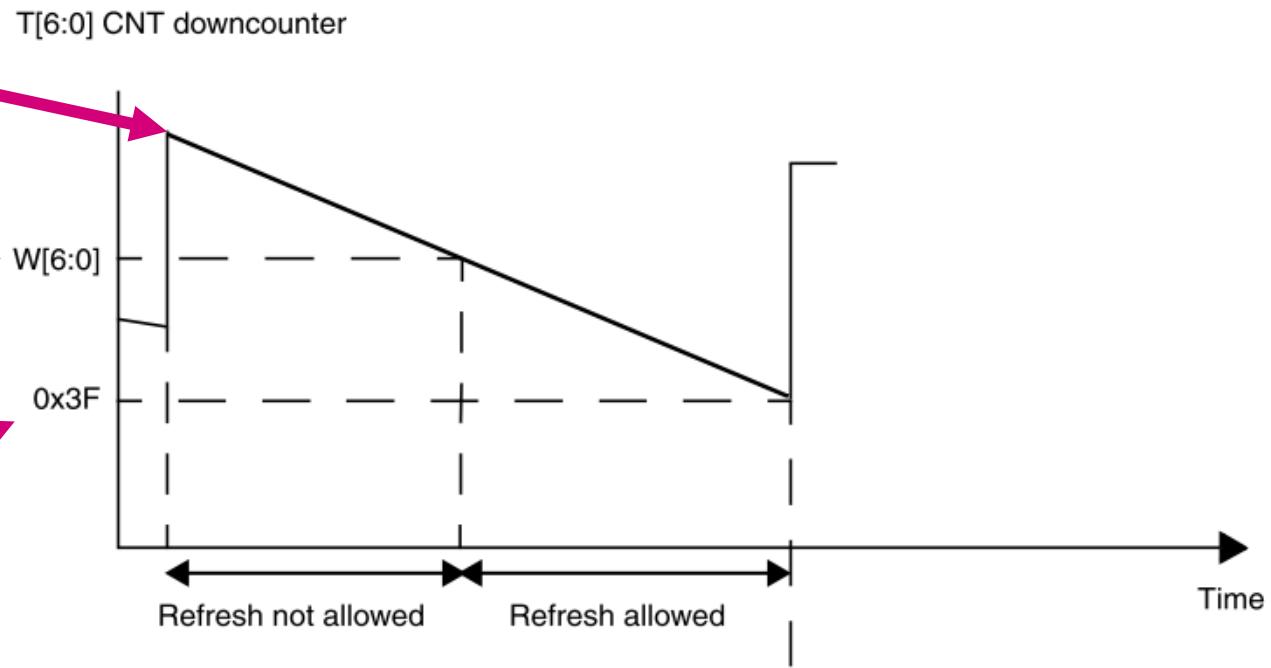
$$= \left( \frac{1}{45 * 10^6} \right) * 4096 * 8 * (127 - 80) = 34.2ms$$

$$t_{wwdg_{max}} = \left( \frac{1}{45 * 10^6} \right) * 4096 * 8 * (127 - 63) = 46.6ms$$

We refresh the  
WWDG to 127

In our case 80

Fixed 63



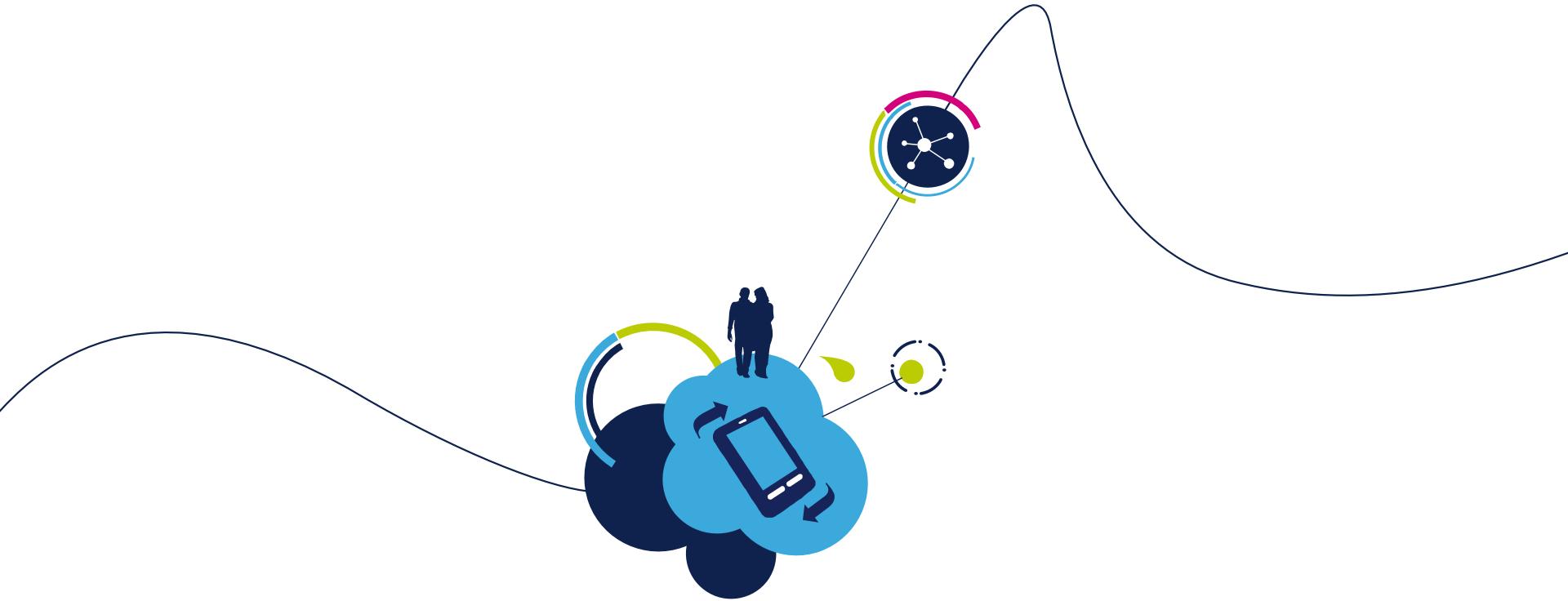
- Solution

- WWDG Start

```
/* USER CODE BEGIN 2 */  
HAL_WWDG_Start(&hwwdg);  
/* USER CODE END 2 */
```

- WWDG refresh

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    //30ms or 50ms is outside the WWDG window, 40ms fits inside the window  
    HAL_Delay(40);  
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);  
    HAL_WWDG_Refresh(&hwwdg, 127);  
}  
/* USER CODE END 3 */
```



# IWDG lab 24

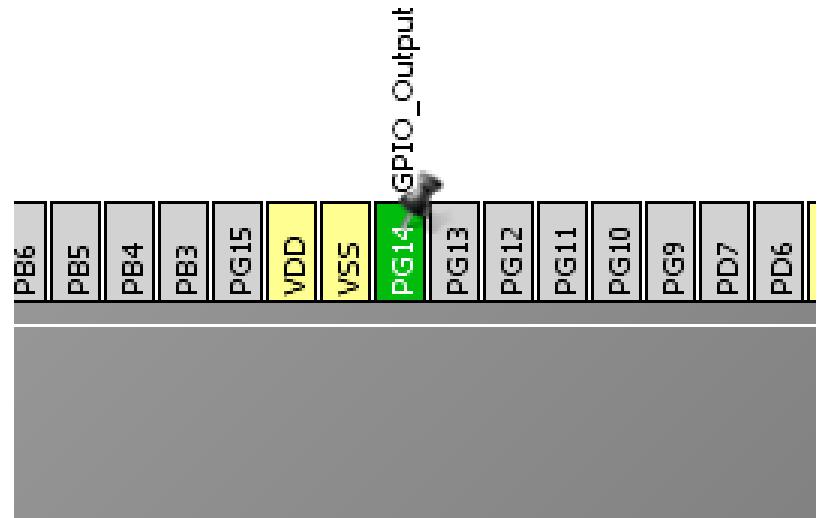
- Objective

- Learn how to setup IWDG in CubeMX
- How to Generate Code in CubeMX and use HAL functions
- Create simple application to test IWDG

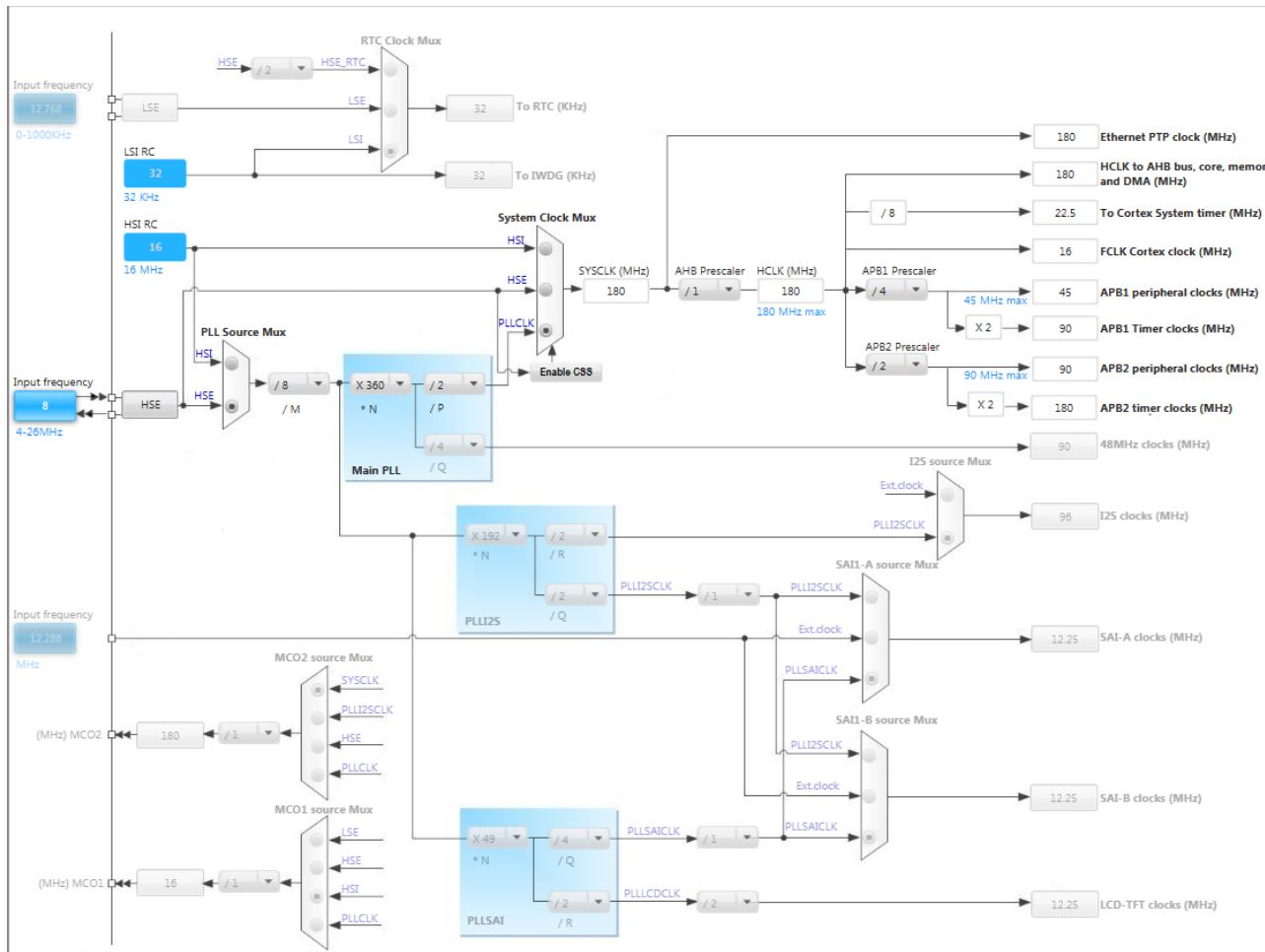
- Goal

- Configure IWDG in CubeMX and Generate Code
- Learn how to start IWDG
- IWDG indication via LED

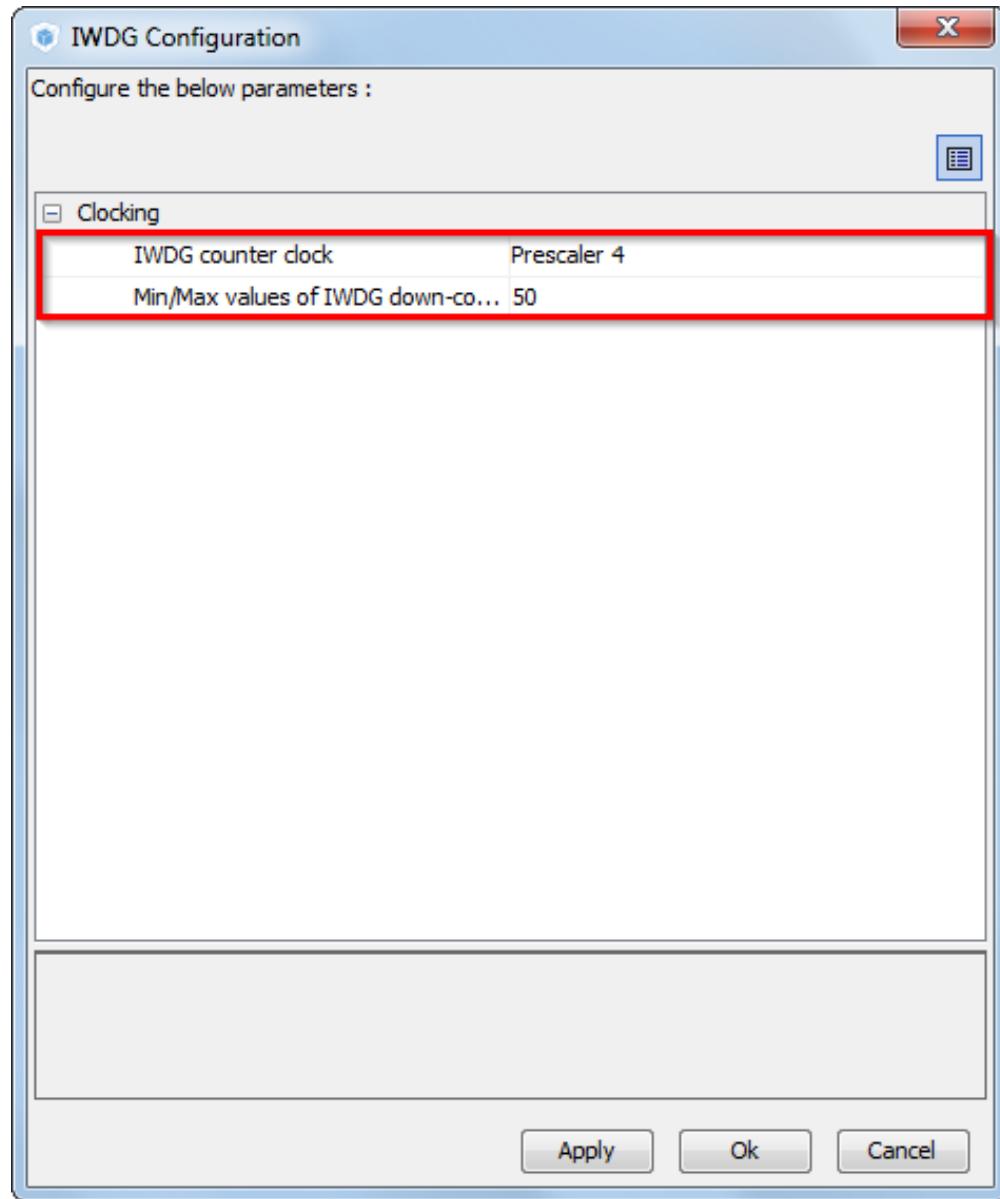
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX IWDG selection
  - Select IWDG
  - Configure PG14 for LED indication



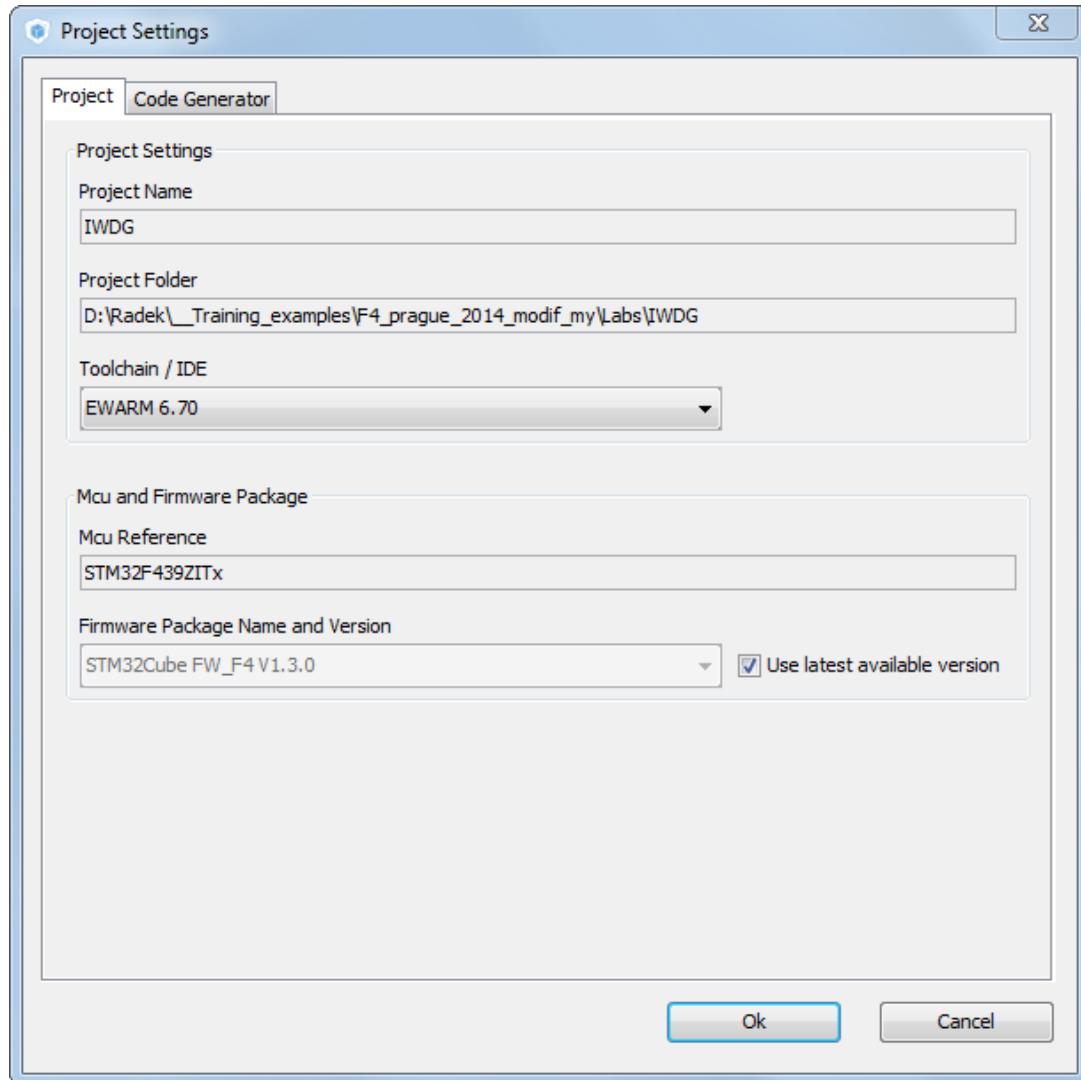
- In order to run on maximum frequency, setup clock system
- Details in lab 0



- CubeMX ADC configuration
  - TAB>Configuration>System>>IWDG>Parameter Settings
  - Set prescaller to 4
  - Max value to 50
  - Button OK



- 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



- IWDG refresh interval

$$t_{\text{iwdg}} = \frac{1}{f_{LSI}} * P_{IWDG\_PRESCALLER} * N_{IWDG\_COUNTERVAL} = \left(\frac{1}{32 * 10^3}\right) * 4 * 50 = 6.25\text{ms}$$

- Solution

- IWDG Start

```
/* USER CODE BEGIN 2 */  
HAL_IWDG_Start(&hiwdg);  
/* USER CODE END 2 */
```

- IWDG refresh

```
/* USER CODE BEGIN 3 */  
/* Infinite loop */  
while (1)  
{  
    HAL_Delay(7); //try delay 6ms and 7ms  
    HAL_GPIO_WritePin(GPIOG,GPIO_PIN_14,GPIO_PIN_SET);  
    HAL_IWDG_Refresh(&hiwdg);  
}  
/* USER CODE END 3 */
```

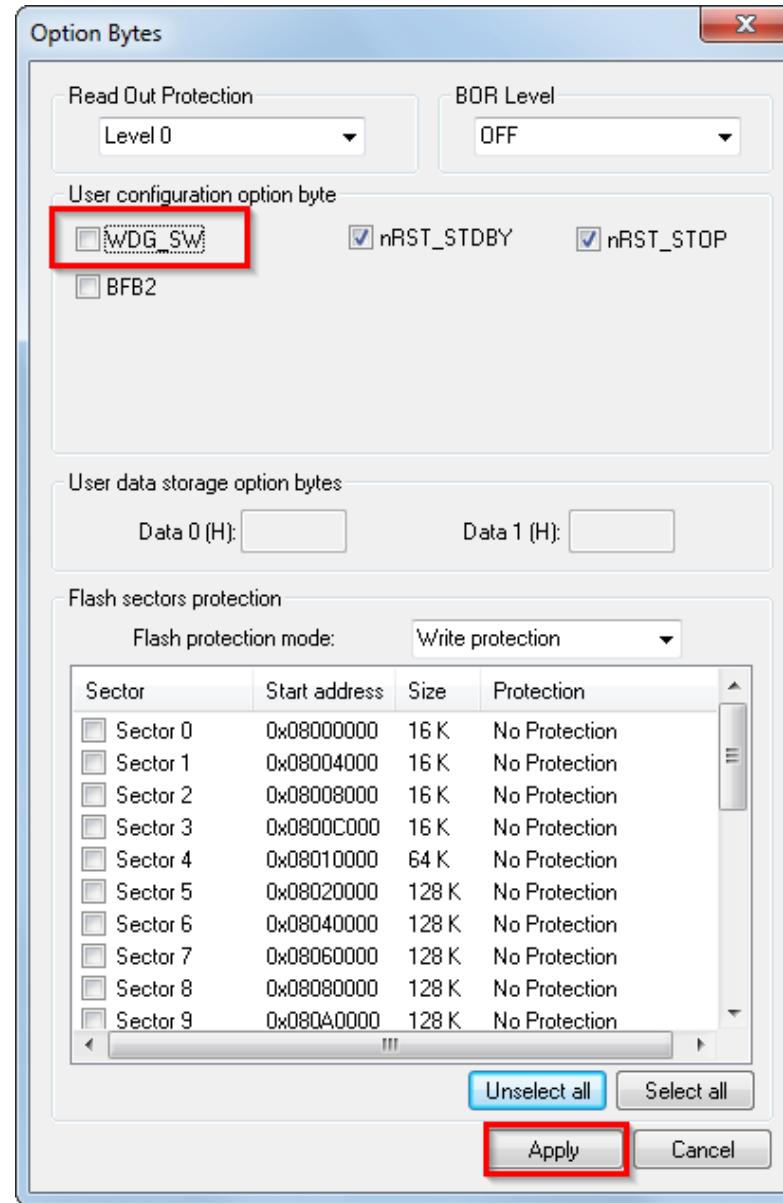
- Hardware IWDG

- Remove IWDG start from project

```
/* USER CODE BEGIN 2 */  
/* USER CODE END 2 */
```

- Use ST-Link utility and enable IWDG Hardware start

- Hardware IWDG
  - Start ST-Link utility
  - Menu>Target>Option bytes or CTRL+B
  - Uncheck the WDG\_WS
  - Button APPLY
  - Now the IWDG is automatically started after reset
- !!! DO NOT FORGET disable IWDG automatic start after you end this example





# BSP SDRAM lab 25

# 25 Use BSP for SDRAM initialization

364

- Objective

- Learn how import BSP into project
- Which part need to by configured in GUI
- Try to write data into SDRAM and read it

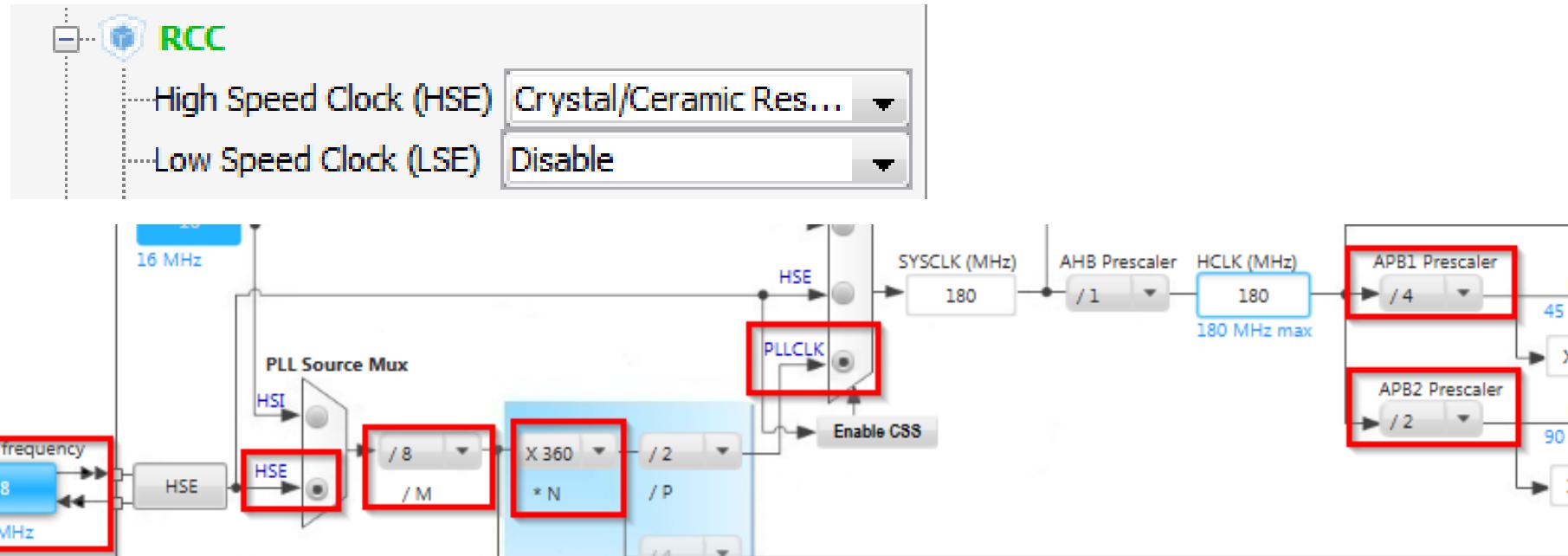
- Goal

- Successfully import BSP into your project
- Learn which part you need to import
- How to setup the project

# 25 Use BSP for SDRAM initialization

365

- 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



# 25

# Use BSP for SDRAM initialization

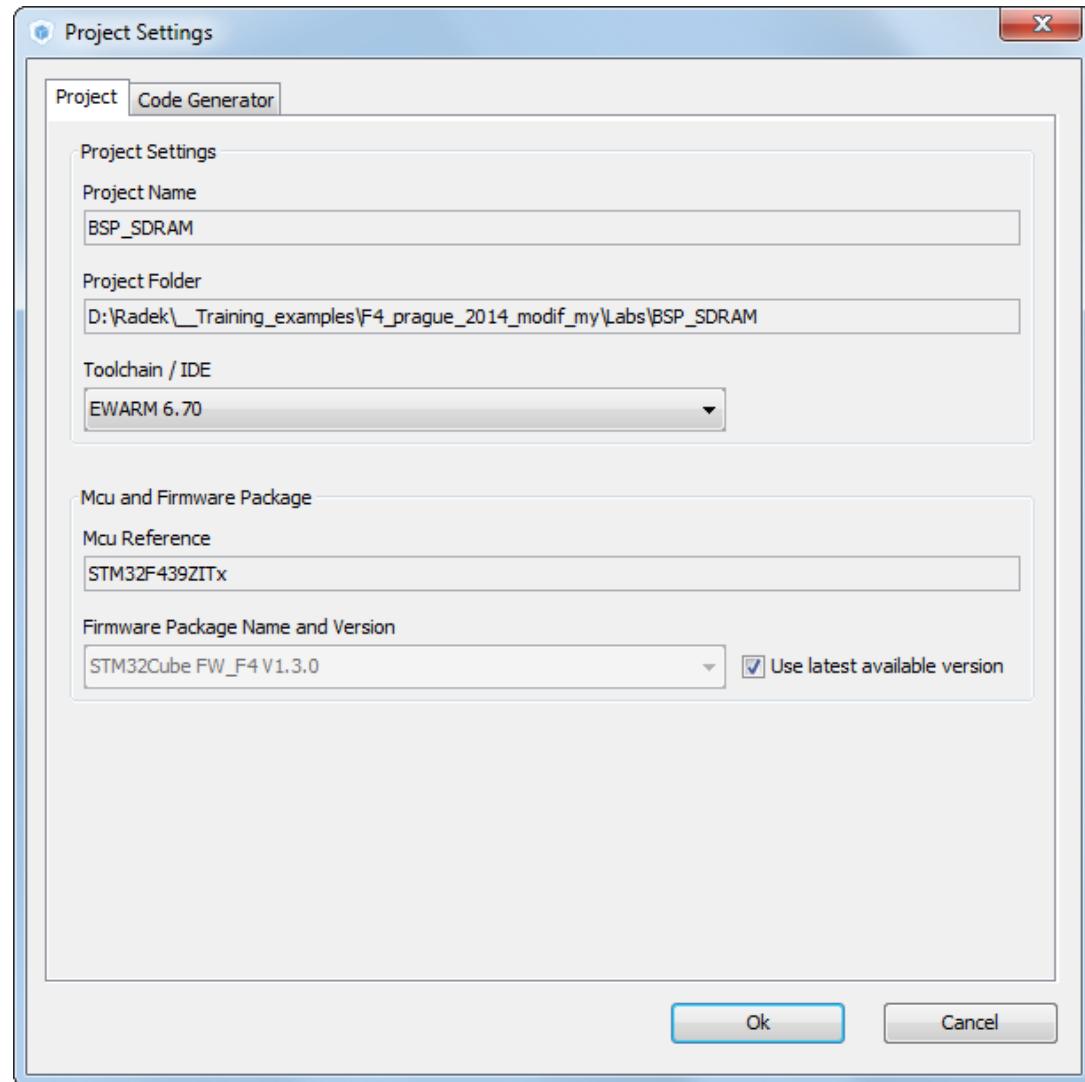
366

- 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

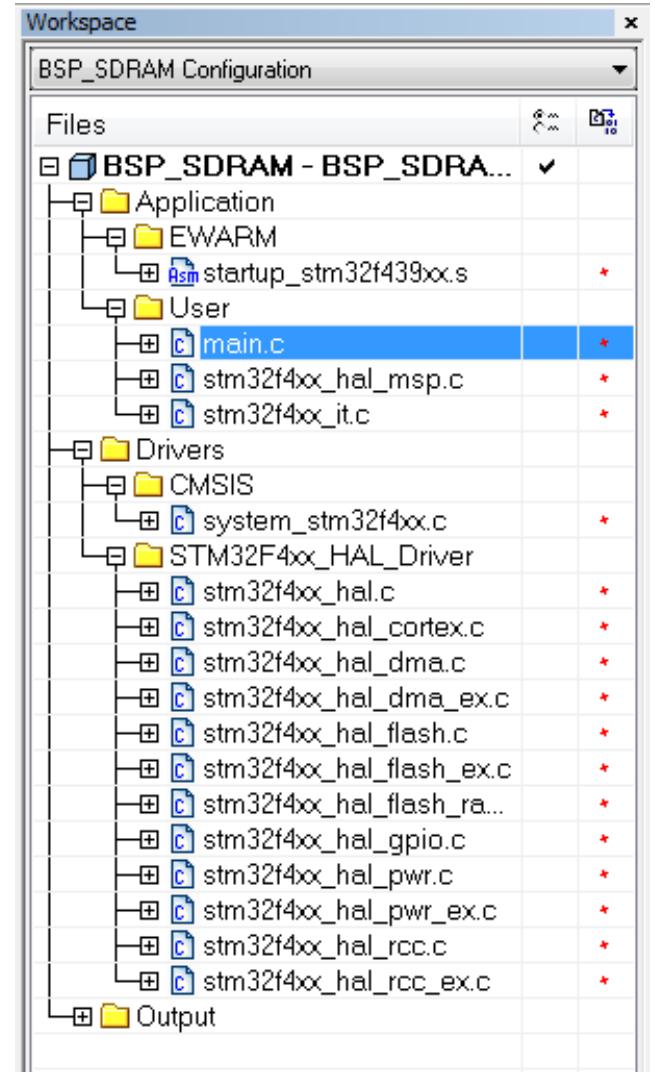
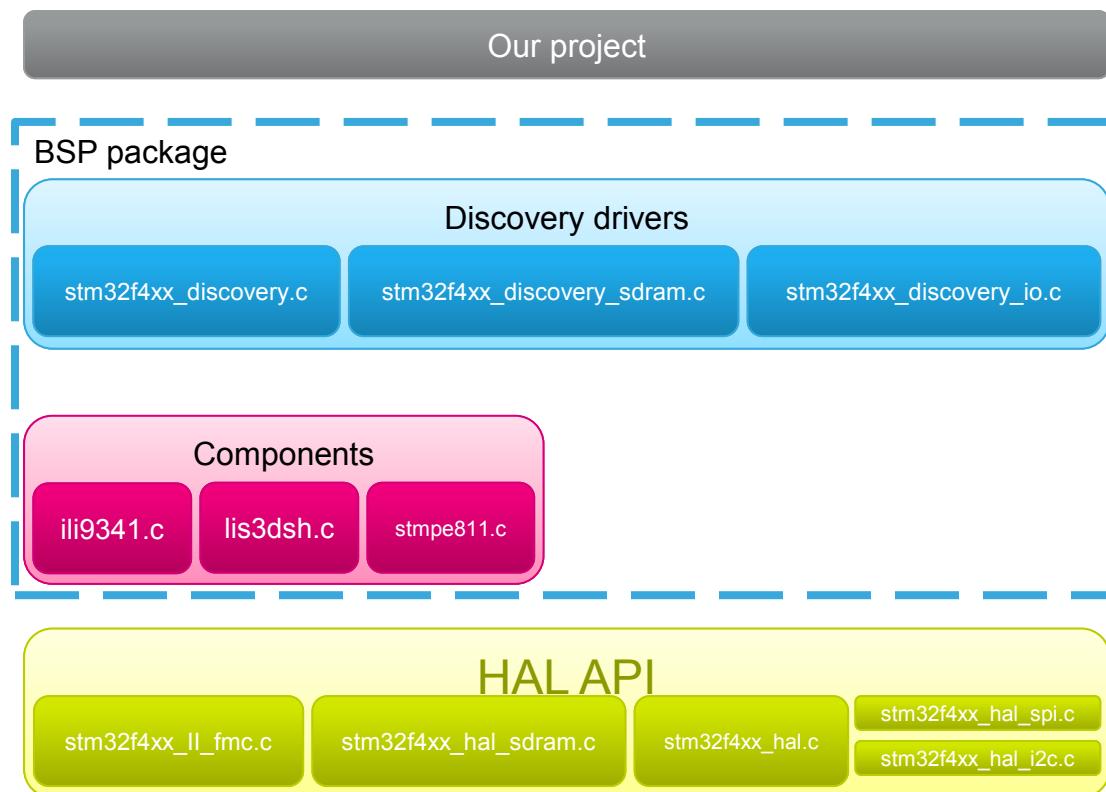
- Menu > Project > Generate Code



# 25 Use BSP for SDRAM initialization

367

- Now we have bank project
- For SDRAM we need to include more parts



## BSP SDRAM organization

## Our project

## BSP package

## Discovery drivers

`stm32f4xx_discovery.c``stm32f4xx_discovery_sdram.c``stm32f4xx_discovery_io.c`

## Components

`ili9341.c``lis3dsh.c``stmpe811.c`

## HAL API

`stm32f4xx_ll_fmc.c``stm32f4xx_hal_sdram.c``stm32f4xx_hal.c``stm32f4xx_hal_spi.c``stm32f4xx_hal_i2c.c`

# 25

# Use BSP for SDRAM initialization

369

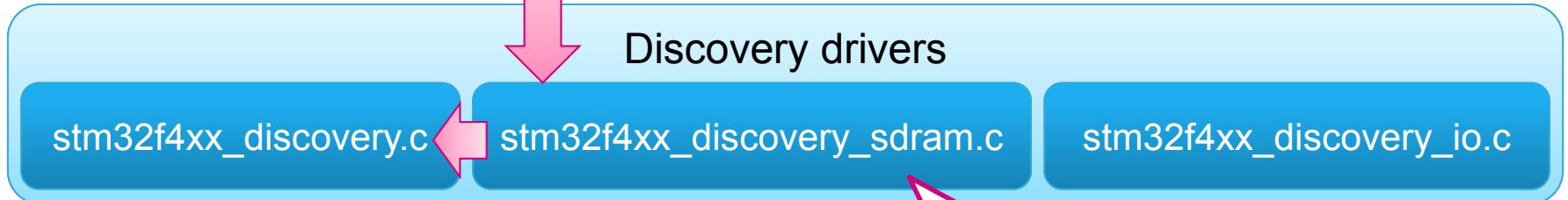
BSP SDRAM organization

Our project

1. include

`stm32f4xx_discovery_sdram.h`

BSP package



Components

`ili9341.c`

`lis3dsh.c`

`stmpe811.c`

2. include

`stm32f4xx_discovery.h`

HAL API

`stm32f4xx_ll_fmc.c`

`stm32f4xx_hal_sdram.c`

`stm32f4xx_hal.c`

`stm32f4xx_hal_spi.c`

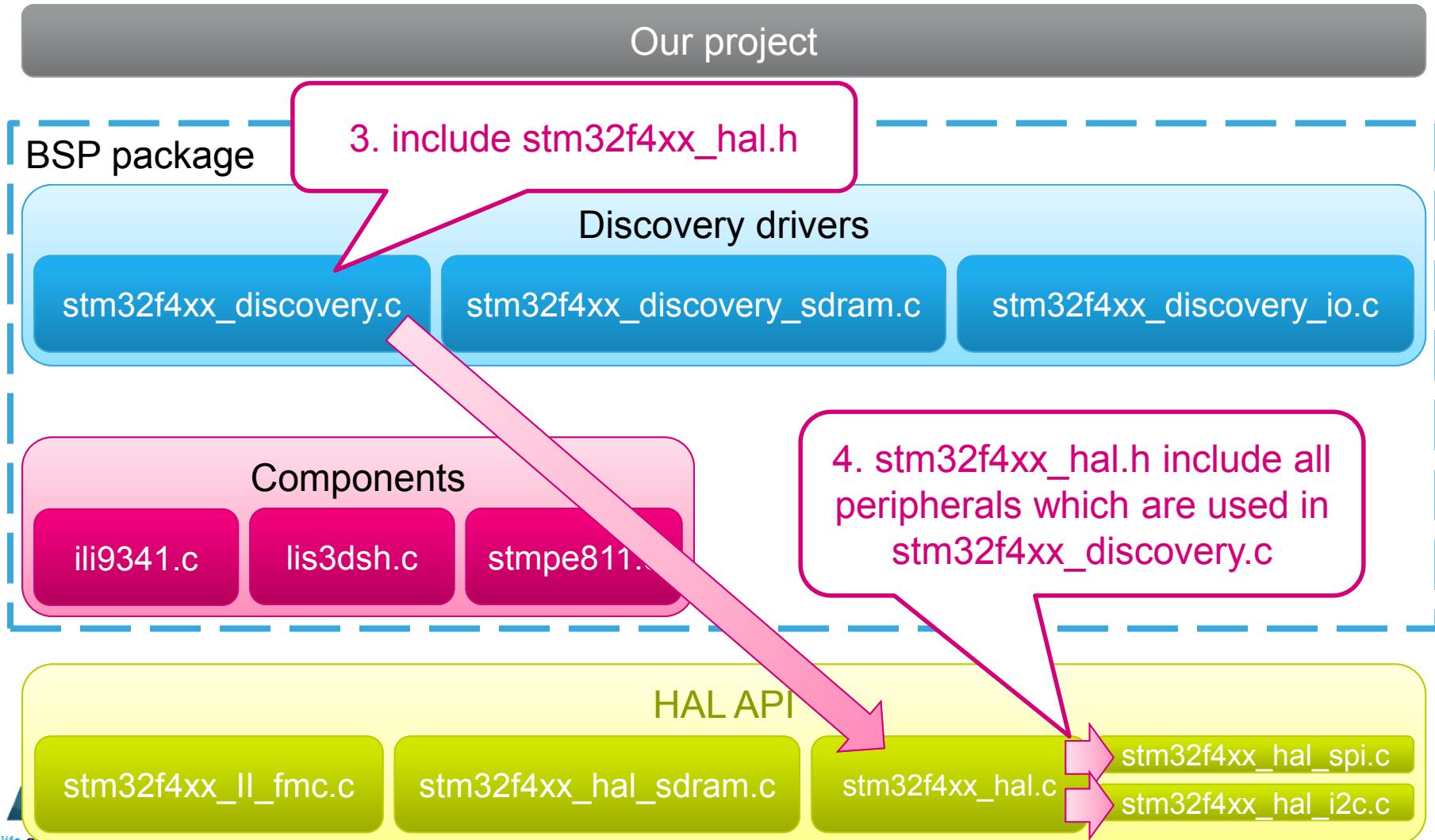
`stm32f4xx_hal_i2c.c`

# 25

# Use BSP for SDRAM initialization

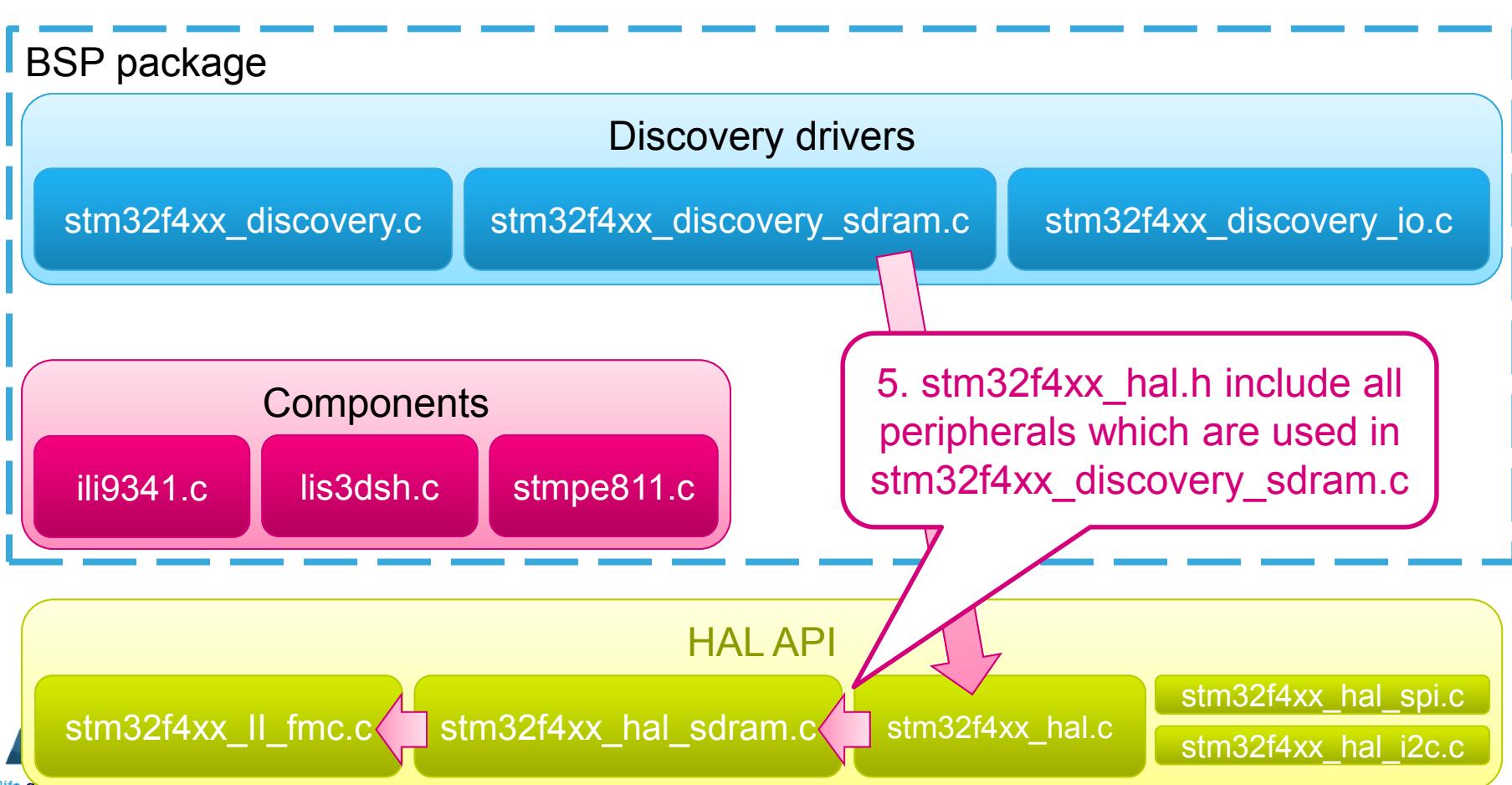
370

## BSP SDRAM organization



## BSP SDRAM organization

## Our project

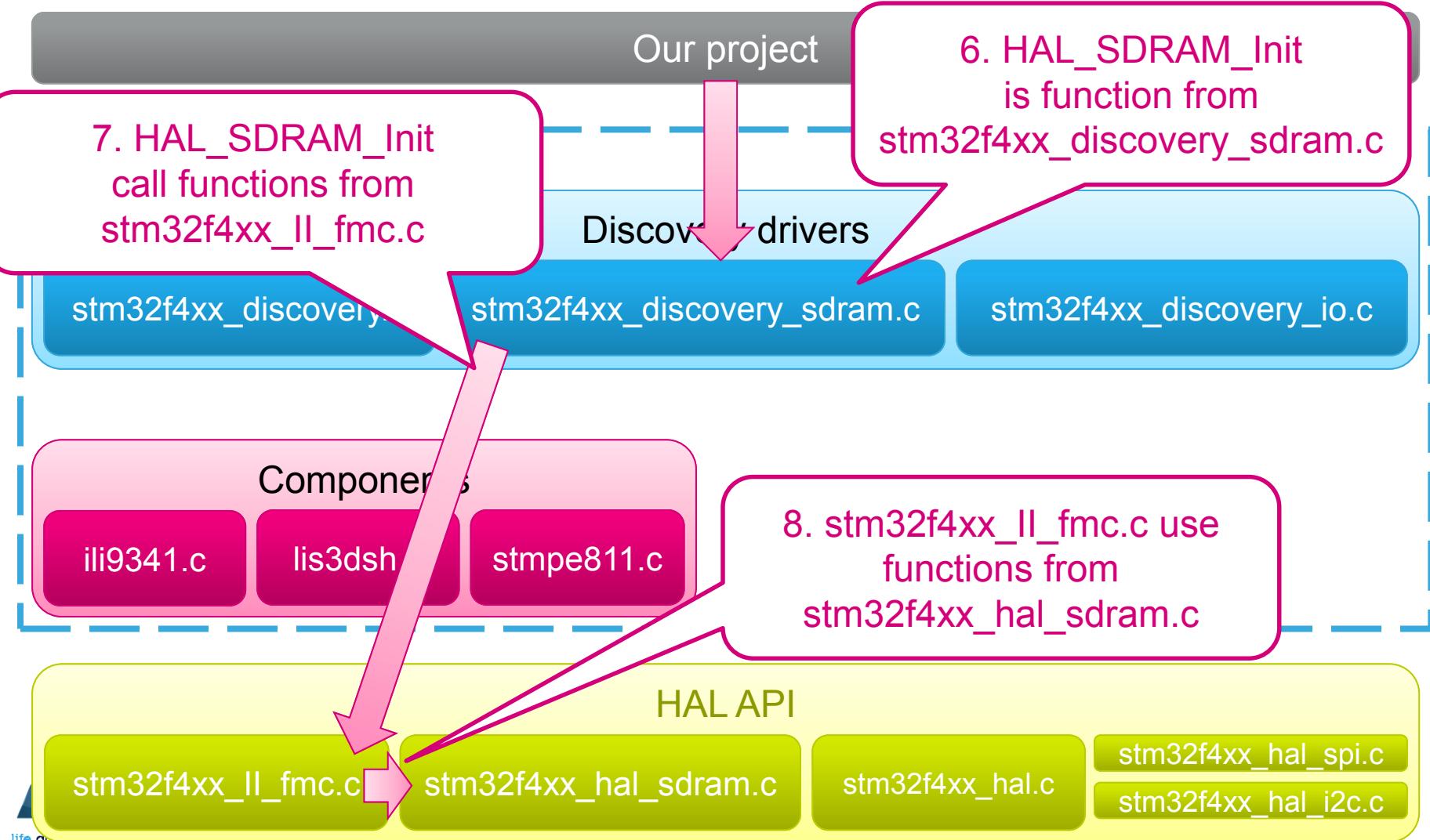


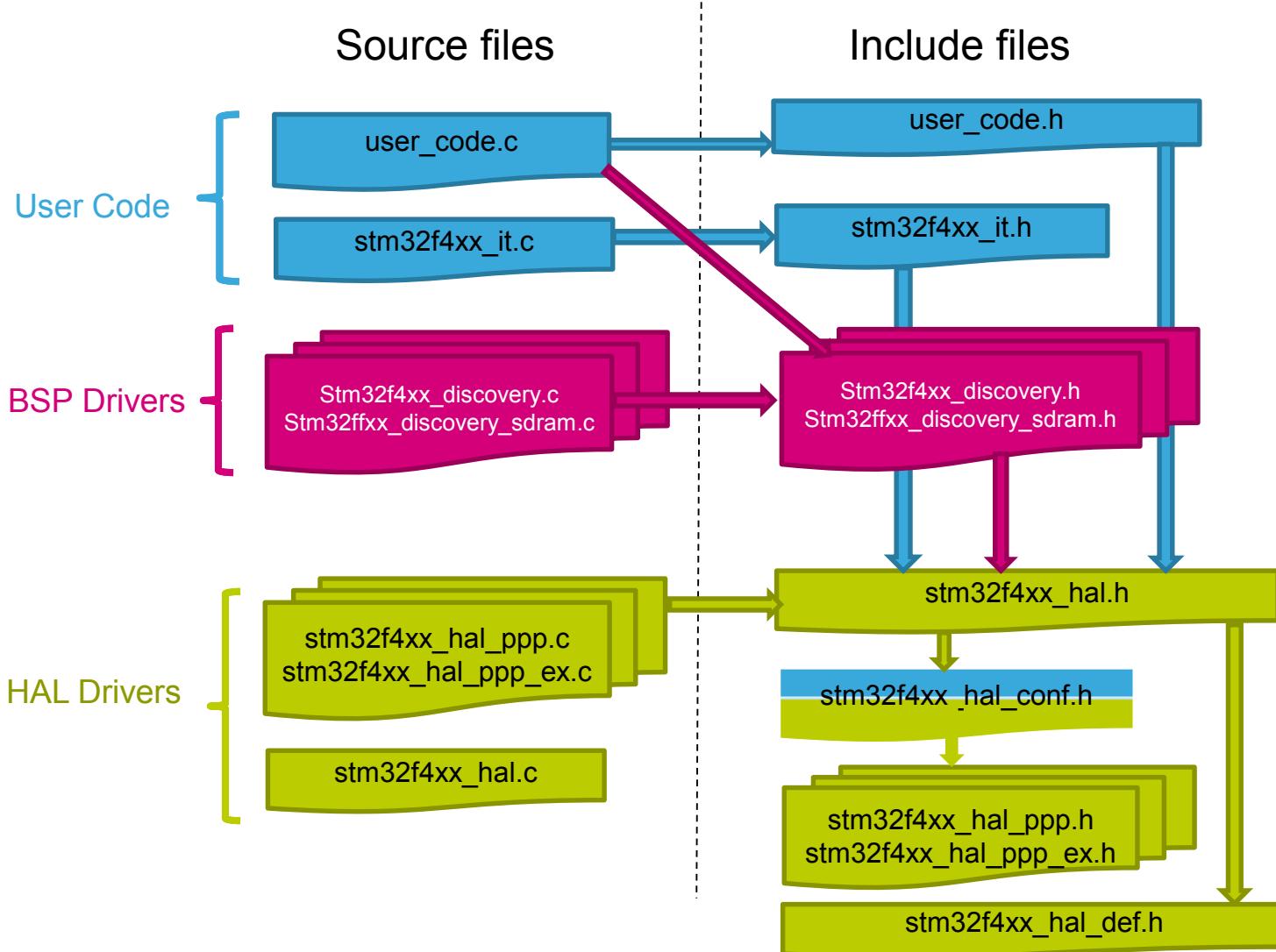
# 25

# Use BSP for SDRAM initialization

372

## BSP SDRAM organization

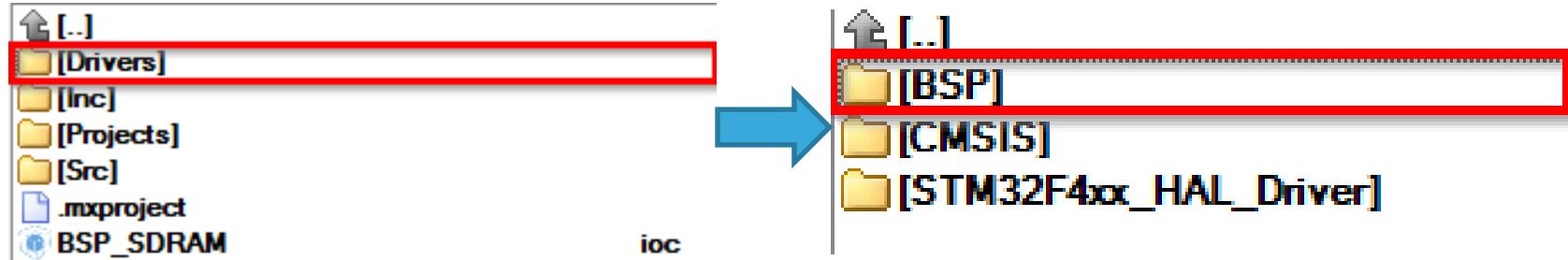




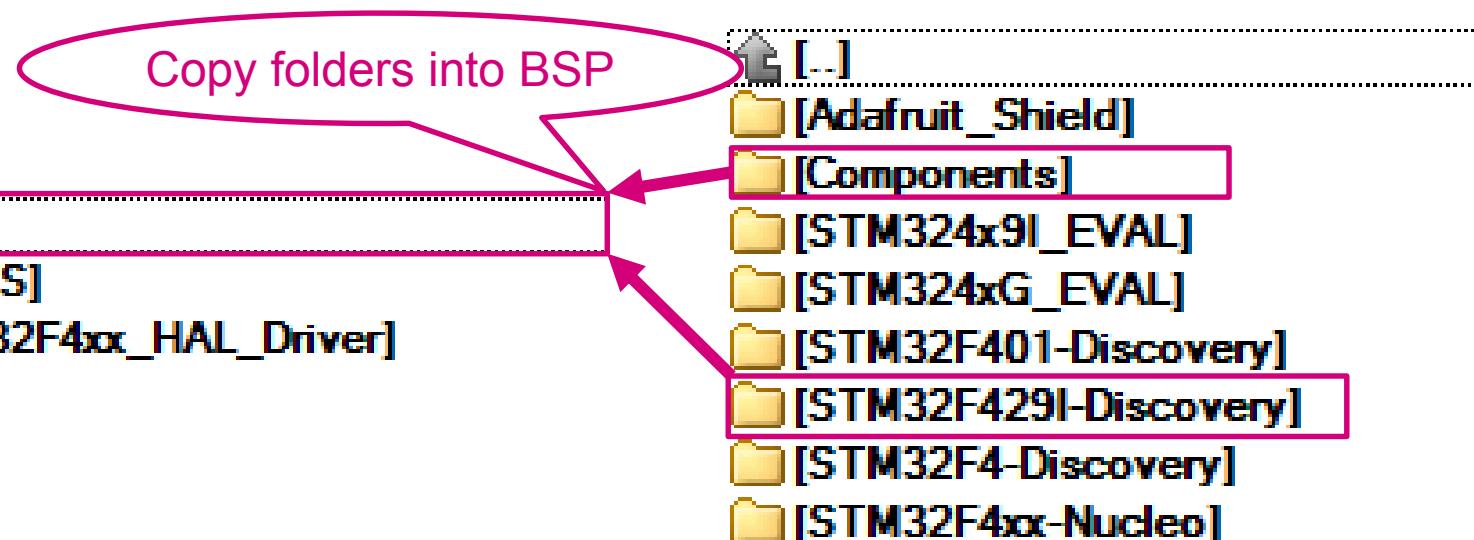
# 25 Use BSP for SDRAM initialization

374

- The copy part
  - In our project in Drivers folder create folder BSP



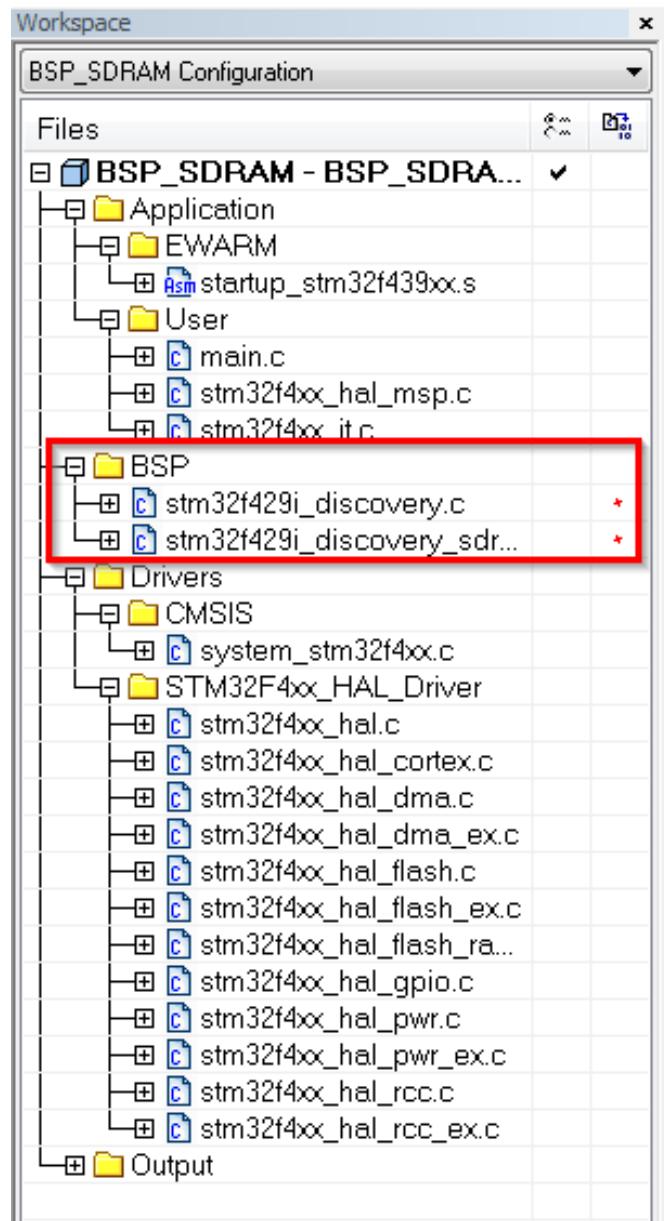
- Now go into CubeMX repository ...\\STM32Cube\_FW\_F4\_V1.3.0\\Drivers\\BSP\\
- And copy Components and STM32F429I-Discovery into BSP folder



# 25 Use BSP for SDRAM initialization

375

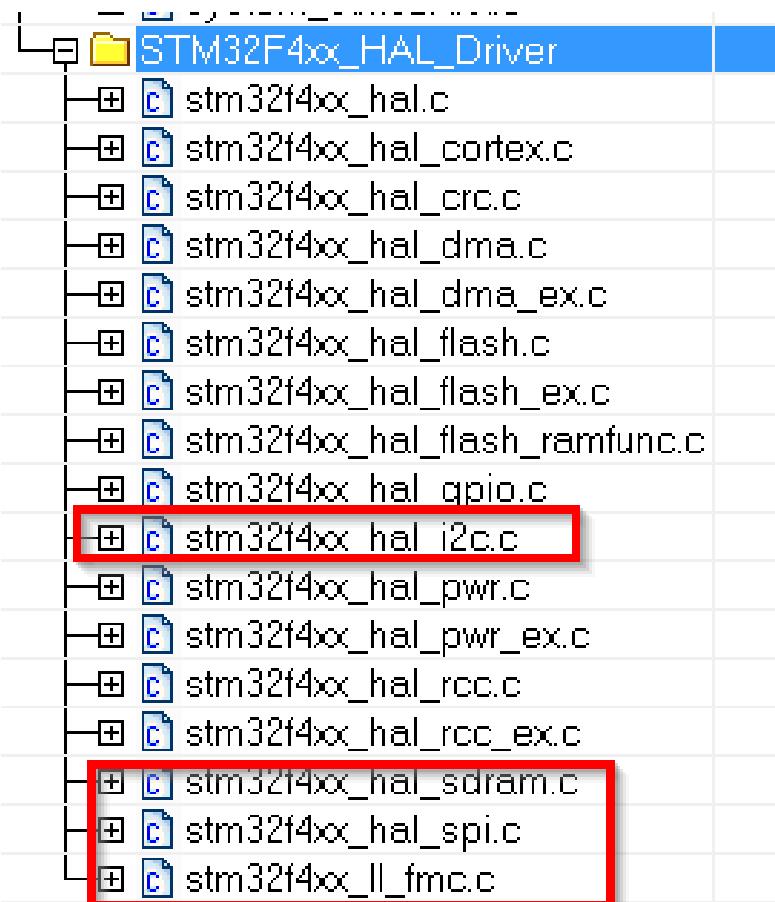
- Now we need to add this files also in project
  - Create BSP folder in project
  - Right click on project in Workplace>ADD>Group
  - Name it BSP
  - Now right click on BSP>ADD>Files
  - From Drivers\BSP\STM32F429I-Discovery\ add stm32f429i\_discovery.c and stm32f429i\_discovery\_sdram.c



# 25 Use BSP for SDRAM initialization

376

- The `stm32f429i_discovery.c` contains functions for all components on discovery kit (LCD, GYRO,...)
- Then we also need add into project HAL library which handle their interface (I2C, SPI, ... )
- Right click on `STM32F4xx_HAL_Drive`>ADD from `\Drivers\STM32F4xx_HAL_Driver\Src`
  - `stm32f4xx_hal_i2c.c`
  - `stm32f4xx_hal_spi.c`
  - `stm32f4xx_hal_sdram.c`
  - `stm32f4xx_ll_fmc.c`



# 25 Use BSP for SDRAM initialization

377

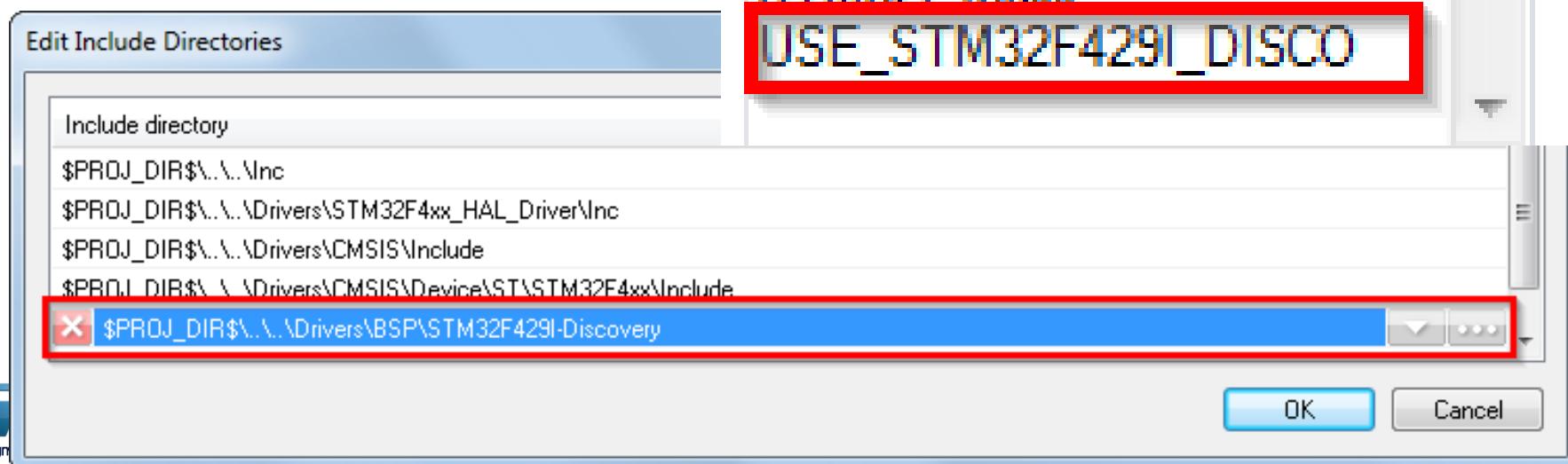
- Now add the include paths for this new files
  - Right click on project>Options>Category C/C++Compiler>Preprocesor
  - Into Defined symbols add USE\_STM32F429I\_DISCO
  - This allow use BSP functions
  - Into additional includes add  
\$PROJ\_DIR\$\..\..\Drivers\BSP\STM32F429I-Discovery
  - Button OK
  - Button OK close project options

Defined symbols: (one per line)

USE\_HAL\_DRIVER

STM32F439xx

USE\_STM32F429I\_DISCO



- Now last thing is allow to include new HAL files which we added
  - Open `stm32f4xx_hal_conf.h` in ..\Inc\
  - Uncomment files which we added
    - `HAL_SDRAM_MODULE_ENABLED`
    - `HAL_I2C_MODULE_ENABLED`
    - `HAL_SPI_MODULE_ENABLED`

```
/* ##### 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
//#define HAL_PCCARD_MODULE_ENABLED
//#define HAL_SRAM_MODULE_ENABLED
#define HAL_SDRAM_MODULE_ENABLED
//#define HAL_HASH_MODULE_ENABLED
#define HAL_I2C_MODULE_ENABLED
//#define HAL_I2S_MODULE_ENABLED
//#define HAL_IWDG_MODULE_ENABLED
//#define HAL_LTDC_MODULE_ENABLED
//#define HAL_RNG_MODULE_ENABLED
//#define HAL_RTC_MODULE_ENABLED
//#define HAL_SAI_MODULE_ENABLED
//#define HAL_SD_MODULE_ENABLED
#define HAL_SPI_MODULE_ENABLED
//#define HAL_TIM_MODULE_ENABLED
//#define HAL_UART_MODULE_ENABLED
//#define HAL_USART_MODULE_ENABLED
//#define HAL_IRDA_MODULE_ENABLED
//#define HAL_SMARTCARD_MODULE_ENABLED
//#define HAL_WWDG_MODULE_ENABLED
//#define HAL_PCD_MODULE_ENABLED
//#define HAL_HCD_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED
```

# 25 Use BSP for SDRAM initialization

379

- Into main.c now we add include of stm32f429i\_discovery\_sdram.h

```
/* USER CODE BEGIN Includes */  
#include "stm32f429i_discovery_sdram.h"  
/* USER CODE END Includes */
```

- Now we can use the SDRAM init functions from BSP

```
/* USER CODE BEGIN 2 */  
BSP_SDRAM_Init();  
/* USER CODE END 2 */
```

- Now you can try to write into SDRAM area

In stm32f429i\_discovery\_sdram.h you can find where is the SDRAM memory and how is their size

- SDRAM\_DEVICE\_ADDR ((uint32\_t)0xD0000000)
- SDRAM\_DEVICE\_SIZE ((uint32\_t)0x800000) /\* SDRAM device size in MBytes \*/

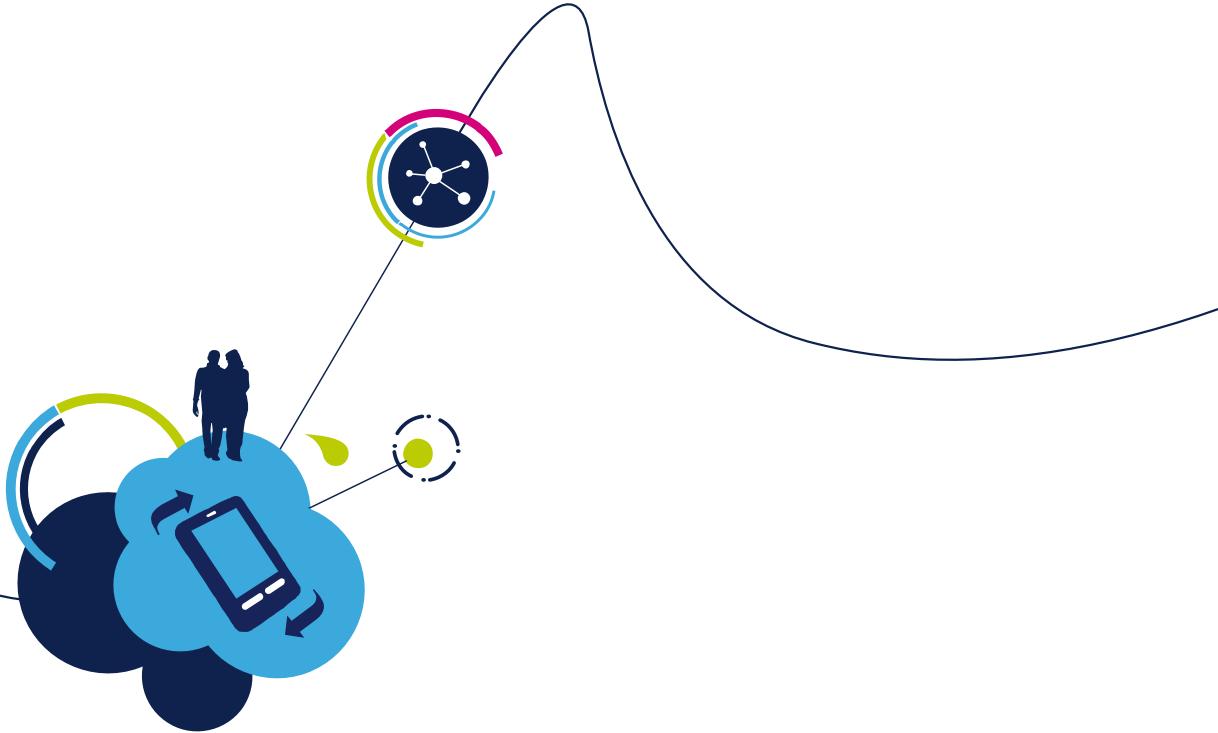
# 25 Use BSP for SDRAM initialization

380

- SDRAM test

```
/* USER CODE BEGIN PV */  
volatile uint32_t value;  
/* USER CODE END PV */
```

```
/* USER CODE BEGIN 2 */  
BSP_SDRAM_Init();  
*((uint32_t*)SDRAM_DEVICE_ADDR)=0x12345678;  
value=*((uint32_t*)SDRAM_DEVICE_ADDR);  
/* USER CODE END 2 */
```



# BSP LCD lab 26

# 26 Use BSP for LCD init and writing

382

- Objective

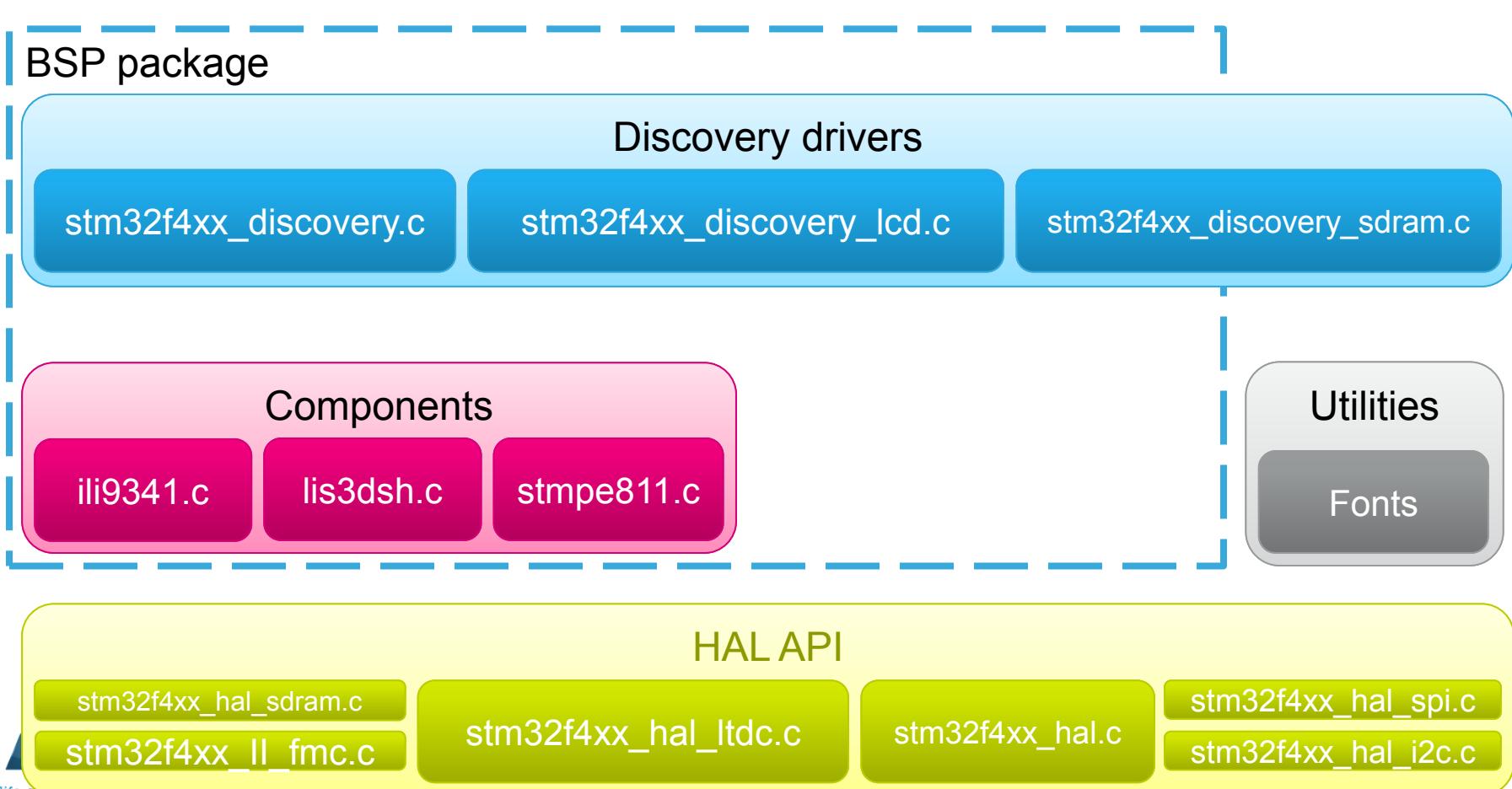
- Learn how import BSP LCD into project
- Because the LCD use the SDRAM we use project from lab 25
- Which part need to by configured in GUI
- Try to write text on LCD

- Goal

- Successfully import BSP LCD into your project
- Learn which part you need to import
- How to setup the project

## BSP LCD organization

## Our project



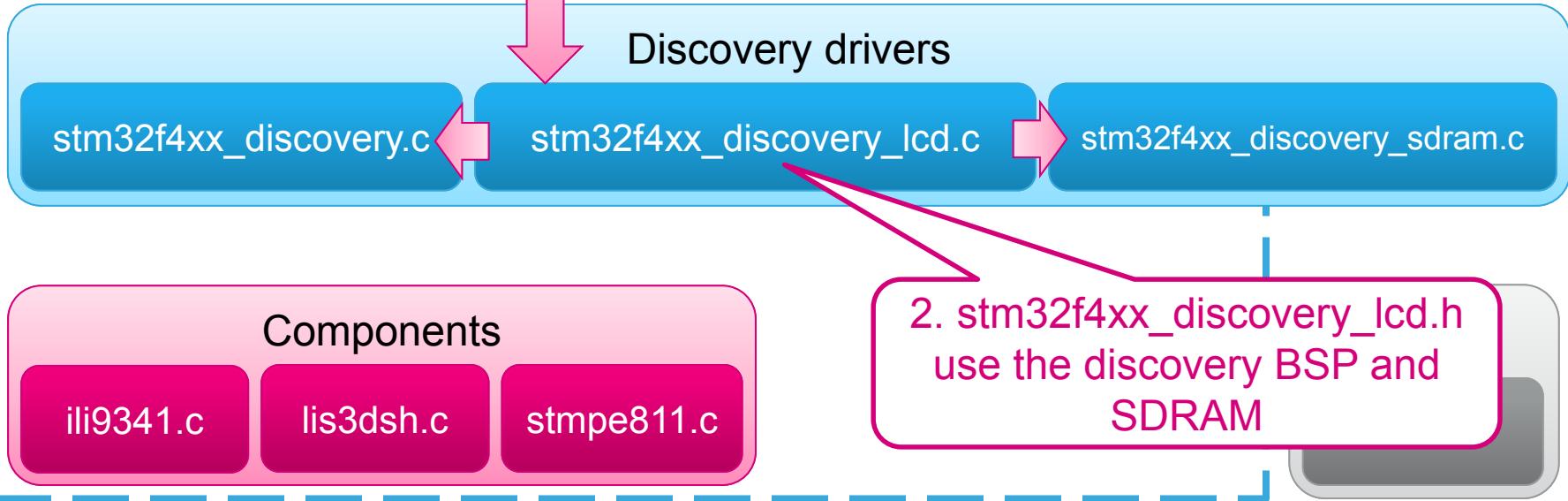
## BSP LCD organization

Our project

1. include

`stm32f4xx_discovery_lcd.h`

BSP package



## HAL API

`stm32f4xx_hal_sram.c``stm32f4xx_ll_fmc.c``stm32f4xx_hal_ltdc.c``stm32f4xx_hal.c``stm32f4xx_hal_spi.c``stm32f4xx_hal_i2c.c`

## BSP LCD organization

## Our project

3. `stm32f4xx_discovery_lcd.h`  
use driver `ili9341.c` which is  
TFT LCD controller

`stm32f4xx_discovery.c`

`stm32f4xx_discovery_lcd.c`

`stm32f4xx_discovery_sram.c`

## Discovery drivers

`ili9341.c`

`lis3dsh.c`

`stmpe811.c`

## Components

4. `stm32f4xx_discovery_lcd.h`  
use LCD controller from HAL

## HAL API

`stm32f4xx_hal_sram.c`

`stm32f4xx_ll_fmc.c`

`stm32f4xx_hal_ltdc.c`

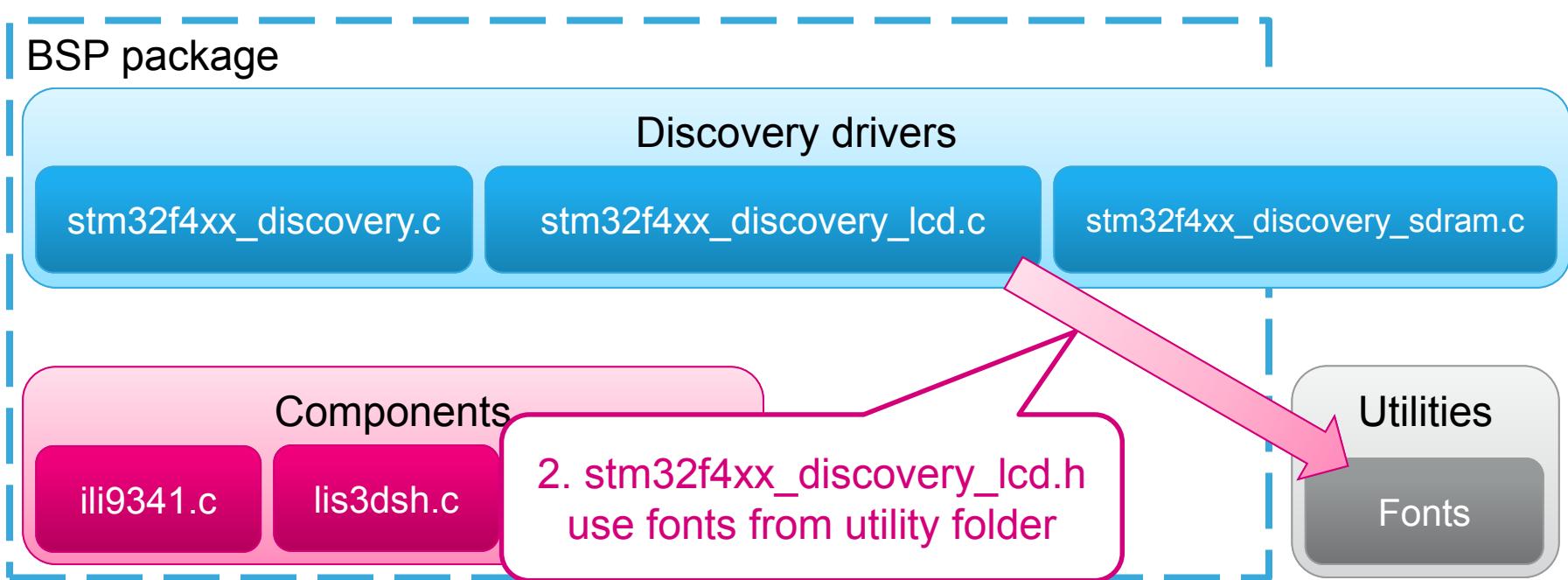
`stm32f4xx_hal.c`

`stm32f4xx_hal_spi.c`

`stm32f4xx_hal_i2c.c`

## BSP LCD organization

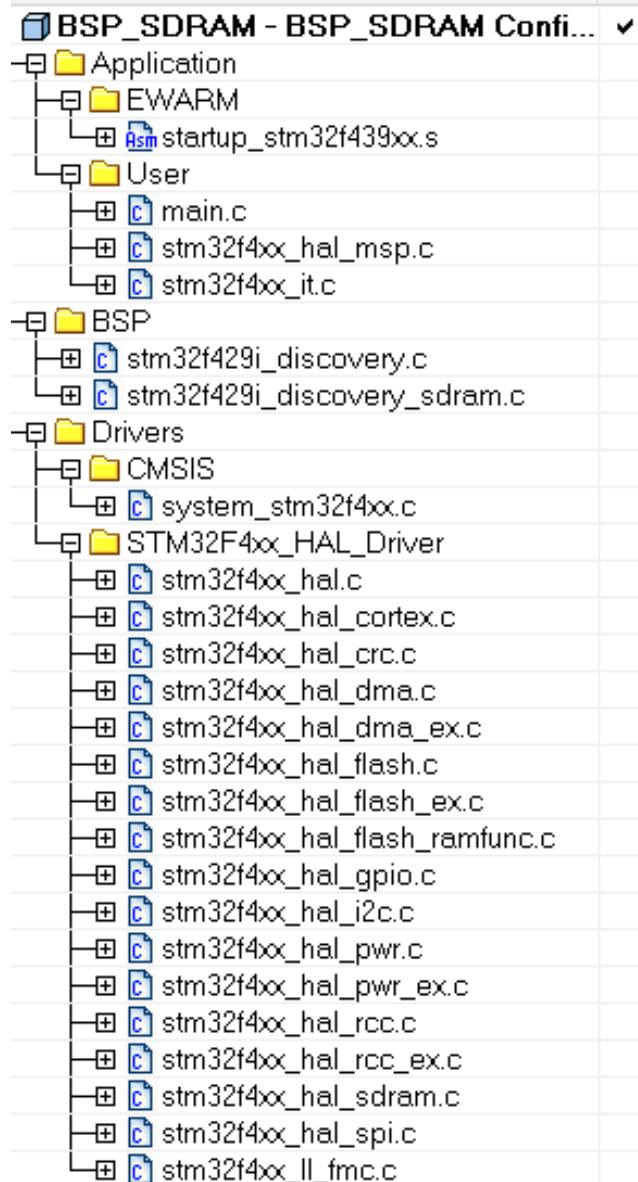
## Our project



## HAL API

`stm32f4xx_hal_sdram.c``stm32f4xx_ll_fmc.c``stm32f4xx_hal_ltdc.c``stm32f4xx_hal.c``stm32f4xx_hal_spi.c``stm32f4xx_hal_i2c.c`

- We use the project from BSP SDRAM because the LCD also use the SDRAM
- We need copy the Fonts from Utilities folder in CubeMX repository



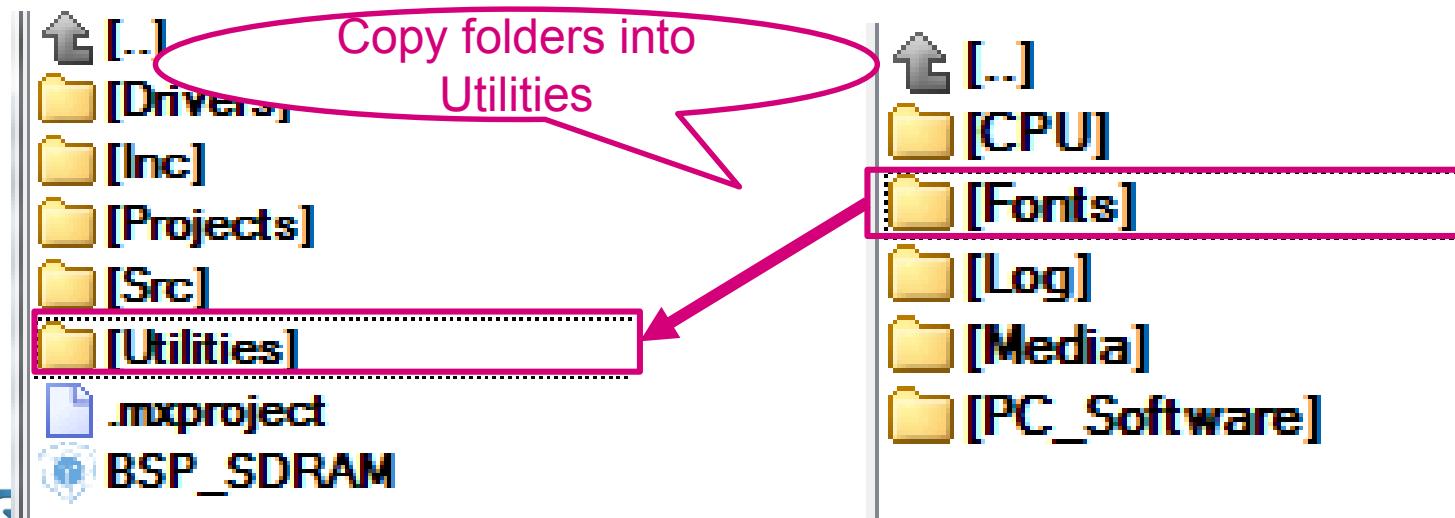
# 26 Use BSP for LCD init and writing

388

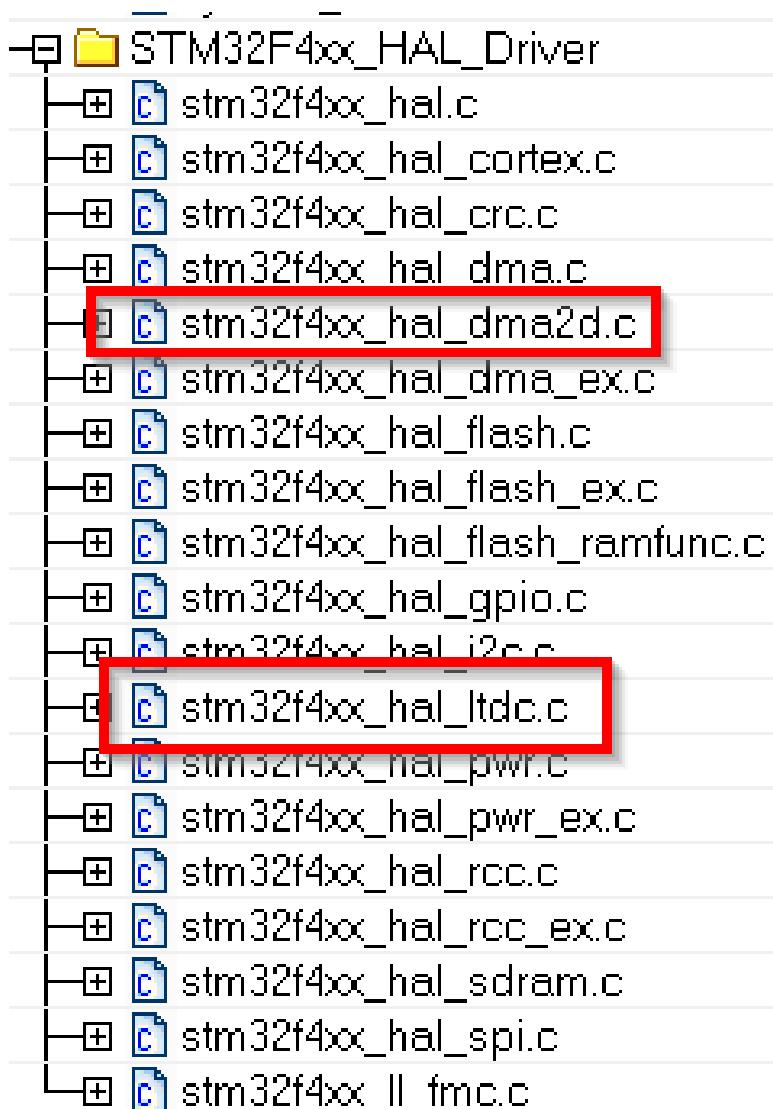
- The copy part
  - In our project in Drivers folder create folder Utilities



- Now go into CubeMX repository ...\\STM32Cube\_FW\_F4\_V1.3.0\\Utilities\\
- And copy **Fonts** into **Utilities** folder



- We add the driver for LCD from HAL
- Right click on STM32F4xx\_HAL\_Drive>ADD from \Drivers\STM32F4xx\_HAL\_Driver\Src
  - `stm32f4xx_hal_ltdc.c`
  - `Stm32f4xx_hal_dma2d.c`



# 26

# Use BSP for LCD init and writing

390

- We add the driver for BSP LDC
- Right click on BSP>ADD from \Drivers\BSP\STM32F429I-Discovery\
  - `stm32f429i_discovery_lcd.c`
- Right click on BSP>ADD from \Drivers\BSP\Components\ili9341\
  - `ili9341.c`



- Now last thing is allow to include new HAL files which we added
  - Open `stm32f4xx_hal_conf.h` in ..\Inc\
  - Uncomment files which we added
    - `HAL_DMA2D_MODULE_ENABLED`
    - `HAL_LTDC_MODULE_ENABLED`

```
/* ##### 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
///#define HAL_PCCARD_MODULE_ENABLED
///#define HAL_SRAM_MODULE_ENABLED
#define HAL_SDRAM_MODULE_ENABLED
///#define HAL_HASH_MODULE_ENABLED
#define HAL_I2C_MODULE_ENABLED
///#define HAL_I2S_MODULE_ENABLED
///#define HAL_IWDG_MODULE_ENABLED
#define HAL_LTDC_MODULE_ENABLED
///#define HAL_RNG_MODULE_ENABLED
///#define HAL_RTC_MODULE_ENABLED
///#define HAL_SAI_MODULE_ENABLED
///#define HAL_SD_MODULE_ENABLED
#define HAL_SPI_MODULE_ENABLED
///#define HAL_TIM_MODULE_ENABLED
///#define HAL_UART_MODULE_ENABLED
///#define HAL_USART_MODULE_ENABLED
///#define HAL_IRDA_MODULE_ENABLED
///#define HAL_SMARTCARD_MODULE_ENABLED
///#define HAL_WWDG_MODULE_ENABLED
///#define HAL_PCD_MODULE_ENABLED
///#define HAL_HCD_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED
```

- Into main.c now we modify include from  
stm32f429i\_discovery\_sram.h to stm32f429i\_discovery\_lcd.h

```
/* USER CODE BEGIN Includes */  
#include "stm32f429i_discovery_lcd.h"  
/* USER CODE END Includes */
```

- And remove the BSP\_SDRAM\_Init()

```
/* USER CODE BEGIN 2 */  
/* USER CODE END 2 */
```

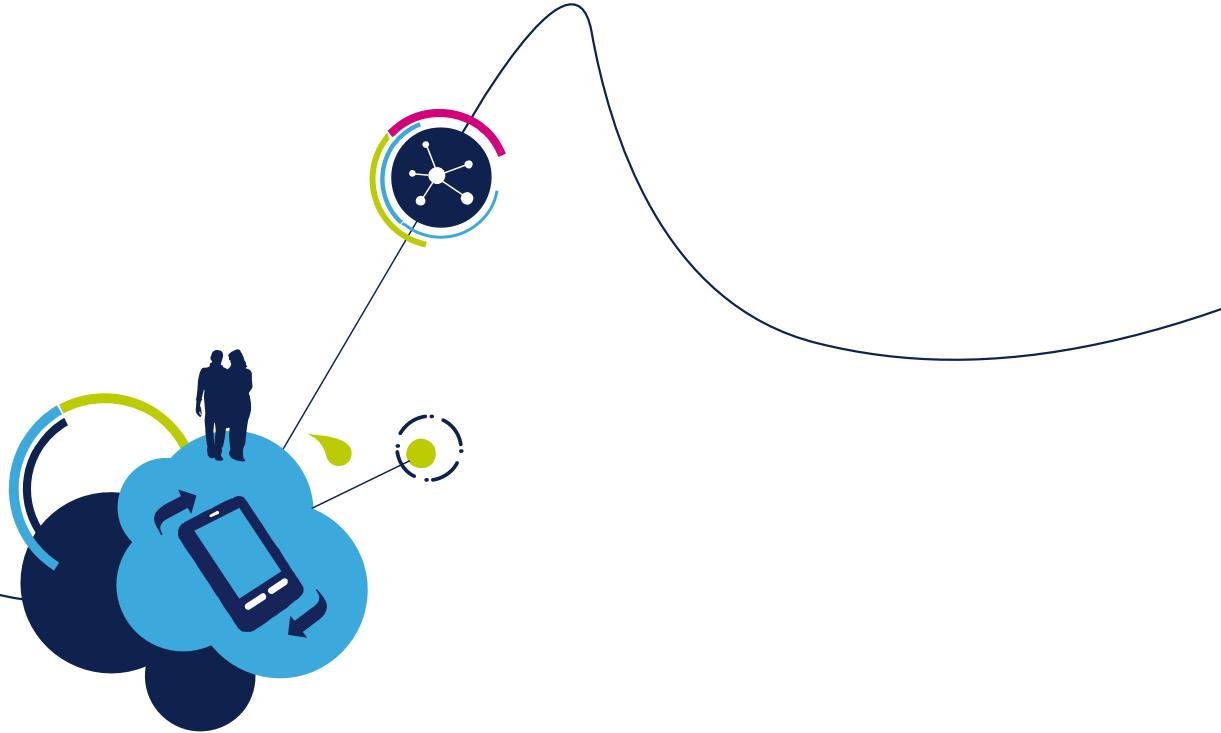
# 26 Use BSP for LCD init and writing

393

- Simple LCD demonstration

```
/* USER CODE BEGIN 2 */
BSP_LCD_Init(); //init LCD
//set the layer buffer address into SDRAM
BSP_LCD_LayerDefaultInit(1, SDRAM_DEVICE_ADDR);
BSP_LCD_SelectLayer(1); //select on which layer we write
BSP_LCD_DisplayOn(); //turn on LCD
BSP_LCD_Clear(LCD_COLOR_BLUE); //clear the LCD on blue color
BSP_LCD_SetBackColor(LCD_COLOR_BLUE); //set text background color
BSP_LCD_SetTextColor(LCD_COLOR_WHITE); //set text color
//write text
BSP_LCD_DisplayStringAtLine(2, "Cube STM32");
BSP_LCD_DisplayStringAtLine(3, "BSP");
BSP_LCD_DisplayStringAtLine(4, "LCD DEMO");
/* USER CODE END 2 */
```





# BSP EEPROM lab 27

- Objective

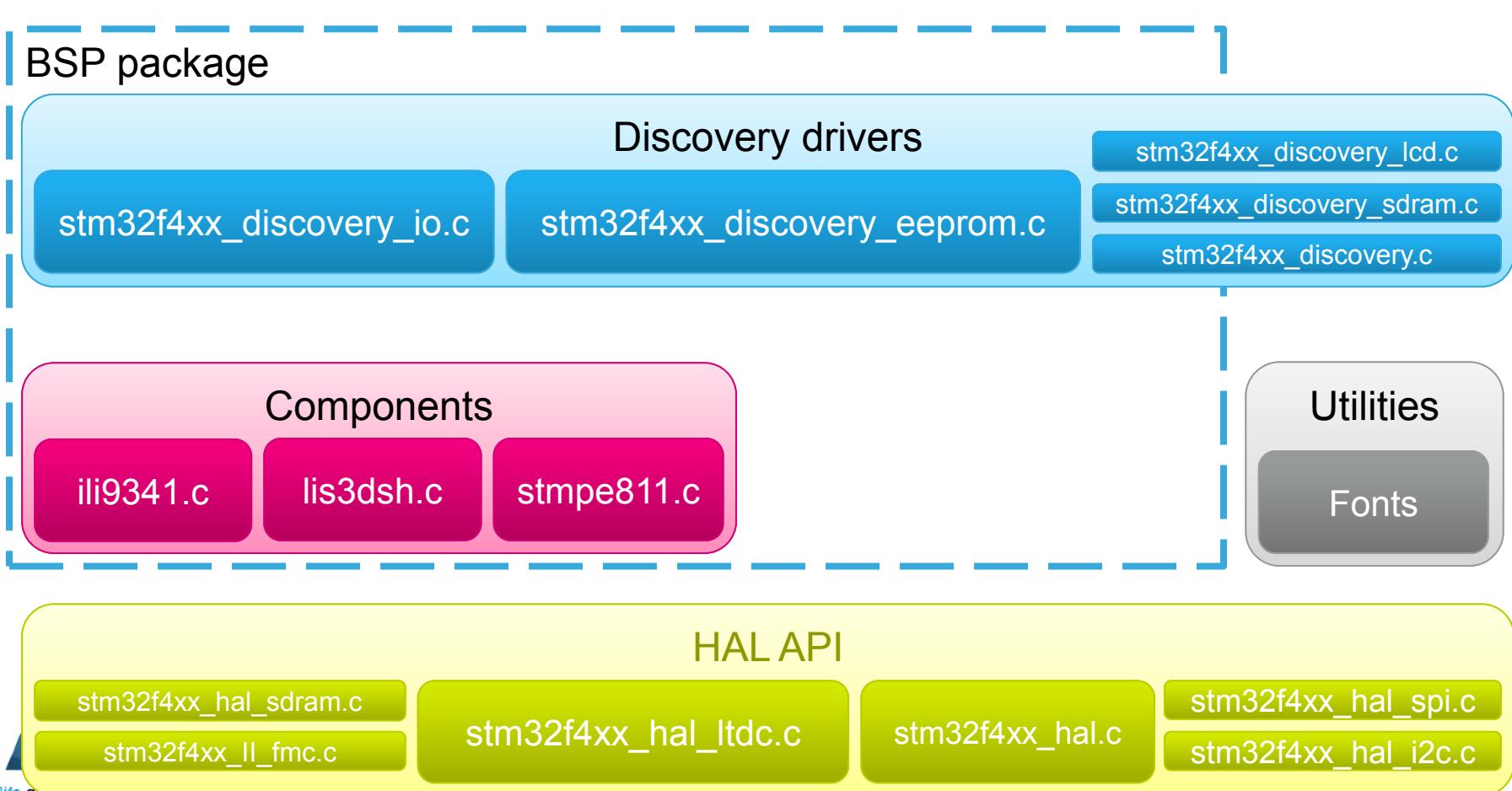
- Learn how import BSP EEPROM into project
- We use the project from lab 26
- Which part need to by configured in GUI
- Try to write text into EEPROM and read it
- Read text from EEPROM and display it on LCD

- Goal

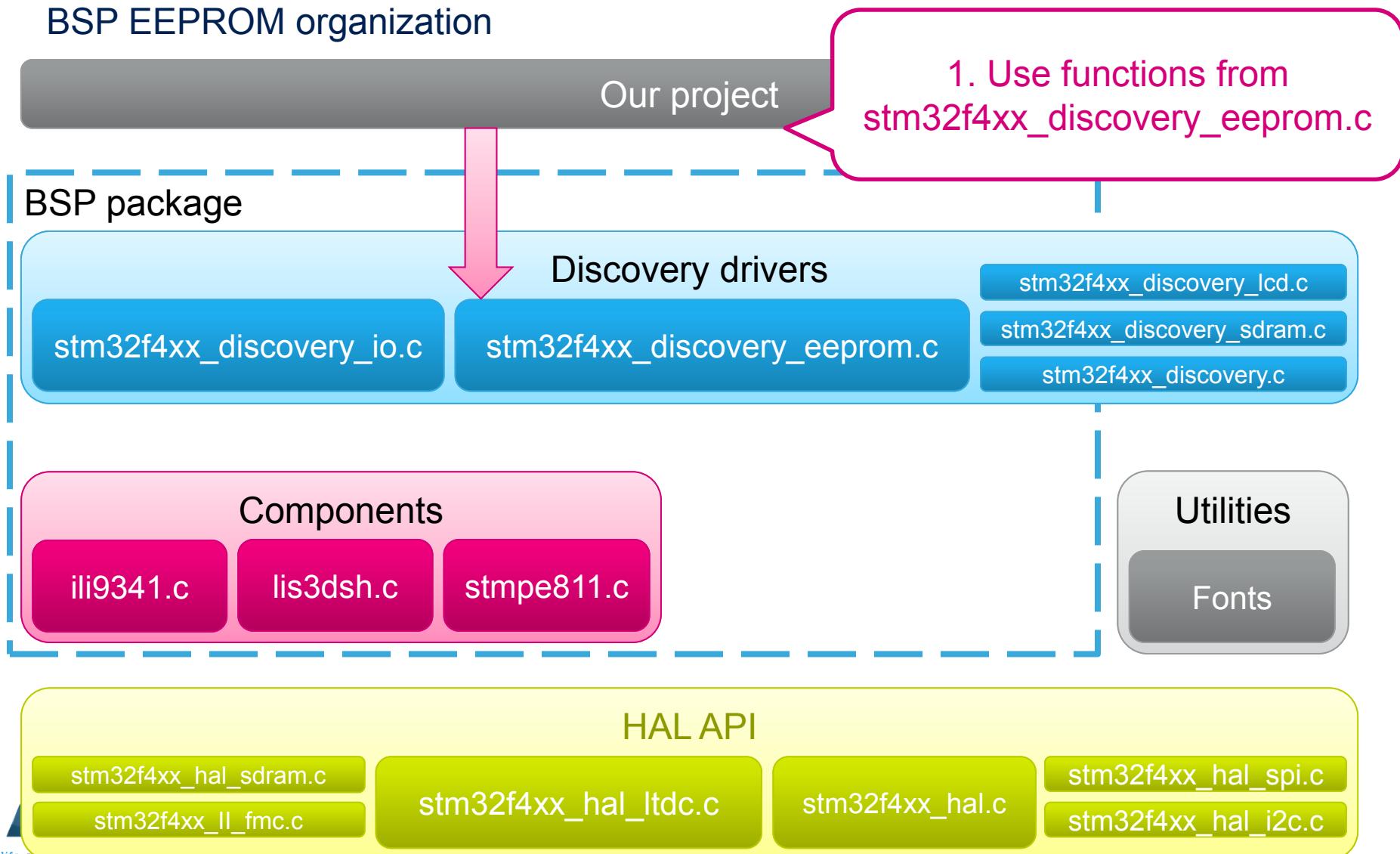
- Successfully import BSP EEPROM drivers into your project
- Learn which part you need to import
- How to setup the project

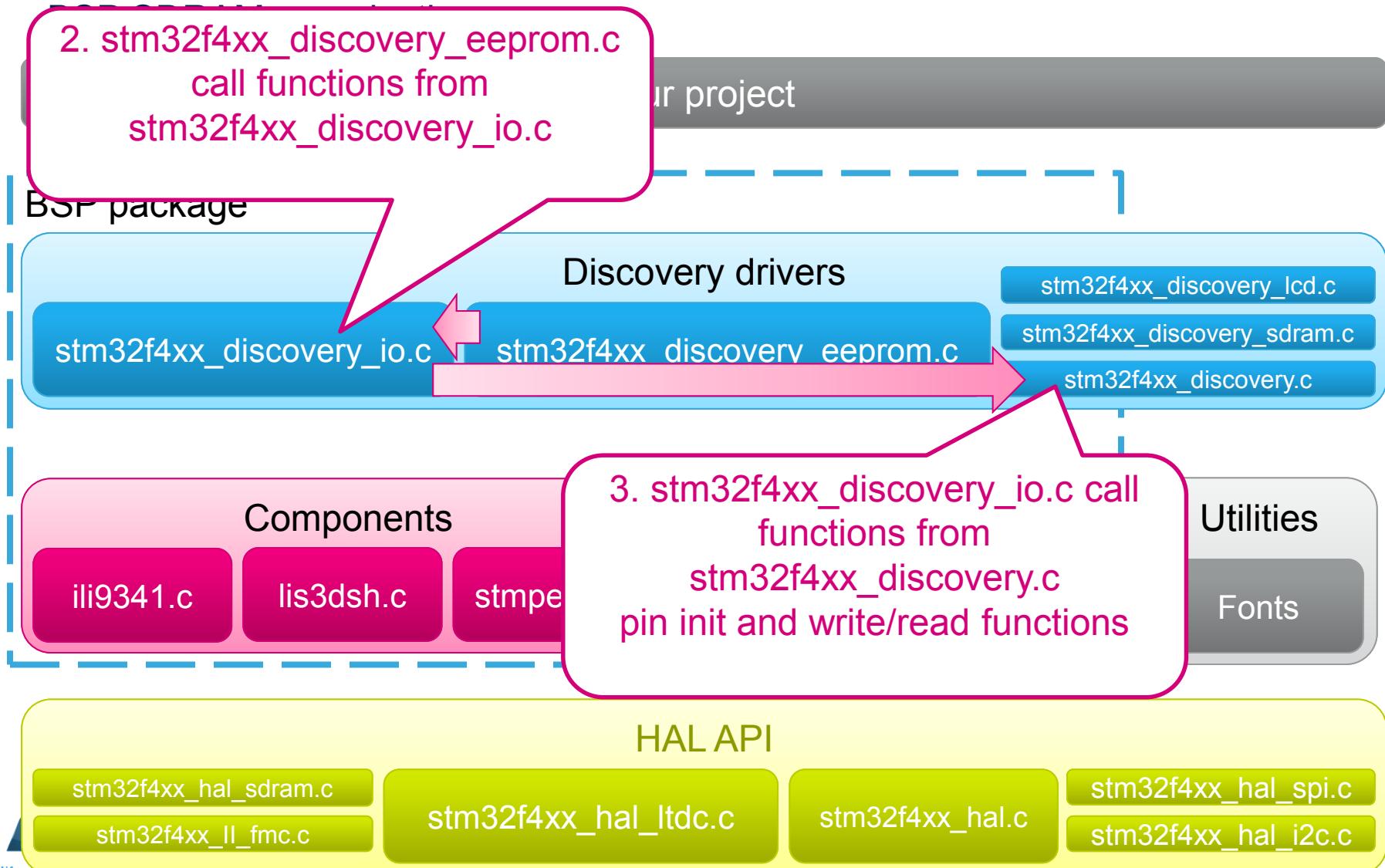
## BSP EEPROM organization

## Our project



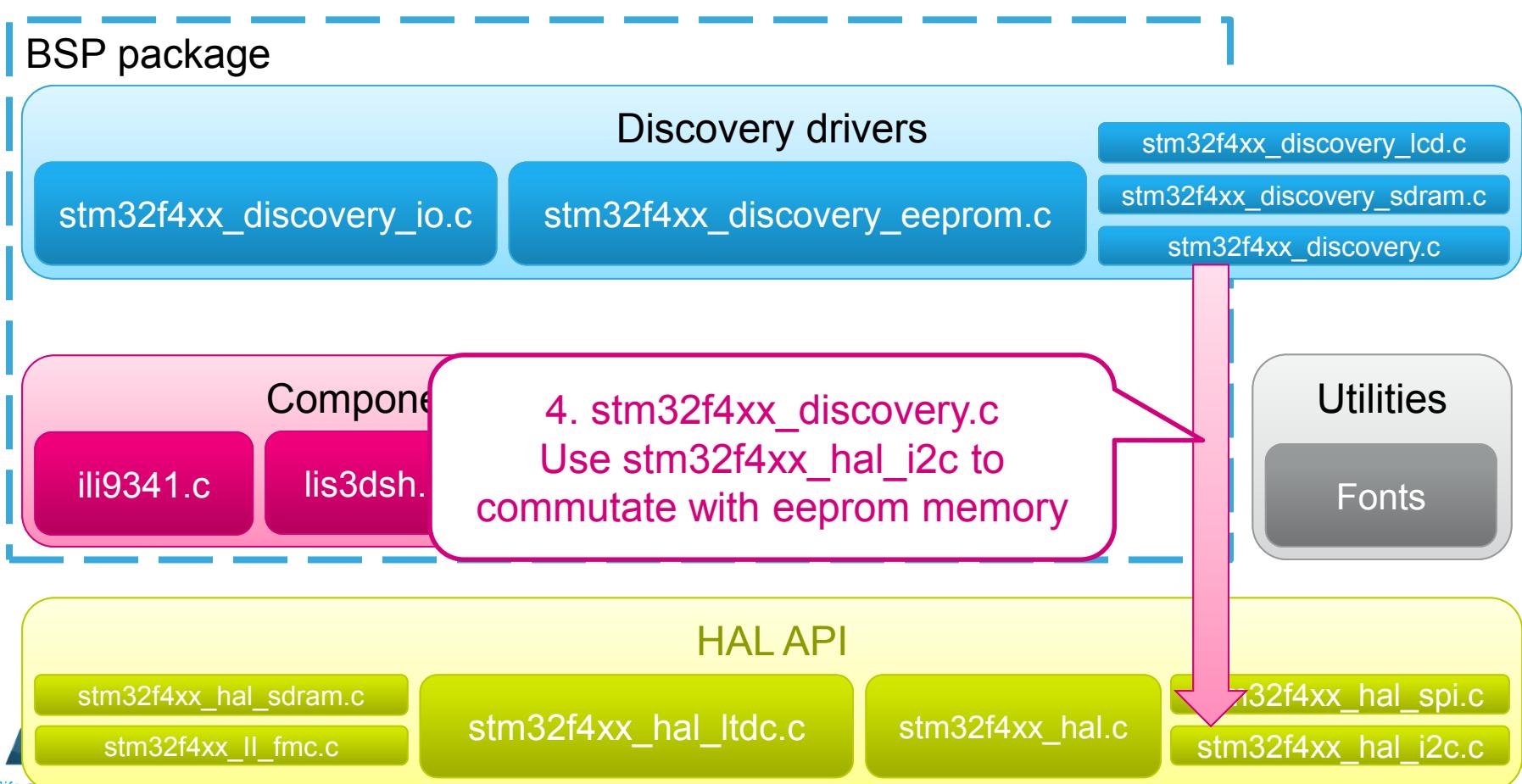
## BSP EEPROM organization



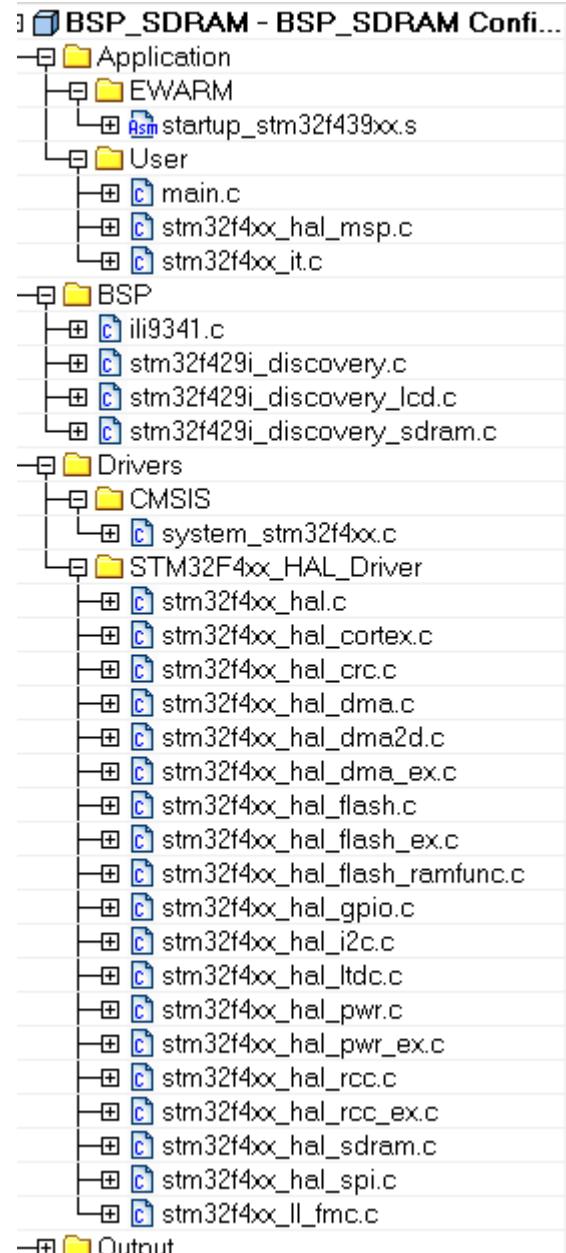


## BSP EEPROM organization

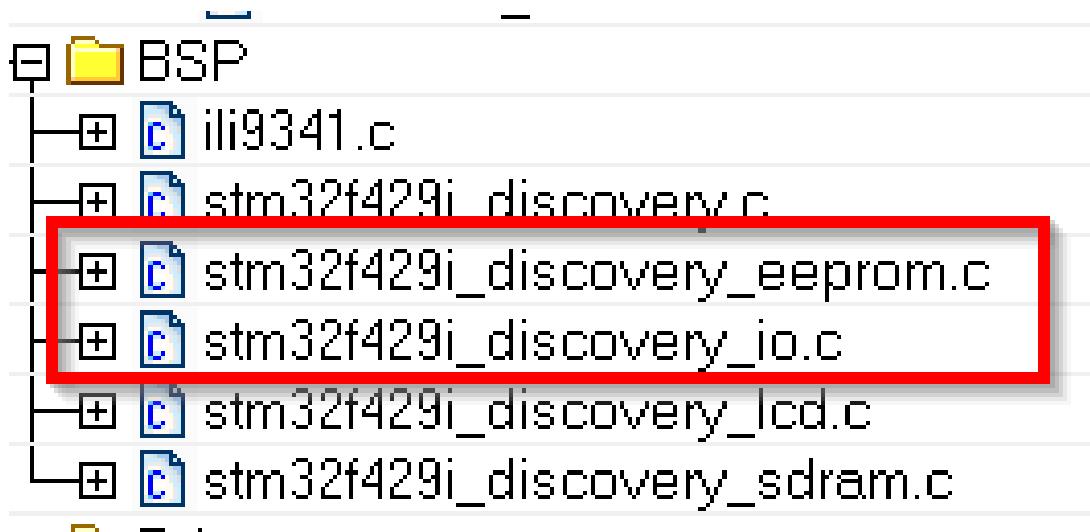
## Our project



- We use the project from BSP LCD lab 26 because we want to display the memory content on LCD

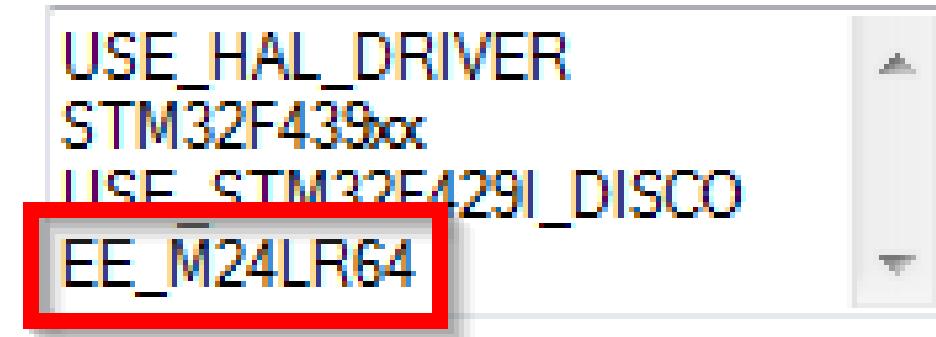


- We add the driver for BSP LDC
- Right click on BSP>ADD from \Drivers\BSP\STM32F429I-Discovery\
  - [stm32f429i\\_discovery\\_eeprom.c](#)
  - [stm32f429i\\_discovery\\_io.c](#)

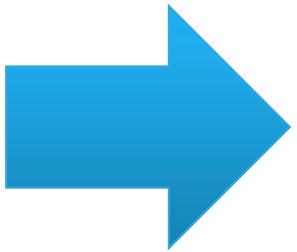
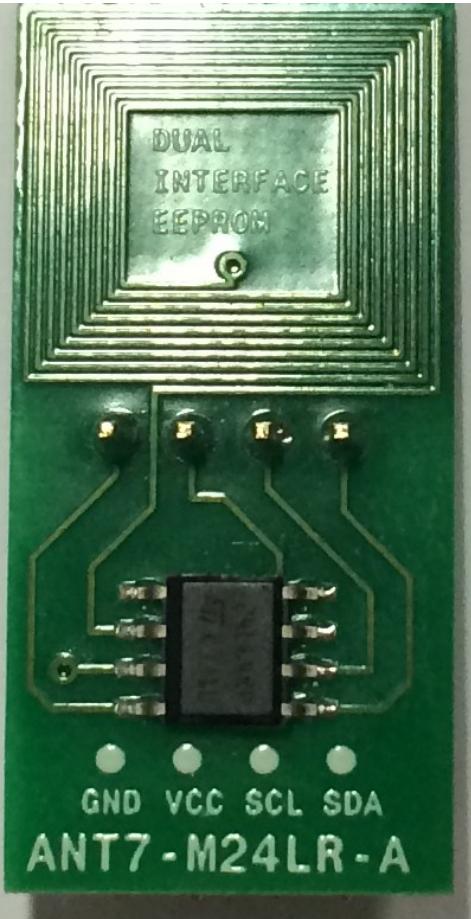


- Add the define of EEPROM into project options
  - Right click on project>Options>Category C/C++Compiler>Preprocesor
  - Into Defined symbols add EE\_M24LR64
  - This allow use EEPROM functions
  - Button OK close project options

Defined symbols: (one per line)



- Use the ATM7-M24LR-A board with M24LR memory and connect it into STM32F429i-Discovery kit



- Into main.c now modify include

```
/* USER CODE BEGIN Includes */  
#include "stm32f429i_discovery_lcd.h"  
#include "stm32f429i_discovery_io.h"  
#include "stm32f429i_discovery_eeprom.h"  
#include <string.h>  
/* USER CODE END Includes */
```

- Define variables

```
/* USER CODE BEGIN PV */  
uint8_t text_to_write[]="test text";//write to eeprom  
uint8_t text_to_read[200];//read from eeprom  
uint32_t address=0;//address in eeprom  
uint16_t read_num=1;//number of bytes which we want to read from  
eeprom  
/* USER CODE END PV */
```

- Into `stm32f4xx_hal_it.c` add global variable for I2C handle

```
/* USER CODE BEGIN 0 */  
extern I2C_HandleTypeDef I2cHandle;  
/* USER CODE END 0 */
```

- and define handler functions for I2C DMA

```
/* USER CODE BEGIN 1 */  
void DMA1_Stream4_IRQHandler()  
{  
    HAL_DMA_IRQHandler(I2cHandle.hdmaTx);  
}  
  
void DMA1_Stream2_IRQHandler()  
{  
    HAL_DMA_IRQHandler(I2cHandle.hdmaRx);  
}  
/* USER CODE END 1 */
```

- Into main.c add

```
/* USER CODE BEGIN 2 */
/*LCD init*/
BSP_LCD_Init();
BSP_LCD_LayerDefaultInit(1, SDRAM_DEVICE_ADDR);
BSP_LCD_SelectLayer(1);
BSP_LCD_DisplayOn();
BSP_LCD_Clear(LCD_COLOR_BLUE);
BSP_LCD_SetBackColor(LCD_COLOR_BLUE);
BSP_LCD_SetTextColor(LCD_COLOR_WHITE);

/*EEPROM init*/
BSP_EEPROM_Init();
/*Write text into EEPROM*/
BSP_EEPROM_WriteBuffer(text_to_write,0,(strlen(text_to_write)+1));
/*Read text from EEPROM*/
do{
    BSP_EEPROM_ReadBuffer((uint8_t*)&(text_to_read[address]),address,(uint16_t*)&read_num);
}while(text_to_read[address++]!=0x0);
/*Display text*/
BSP_LCD_DisplayStringAtLine(2,text_to_read);
/* USER CODE END 2 */
```





# BSP GYRO lab 28

# 28 Use BSP to access GYROSCOPE

408

- Objective

- Learn how import BSP GYROSCOPE into project
- We use the project from lab 26
- Which part need to by configured in GUI
- Read data from GYROSCOPE and display it on LCD

- Goal

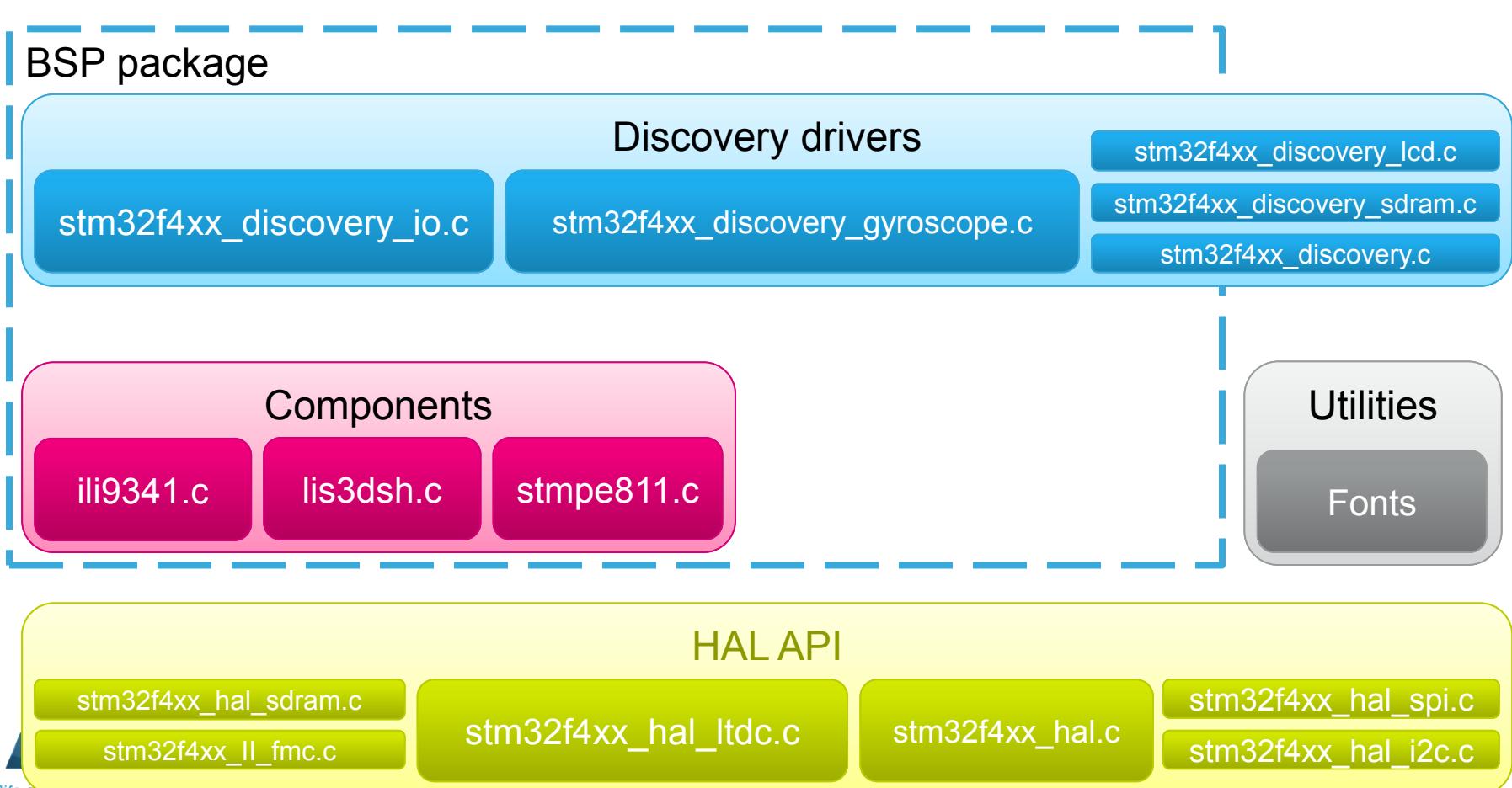
- Successfully import BSP GYROSCOPE drivers into your project
- Learn which part you need to import
- How to setup the project

# 28 Use BSP to access GYROSCOPE

409

## BSP GYRO organization

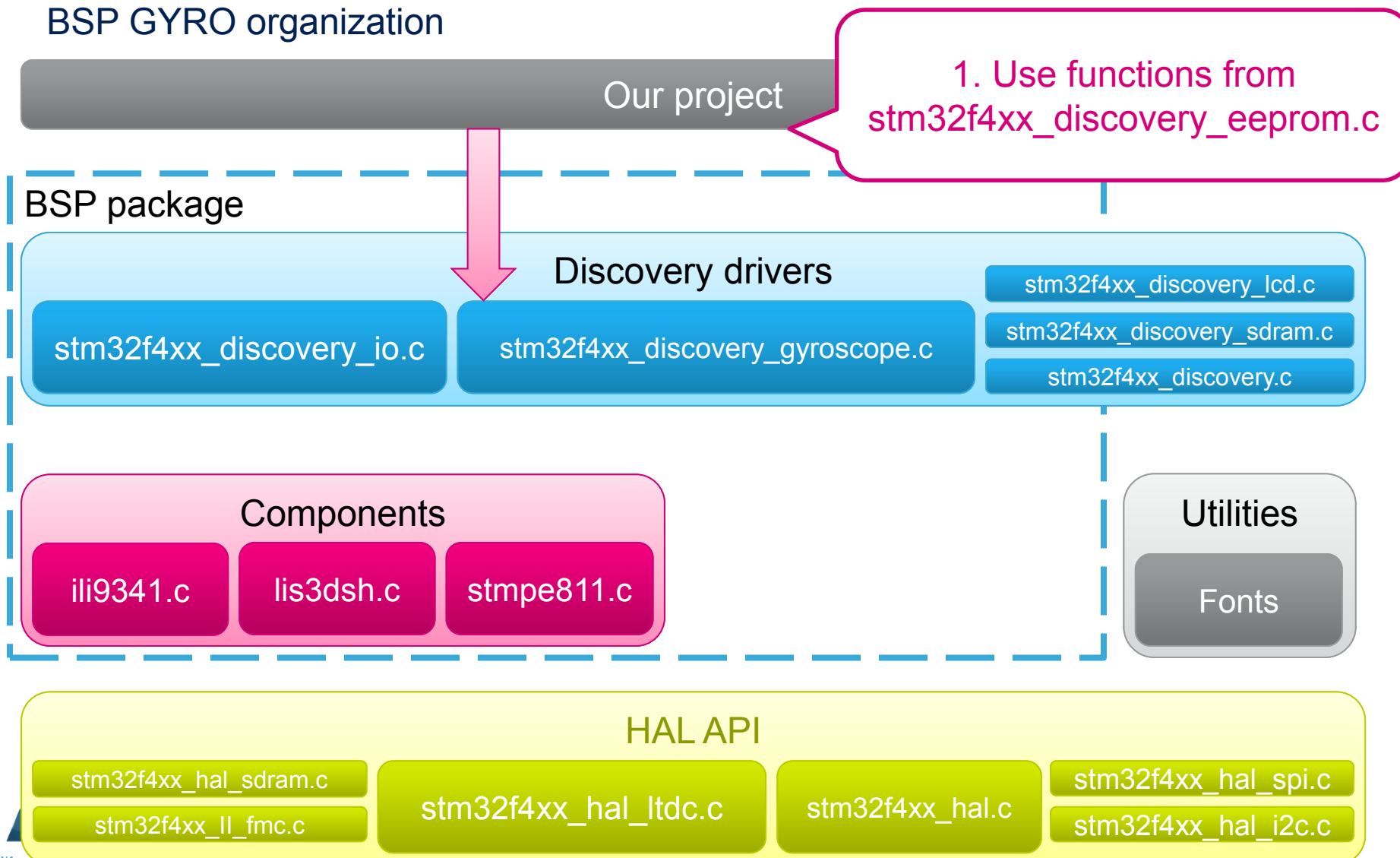
### Our project



# 28 Use BSP to access GYROSCOPE

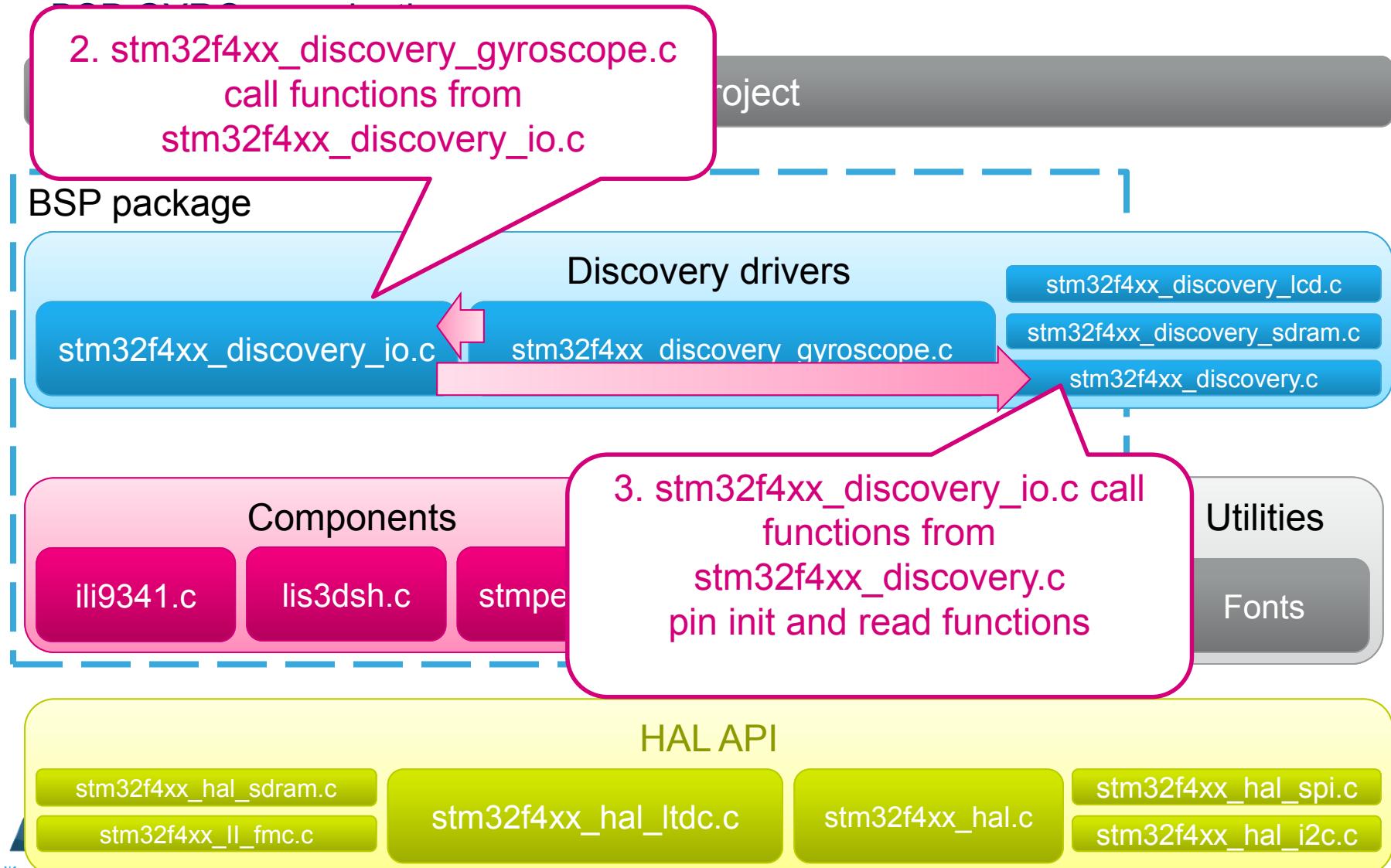
410

## BSP GYRO organization



# 28 Use BSP to access GYROSCOPE

411

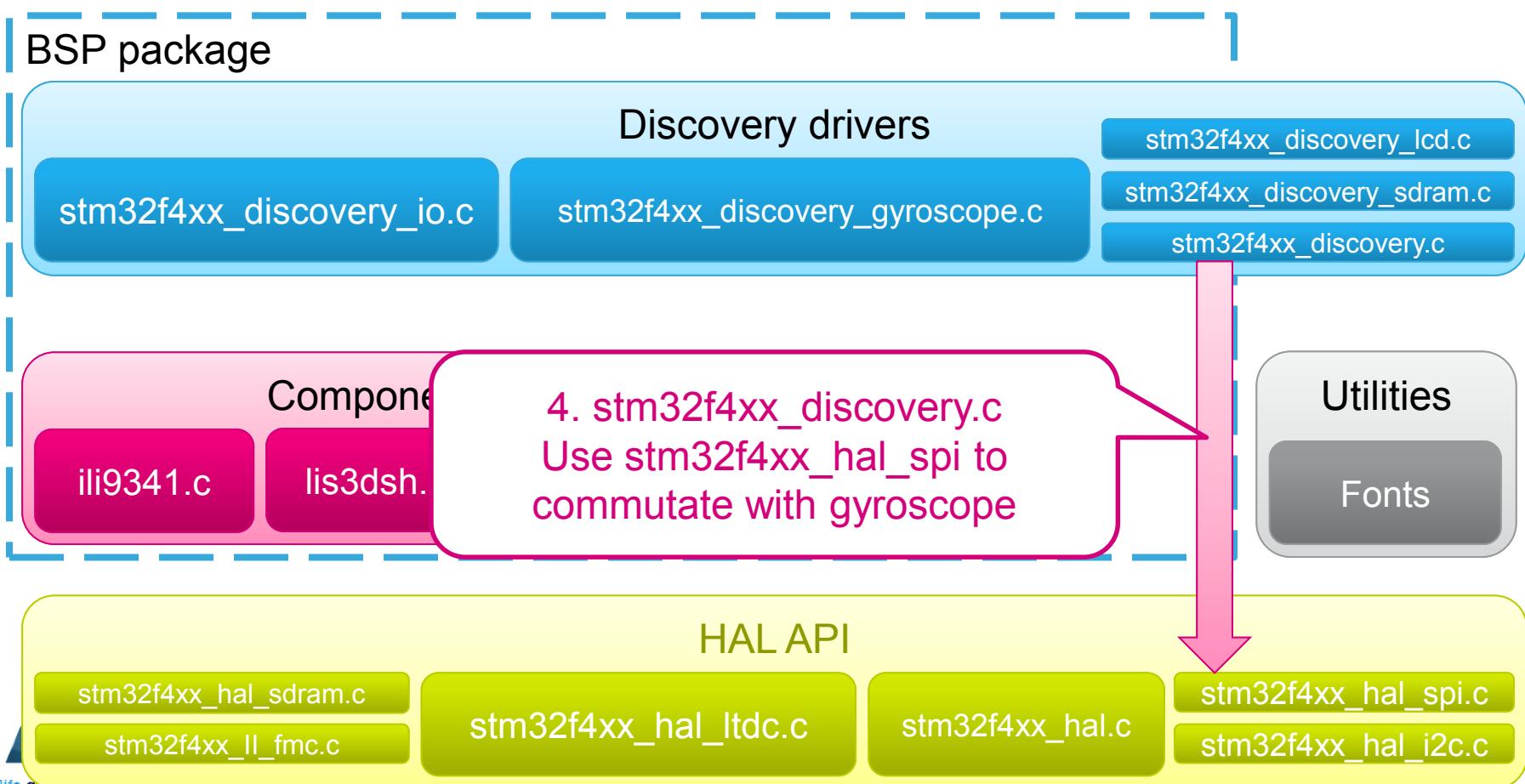


# 28 Use BSP to access GYROSCOPE

412

## BSP GYRO organization

### Our project



# 28 Use BSP to access GYROSCOPE

413

- We use the project from BSP LCD lab 26 because we want to display gyro values on LCD

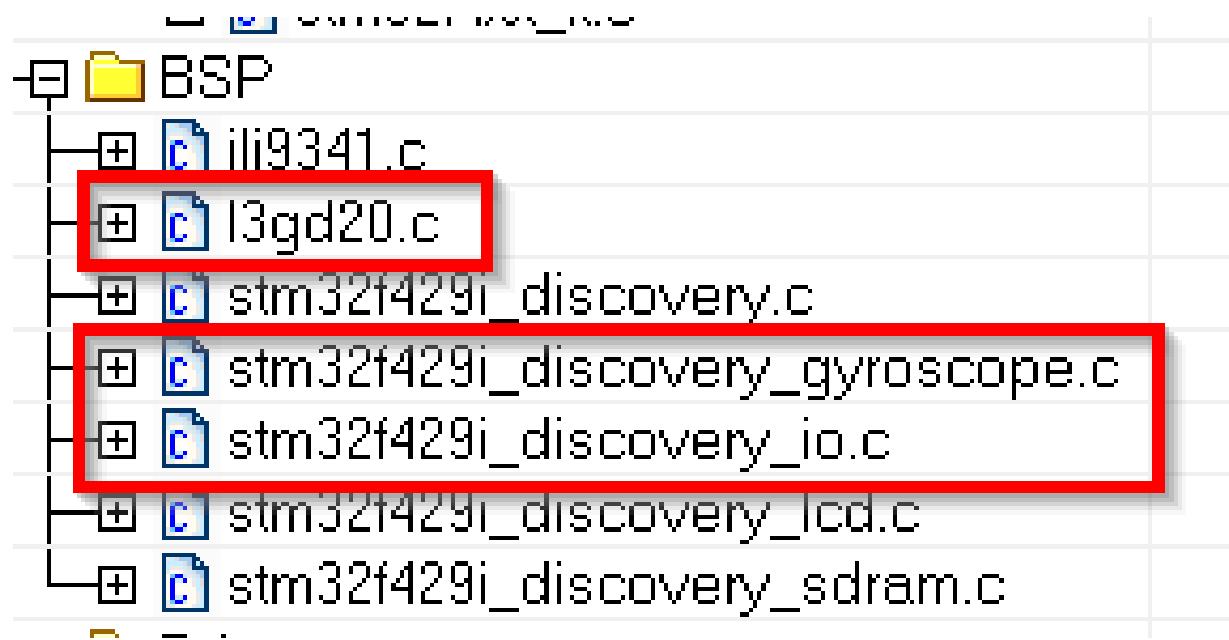
BSP\_SDRAM - BSP\_SDRAM Config...

```
Application
    └── EWARM
        └── startup_stm32f439xx.s
    └── User
        ├── main.c
        ├── stm32f4xx_hal_msp.c
        └── stm32f4xx_it.c
BSP
    ├── ili9341.c
    ├── stm32f429i_discovery.c
    ├── stm32f429i_discovery_lcd.c
    └── stm32f429i_discovery_sram.c
Drivers
    ├── CMSIS
        └── system_stm32f4xx.c
    └── STM32F4xx_HAL_Driver
        ├── stm32f4xx_hal.c
        ├── stm32f4xx_hal_cortex.c
        ├── stm32f4xx_hal_crc.c
        ├── stm32f4xx_hal_dma.c
        ├── stm32f4xx_hal_dma2d.c
        ├── stm32f4xx_hal_dma_ex.c
        ├── stm32f4xx_hal_flash.c
        ├── stm32f4xx_hal_flash_ex.c
        ├── stm32f4xx_hal_flash_ramfunc.c
        ├── stm32f4xx_hal_gpio.c
        ├── stm32f4xx_hal_i2c.c
        ├── stm32f4xx_hal_ltdc.c
        ├── stm32f4xx_hal_pwr.c
        ├── stm32f4xx_hal_pwr_ex.c
        ├── stm32f4xx_hal_rcc.c
        ├── stm32f4xx_hal_rcc_ex.c
        ├── stm32f4xx_hal_sdram.c
        ├── stm32f4xx_hal_spi.c
        └── stm32f4xx_ll_fmc.c
Output
```

# 28 Use BSP to access GYROSCOPE

414

- We add the driver for BSP LDC
- Right click on BSP>ADD from \Drivers\BSP\STM32F429I-Discovery\
  - `stm32f429i_discovery_gyroscope.c`
  - `stm32f429i_discovery_io.c`
- Right click on BSP>ADD from \Drivers\BSP\Components\
  - `I3gd20.c`



# 28 Use BSP to access GYROSCOPE

415

- Into main.c now modify include

```
/* USER CODE BEGIN Includes */  
#include "stm32f429i_discovery_lcd.h"  
#include "stm32f429i_discovery_gyroscope.h"  
#include "stm32f429i_discovery_io.h"  
#include <stdio.h>  
/* USER CODE END Includes */
```

- Define variables

```
/* USER CODE BEGIN PV */  
float valxyz[3];//gyroscope values  
uint8_t buffer[200];//text buffer  
/* USER CODE END PV */
```

# 28 Use BSP to access GYROSCOPE

416

- Into main.c add

```
/* USER CODE BEGIN 2 */  
/*LCD init*/  
BSP_LCD_Init();  
BSP_LCD_LayerDefaultInit(1, SDRAM_DEVICE_ADDR);  
BSP_LCD_SelectLayer(1);  
BSP_LCD_DisplayOn();  
BSP_LCD_Clear(LCD_COLOR_BLUE);  
BSP_LCD_SetBackColor(LCD_COLOR_BLUE);  
BSP_LCD_SetTextColor(LCD_COLOR_WHITE);  
/*Gyroscope init*/  
BSP_GYRO_Init();  
/* USER CODE END 2 */
```

# 28 Use BSP to access GYROSCOPE

417

- Into main.c add

```
/* USER CODE BEGIN 3 */
/* Infinite loop */
while (1)
{
    /*Get Gyrospope value*/
    BSP_GYRO_GetXYZ(valxyz);
    /*Display X*/
    sprintf(buffer, "x:%f",valxyz[0]);
    BSP_LCD_DisplayStringAtLine(2,buffer);
    /*Display Y*/
    sprintf(buffer, "y:%f",valxyz[1]);
    BSP_LCD_DisplayStringAtLine(3,buffer);
    /*Display Z*/
    sprintf(buffer, "z:%f",valxyz[2]);
    BSP_LCD_DisplayStringAtLine(4,buffer);
    /*Delay*/
    HAL_Delay(1000);
}
/* USER CODE END 3 */
```

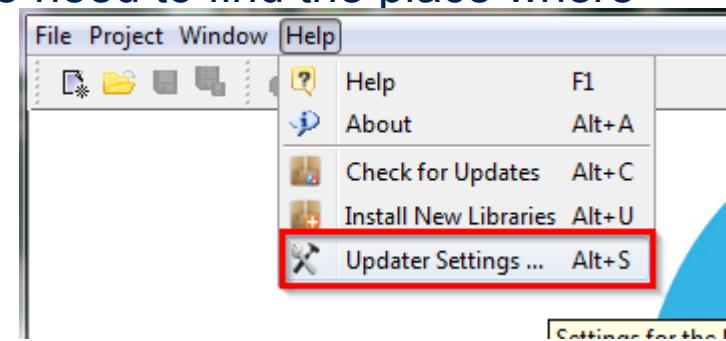




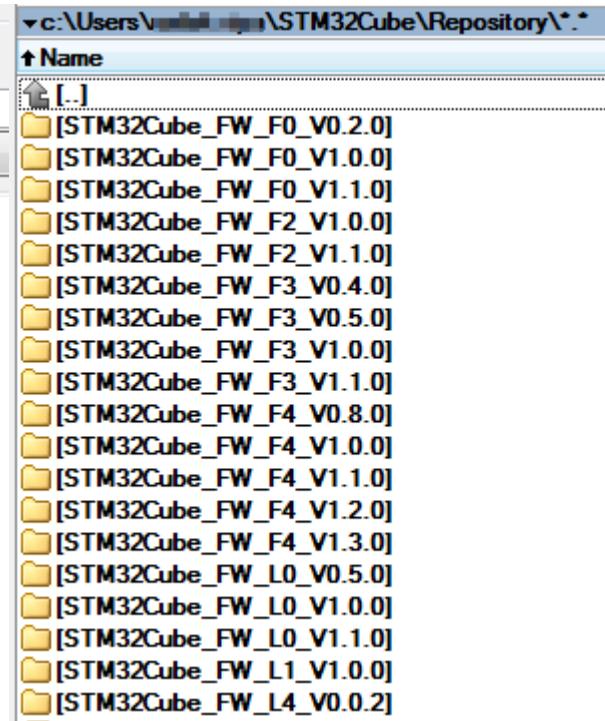
# Appendix A CubeMX install

- CubeMX tool
  - [http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242?s\\_searchtype=partnumber](http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242?s_searchtype=partnumber)
- The CubeMX tool need the java
  - Please check if you have actual 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 throre CubeMX
  - [STM32CubeL0](#)
  - [STM32CubeL1](#)
  - [STM32CubeF0](#)
  - [STM32CubeF2](#)
  - [STM32CubeF3](#)
  - [STM32CubeF4](#)

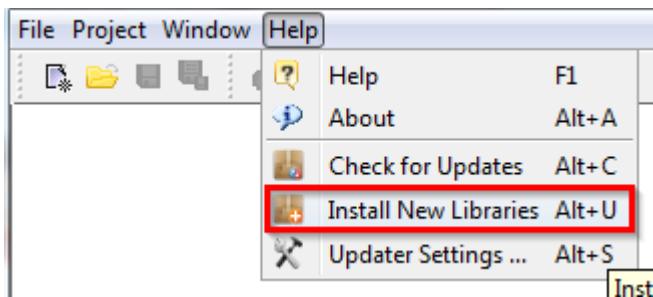
- Install the CubeMX
- After installation run CubeMX
- In case you download the package from web we need to find the place where they need to be stored
- MENU>Help>Updater Settings...
- You will see where is the repository folder
  - Default is [C:/User/Acc\\_name/STM32Cube/Repository/](C:/User/Acc_name/STM32Cube/Repository/)
- You need to download STM32 packages into this folder
- Or CubeMX automatically download them into this folder



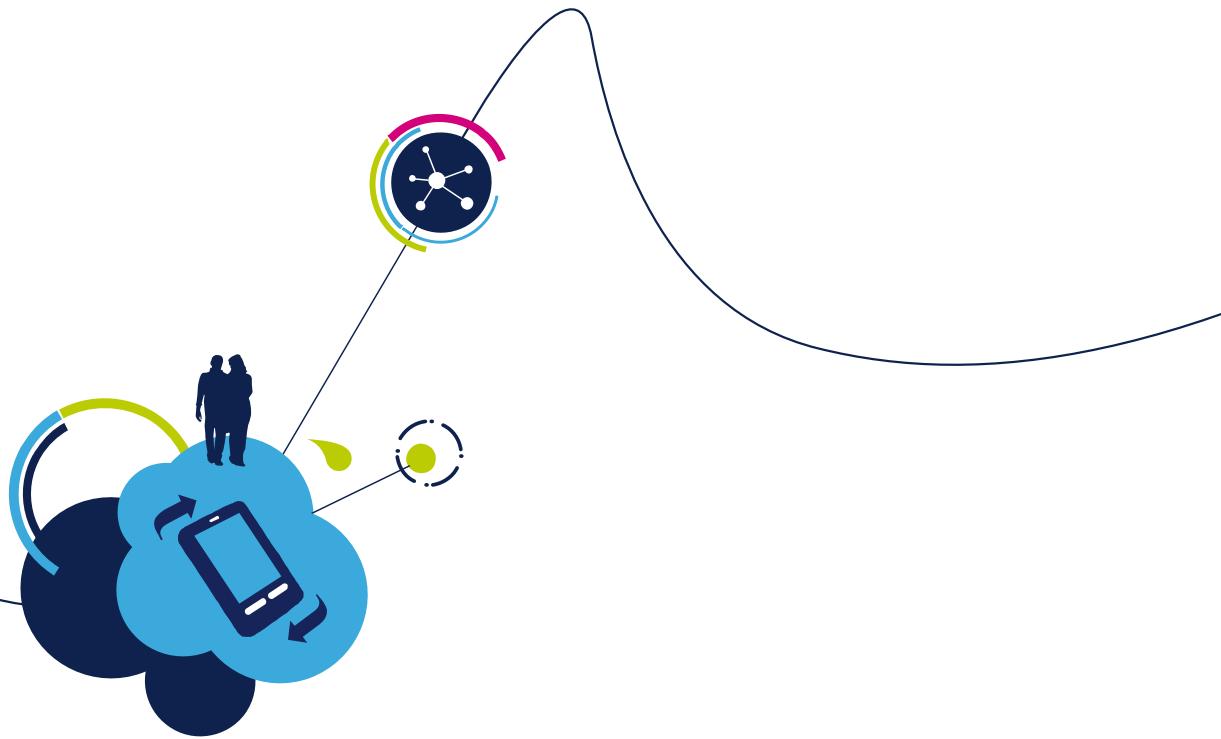
- The comparison of the CubeMX repository settings and structure in this folder



- In case you want to download these files automatically use in CubeMX
  - MENU>Help>Install New Libraries
  - Select libraries which you want
  - Force download with button Install Now



- 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

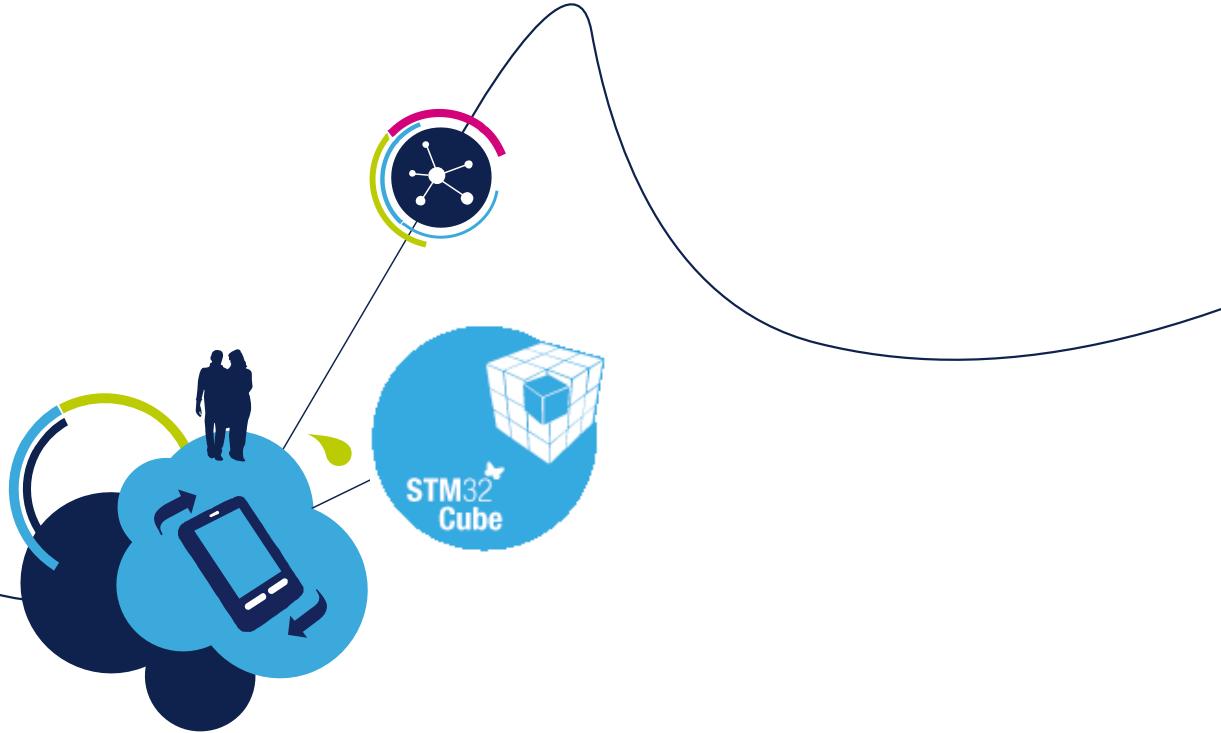


# Appendix B Documents

- CubeMX user manual UM1718
  - [http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user\\_manual/DM00104712.pdf](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00104712.pdf)
- CubeMX release note RN0094
  - [http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user\\_manual/DM00104712.pdf](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00104712.pdf)
- CubeMX technical note TN0072
  - [http://www.st.com/st-web-ui/static/active/en/resource/technical/document/technical\\_note/CD00214439.pdf](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/technical_note/CD00214439.pdf)

- STM32F429ZI web page
  - <http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1577/LN1806/PF255419#>
- STM32F429 Datasheet
  - <http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00071990.pdf>
- STM32F429 Reference Manual
  - [http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference\\_manual/DM00031020.pdf](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf)
- STM32F429 Programming manual
  - [http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming\\_manual/DM00046982.pdf](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00046982.pdf)

- STM32F429i-Discovery page
  - [http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/LN1848/PF259090?s\\_searchtype=keyword](http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/LN1848/PF259090?s_searchtype=keyword)
- STM32F429i-Discovery user manual with discovery schematics
  - [http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user\\_manual/DM00093903.pdf](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00093903.pdf)



[www.st.com/stm32](http://www.st.com/stm32)