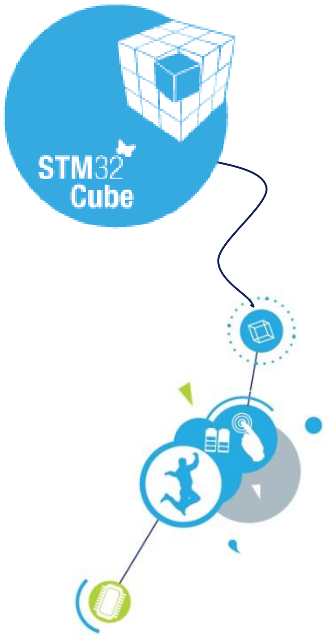
STM32F4 Labs

T.O.M.A.S – Technically Oriented Microcontroller Application Services
V1.09

life.augmented

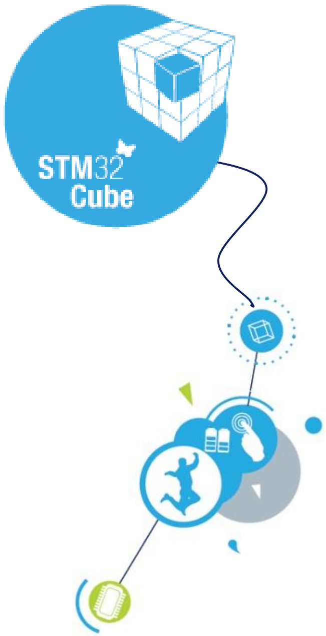
1 System Peripherals

1. GPIO & EXTI (1.1)
    1. GPIO lab (1.1.1)
    2. EXTI lab (1.1.2)
2. PWR (1.2)
    1. SLEEP lab (1.2.1)
    2. STOP lab (1.2.2)
    3. STANDBY lab (1.2.3)
3. DMA (1.3)
    1. DMA Poll lab (1.3.1)
    2. DMA Interrupt lab (1.3.2)

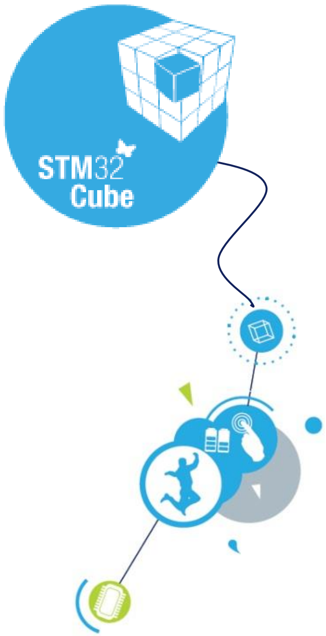


2 Basic communication peripherals

1. UART (2.1)
    1. UART Poll lab (2.1.1)
    2. UART Interrupt lab (2.1.2)
    3. UART DMA lab (2.1.3)
2. SPI (2.2)
    1. SPI Poll lab (2.2.1)
    2. SPI Interrupt lab (2.2.2)
    3. SPI DMA lab (2.2.3)
3. I2C (2.3)
    1. I2C Poll lab (2.3.1)
    2. I2C Interrupt lab (2.3.2)
    3. I2C DMA lab (2.3.3)
    4. I2C Memory mode lab (2.3.4)

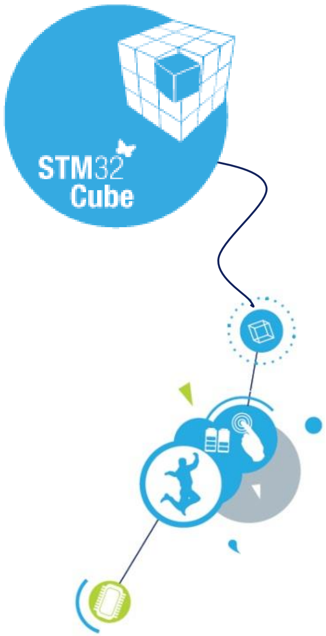


3 Timing peripherals



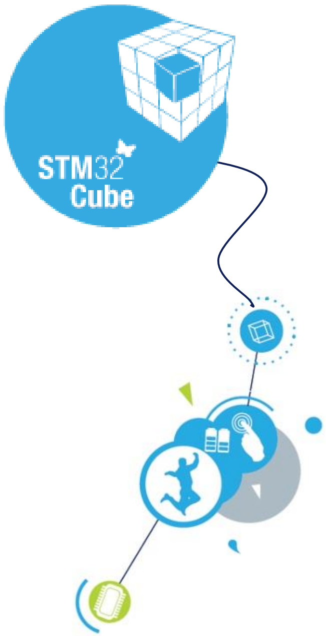
4 Analog peripherals

1. DAC (4.1)
    1. DAC wave generation lab (4.1.1)
2. ADC (4.2)
    1. ADC Poll lab (4.2.1)
    2. ADC Interrupt lab (4.2.2)
    3. ADC DMA lab (4.2.3)



5. BSP package

1. FMC SDRAM BSP lab (5.1)
2. LCD BSP Print text lab (5.2)
3. I2C BSP EEPROM lab (5.3)
4. SPI BSP GYROSCOPE lab (5.4)



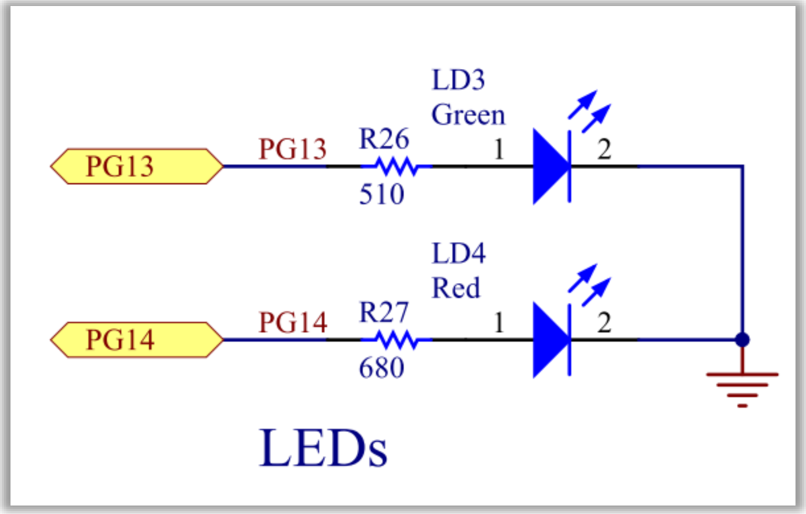
1.1.1 GPIO Lab

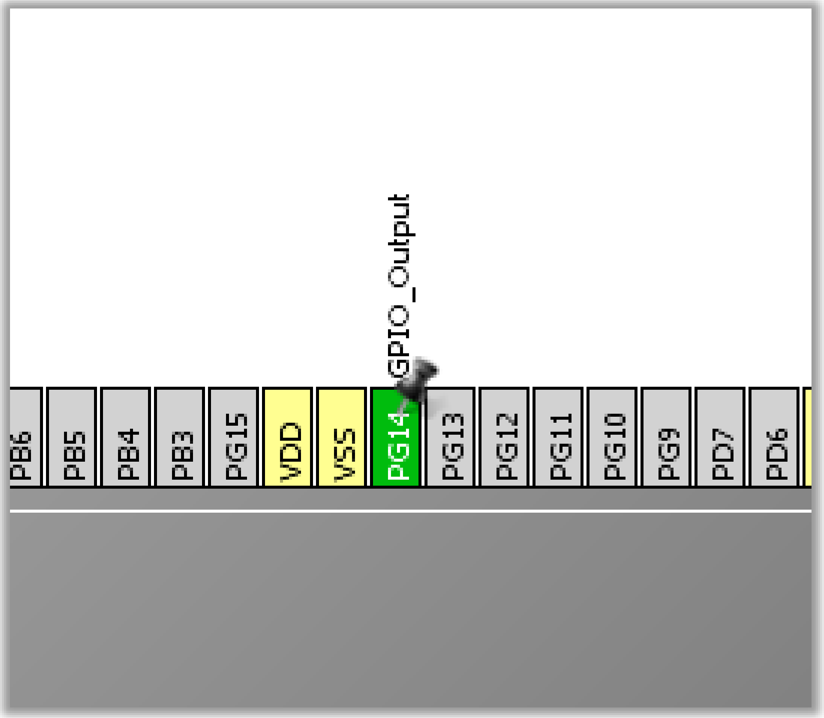
1.1.1 Configure GPIO for LED toggling

- 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

1.1.1 Configure GPIO for LED toggling

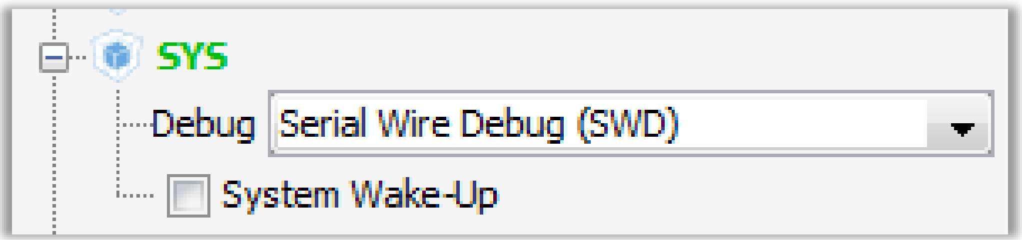
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Configure LED pin as GPIO_Output

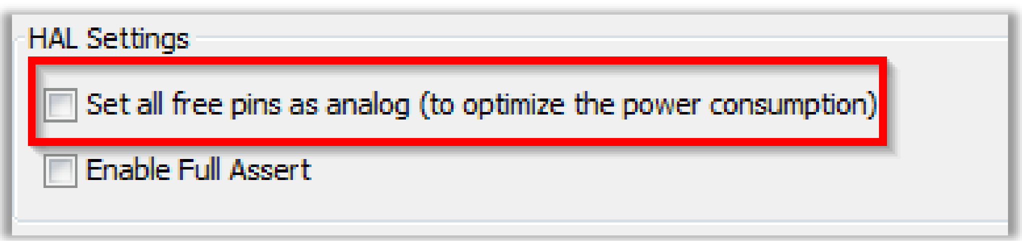


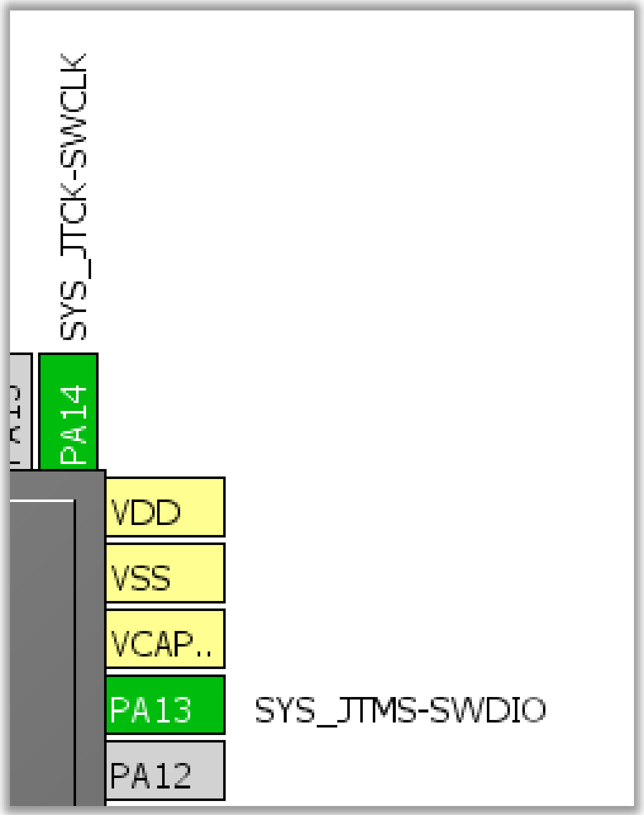


1.1.1 Configure GPIO for LED toggling

- 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**





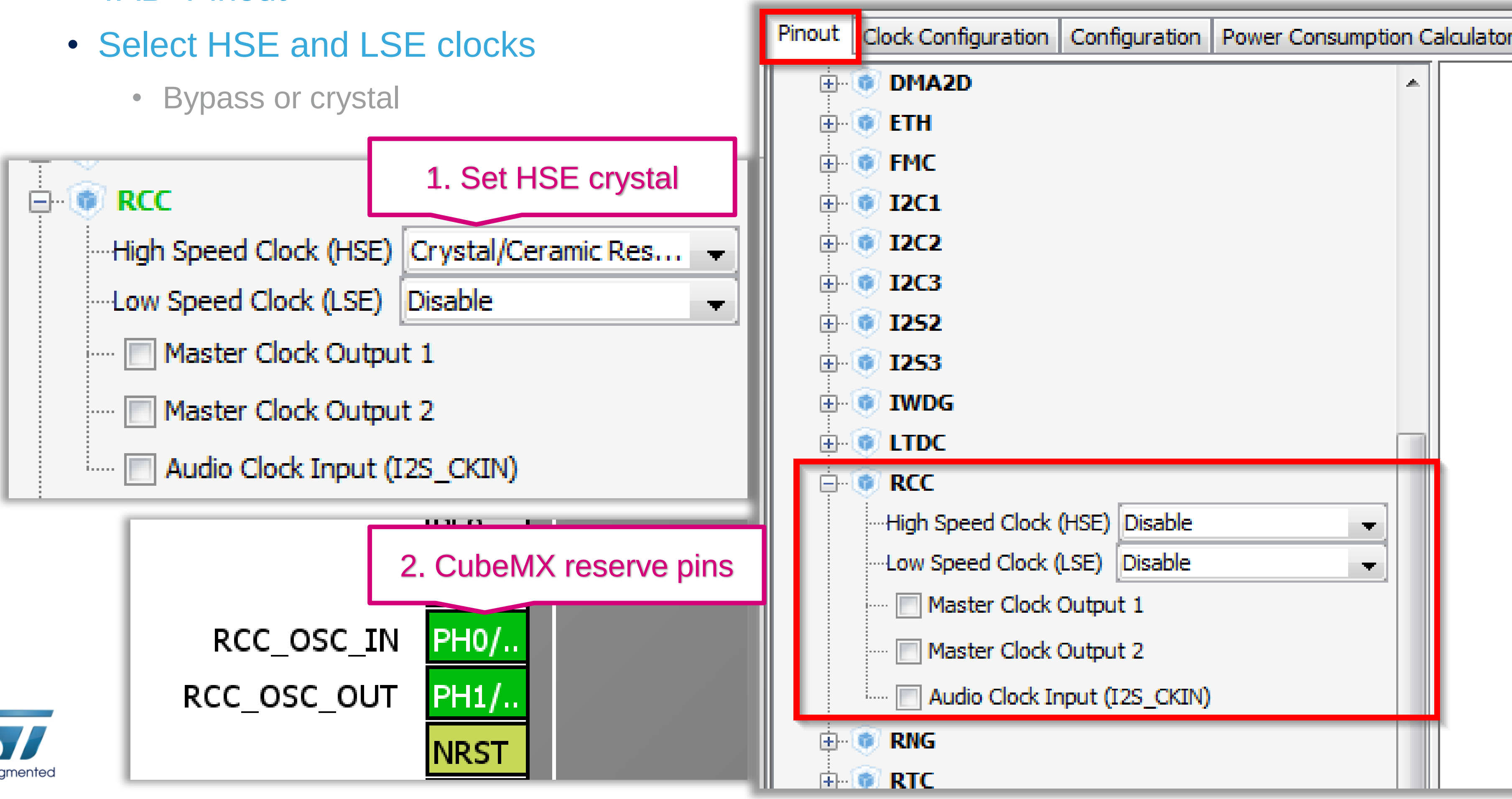


life.augmented

1.1.1 Configure GPIO for LED toggling

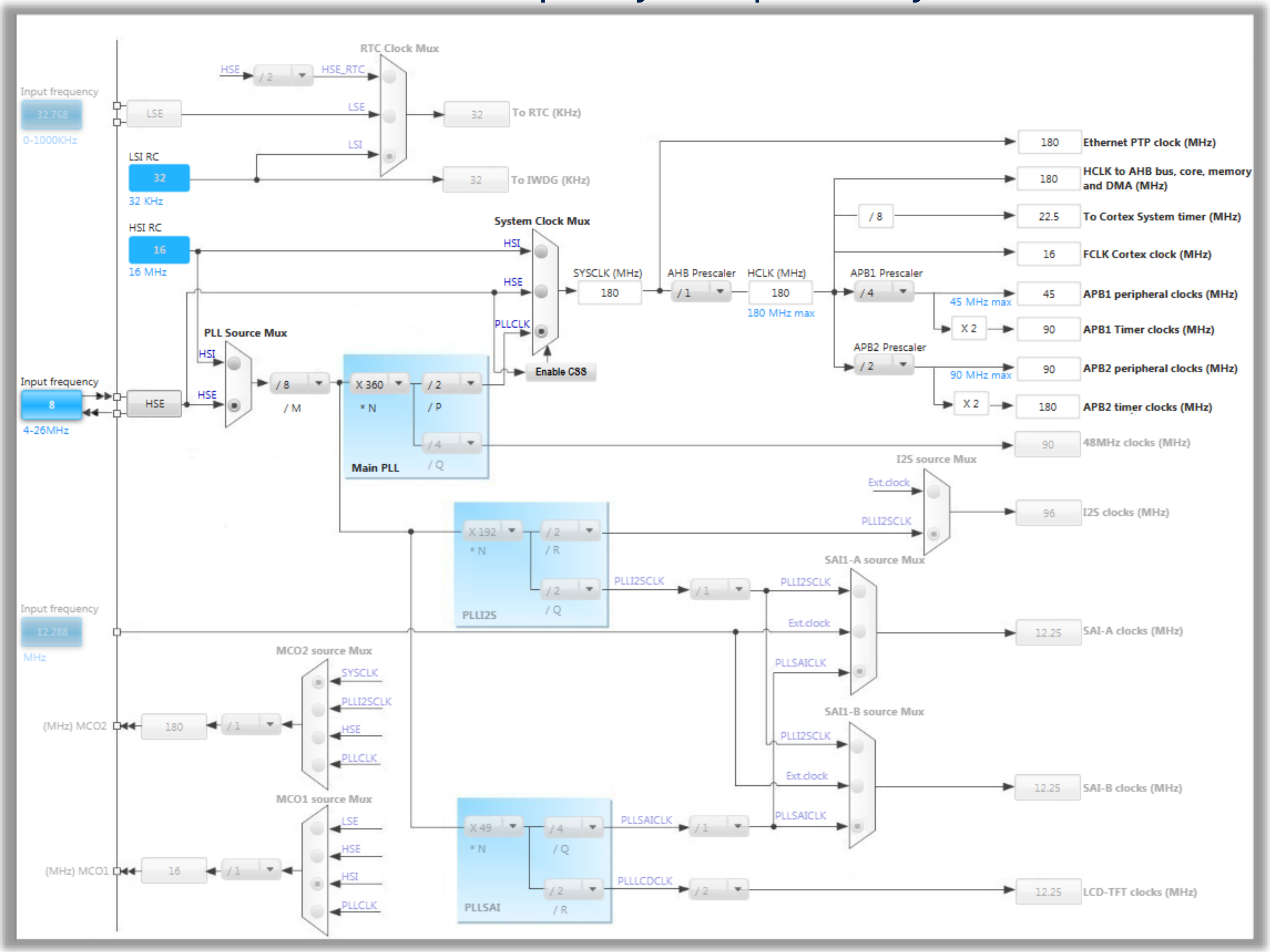
Clock Configuration overview 10

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



1.1.1 Configure GPIO for LED toggling

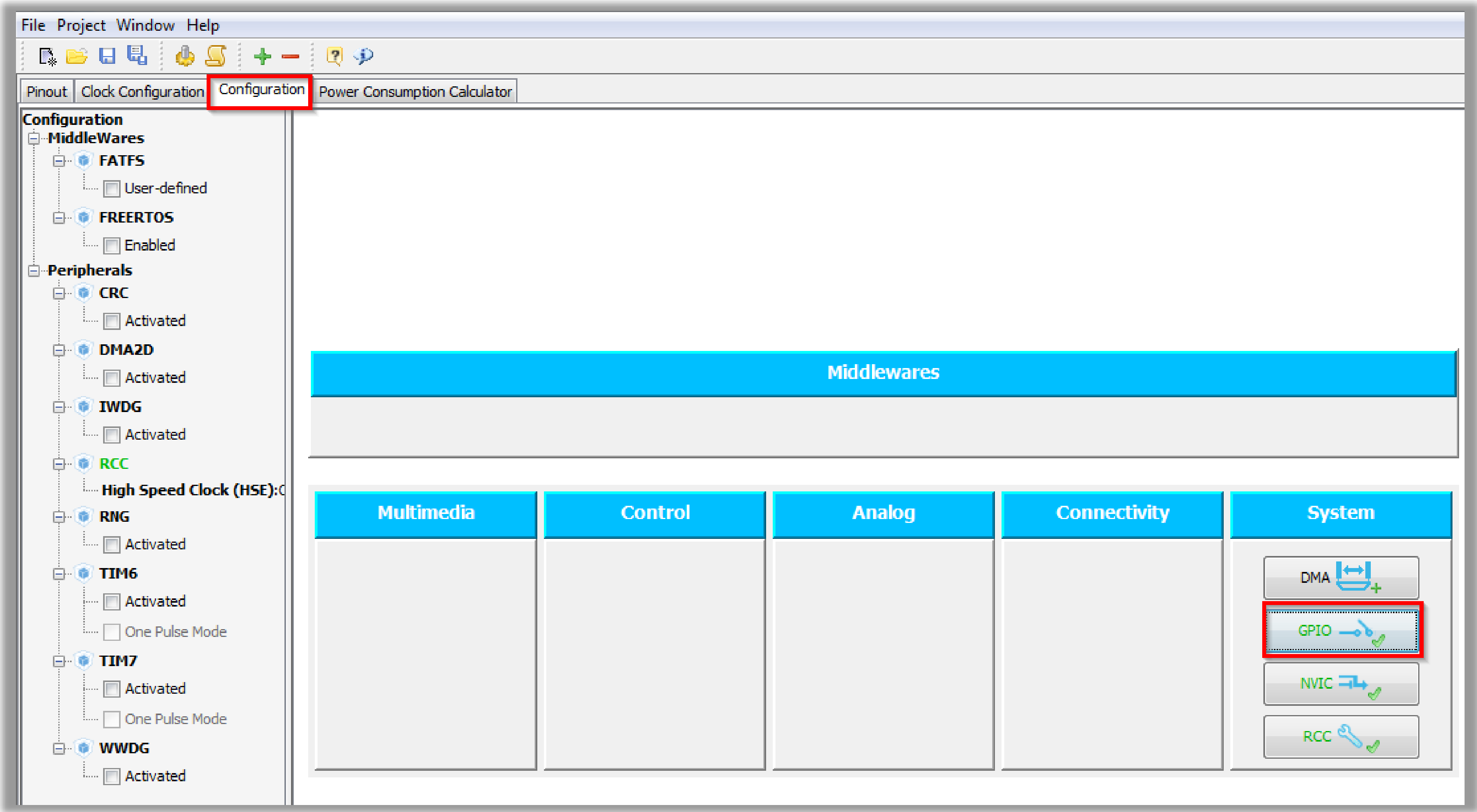
- In order to run on maximum frequency, setup clock system



life.augmented

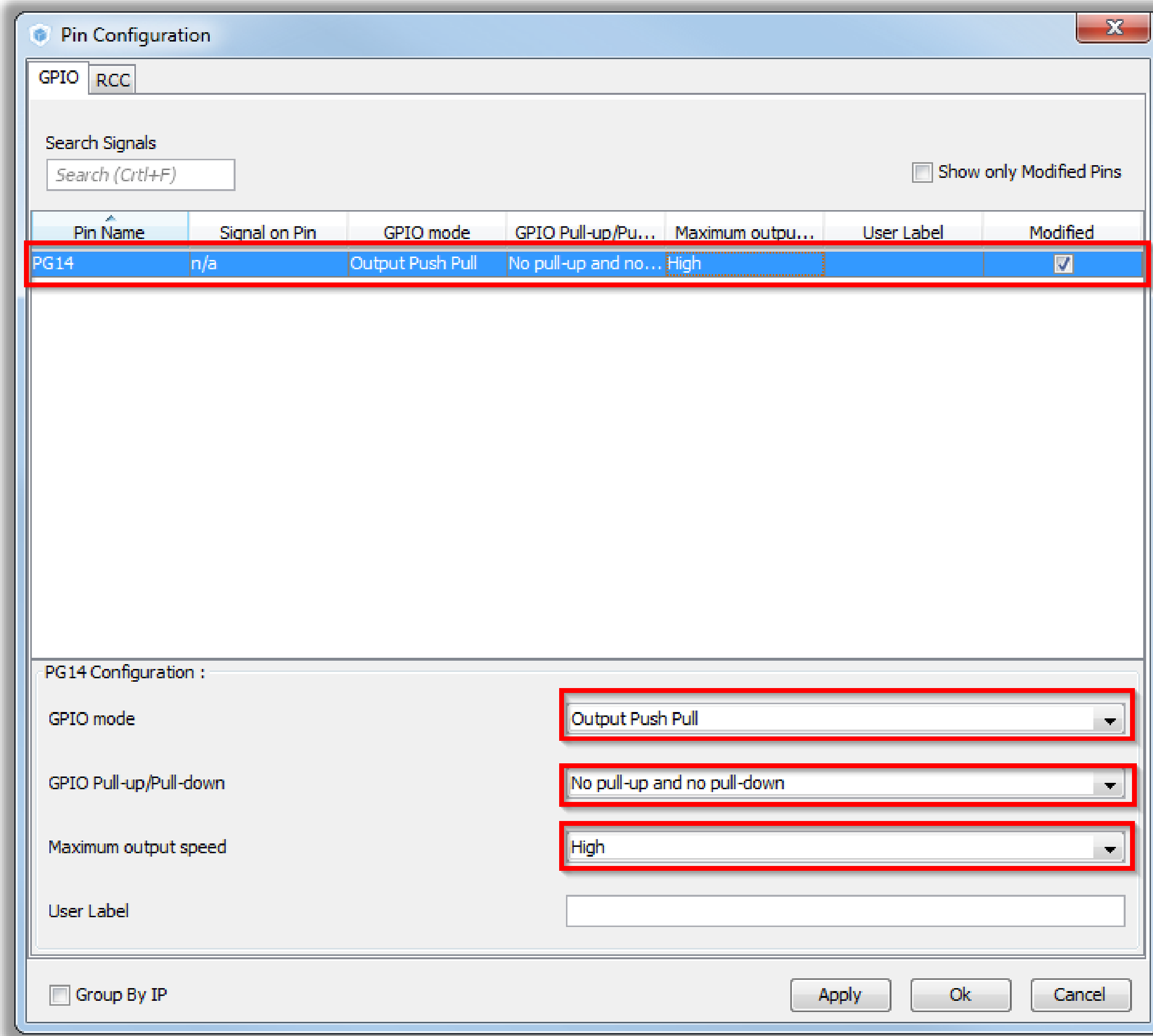
1.1.1 Configure GPIO for LED toggling

- GPIO Configuration
  - TAB>Configuration>System>GPIO



1.1.1 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

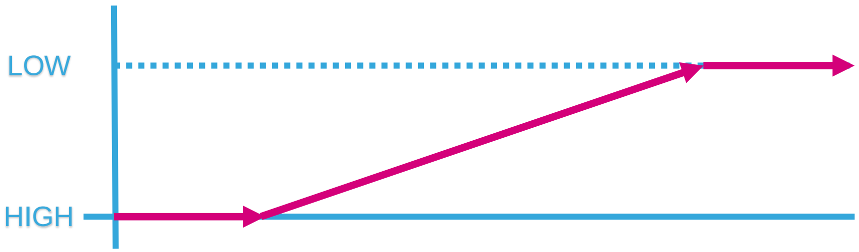


life.augmented

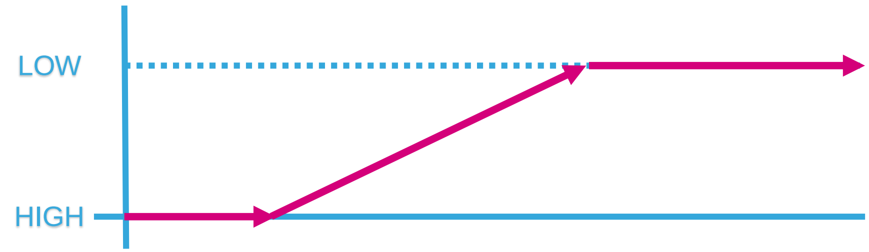
1.1.1 Configure GPIO for LED toggling

- 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

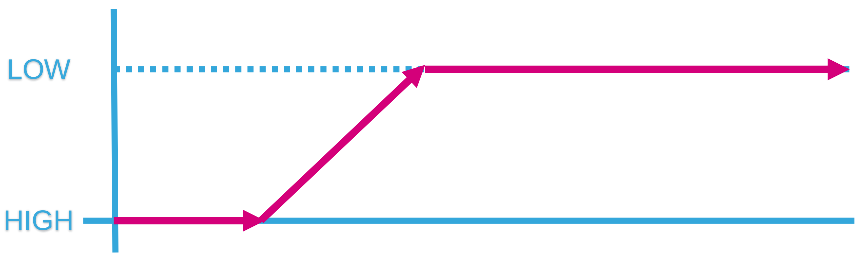
GPIO output LOW speed



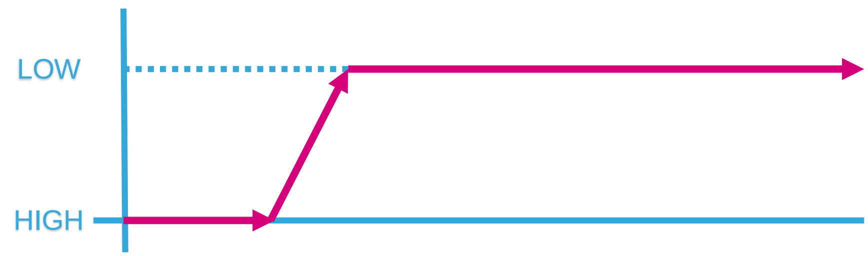
GPIO output MEDIUM speed



GPIO output FAST speed

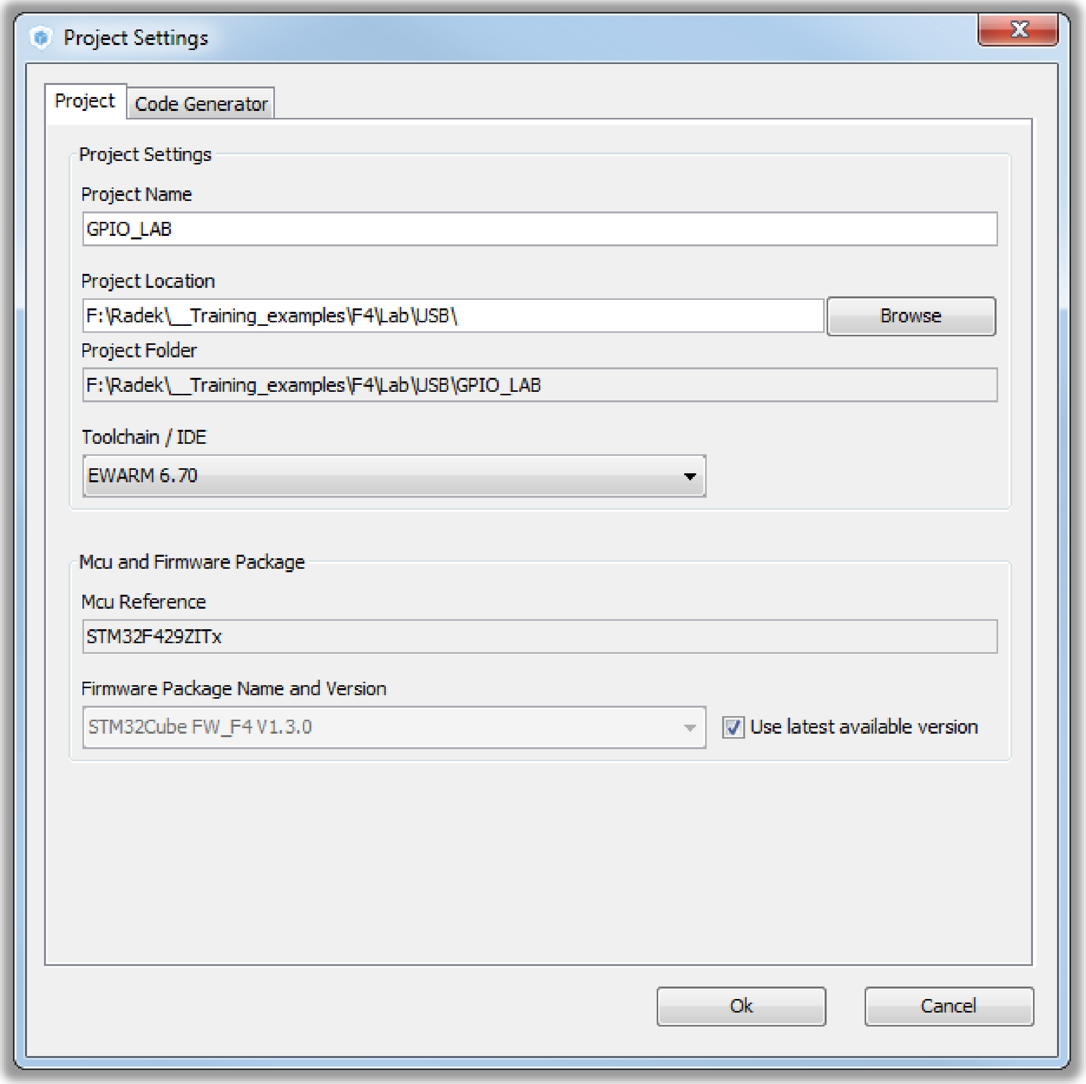


GPIO output HIGH speed



1.1.1 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



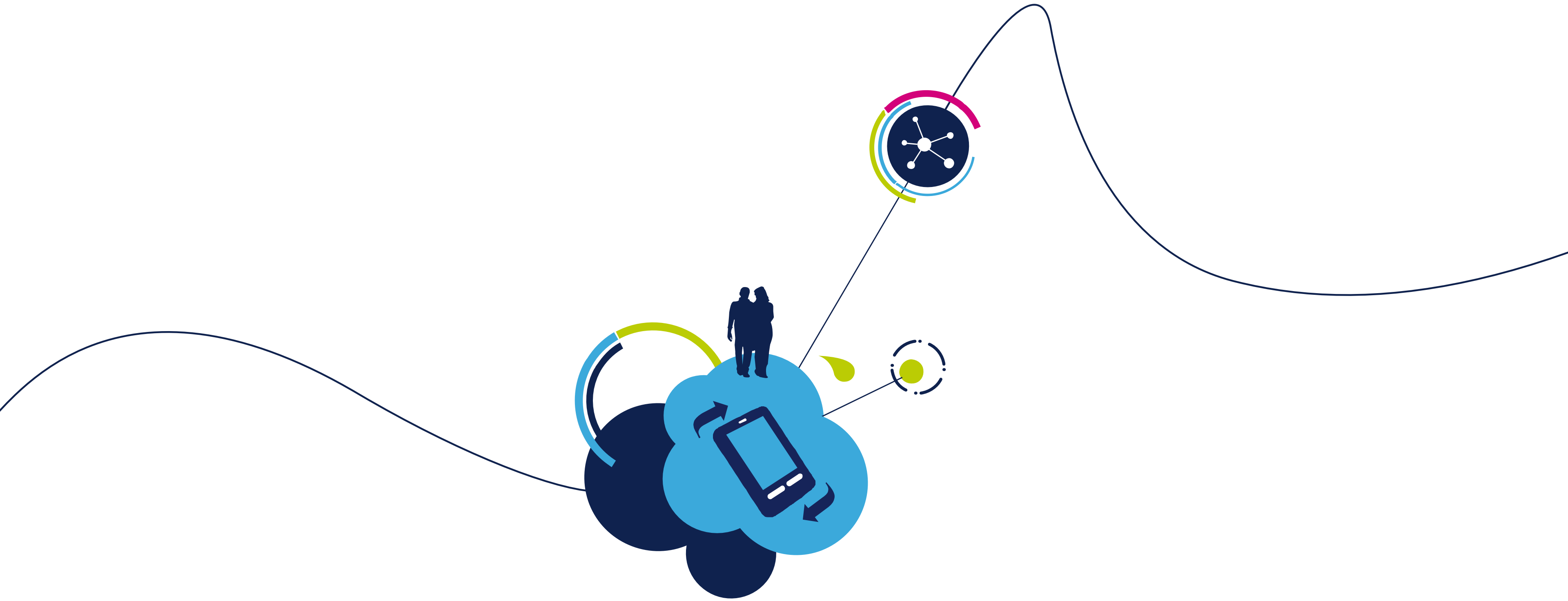
1.1.1 Configure GPIO for LED toggling

- 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*

1.1.1 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*

```
/* USER CODE BEGIN 3 */
 /* Infinite loop */
 while (1)
 {
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
   HAL_Delay(500);

   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);
   HAL_Delay(500);
 }
 /* USER CODE END 3 */
```



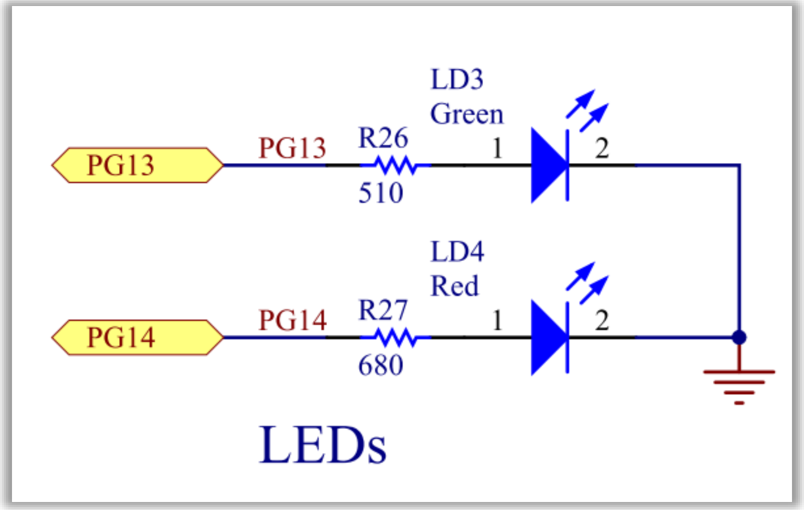
1.1.2 EXTI lab

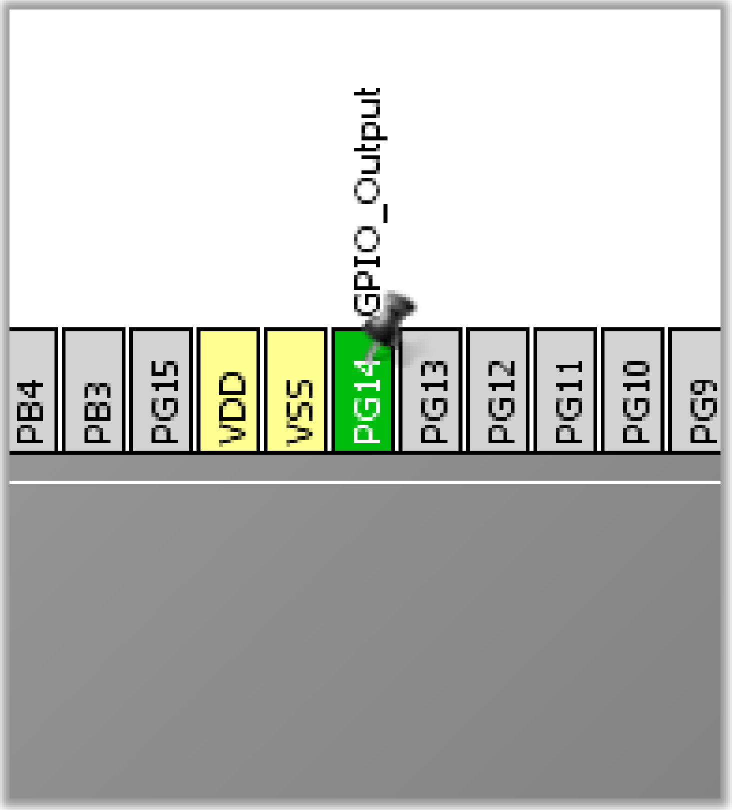
1.1.2 Configure EXTI to turn on LED

- 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

1.1.2 Configure EXTI to turn on LED

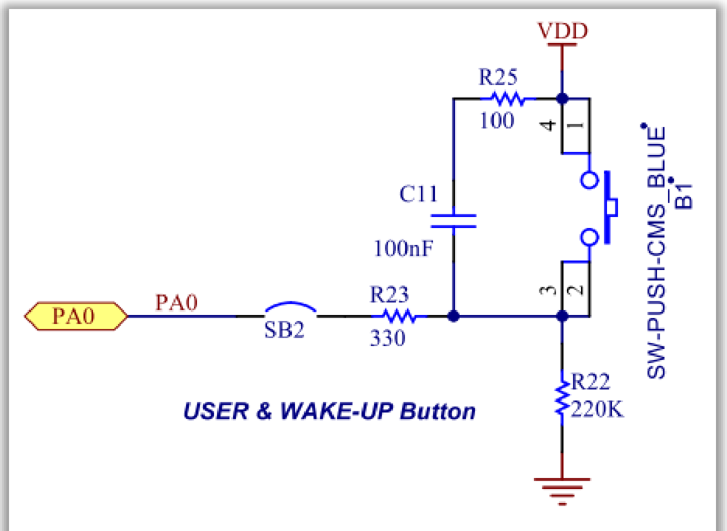
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Configure LED pin as GPIO_Output
- Configure Button pin as GPIO_EXTIX

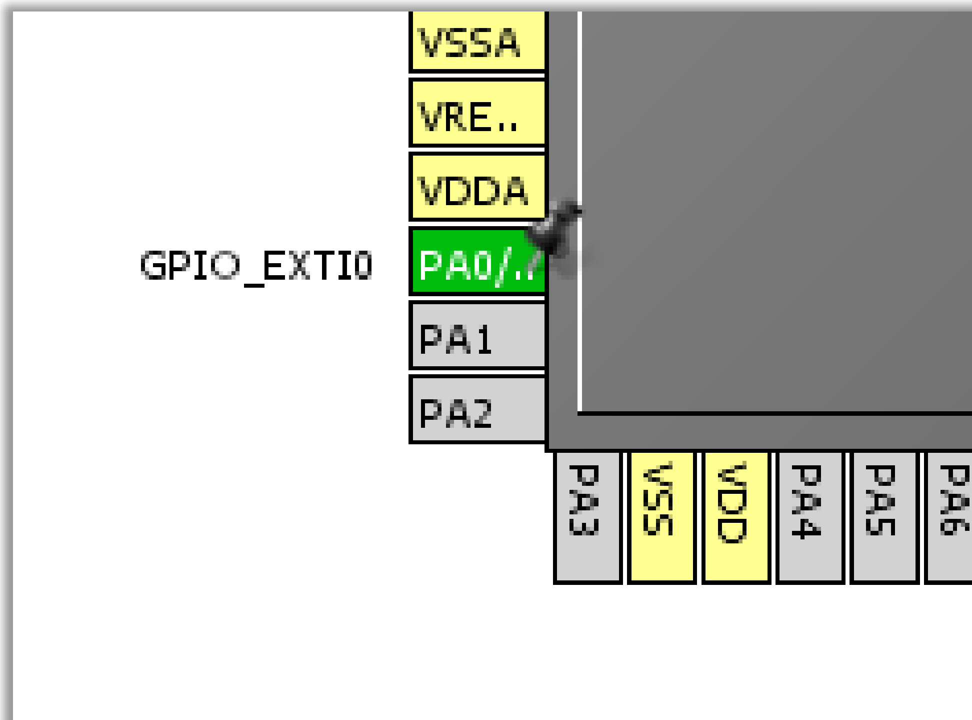




1.1.2 Configure EXTI to turn on LED

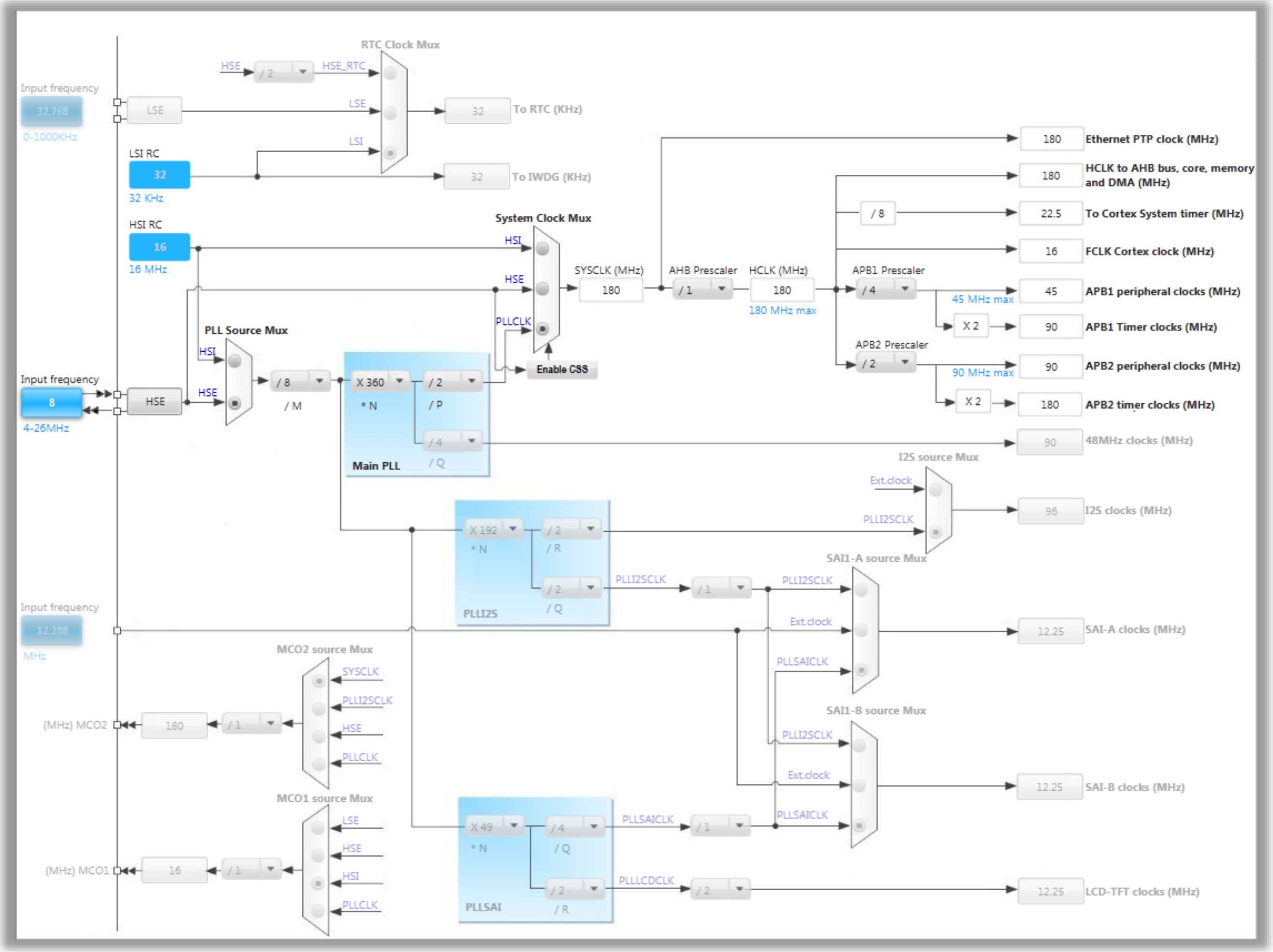
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Configure LED pin as GPIO_Output
- Configure Button pin as GPIO_EXTIX





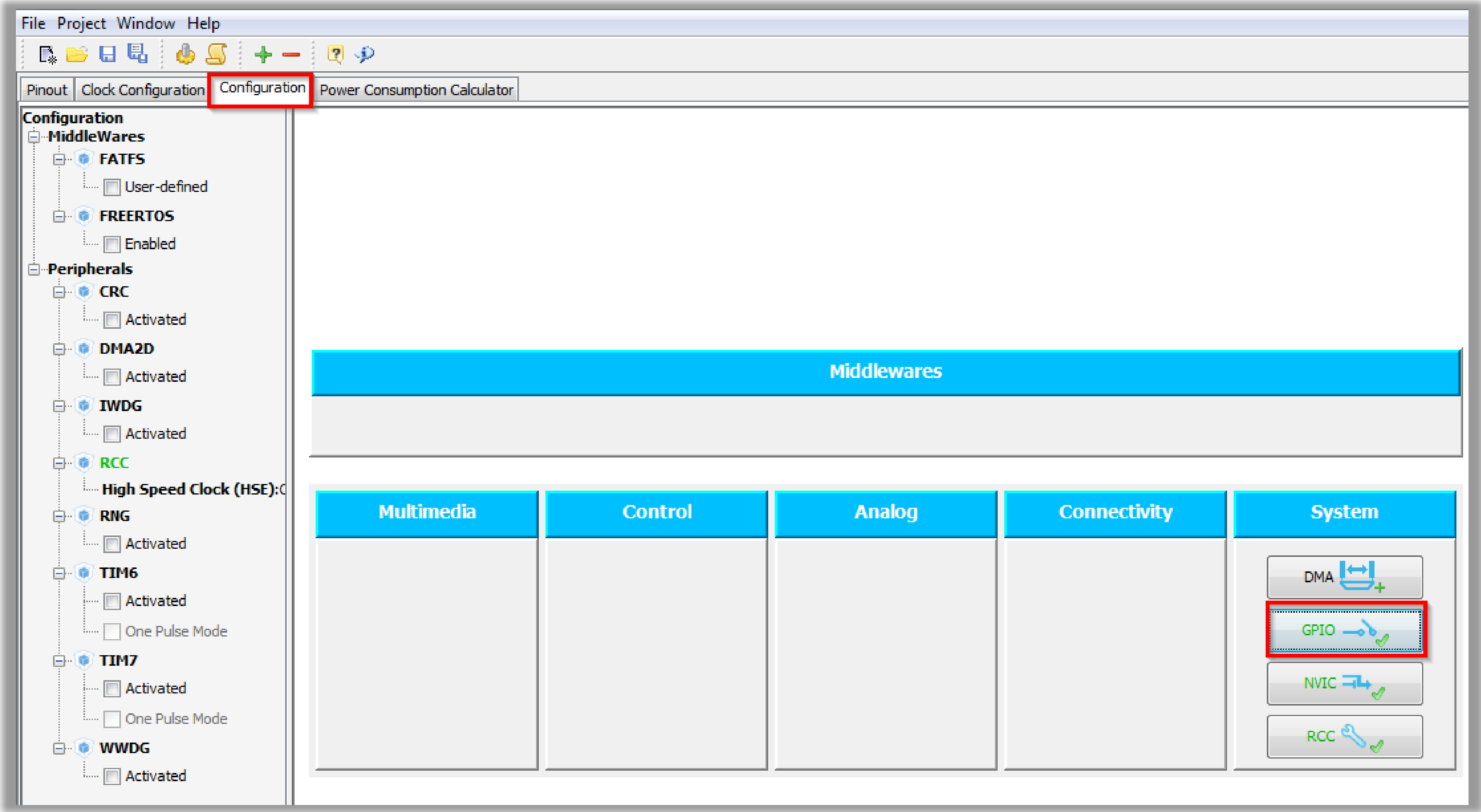
1.1.2 Configure EXTI to turn on LED

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



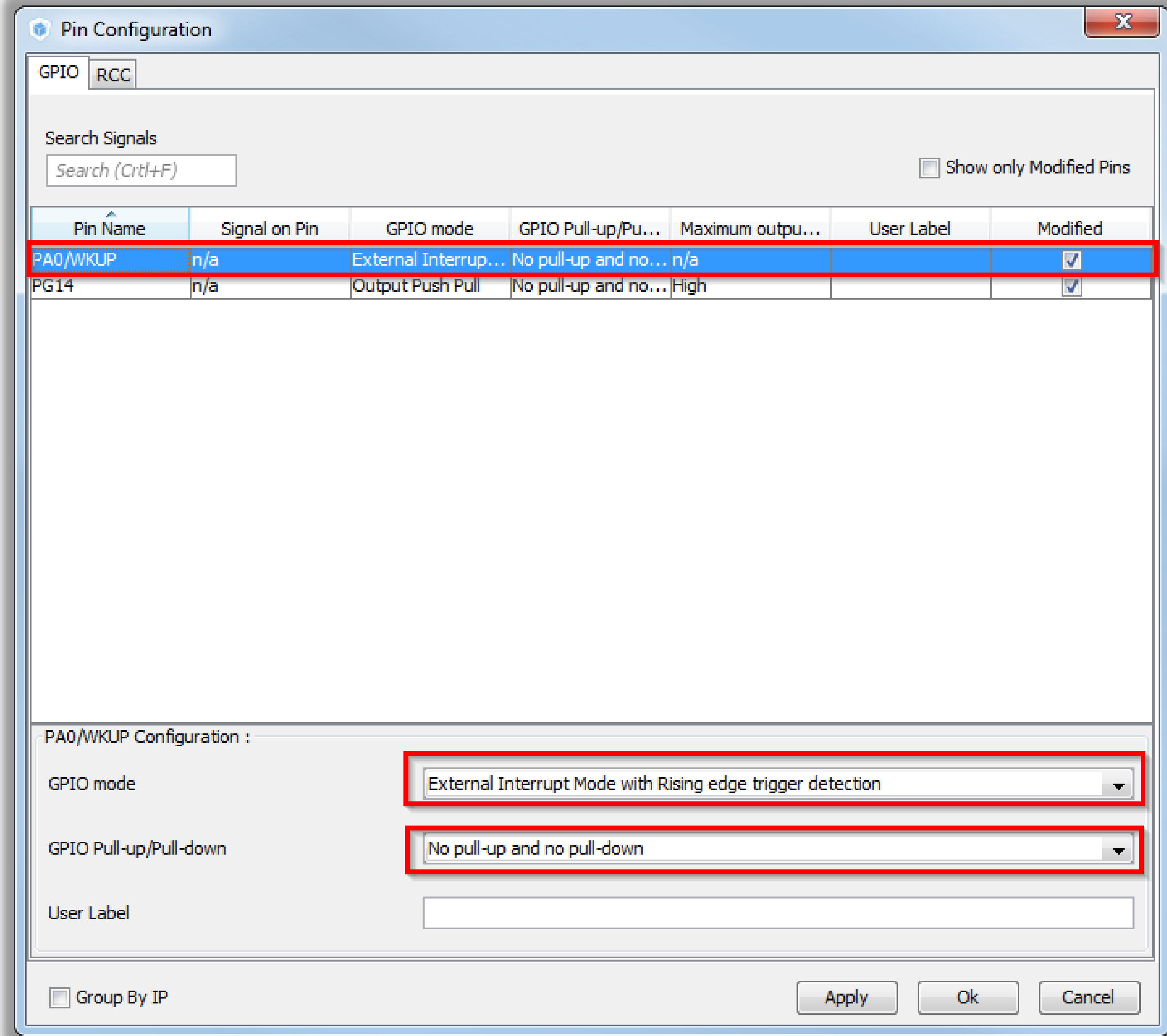
1.1.2 Configure EXTI to turn on LED

- GPIO Configuration
  - TAB>Configuration>System>GPIO



1.1.2 Configure EXTI to turn on LED

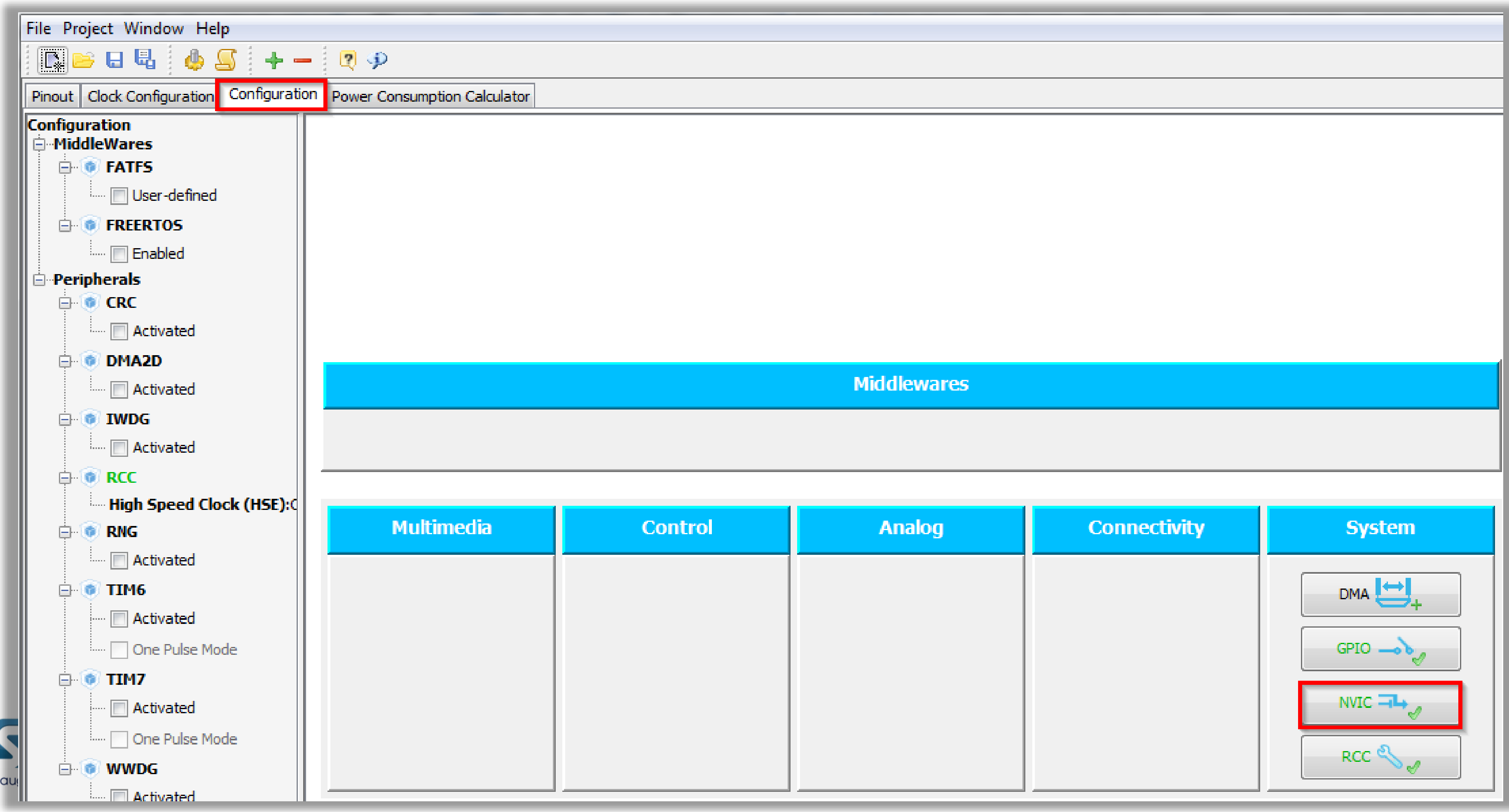
- GPIO(Pin) Configuration
  - Select External Interrupt Mode with Rising edge trigger detection
  - No pull-up or pull-down
  - PG14 can be let in default settings
  - Button OK



life.augmented

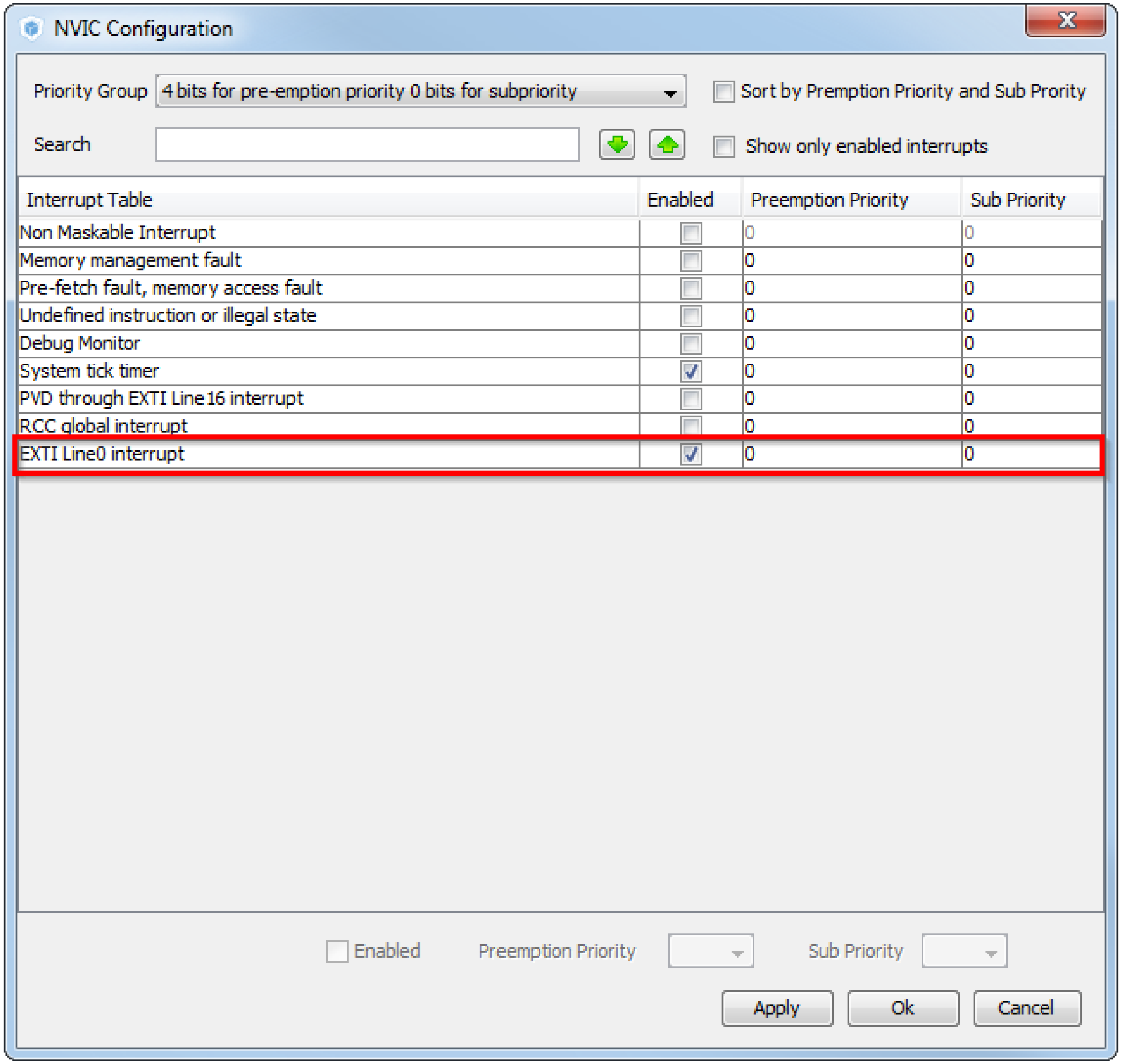
1.1.2 Configure EXTI to turn on LED

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



1.1.2 Configure EXTI to turn on LED

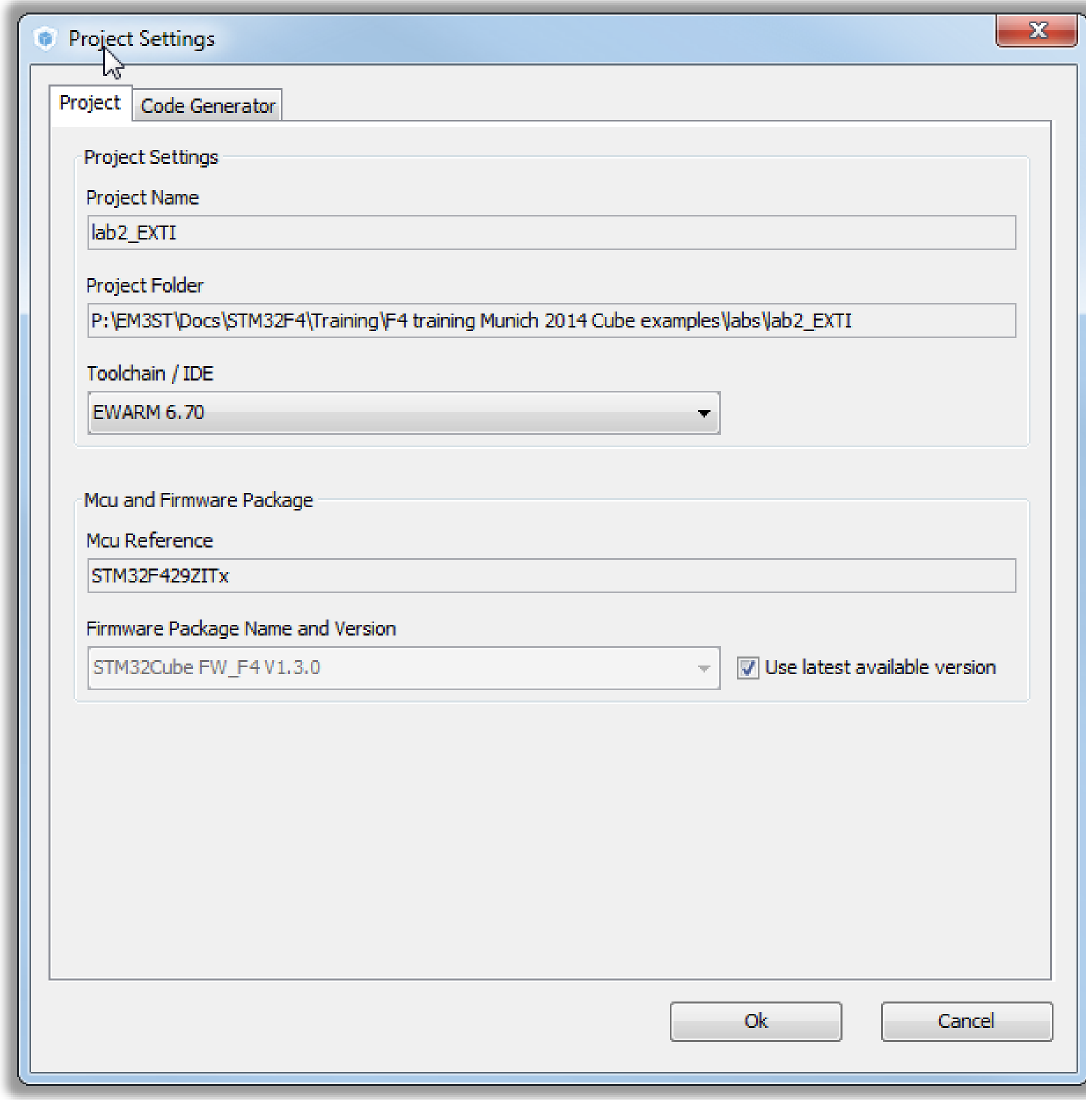
- NVIC Configuration
  - Enable interrupt for EXTI Line0
  - Button OK



life.augmented

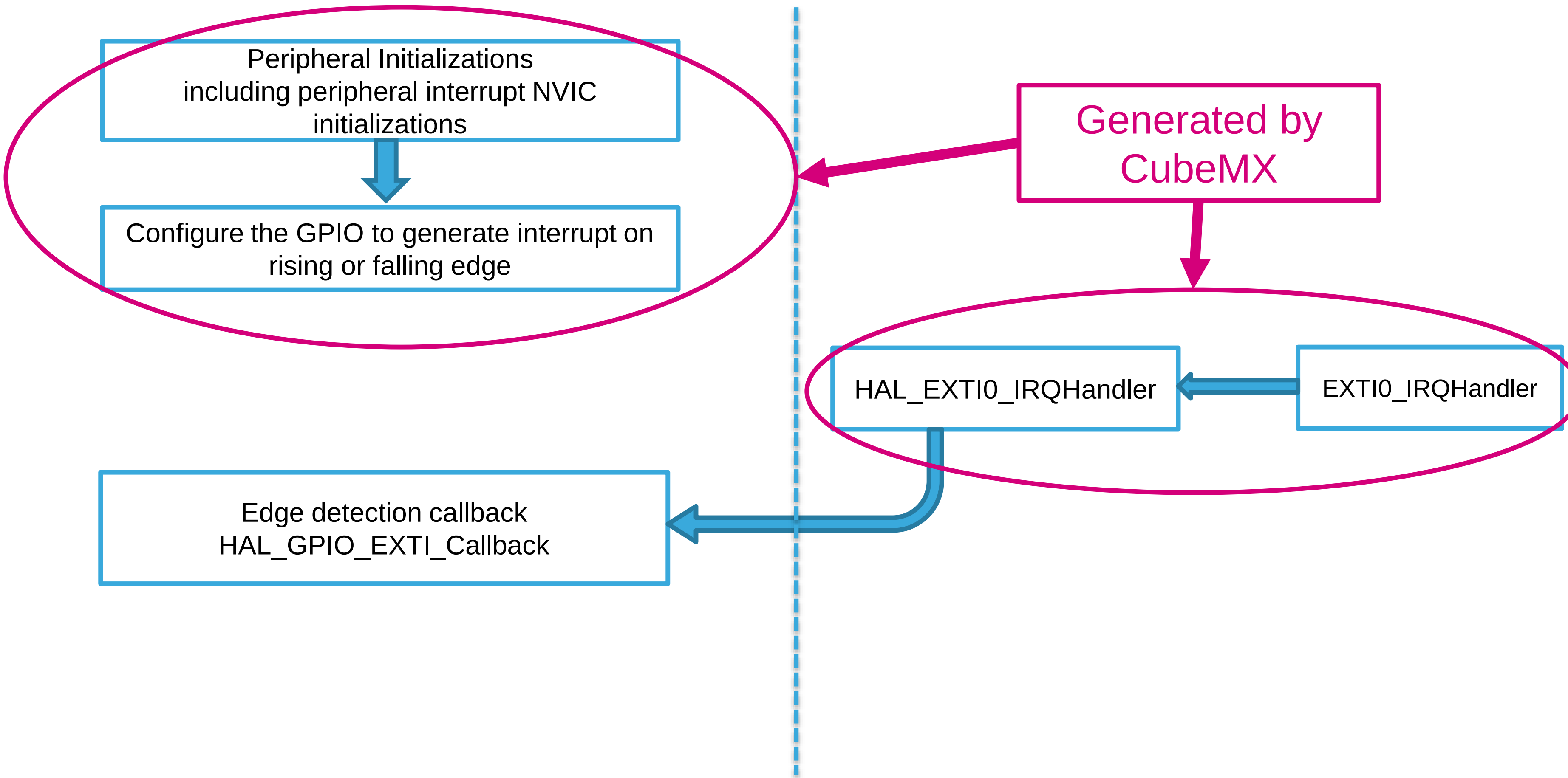
1.1.2 Configure EXTI to turn on LED

- 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



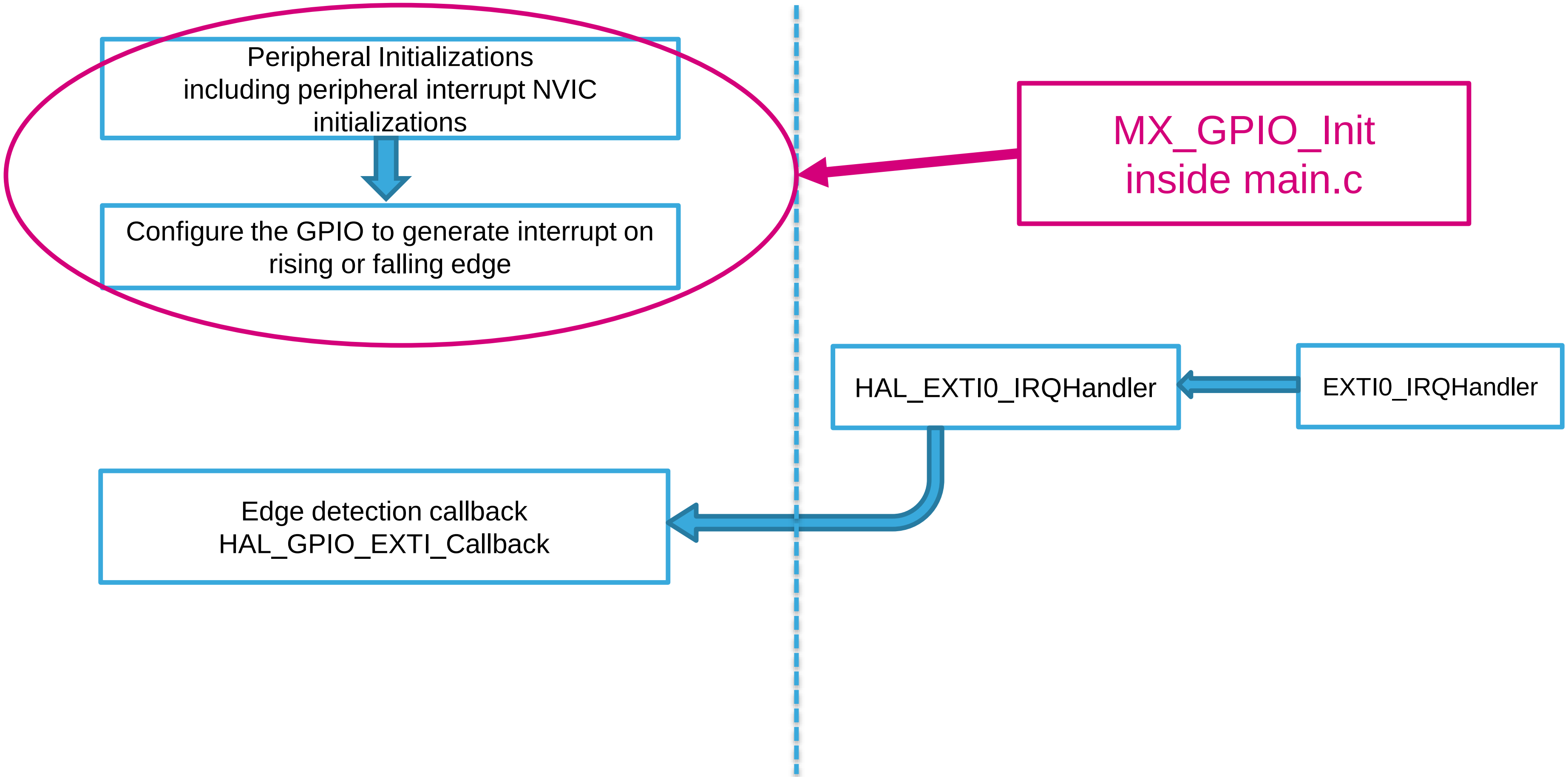
1.1.2 Configure EXTI to turn on LED

HAL Library work flow 1



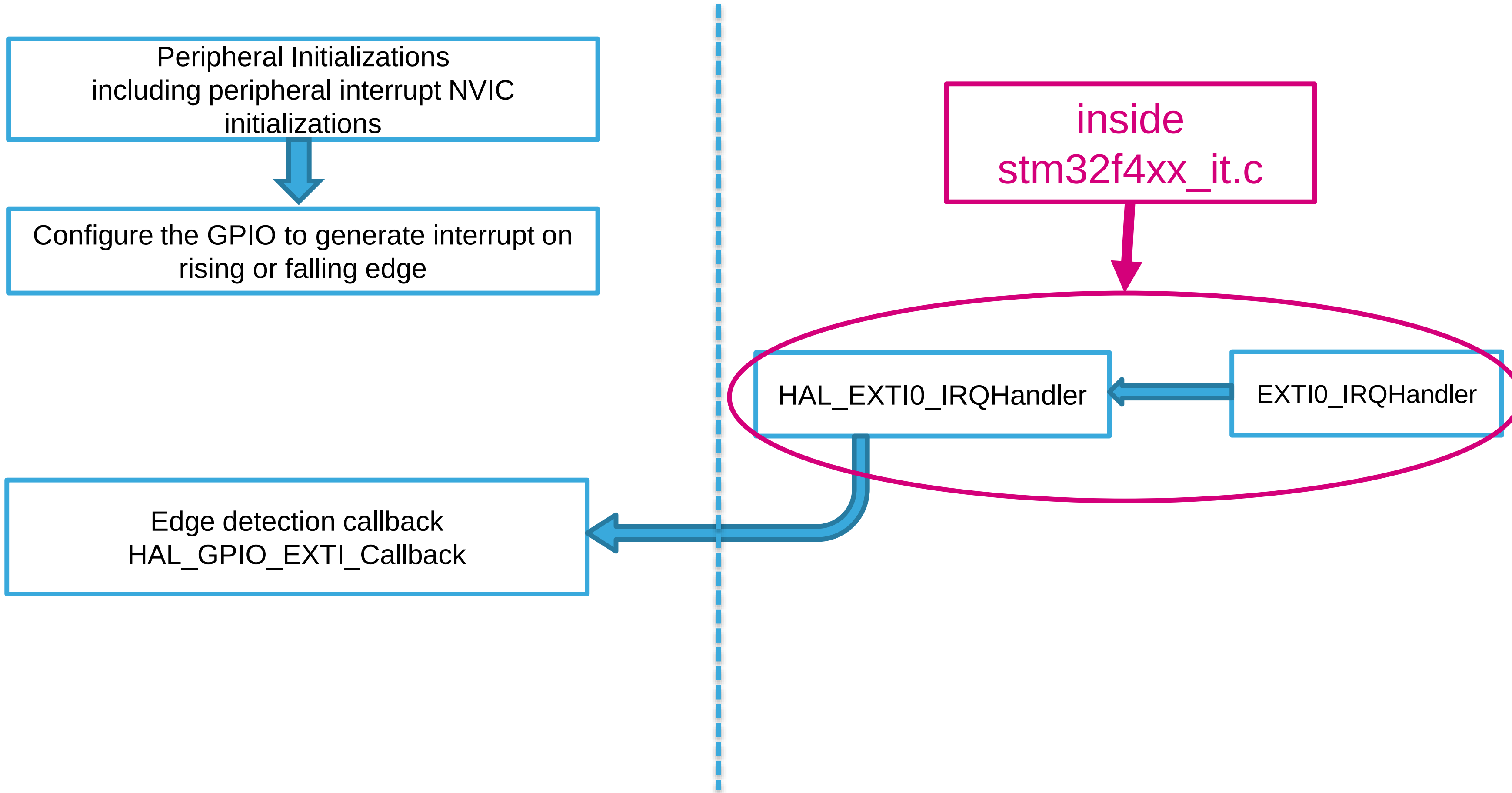
1.1.2 Configure EXTI to turn on LED

HAL Library work flow 2



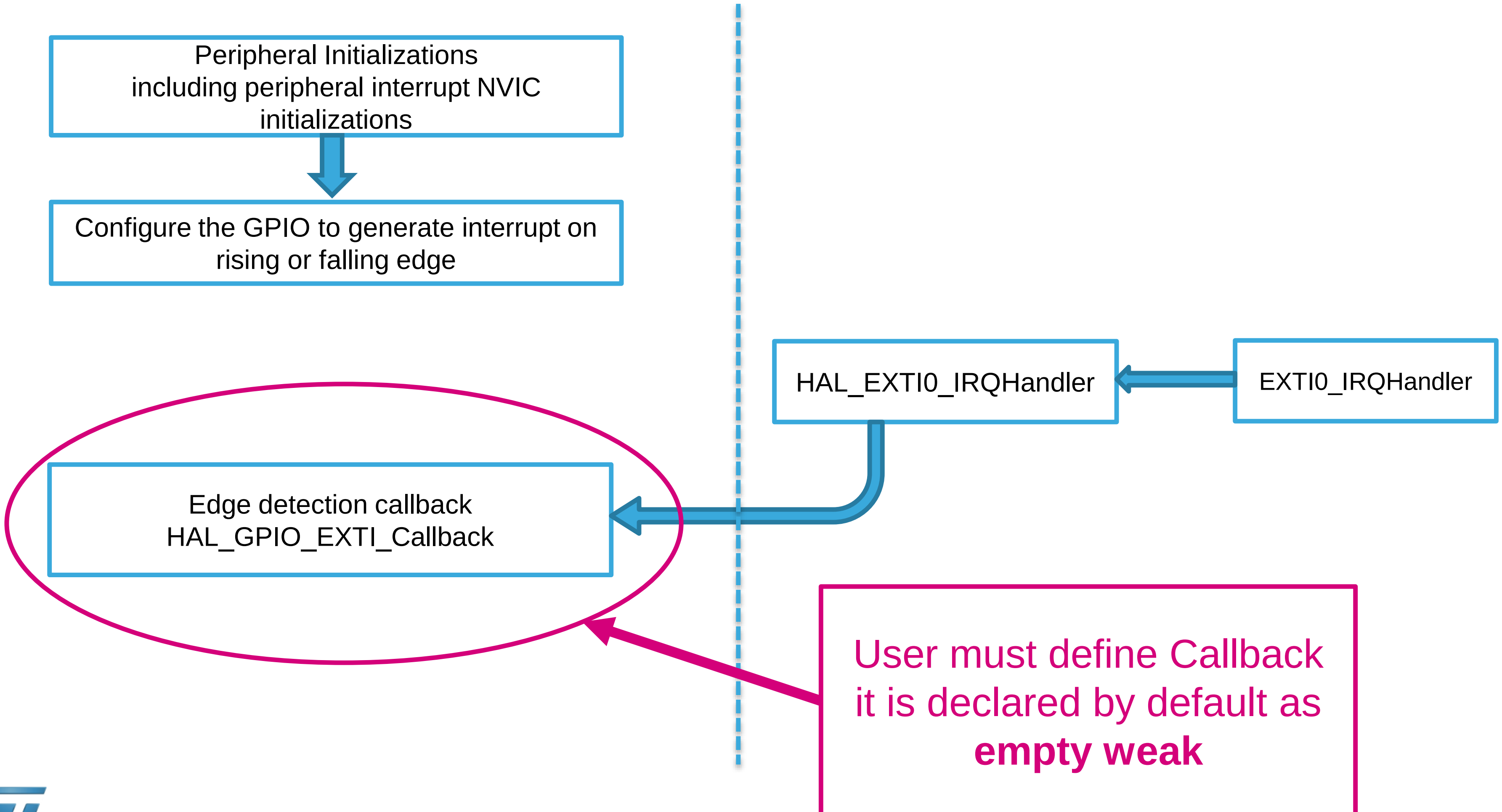
1.1.2 Configure EXTI to turn on LED

HAL Library working flow 3



1.1.2 Configure EXTI to turn on LED

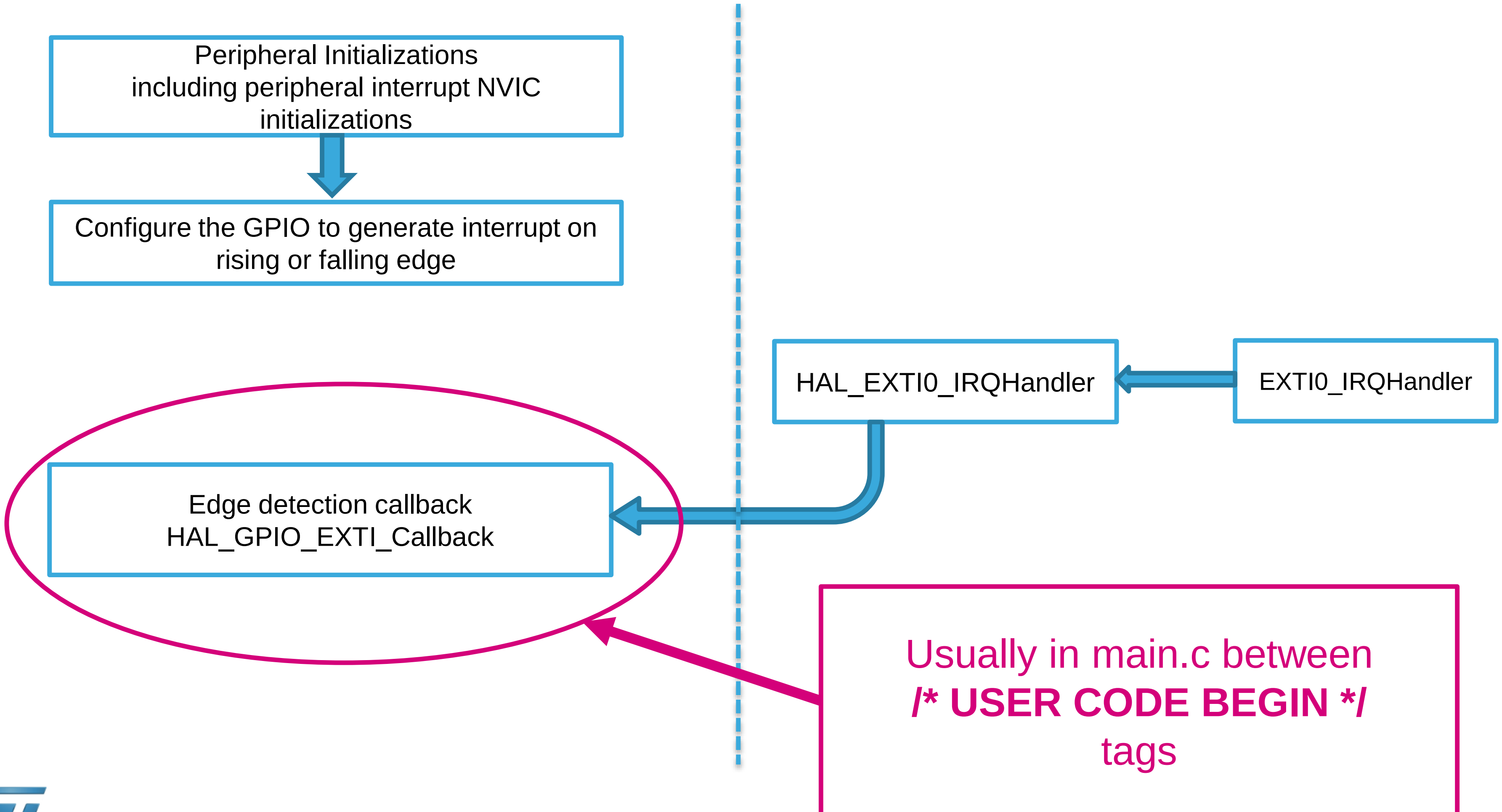
HAL Library work flow 4



life.augmented

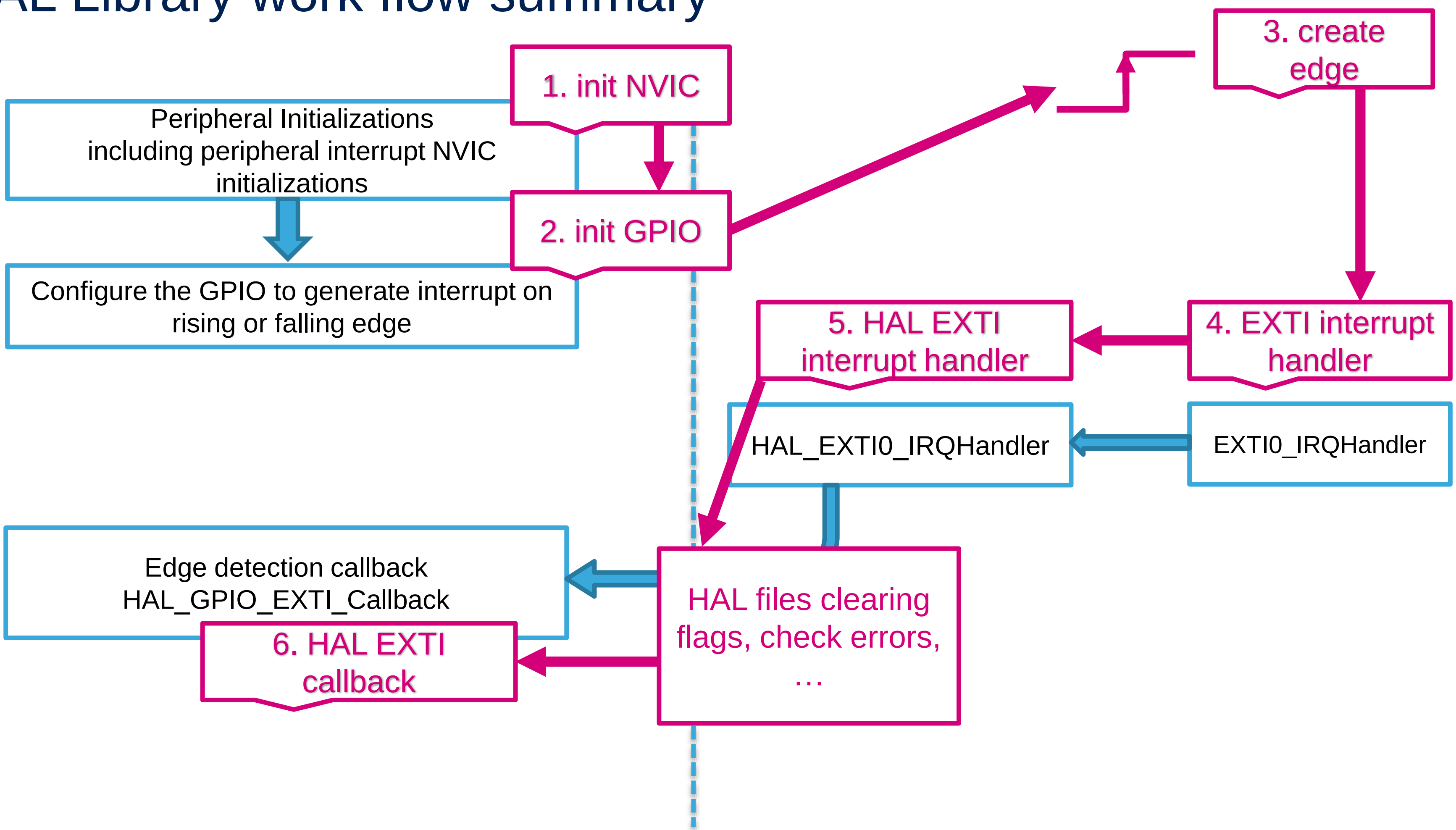
1.1.2 Configure EXTI to turn on LED

HAL Library work flow 5



1.1.2 Configure EXTI to turn on LED

HAL Library work flow summary



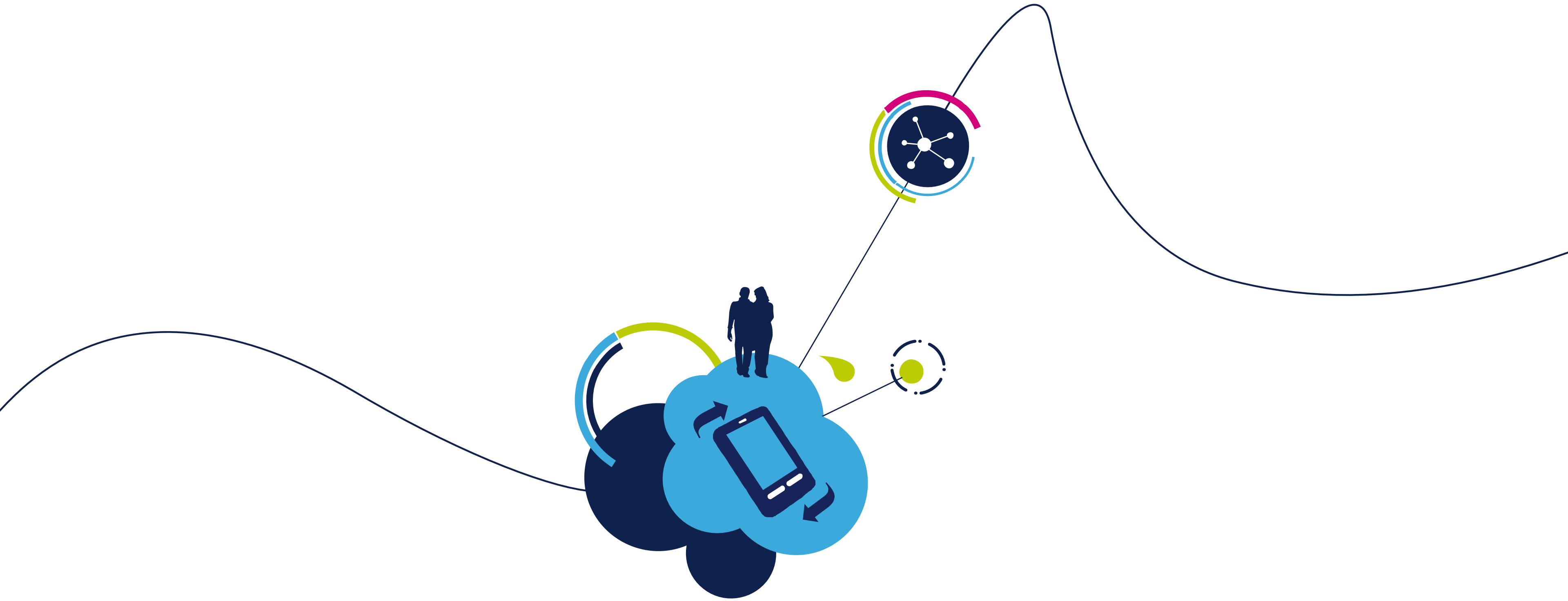
1.1.2 Configure EXTI to turn on LED

- 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*

1.1.2 Configure EXTI to turn on LED

- 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 */
```

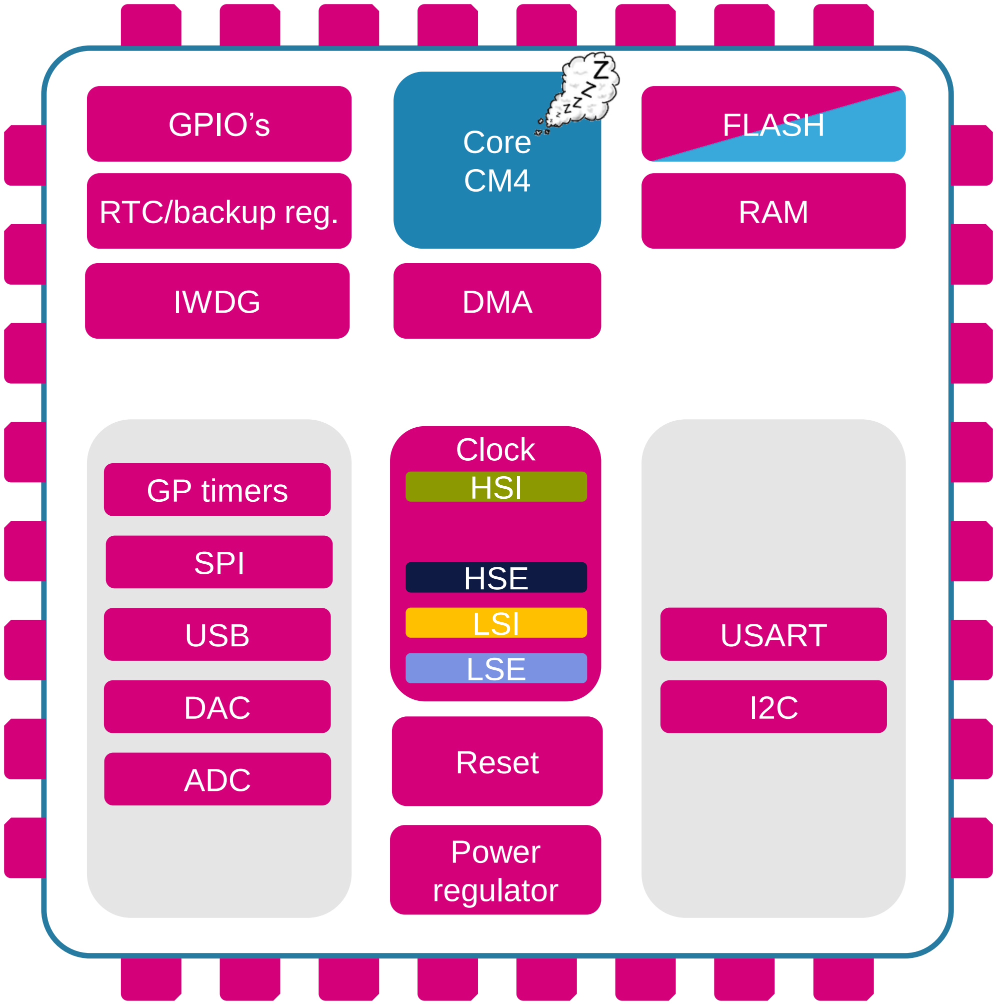


1.2.1 Low Power mode SLEEP lab

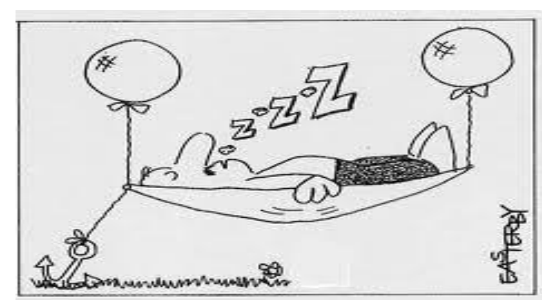
1.2.1 Use SLEEP mode with EXTI

- 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

1.2.1 **SLEEP** Mode



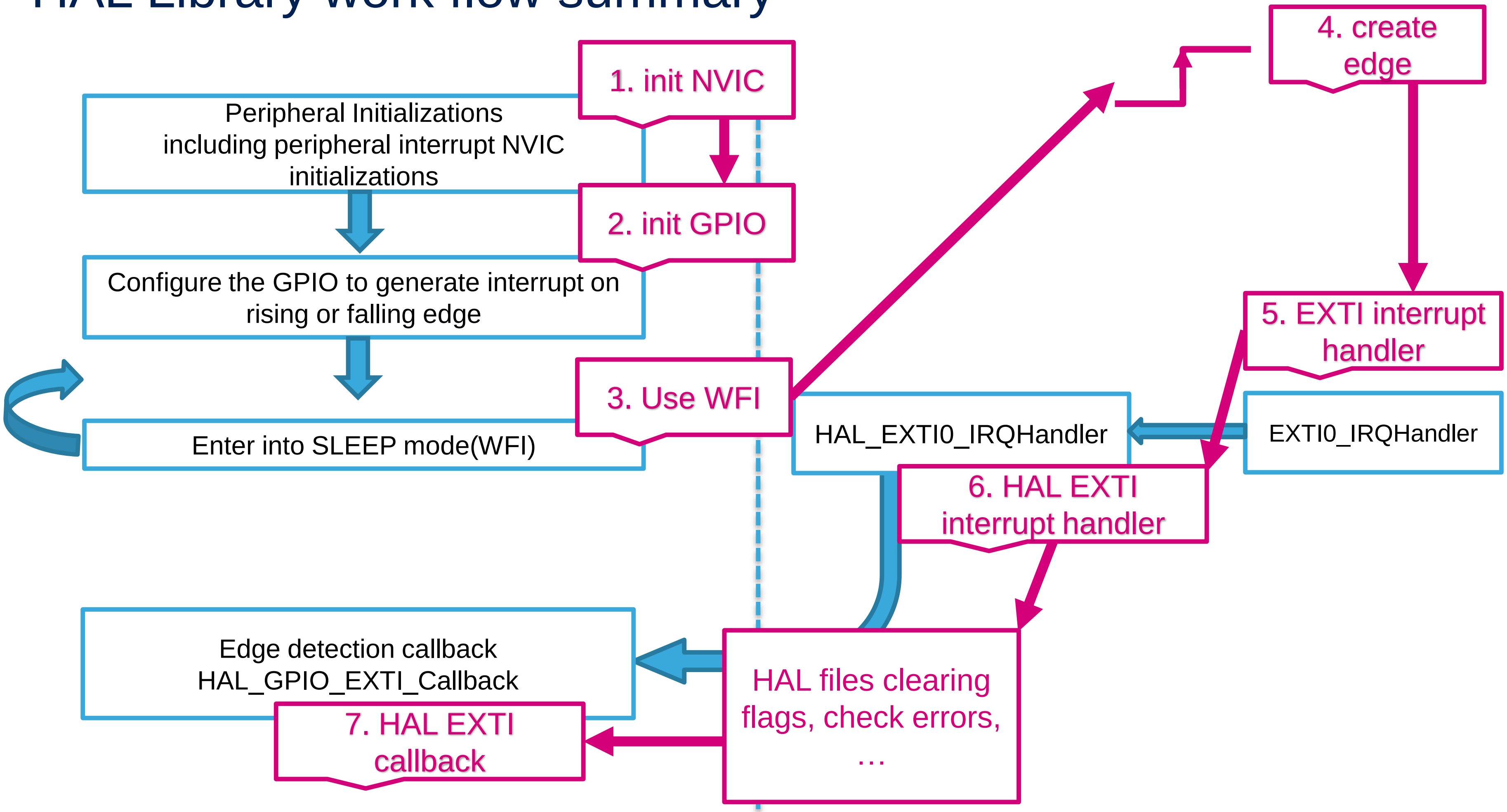
- **Core is stopped**
- **Peripherals are running**



life.augmented

1.2.1 Use SLEEP mode with EXTI

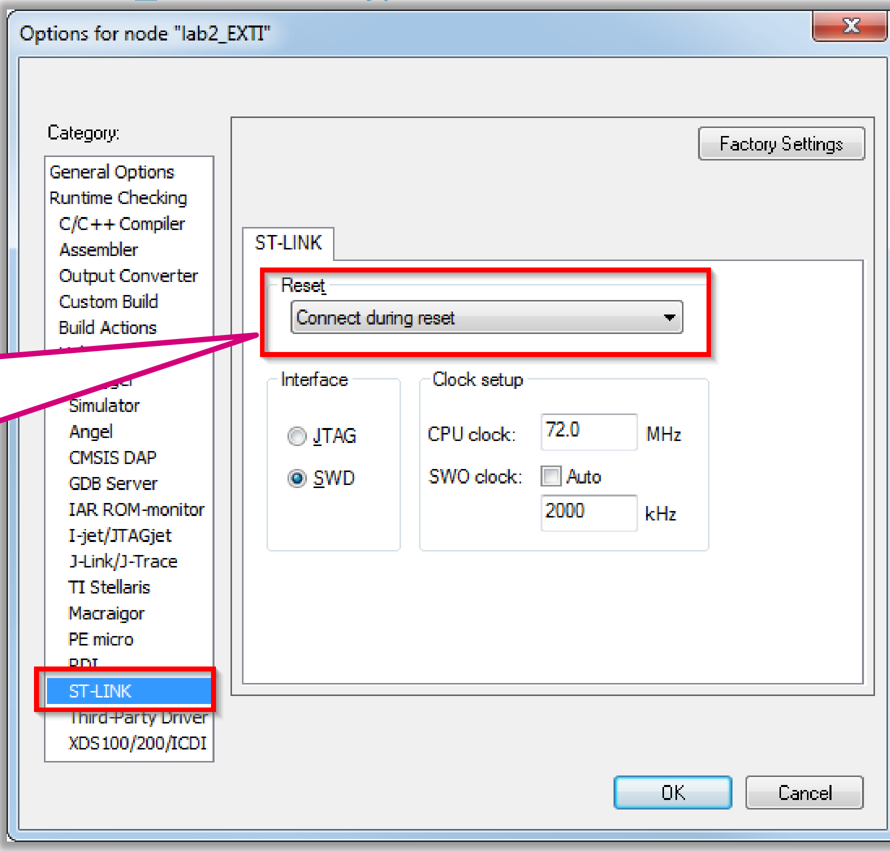
HAL Library work flow summary



1.2.1 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



life.augmented

1.2.1 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

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

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

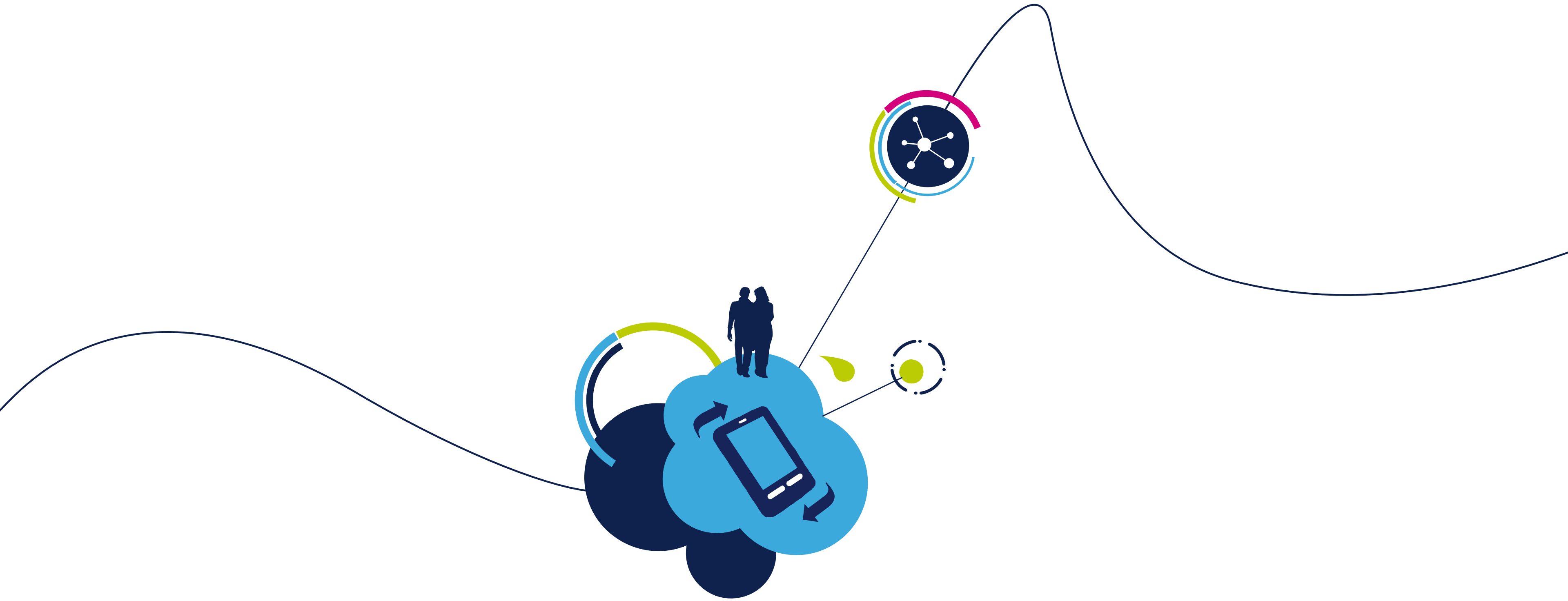
1.2.1 Use SLEEP mode with EXTI

- 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?

life.augmented

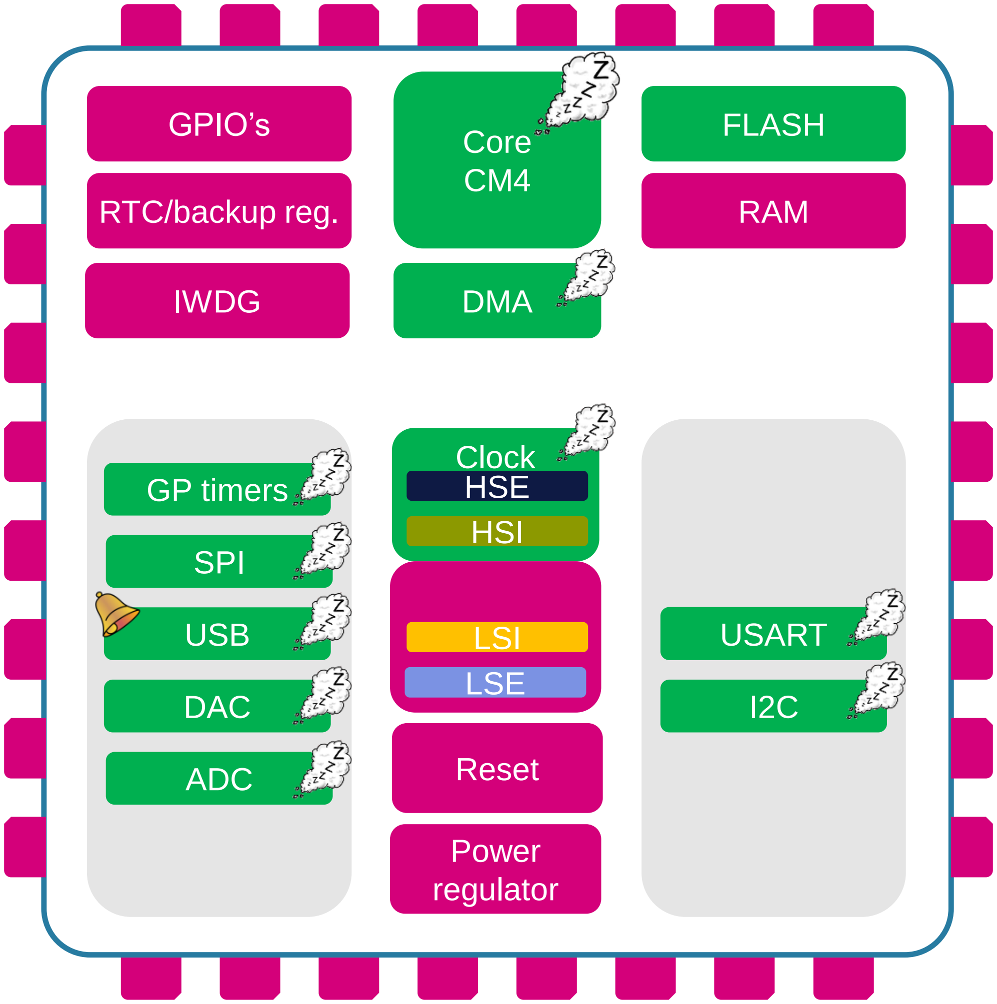


1.2.2 Low Power mode STOP lab

1.2.2 Use STOP mode with EXTI

- 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

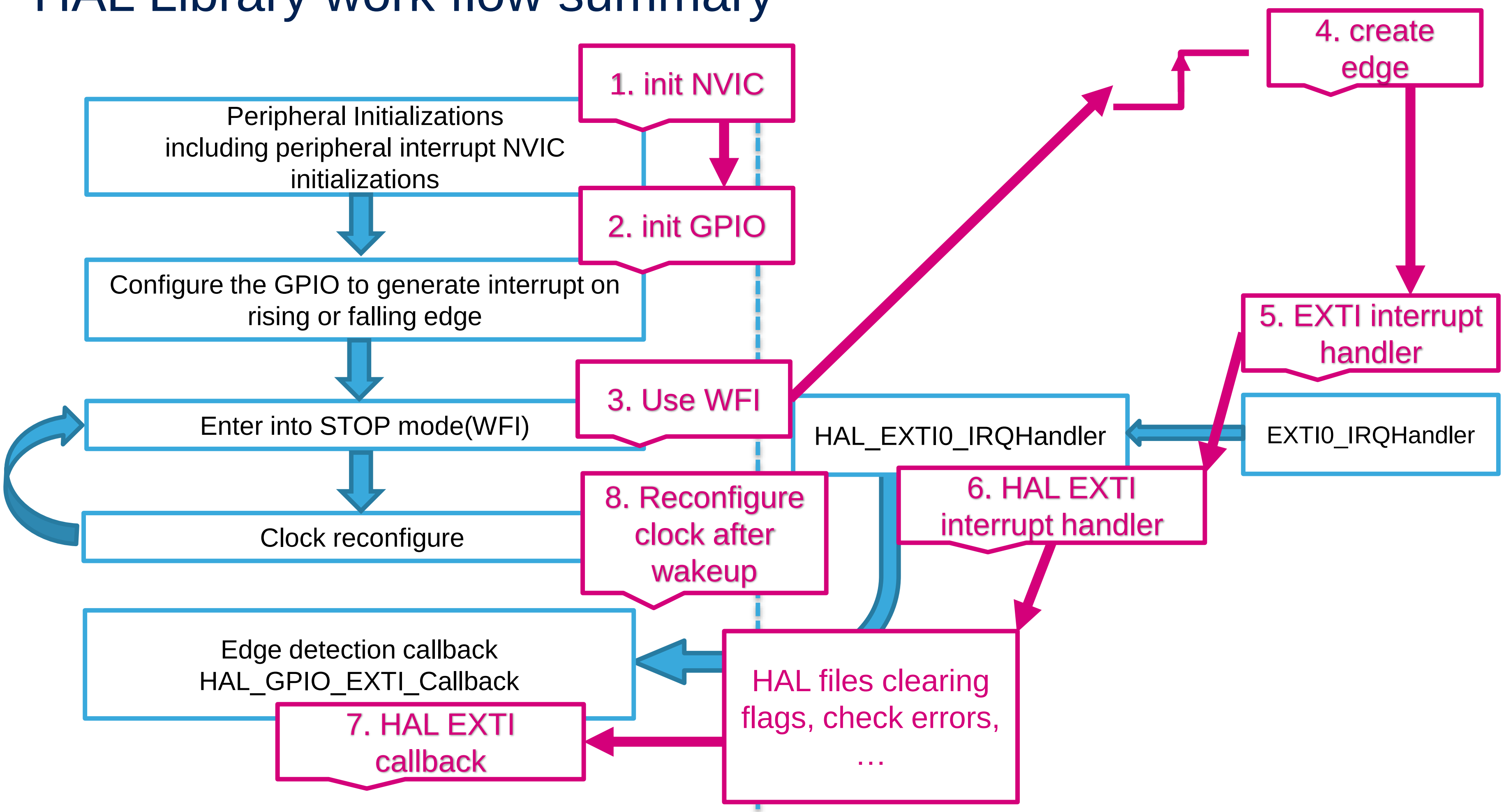
1.2.2 STOP Mode



- 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

1.2.2 Use STOP mode with EXTI

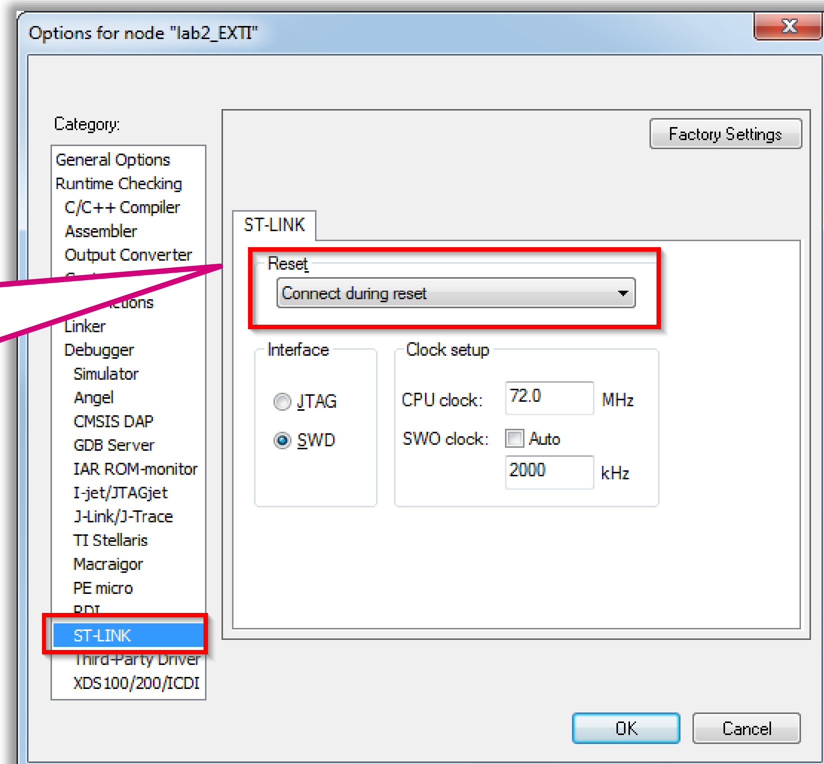
HAL Library work flow summary



1.2.2 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



1.2.2 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

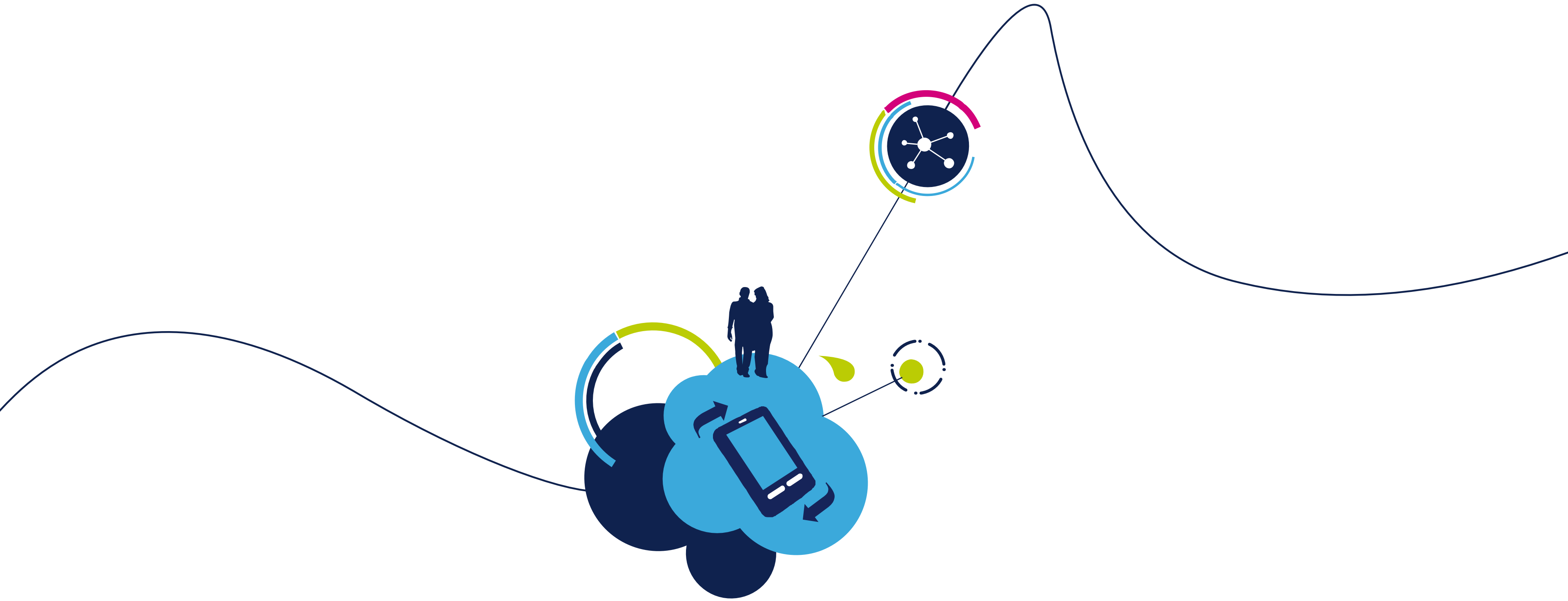
```
/* USER CODE BEGIN 3 */
 /* Infinite loop */
 while (1)
 {
   HAL_Delay(1000);
   HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,PWR_STOPENTRY_WFI);
   SystemClock_Config();
 }
 /* USER CODE END 3 */
```

life.augmented

1.2.2 Use STOP mode with EXTI

- 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 */
```

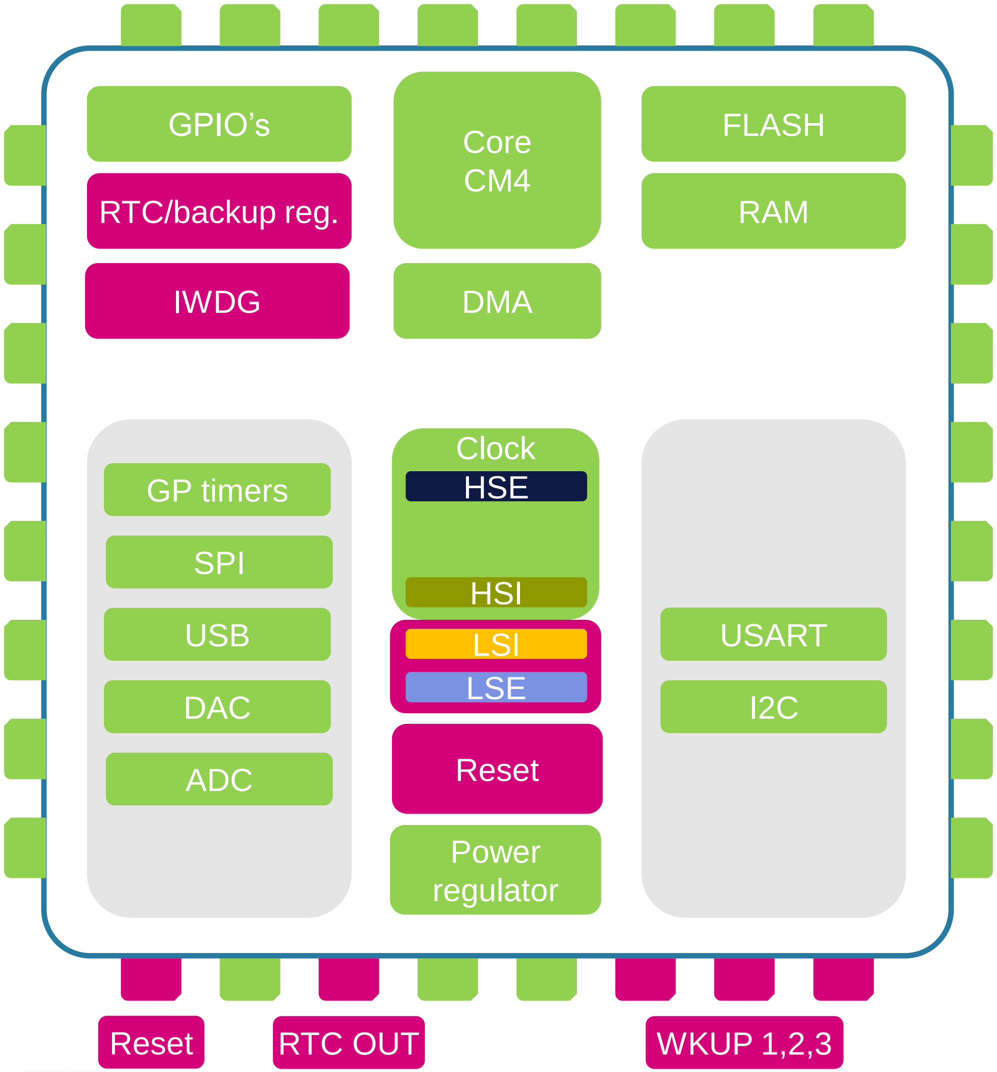


1.2.3 Low Power mode STANDBY lab

1.2.3 Use STANDBY mode

- 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

1.2.3 STANDBY Mode

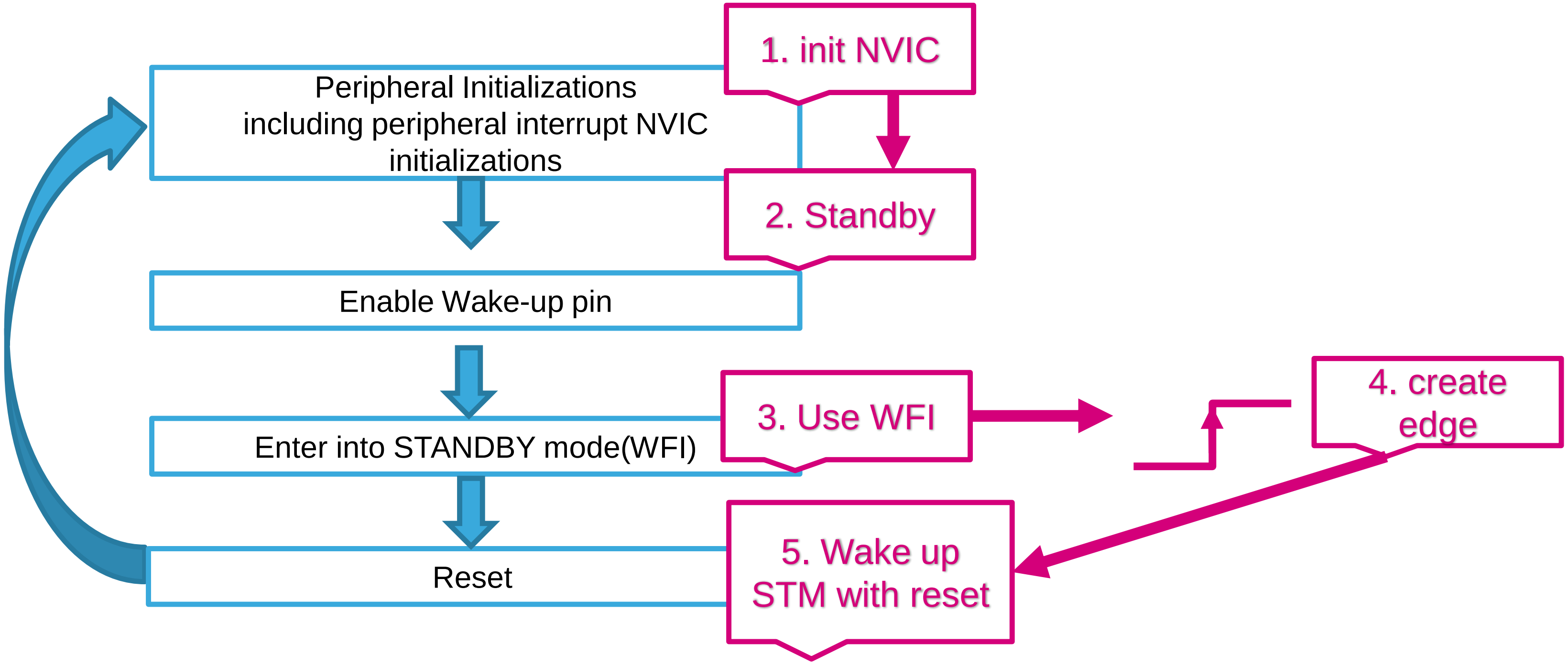


- Core and all peripherals are OFF, **except RTC and IWDG if enabled**
- HSE, HSI clocks are OFF, **LSI LSE can be ON**
- SRAM and registers content is lost, **except RTC, and standby circuitry**
- GPIO's are in high Z, **except Reset, RTC OUT and WKUP 1,2,3**

life.augmented

1.2.3 Use STANDBY mode

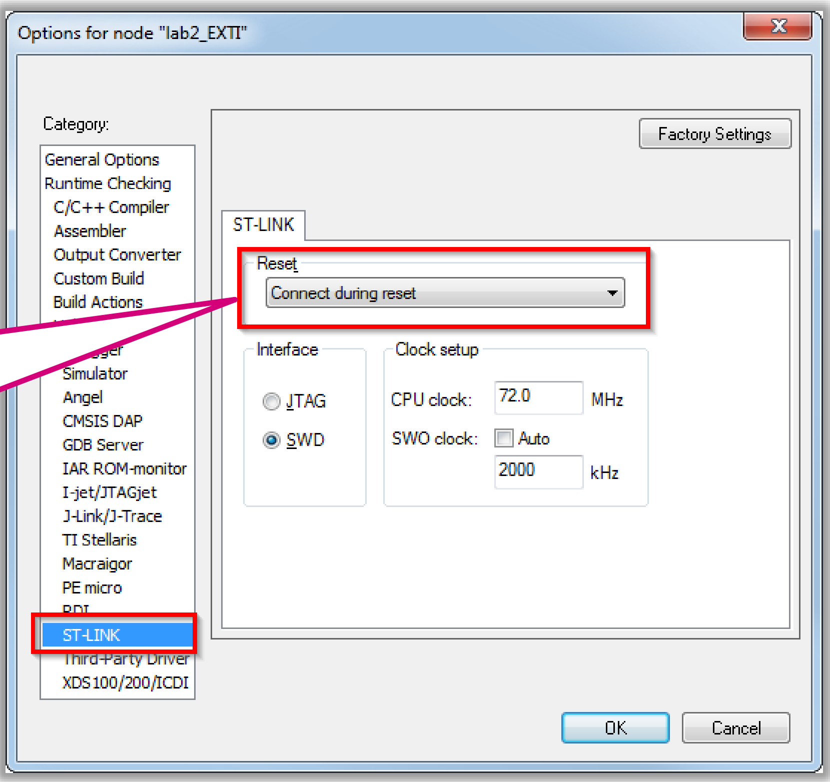
HAL Library work flow summary



1.2.3 Use STANDBY mode

- 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



life.augmented

1.2.3 Use STANDBY mode

- 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 */
```

life.augmented

1.2.3 Use STANDBY mode

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

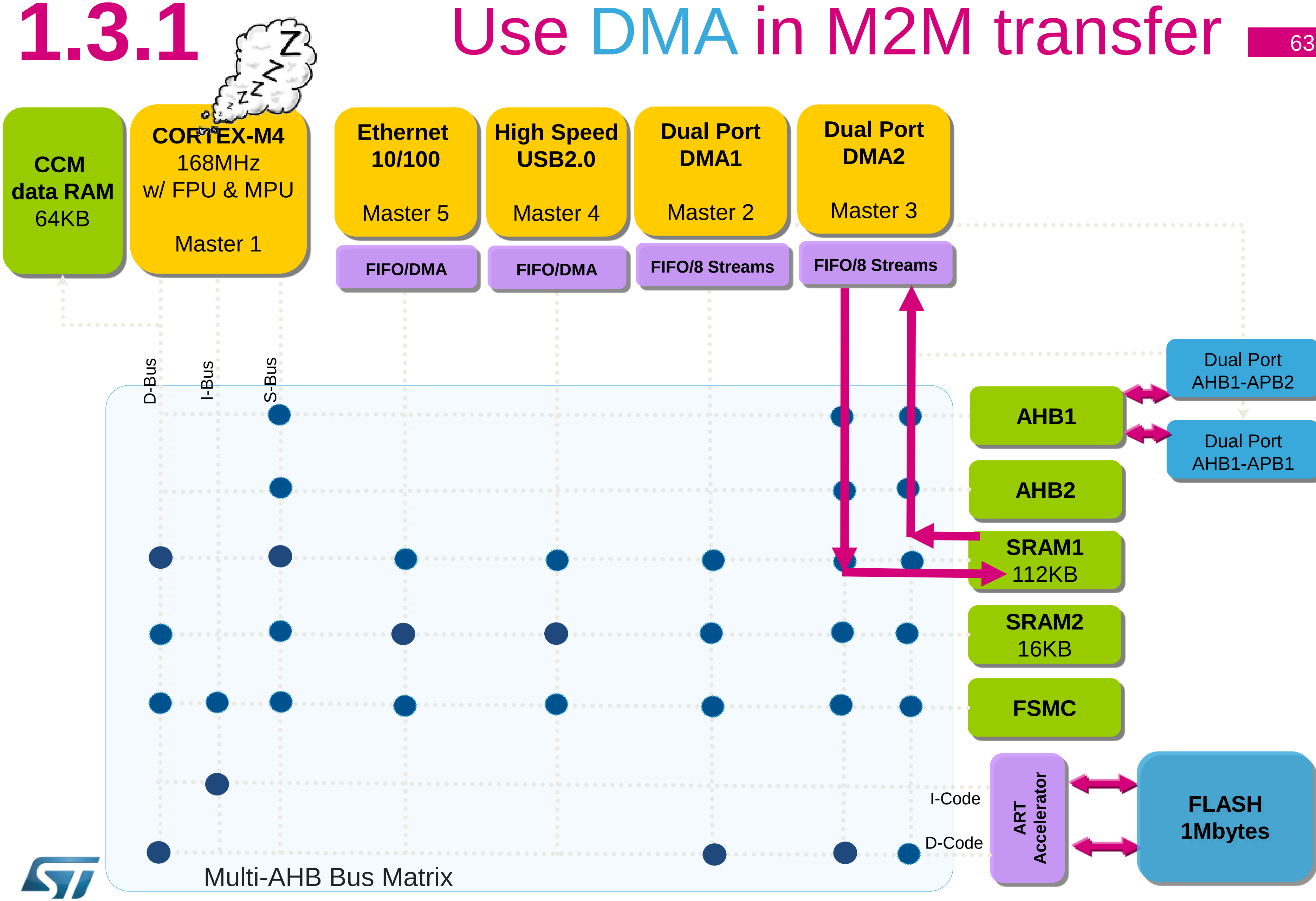
```
 /* USER CODE BEGIN 2 */

__HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
  /* USER CODE END 2 */
```

1.3.1 Data transfer over DMA lab

1.3.1 Use DMA in M2M transfer

- 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

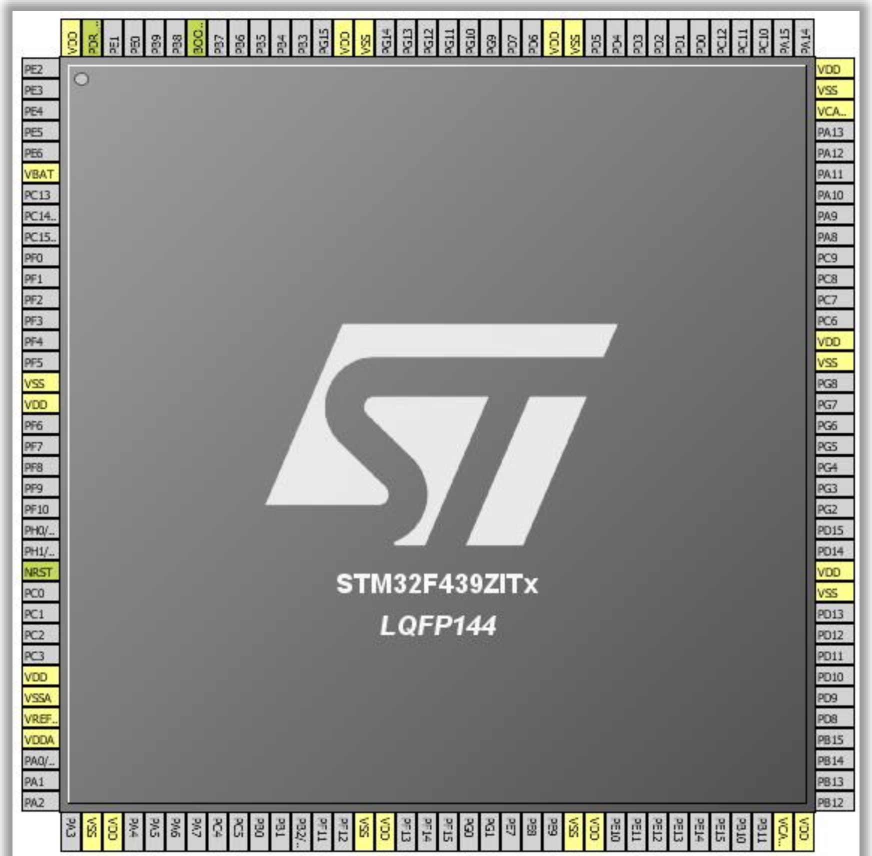
1.3.1 Use DMA in M2M transfer



life.augmented

1.3.1 Use DMA in M2M transfer

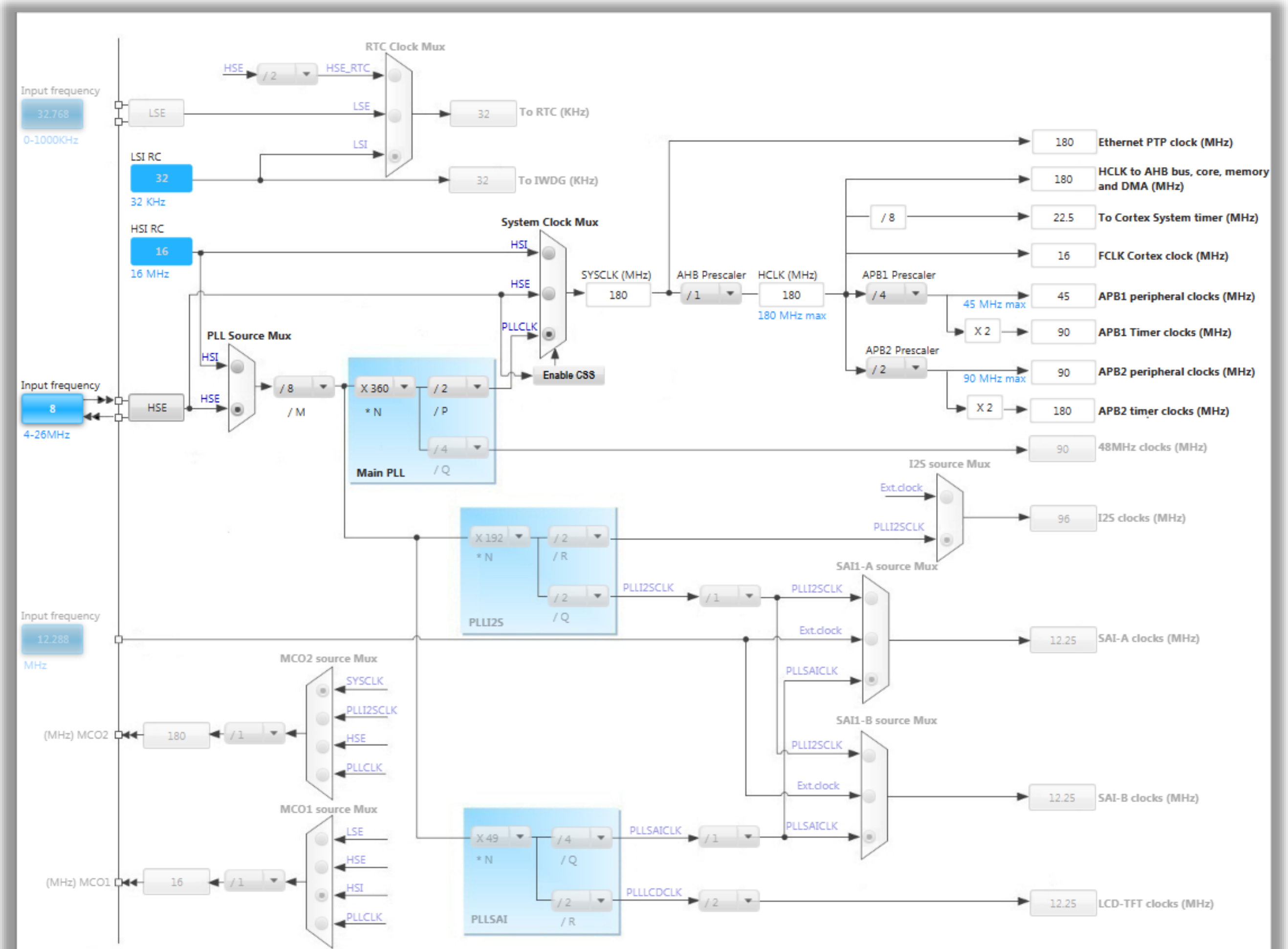
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- For DMA we don't need to configure any pins



life.augmented

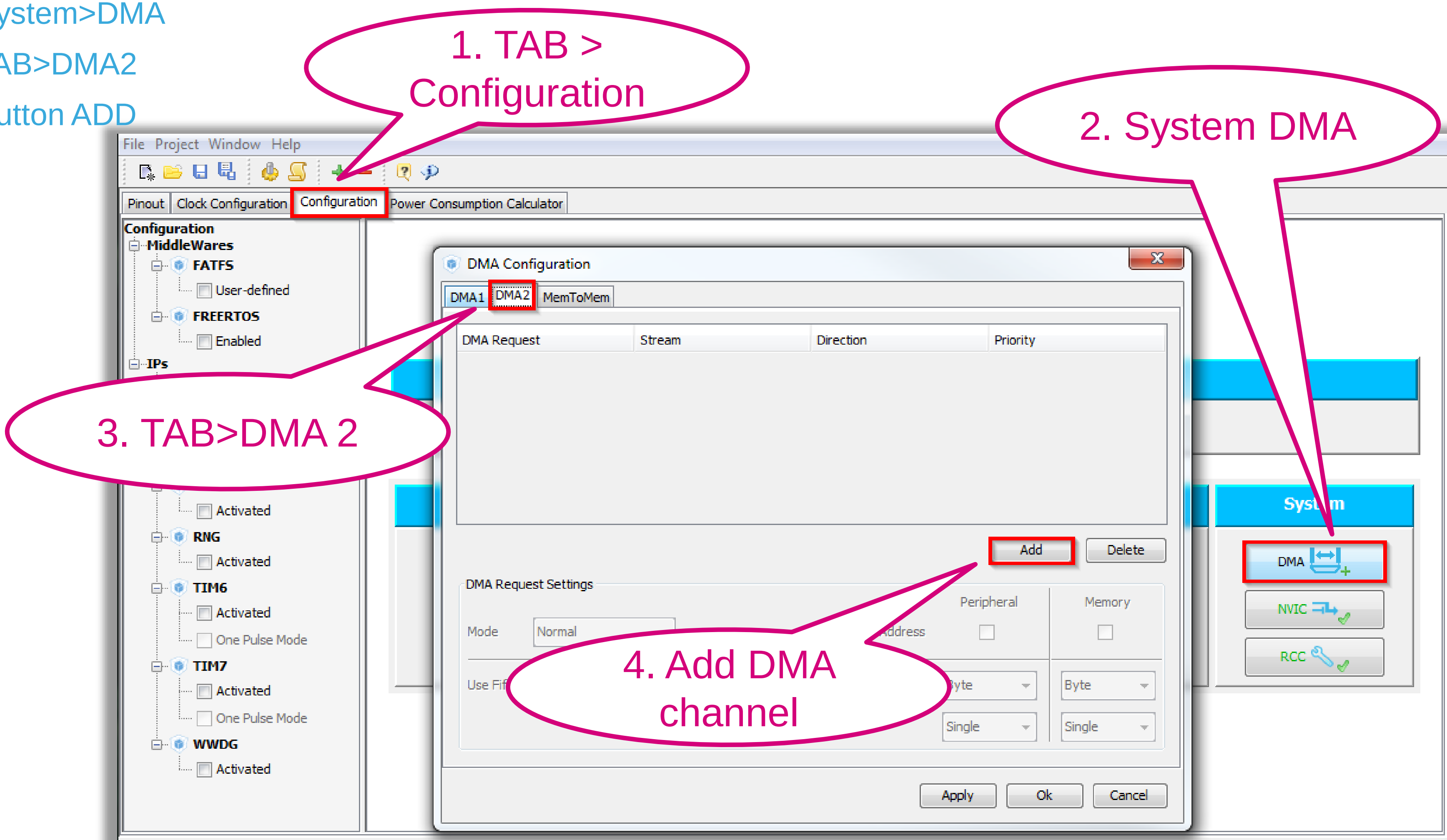
1.3.1 Use DMA in M2M transfer

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



1.3.1 Use DMA in M2M transfer

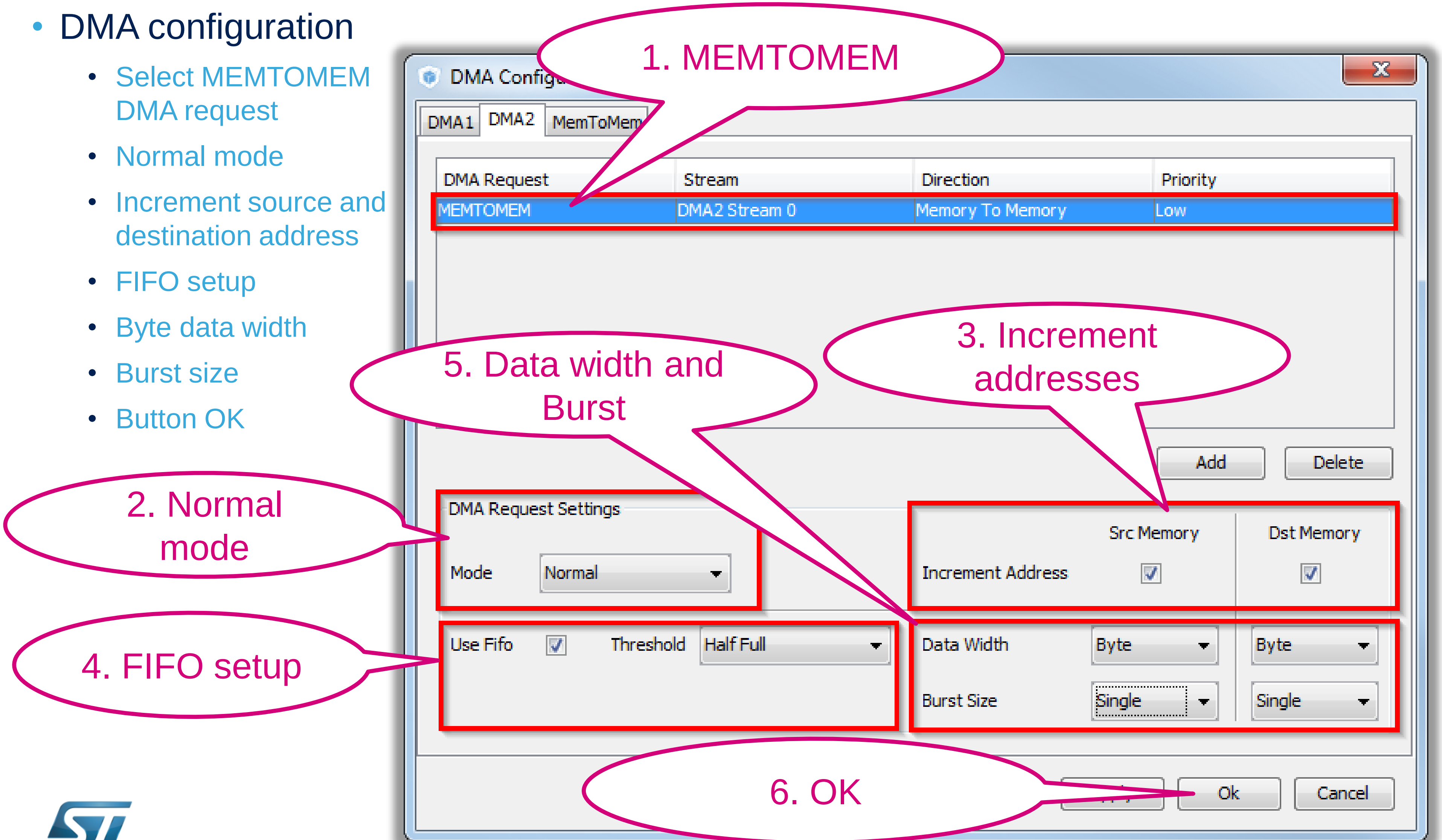
- DMA configuration
  - TAB>Configuration
  - System>DMA
  - TAB>DMA2
  - Button ADD



life.augmented

1.3.1 Use DMA in M2M transfer

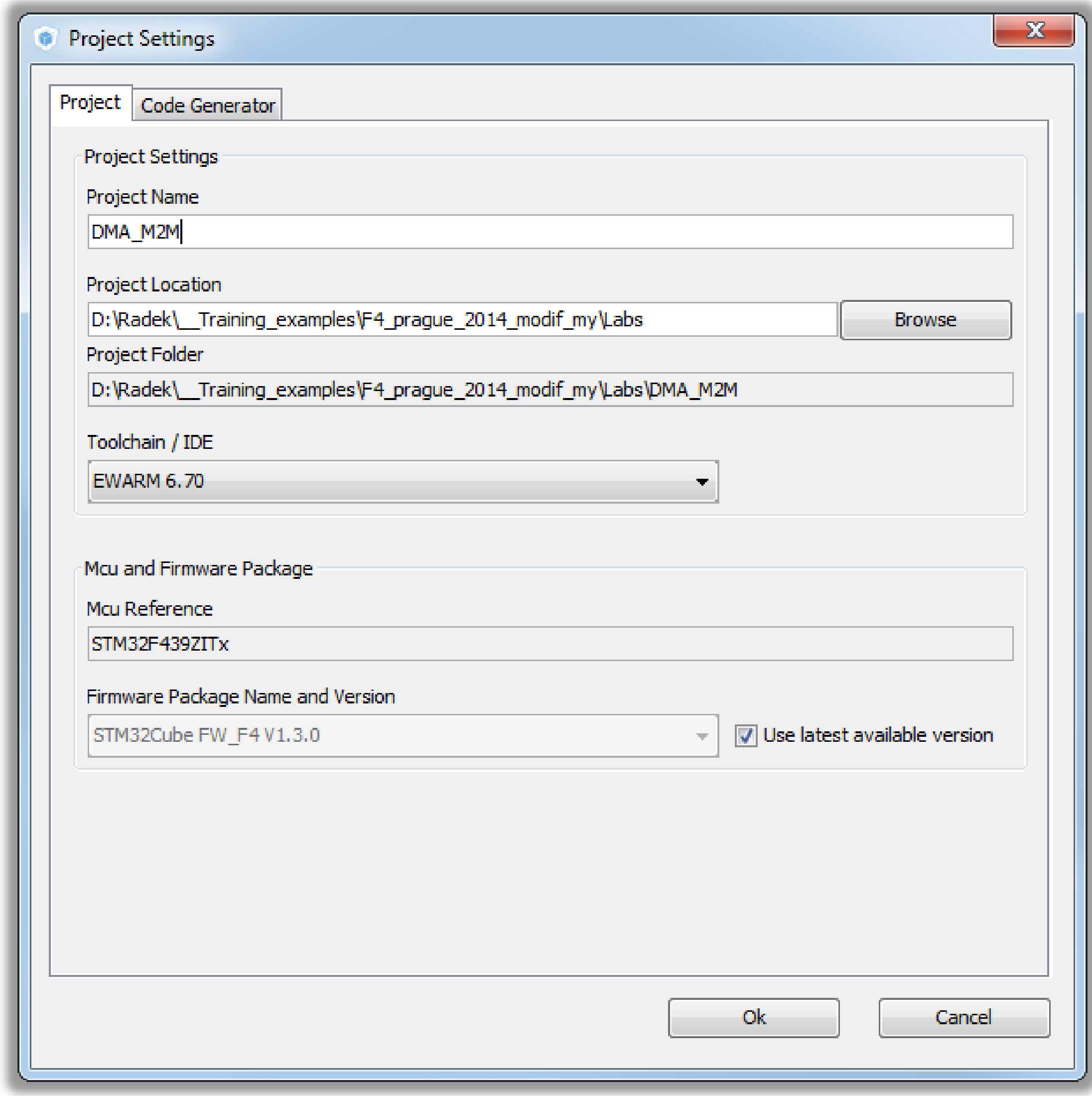
- DMA configuration
  - Select MEMTOMEM DMA request
  - Normal mode
  - Increment source and destination address
  - FIFO setup
  - Byte data width
  - Burst size
  - Button OK



life.augmented

1.3.1 Use DMA in M2M transfer

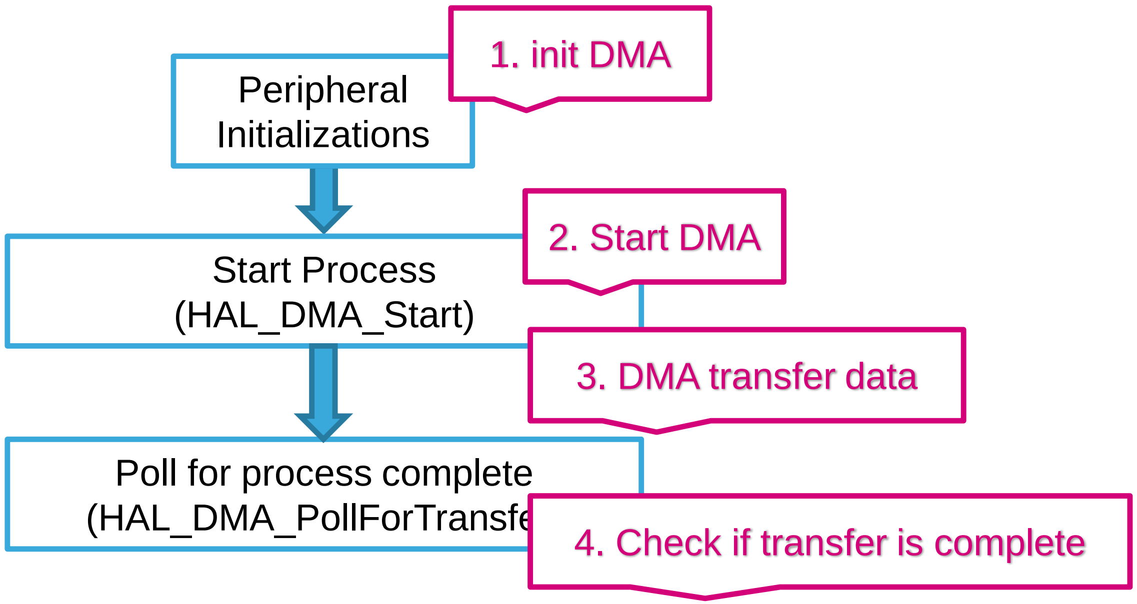
- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



life.augmented

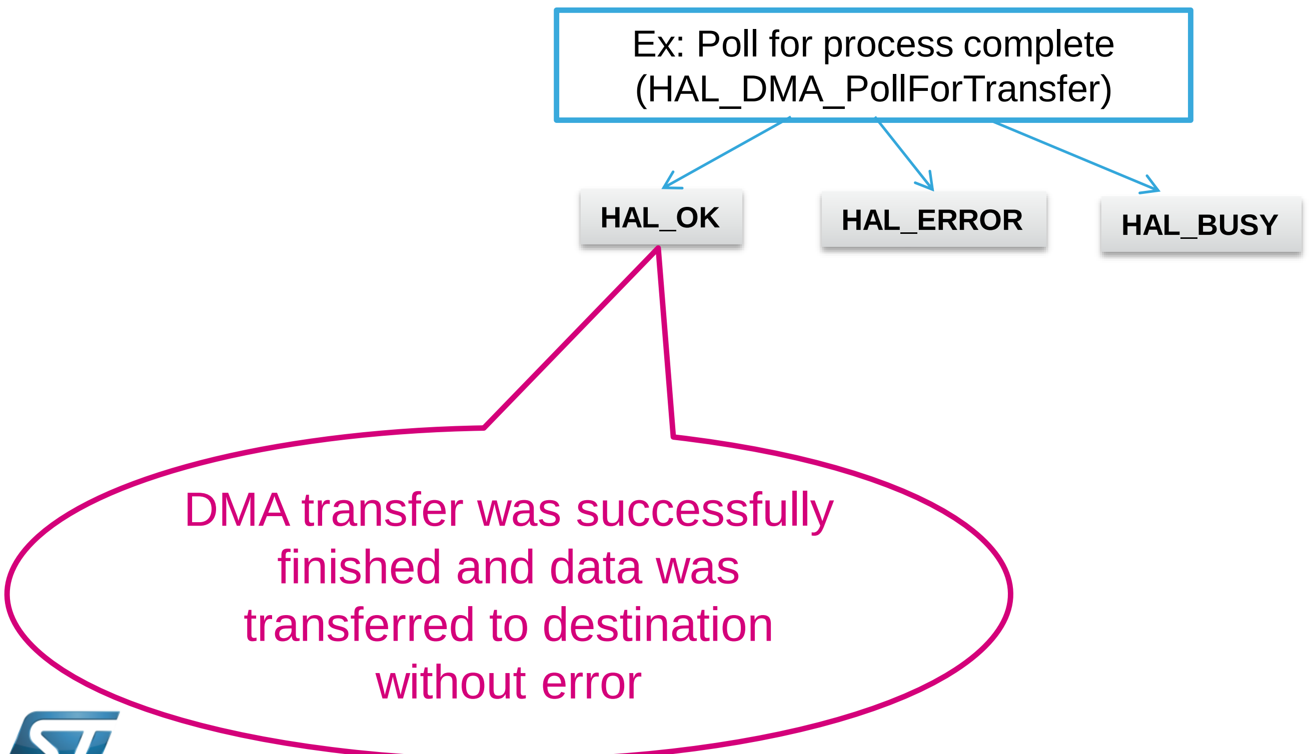
1.3.1 Use DMA in M2M transfer

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



1.3.1 Use DMA in M2M transfer

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



1.3.1 Use DMA in M2M transfer

- Return values
  - Most of CubeMX functions have return values, which indicate, if operation was successful, timeout occurs of 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

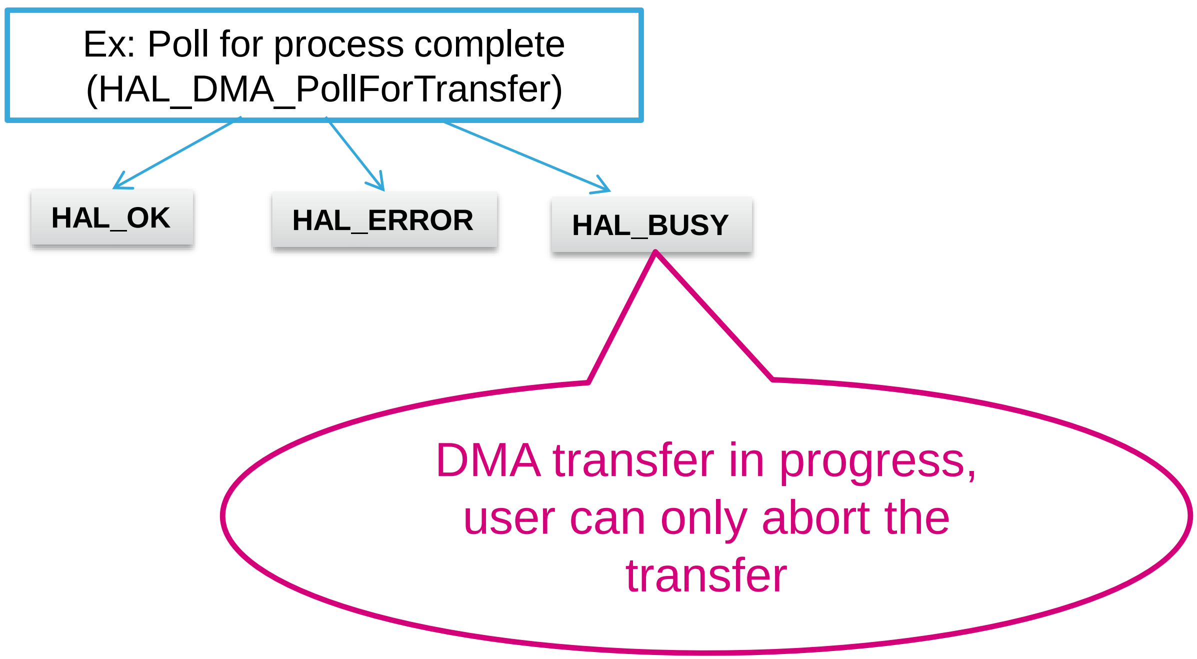
HAL_ERROR

HAL_BUSY

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

1.3.1 Use DMA in M2M transfer

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



1.3.1 Use DMA in M2M transfer

- 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)

1.3.1 Use DMA in M2M transfer

- 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 */
```

1.3.1 Use DMA in M2M transfer

- 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 */
```

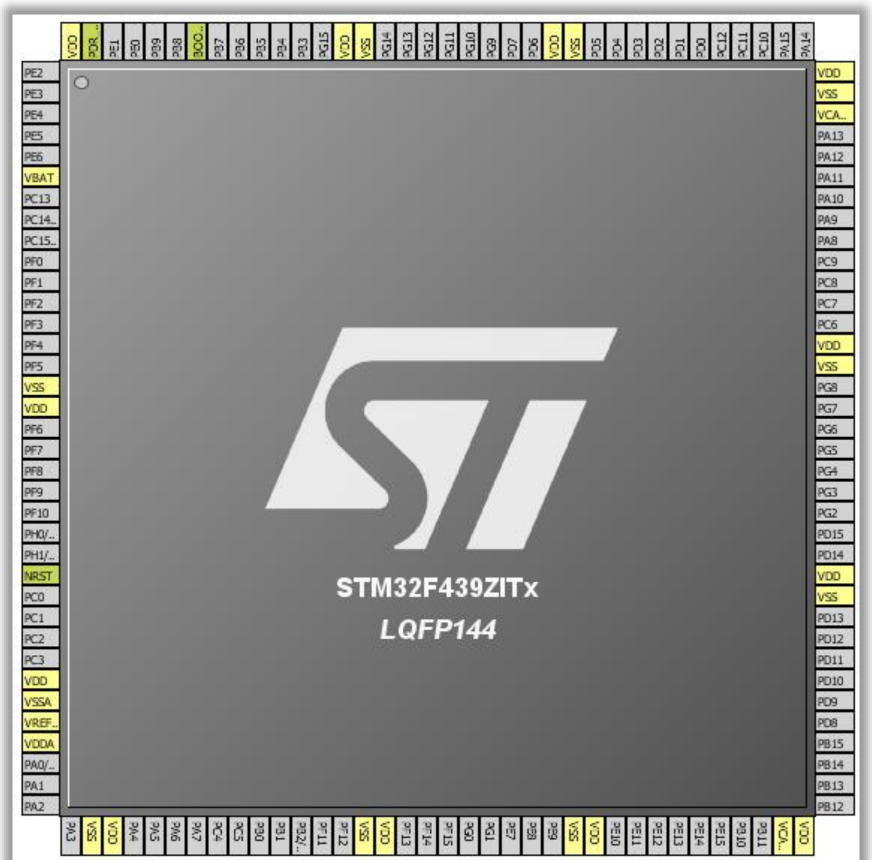
1.3.2 Data transfer over DMA with interrupt lab

1.3.2 Use DMA M2M with interrupt

- 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

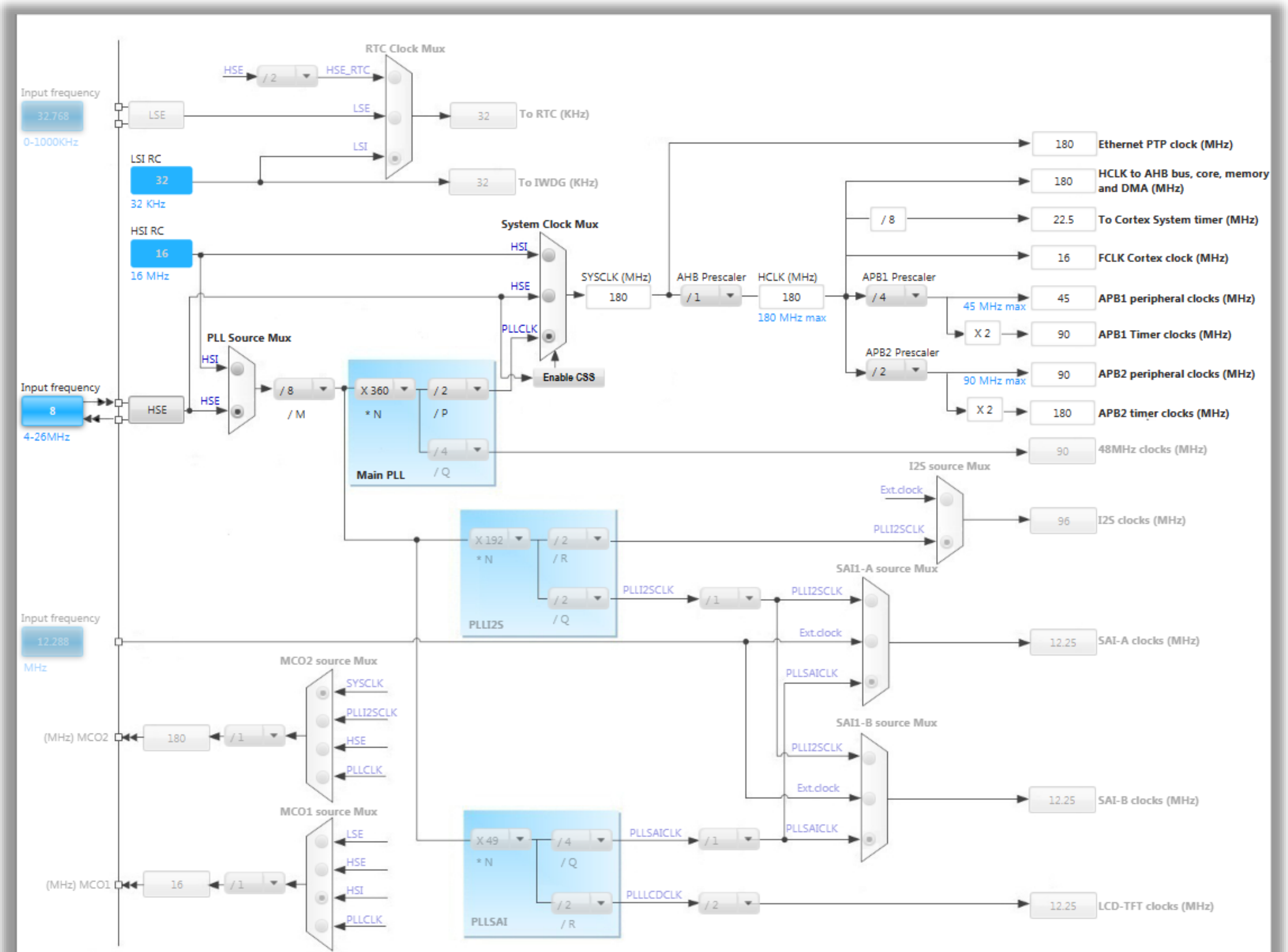
1.3.2 Use DMA M2M with interrupt

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



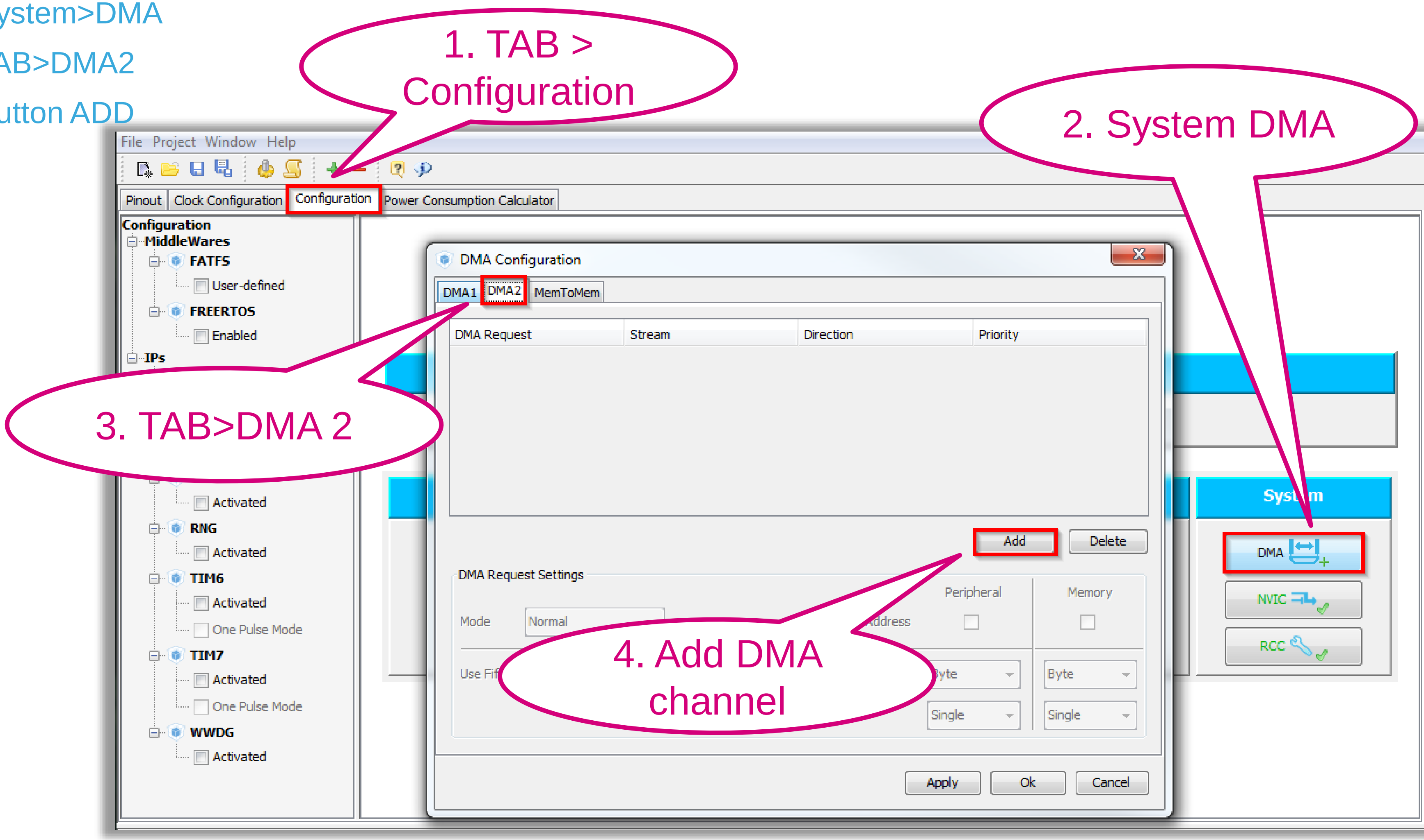
1.3.2 Use DMA M2M with interrupt

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



1.3.2 Use DMA M2M with interrupt

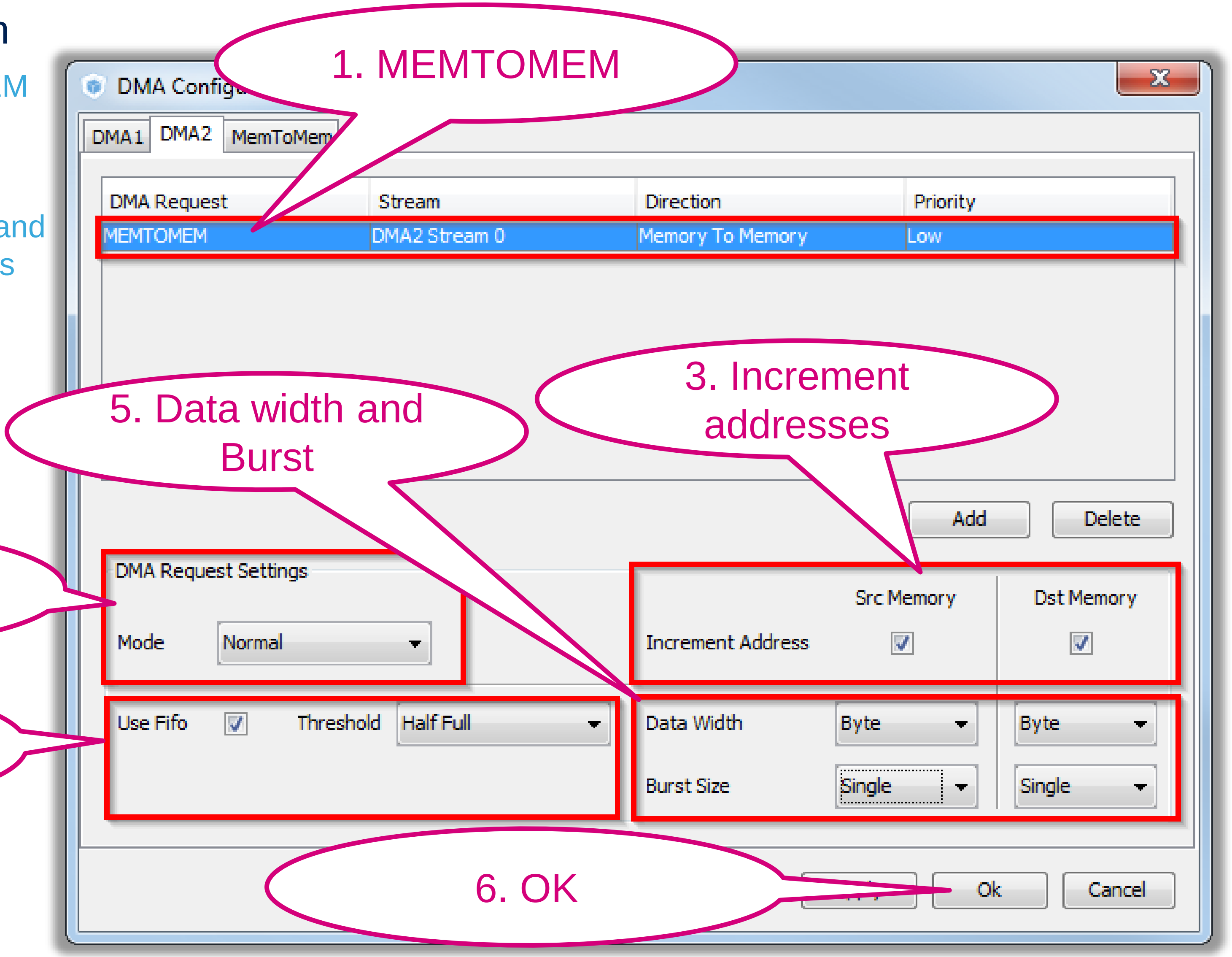
- DMA configuration
  - TAB>Configuration
  - System>DMA
  - TAB>DMA2
  - Button ADD



life.augmented

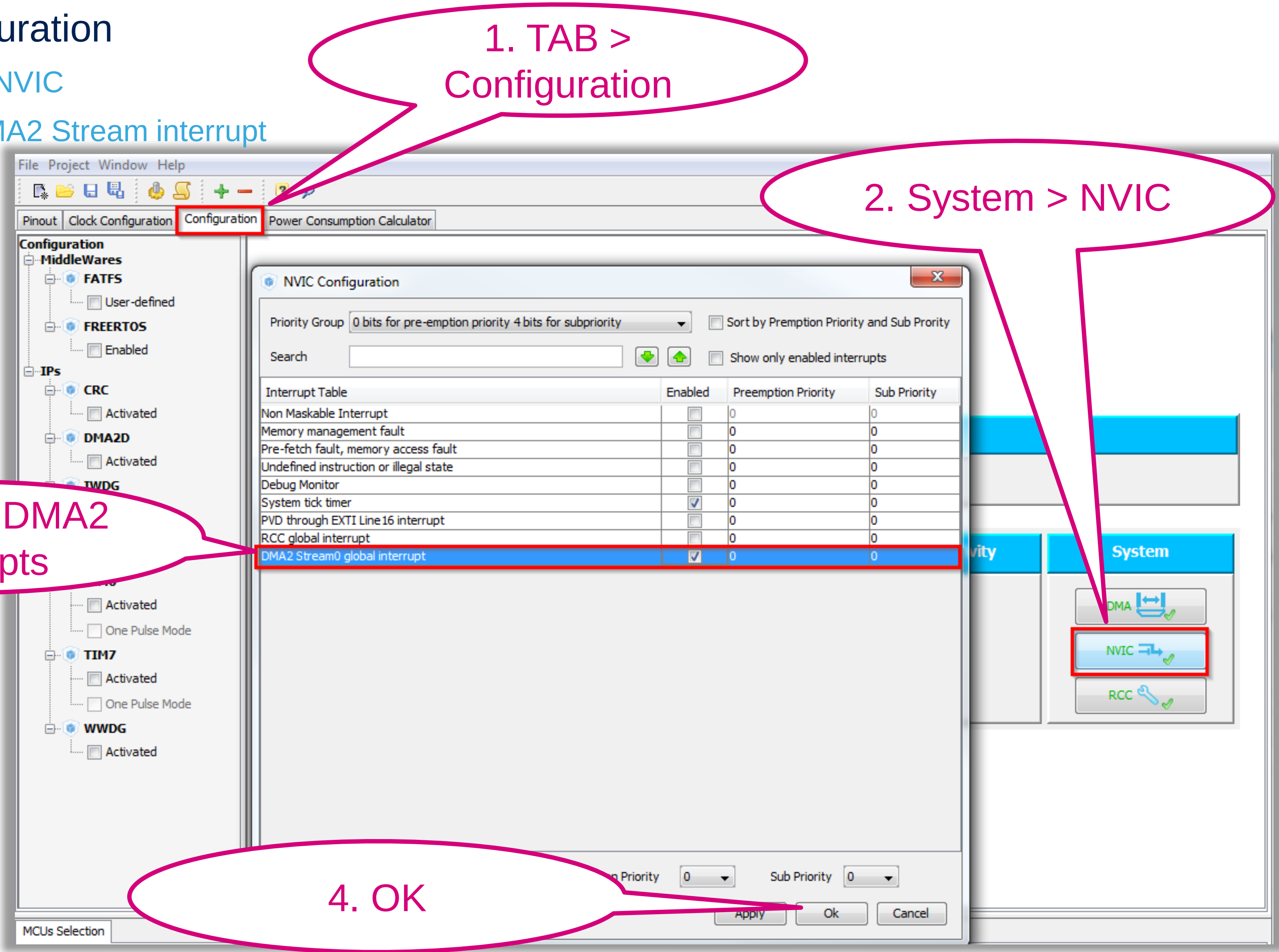
1.3.2 Use DMA M2M with interrupt

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



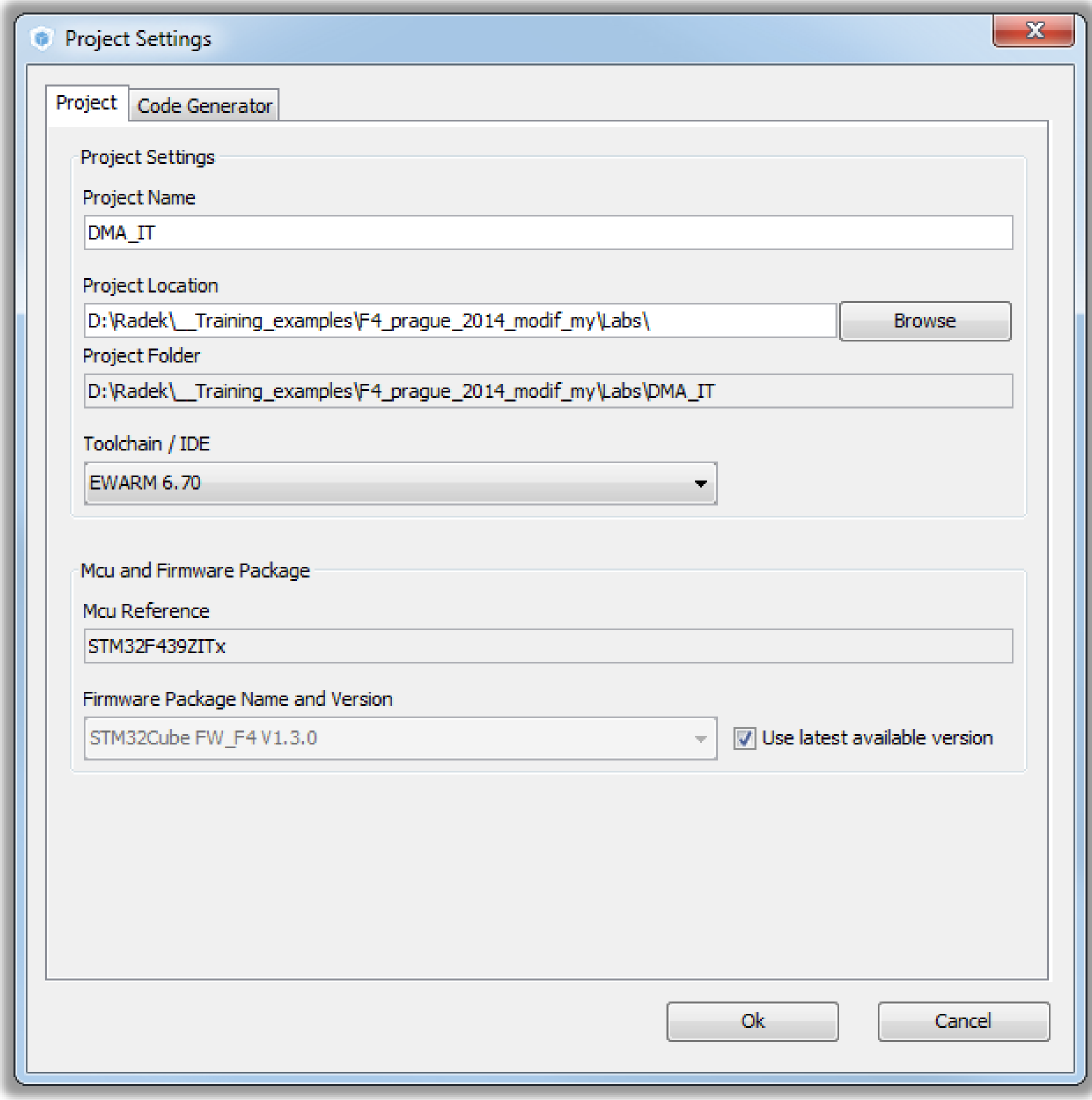
1.3.2 Use DMA M2M with interrupt

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



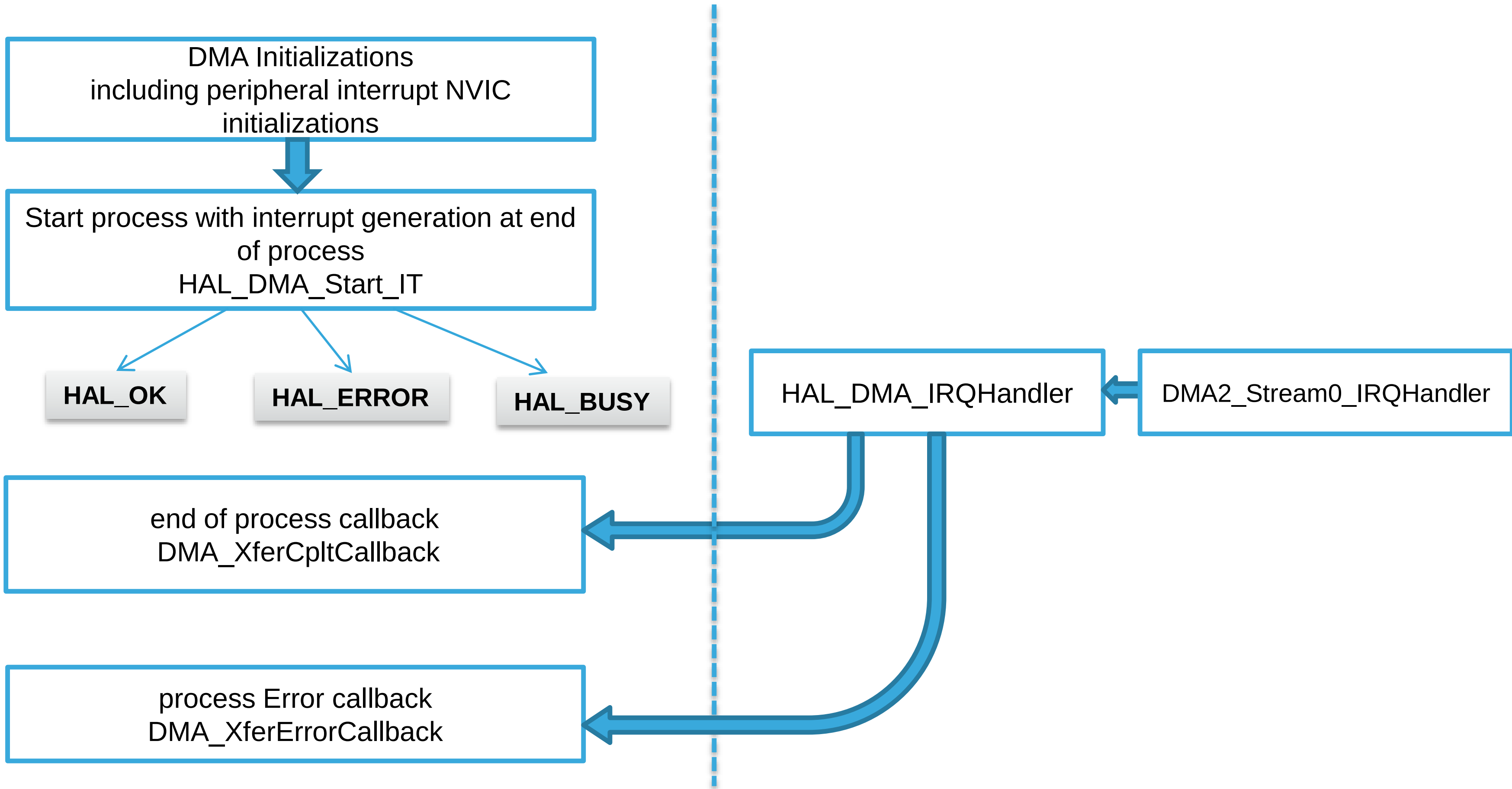
1.3.2 Use DMA M2M with interrupt

- 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



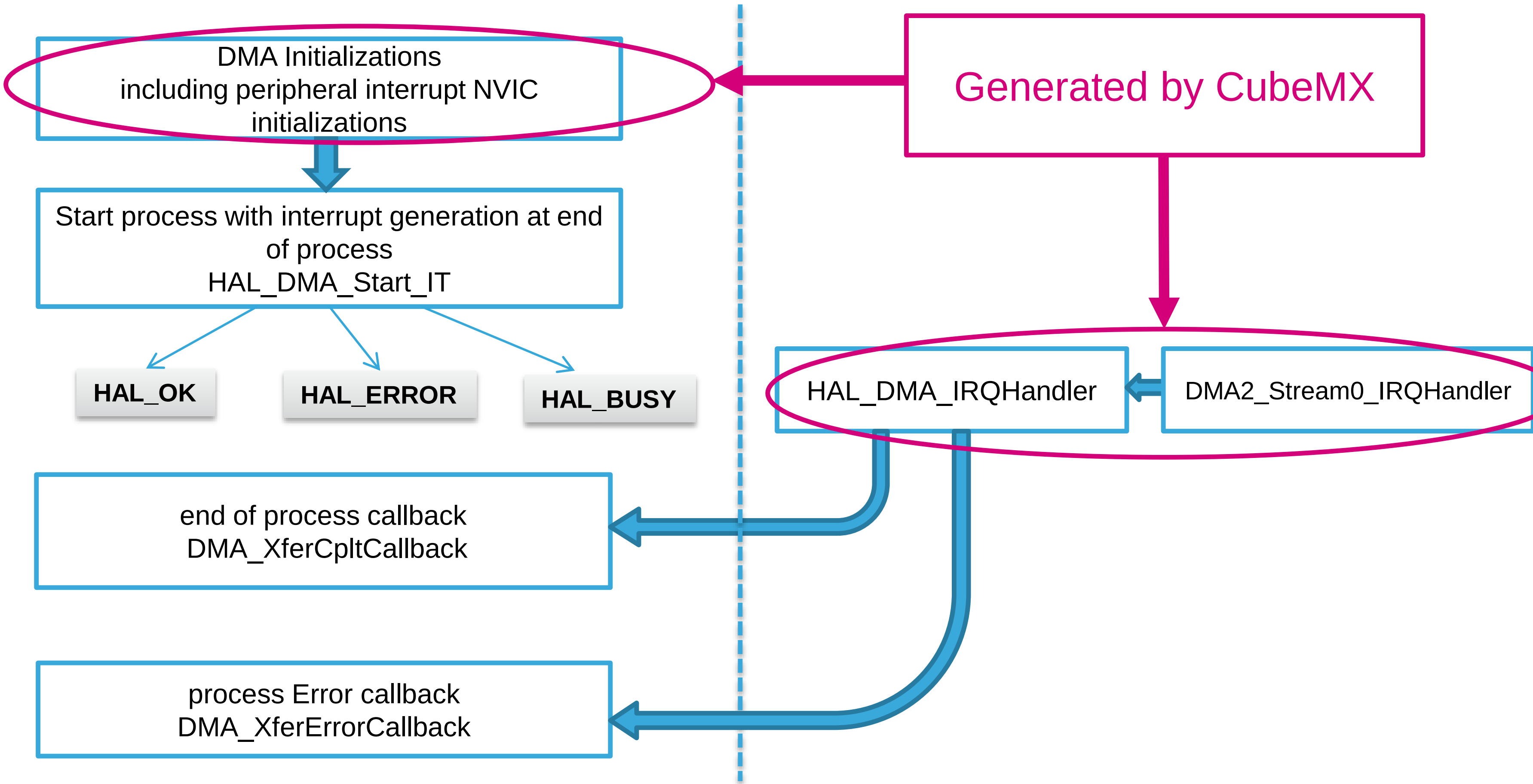
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



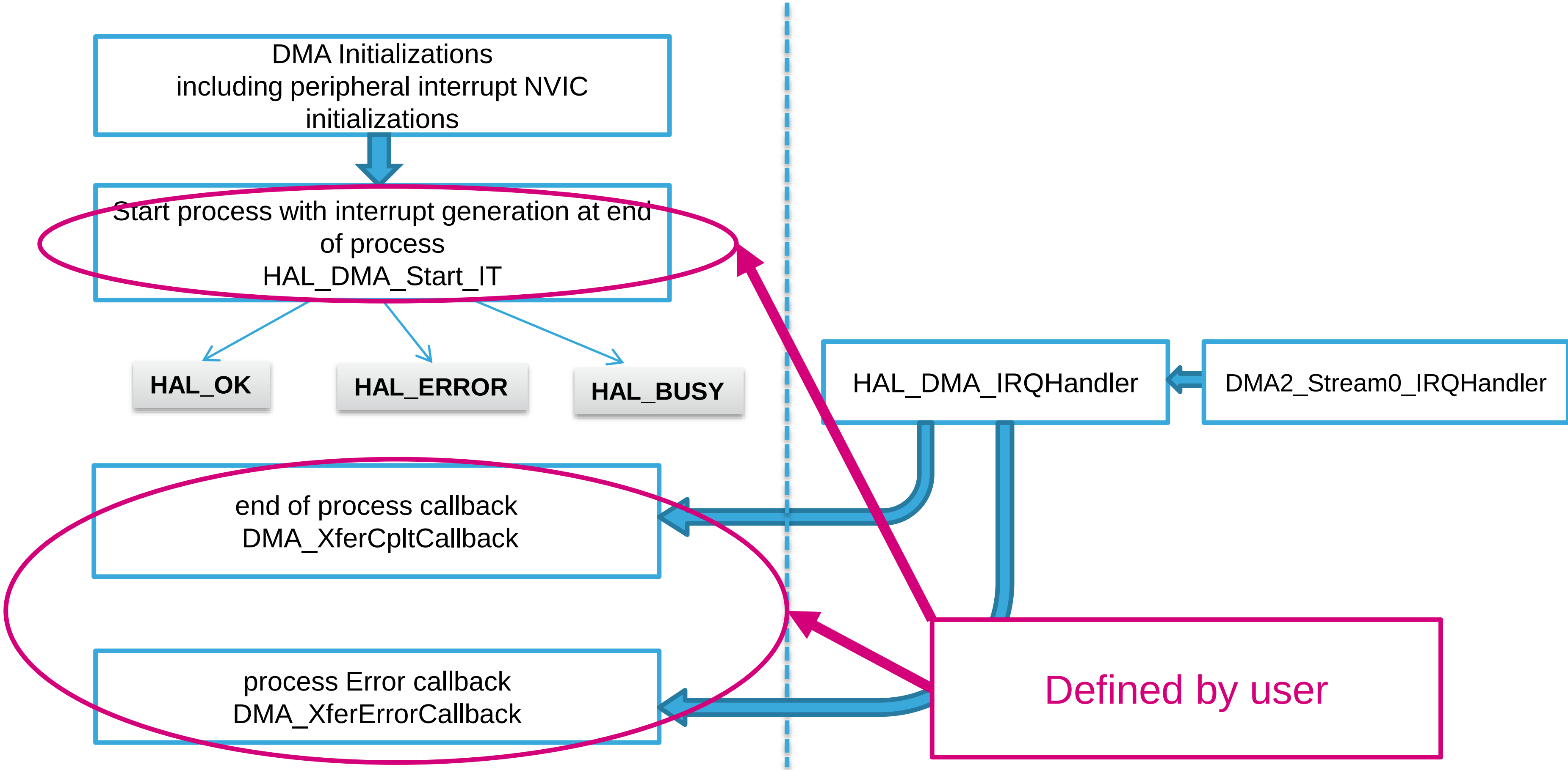
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



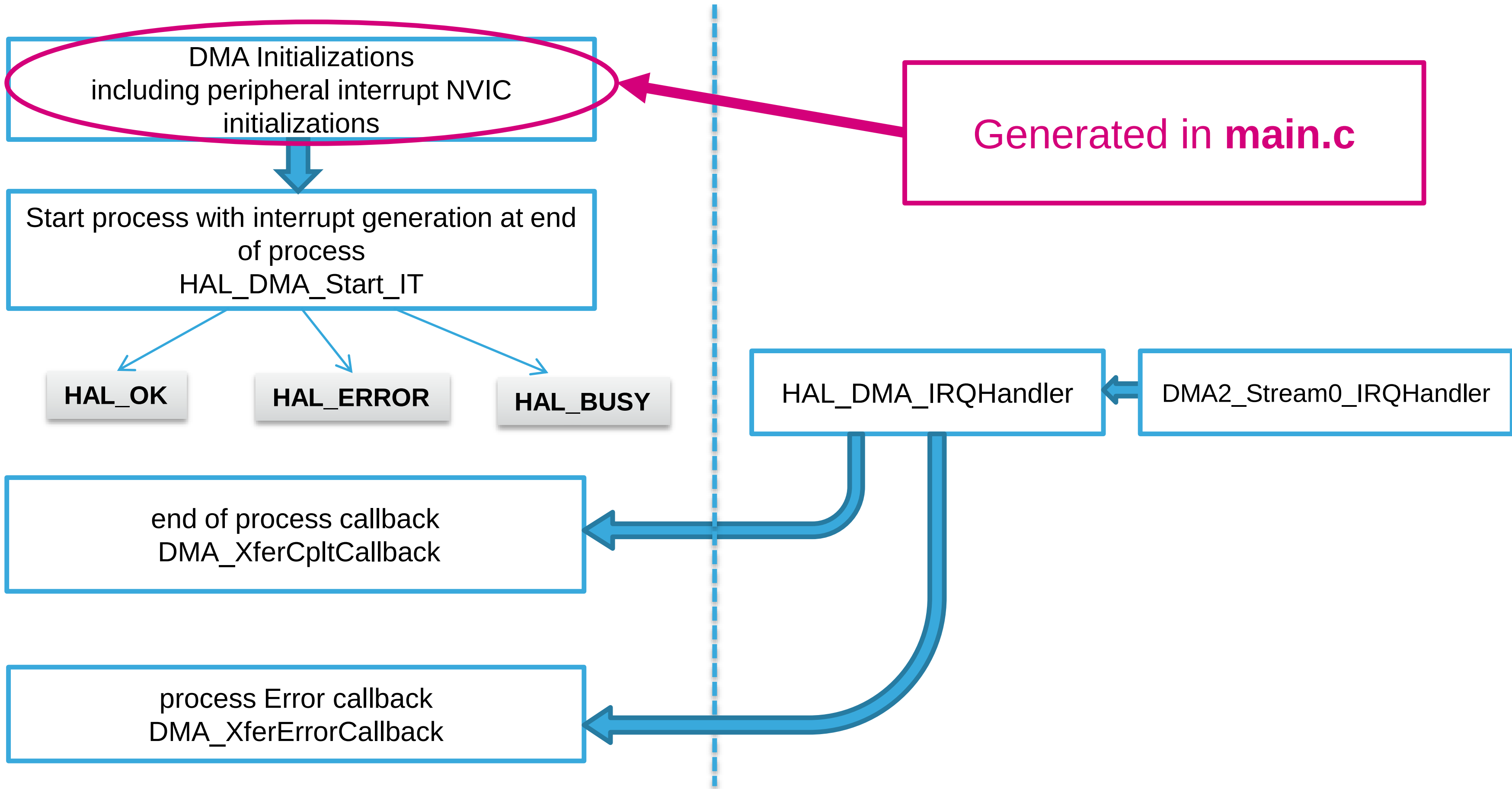
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



1.3.2 Use DMA M2M with interrupt

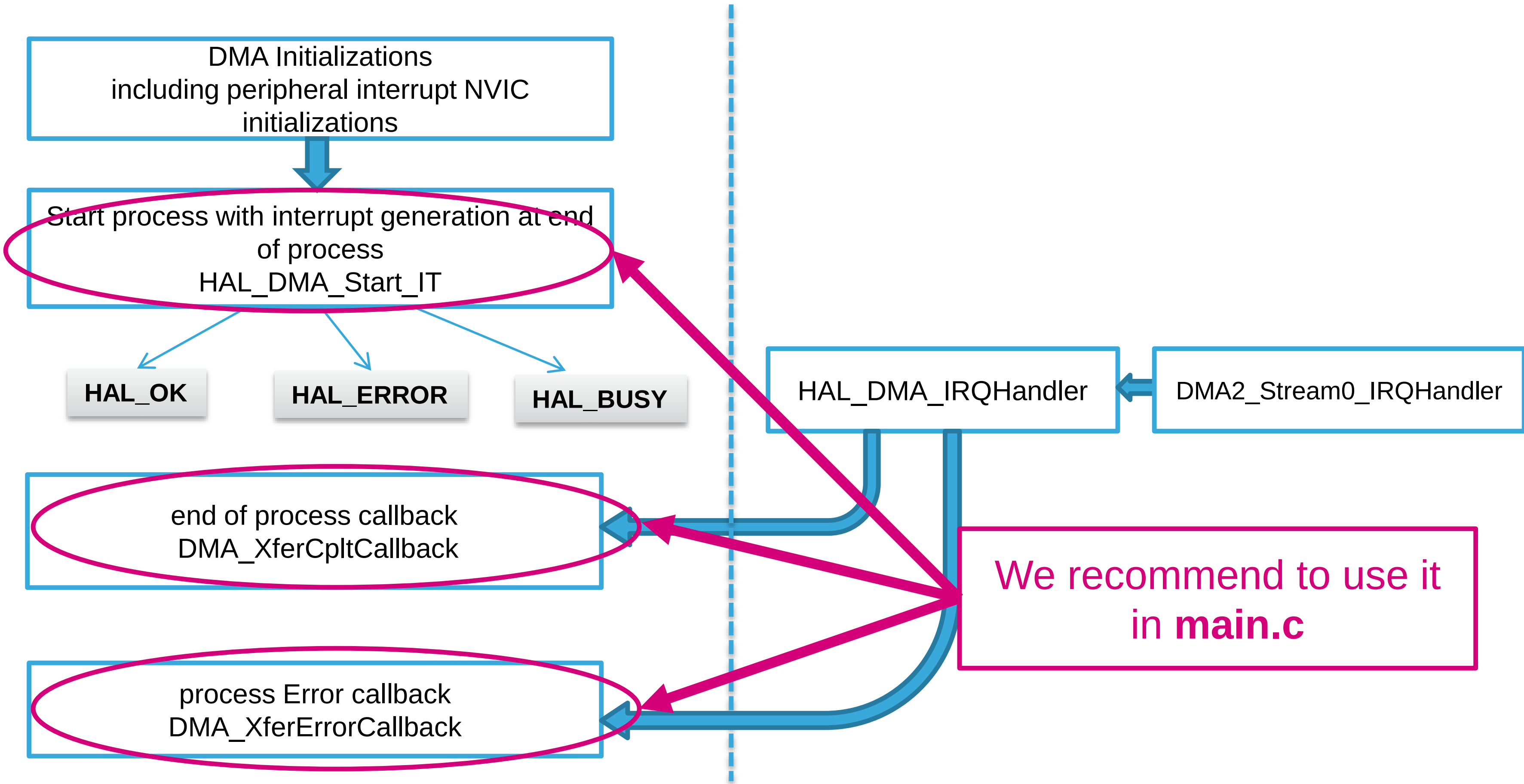
HAL Library DMA with IT flow



life.augmented

1.3.2 Use DMA M2M with interrupt

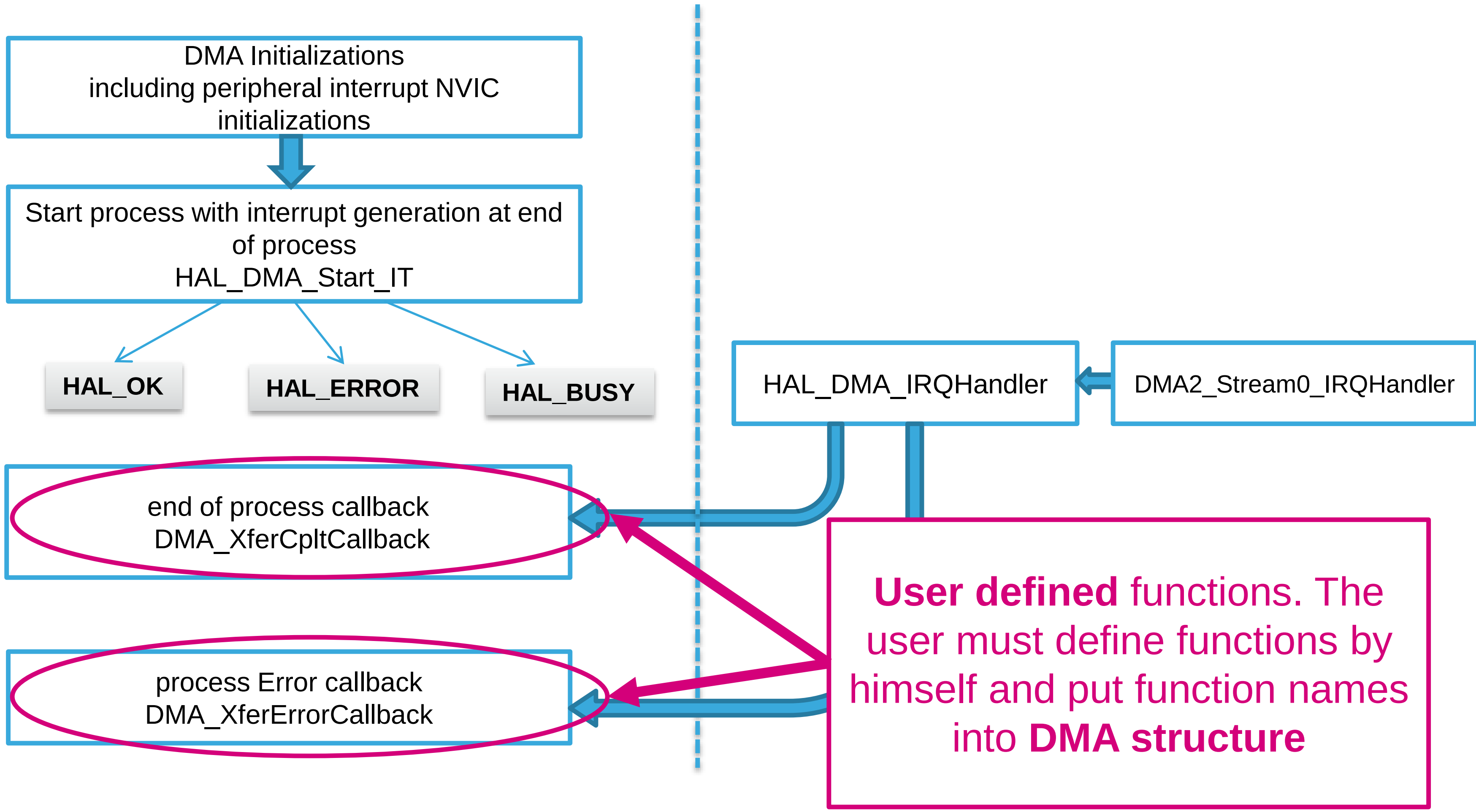
HAL Library DMA with IT flow



life.augmented

1.3.2 Use DMA M2M with interrupt

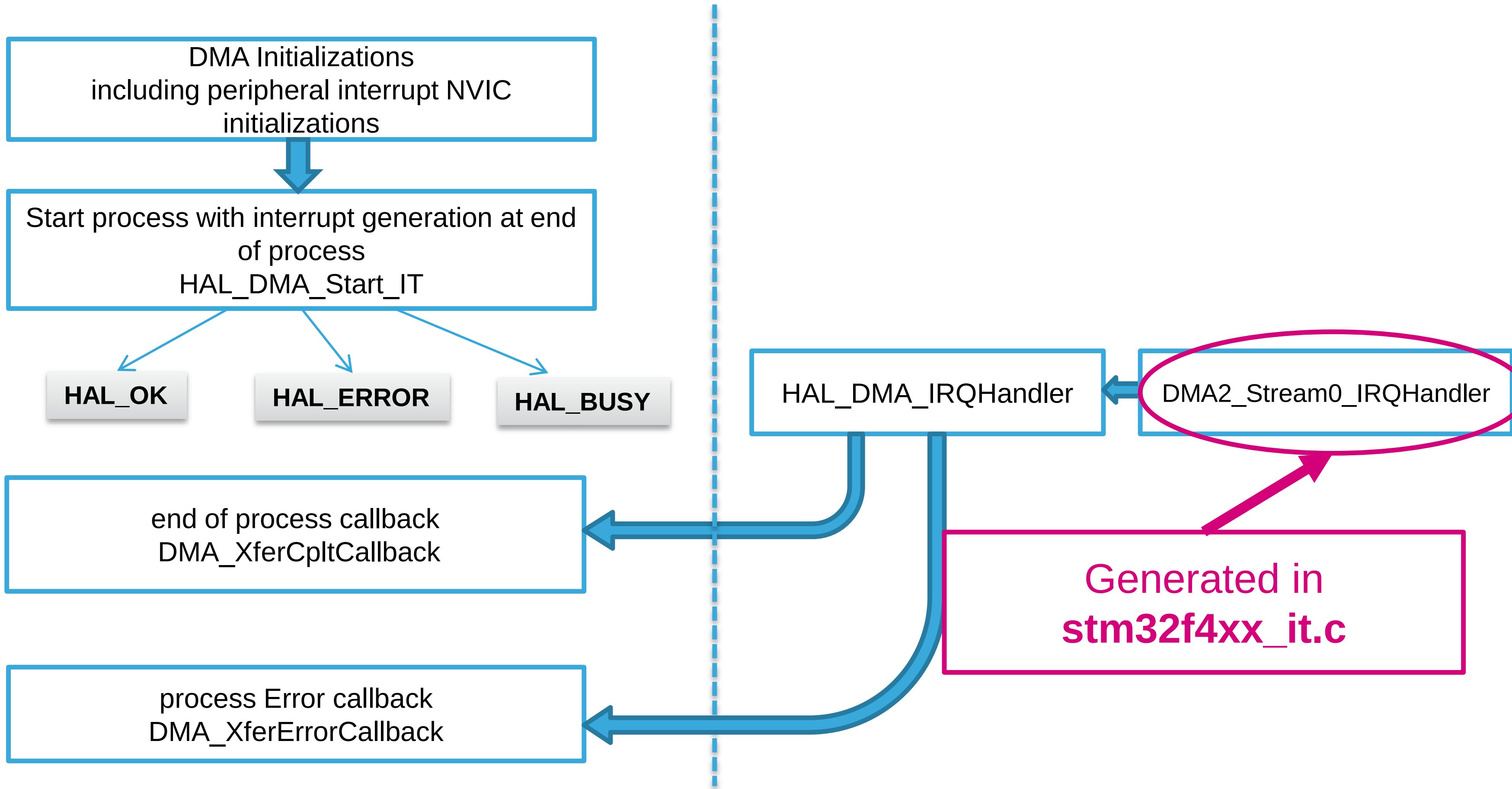
HAL Library DMA with IT flow



life.augmented

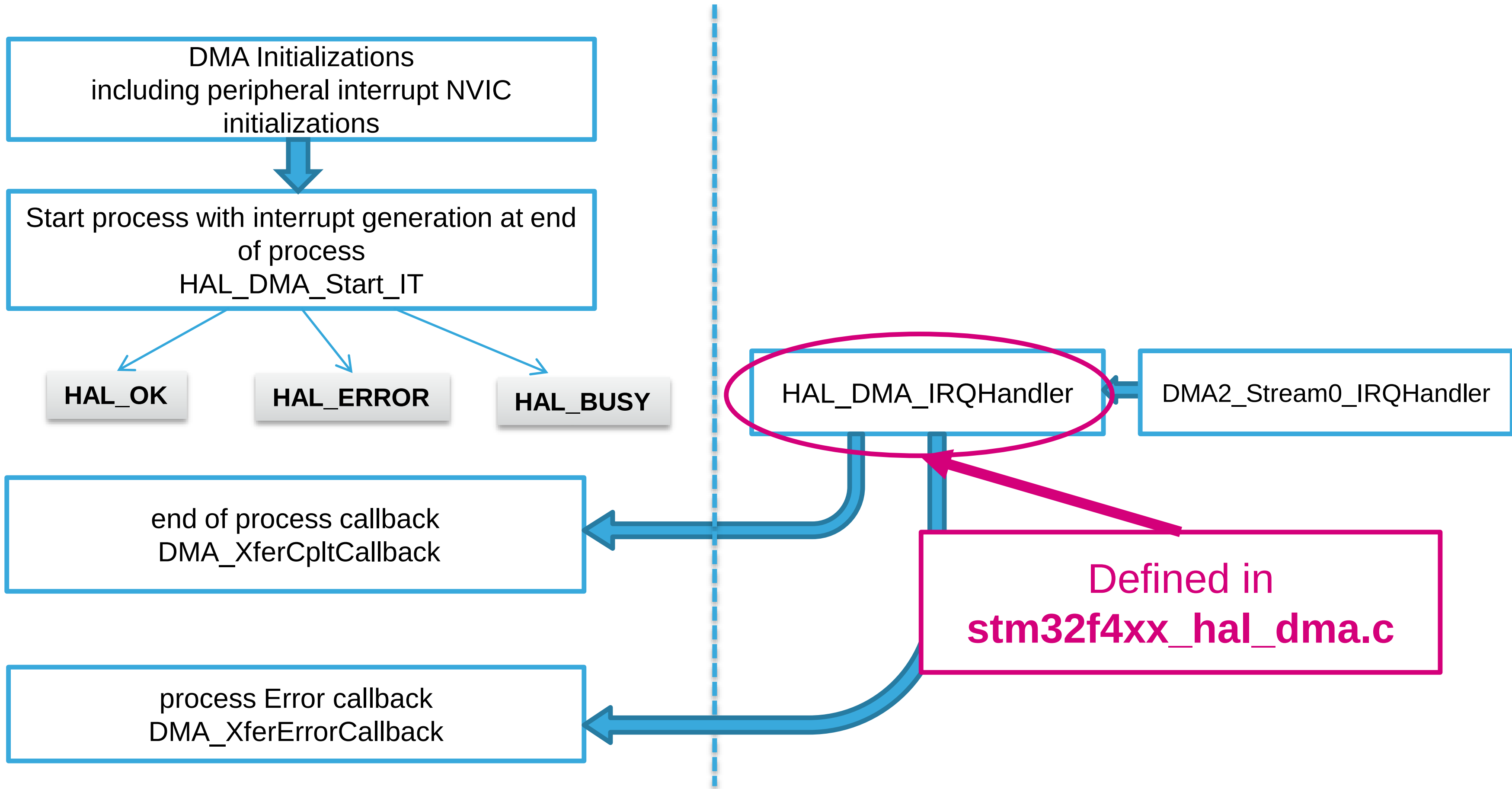
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



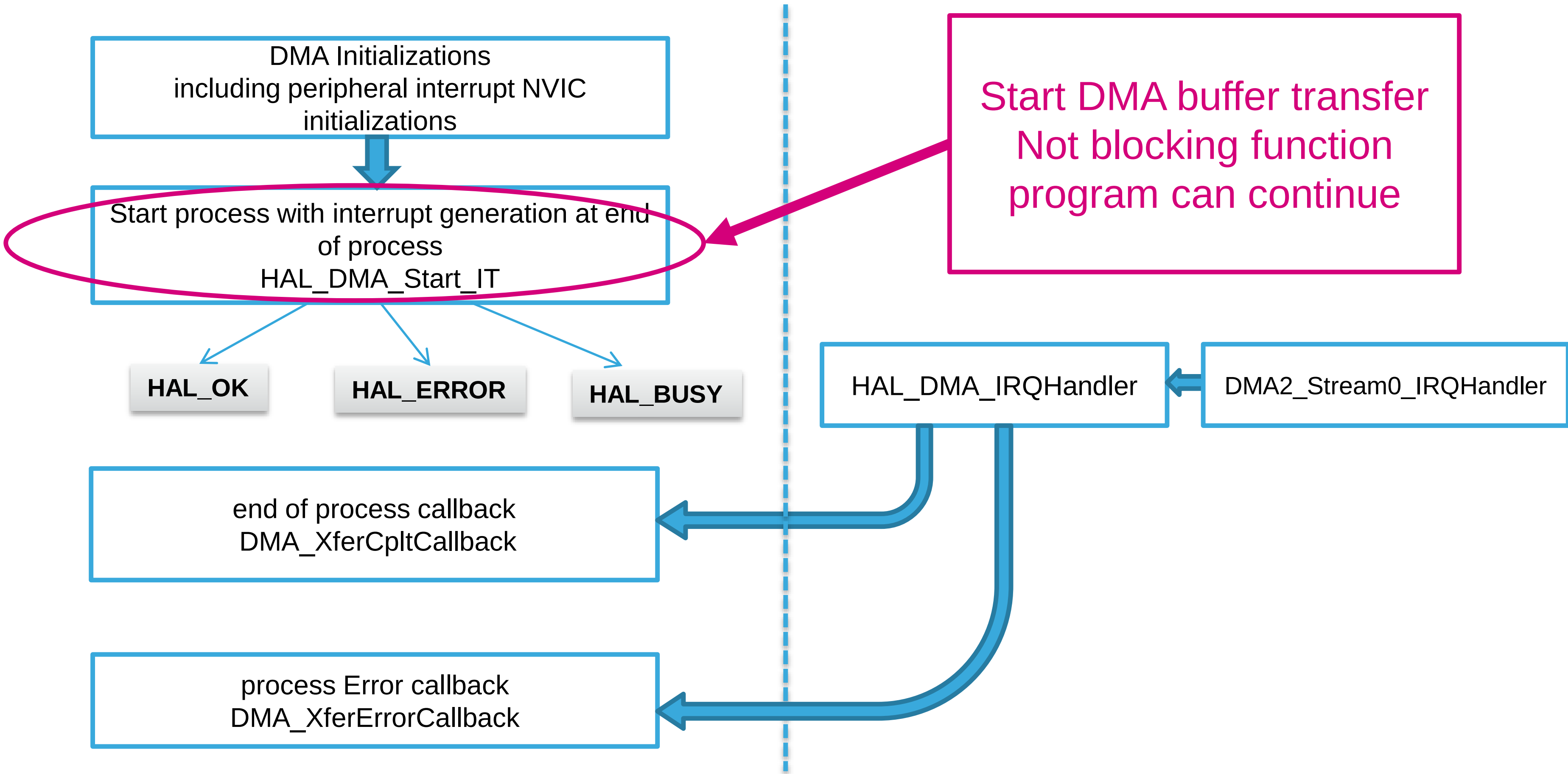
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



1.3.2 Use DMA M2M with interrupt

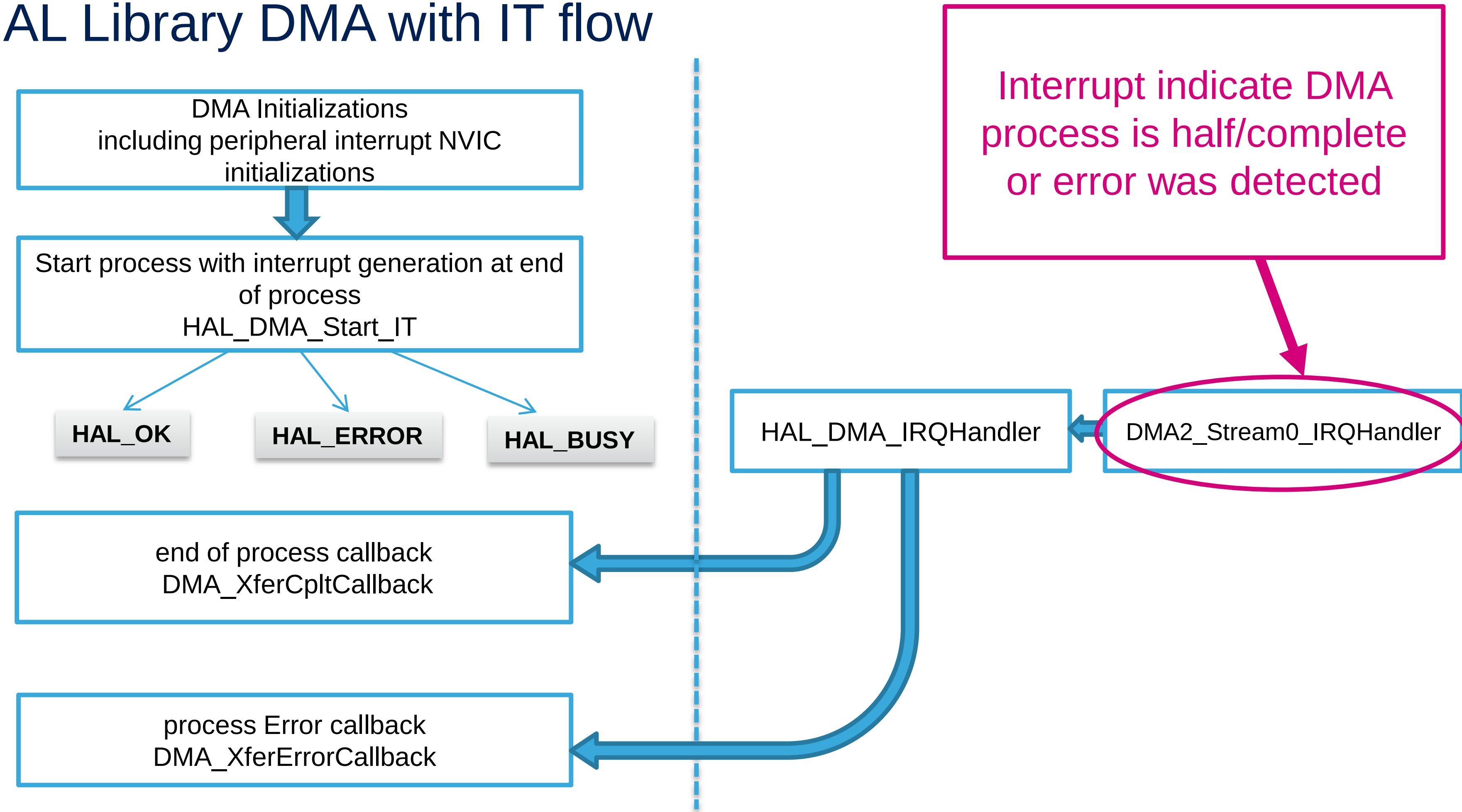
HAL Library DMA with IT flow



life.augmented

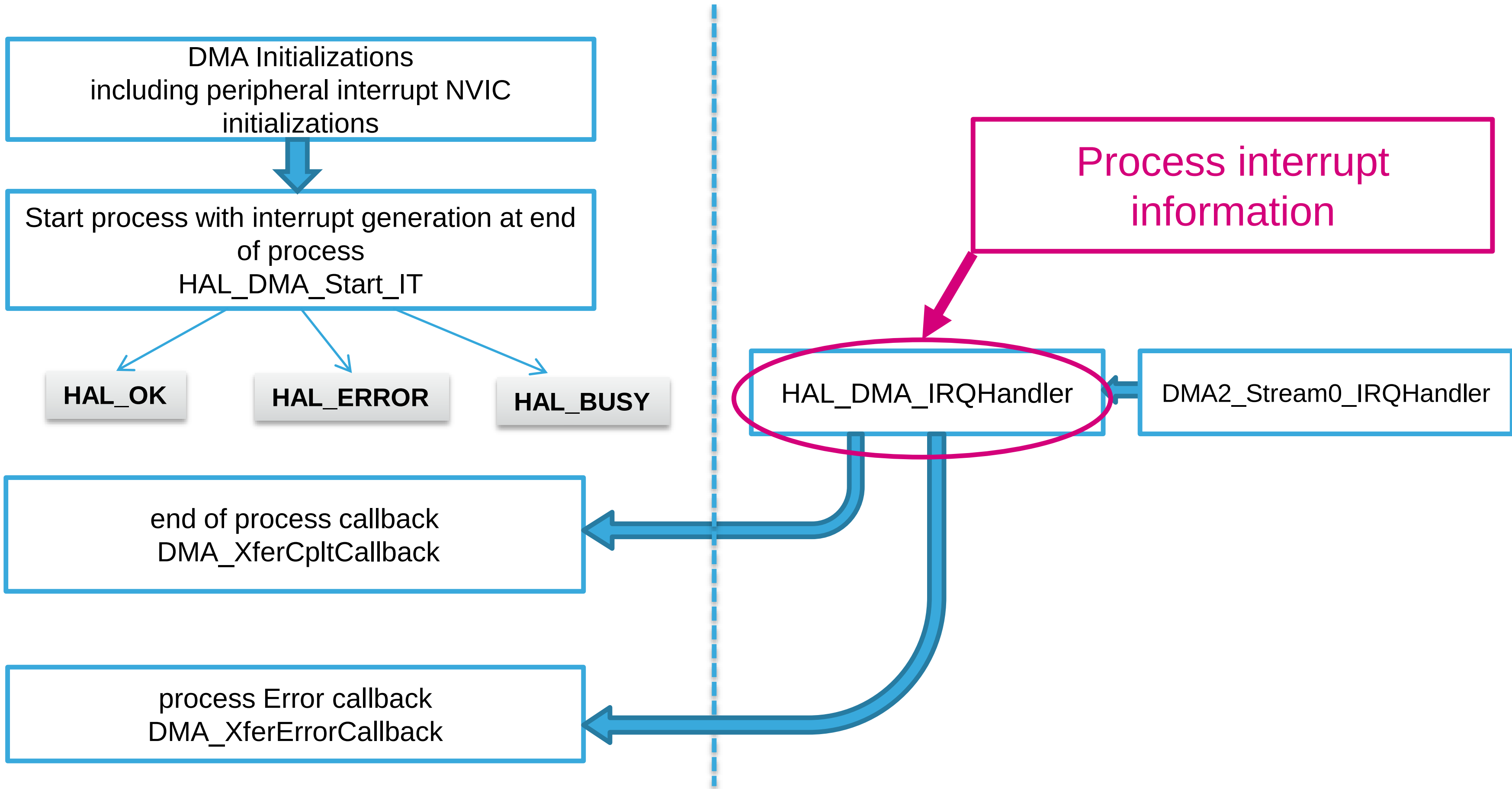
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



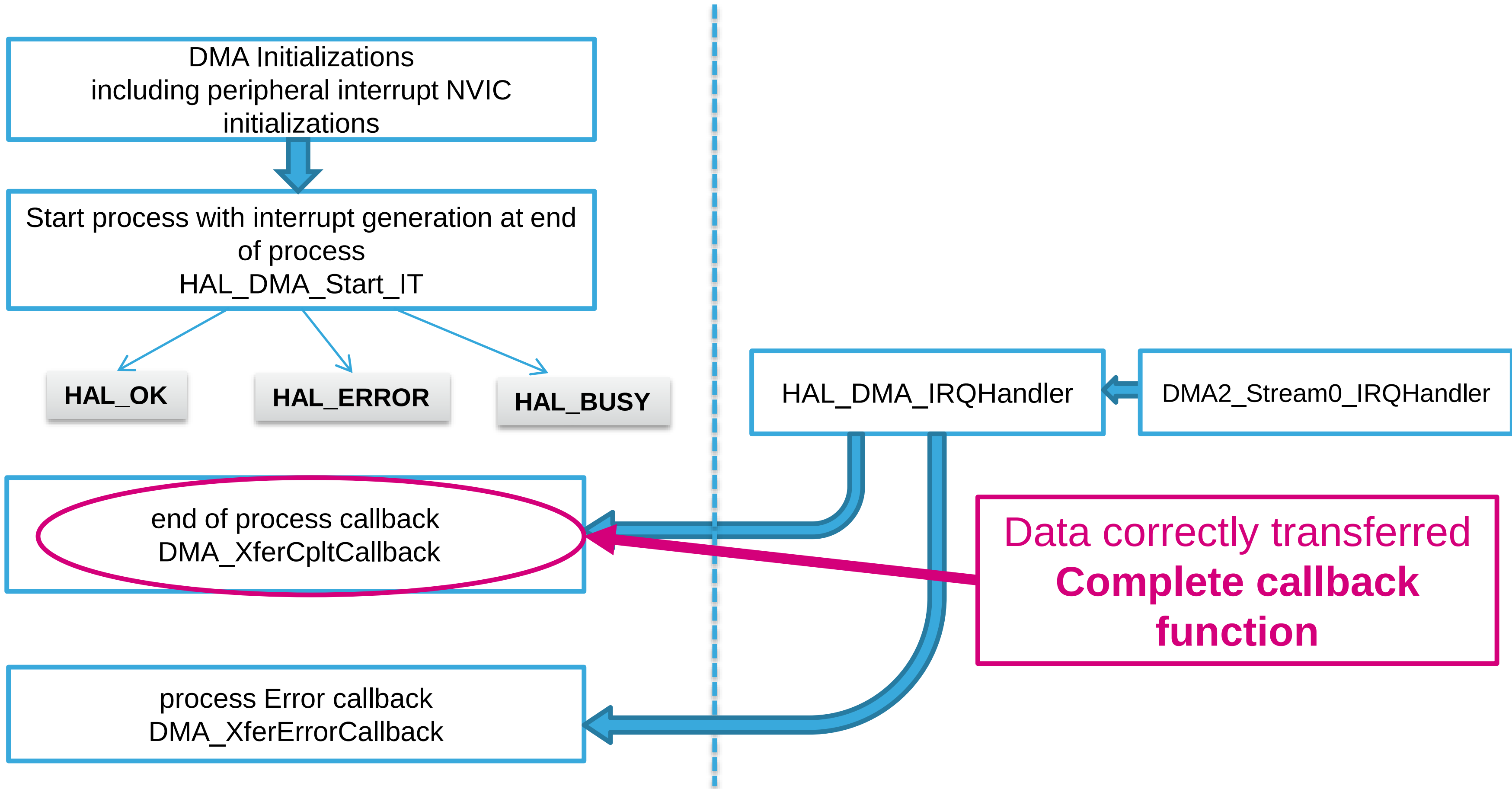
Use DMA M2M with interrupt

HAL Library DMA with IT flow



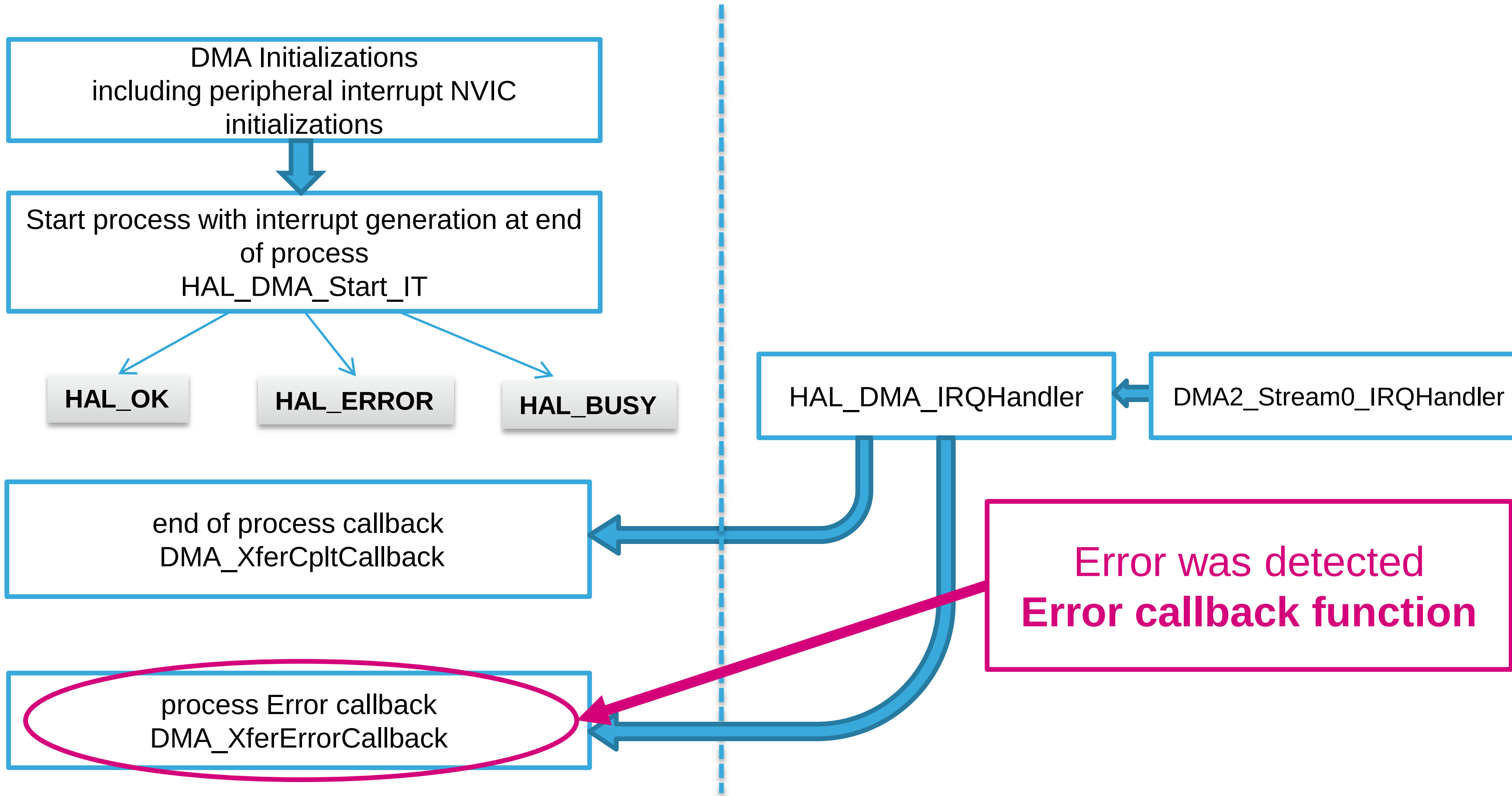
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



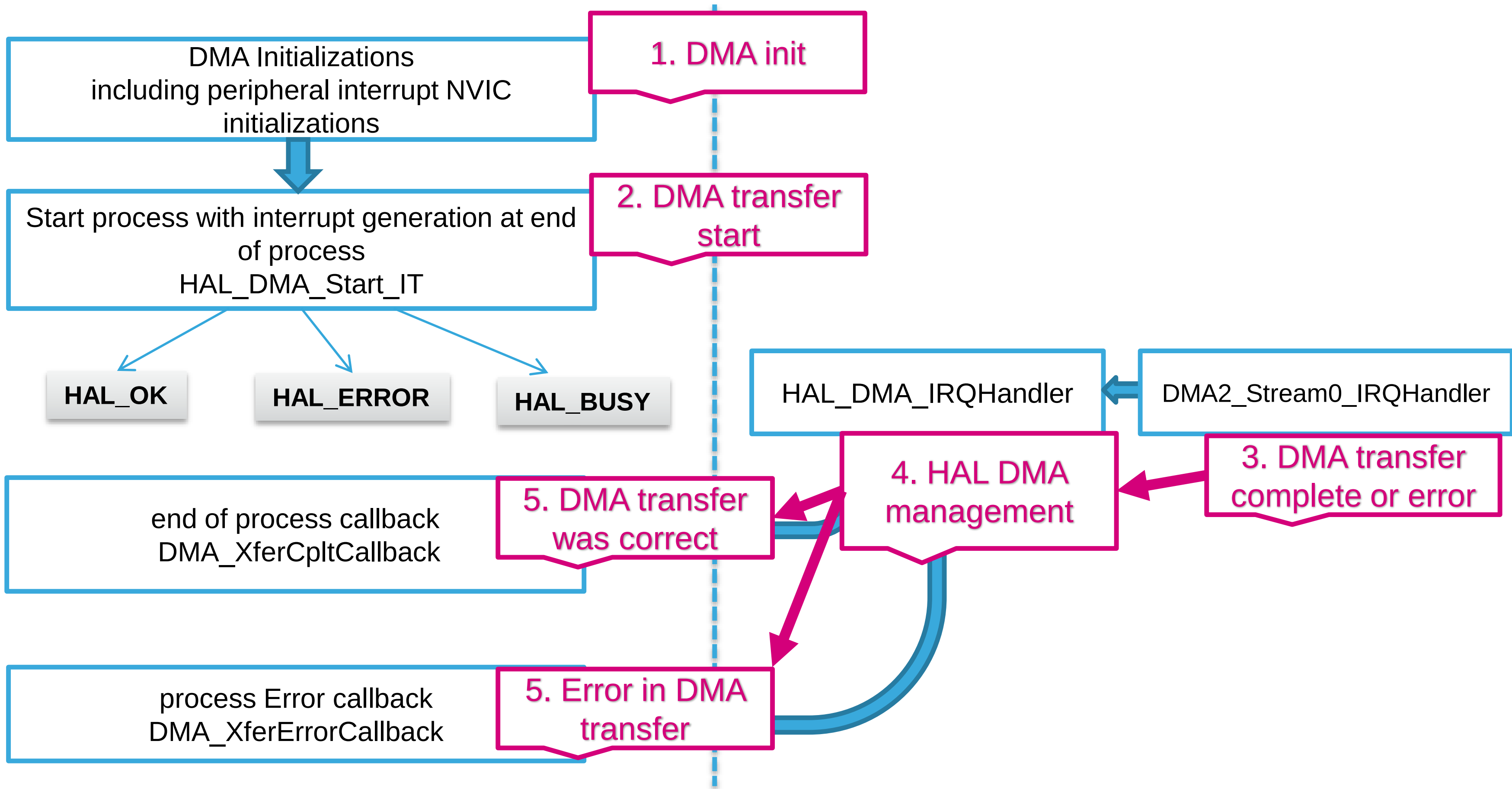
1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



1.3.2 Use DMA M2M with interrupt

HAL Library DMA with IT flow



1.3.2 Use DMA M2M with interrupt

- 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)

1.3.2 Use DMA M2M with interrupt

- 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 */
```

1.3.2 Use DMA M2M with interrupt

- 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 */
```

1.3.2 Use DMA M2M with interrupt

- 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 */
```

2.1.1 UART Poll lab

2.1.1 Simple UART communication

- 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

2.1.1 Simple UART communication

- 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

2.1.1 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																	
Main function	LQFP144	System	SDRAM	LCD-TFT	LCD-RGB	LCD-SPI	L3GD20	USB	LED	Puchbutton	ACP/RF	Touch panel	Free I/O	Power supply	CN2	CN3	CN6	P1	P2
BOOT0	138	BOOT0																21	
NRST	25	NRST		RESET	RESET	RESET				B2					5				12
PA0	34									B1									18
PA1	35						INT1												17
PA2	36						INT2												20
PA3	37			DB3	B5														19
PA4	40			VSYNC	VSYNC														22
PA5	41																		21
PA6	42			DB6	G2														24
PA7	43										ACP_RST					4			23
PA8	100										SCL	SCL				3		53	
PA9	101																	52	
PA10	102																	51	
PA11	103			DB14	R4													50	
PA12	104			DB15	R5													49	
PA13	105	SWDIO													4			48	

2.1.1 Simple UART communication

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

Image from STM32F429 datasheet

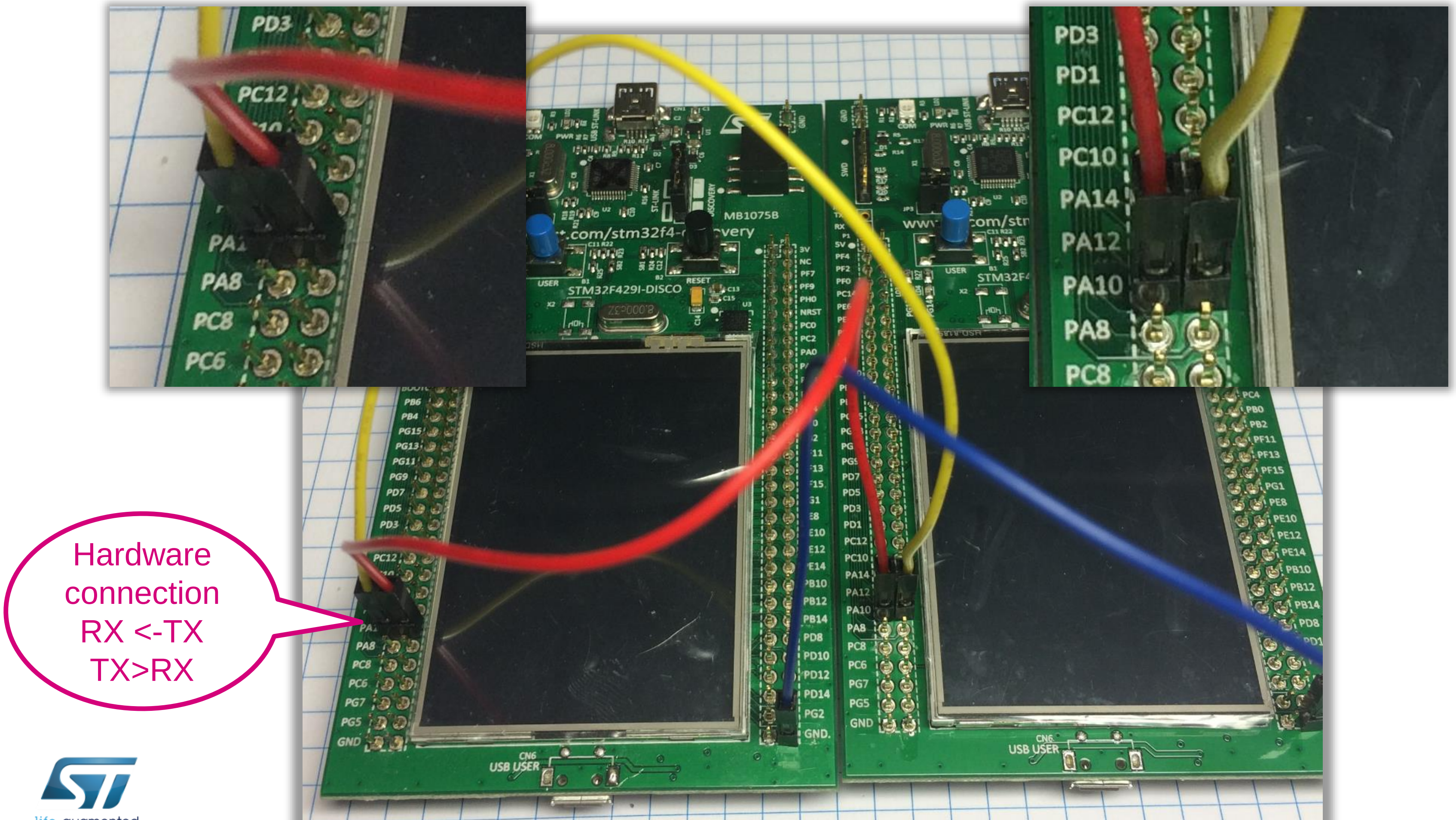
Table 12. STM32F427xx and STM32F429xx alternate function mapping

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/2/3/4/5/6	SPI2/3/SAI1	SPI3/USART1/2/3	USART6/UART4/5/7/8	CAN1/2/TIM12/13/14/LCD	OTG2_HS/OTG1_FS	ETH	FMC/SDIO/OTG2_FS	DCMI	LCD	SYS
Port A	PA0	-	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	-	ETH_MII_CRS	-	-	-	EVENTOUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_RTS	UART4_RX	-	-	ETH_MII_RX_CLK/ETH_RMII_REF_CLK	-	-	-	EVENTOUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX	-	-	-	ETH_MDIO	-	-	-	EVENTOUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	-	USART2_RX	-	-	OTG_HS_ULPI_D0	ETH_MII_COL	-	-	LCD_B5	EVENTOUT
	PA4	-	-	-	-	-	SPI1_NSS	SPI3_NSS/I2S3_WS	USART2_CK	-	-	-	-	OTG_HS_SOF	DCMI_HSYNC	LCD_VSYNC	EVENTOUT
	PA5	-	TIM2_CH1/TIM2_ETR	-	TIM8_CH1N	-	SPI1_SCK	-	-	-	-	OTG_HS_ULPI_CK	-	-	-	-	EVENTOUT
	PA6	-	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	-	SPI1_MISO	-	-	-	TIM13_CH1	-	-	-	DCMI_PIXCLK	LCD_G2	EVENTOUT
	PA7	-	TIM1_CH1N	TIM3_CH2	TIM8_CH1N	-	SPI1_MOSI	-	-	-	TIM14_CH1	-	ETH_MII_RX_DV/ETH_RMII_CRS_DV	-	-	-	EVENTOUT
	PA8	MCO1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_CK	-	-	OTG_FS_SOF	-	-	-	LCD_R6	EVENTOUT
	PA9	-	TIM1_CH2	-	-	I2C3_SMBA	-	-	USART1_TX	-	-	-	-	-	DCMI_D0	-	EVENTOUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_RX	-	-	OTG_FS_ID	-	-	DCMI_D1	-	EVENTOUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_CTS	-	CAN1_RX	OTG_FS_DM	-	-	-	LCD_R4	EVENTOUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_RTS	-	CAN1_TX	OTG_FS_DP	-	-	-	LCD_R5	EVENTOUT

life.augmented

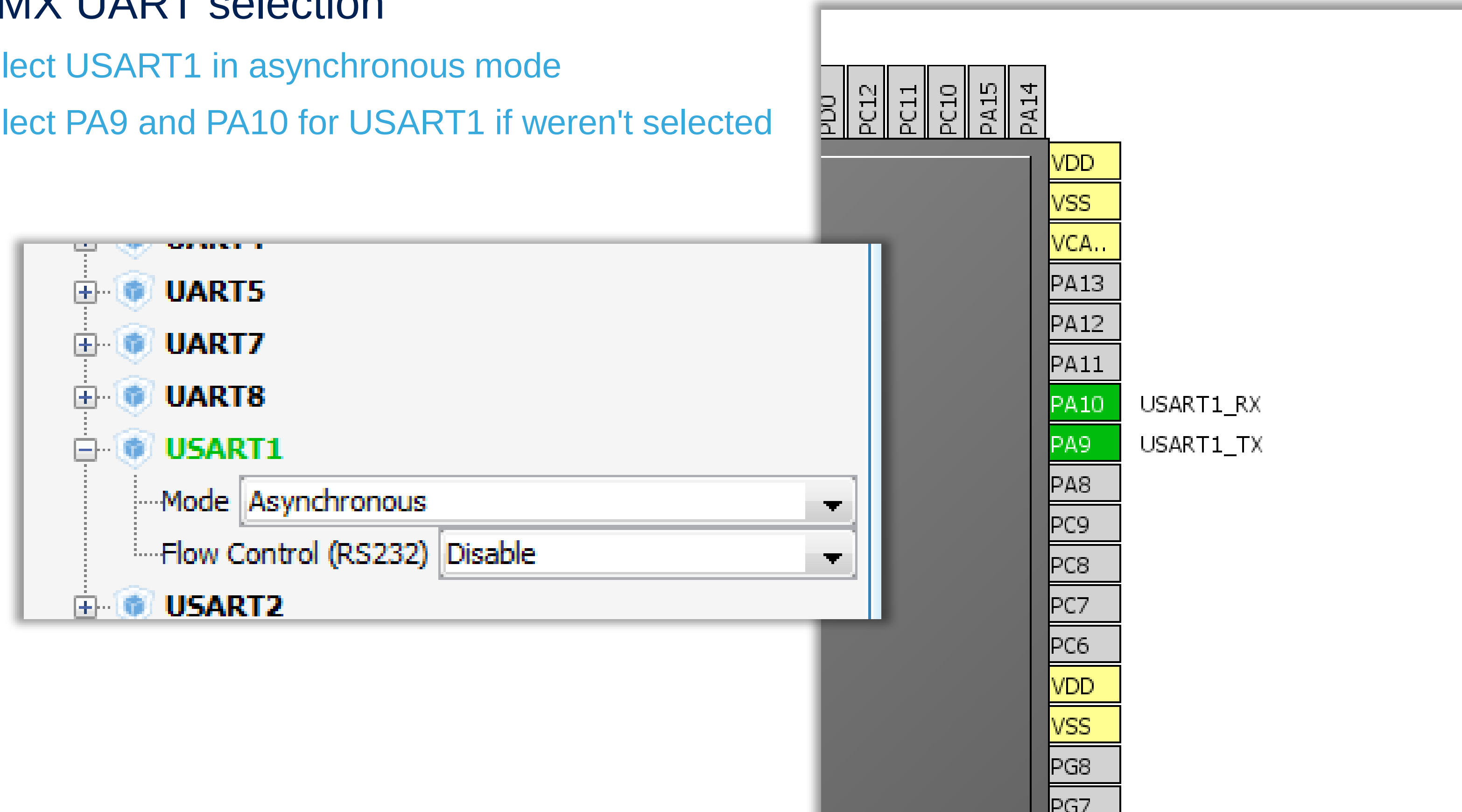
2.1.1 Simple UART communication

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



2.1.1 Simple UART communication

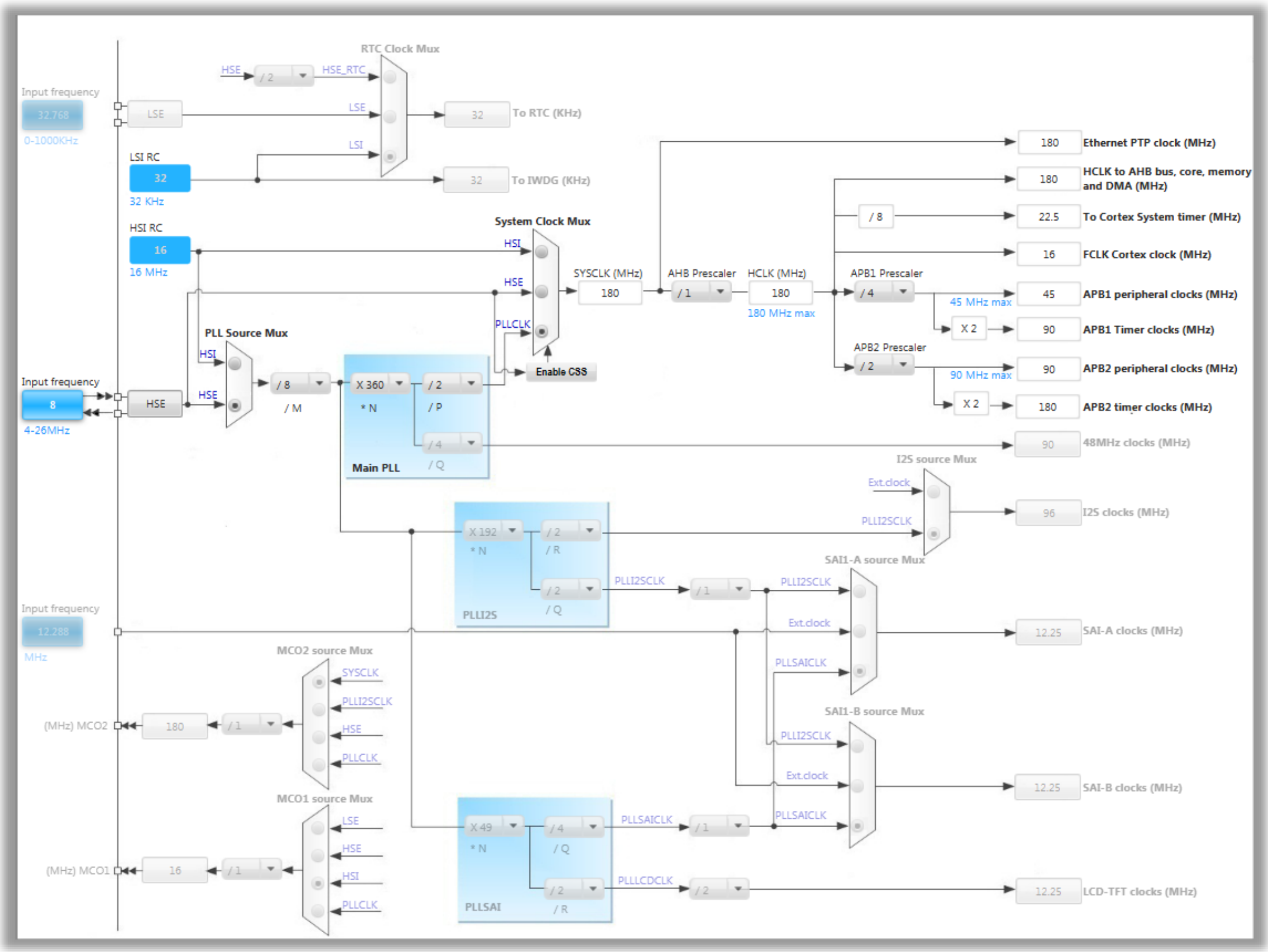
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX UART selection
  - Select USART1 in asynchronous mode
  - Select PA9 and PA10 for USART1 if weren't selected



life.augmented

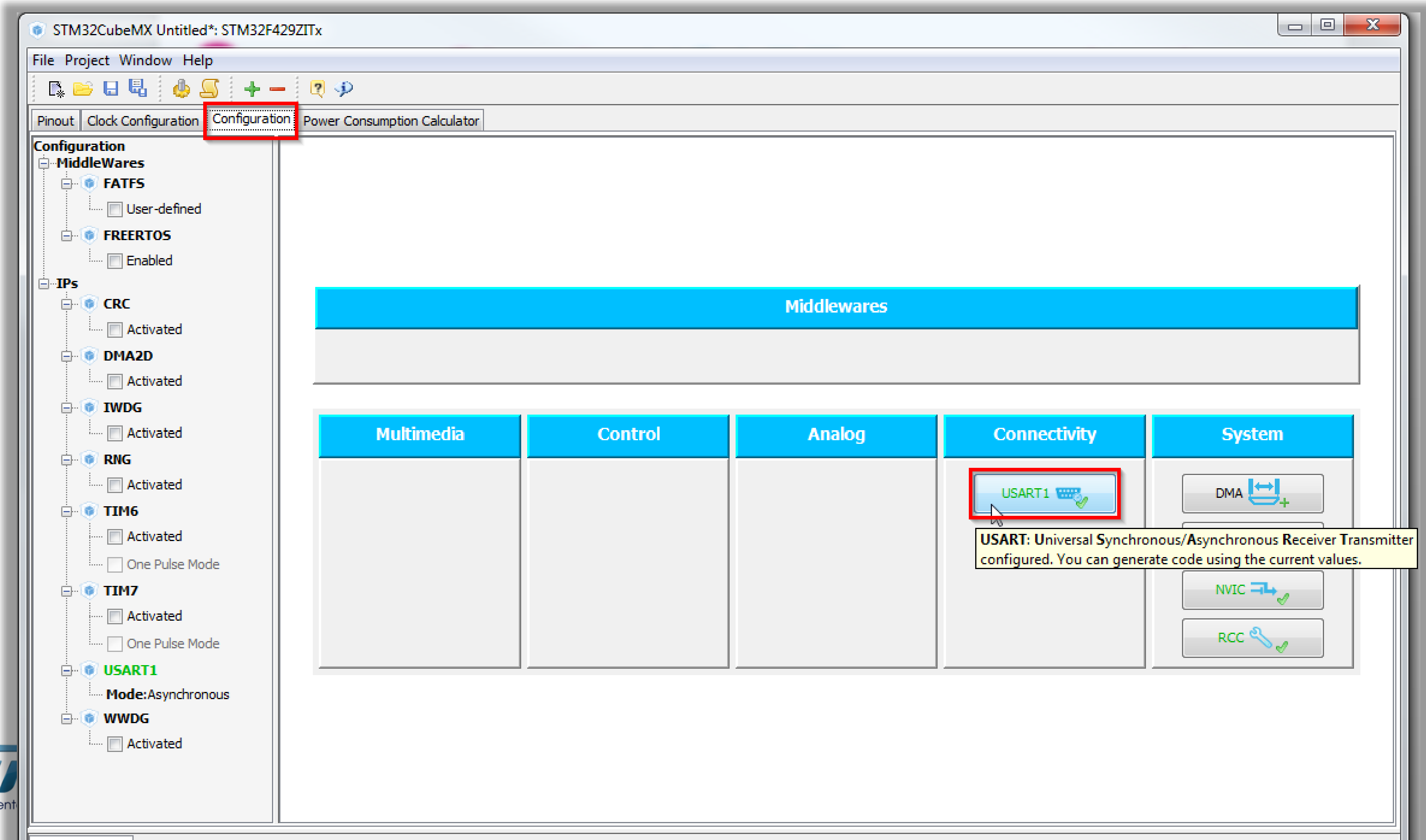
2.1.1 Simple UART communication

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



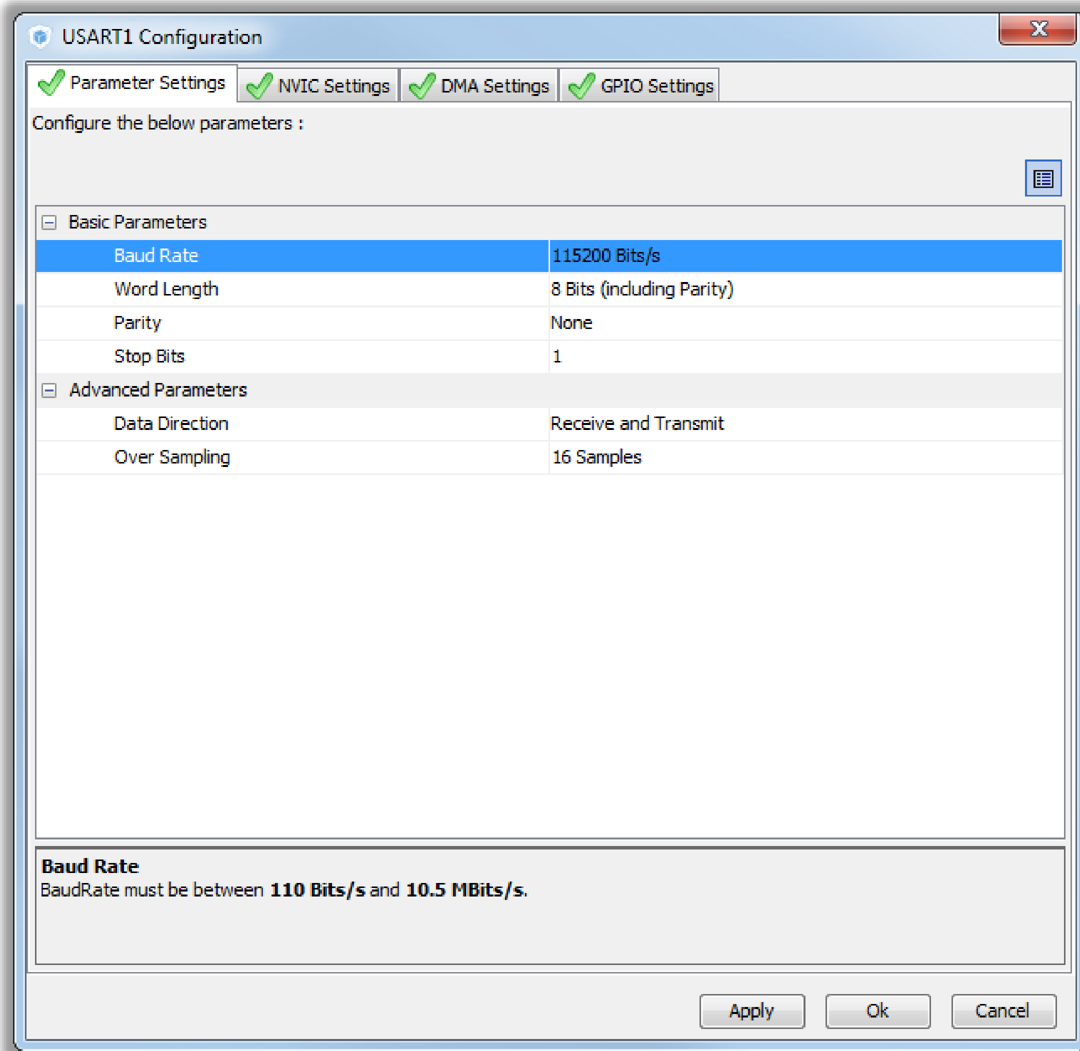
2.1.1 Simple UART communication

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



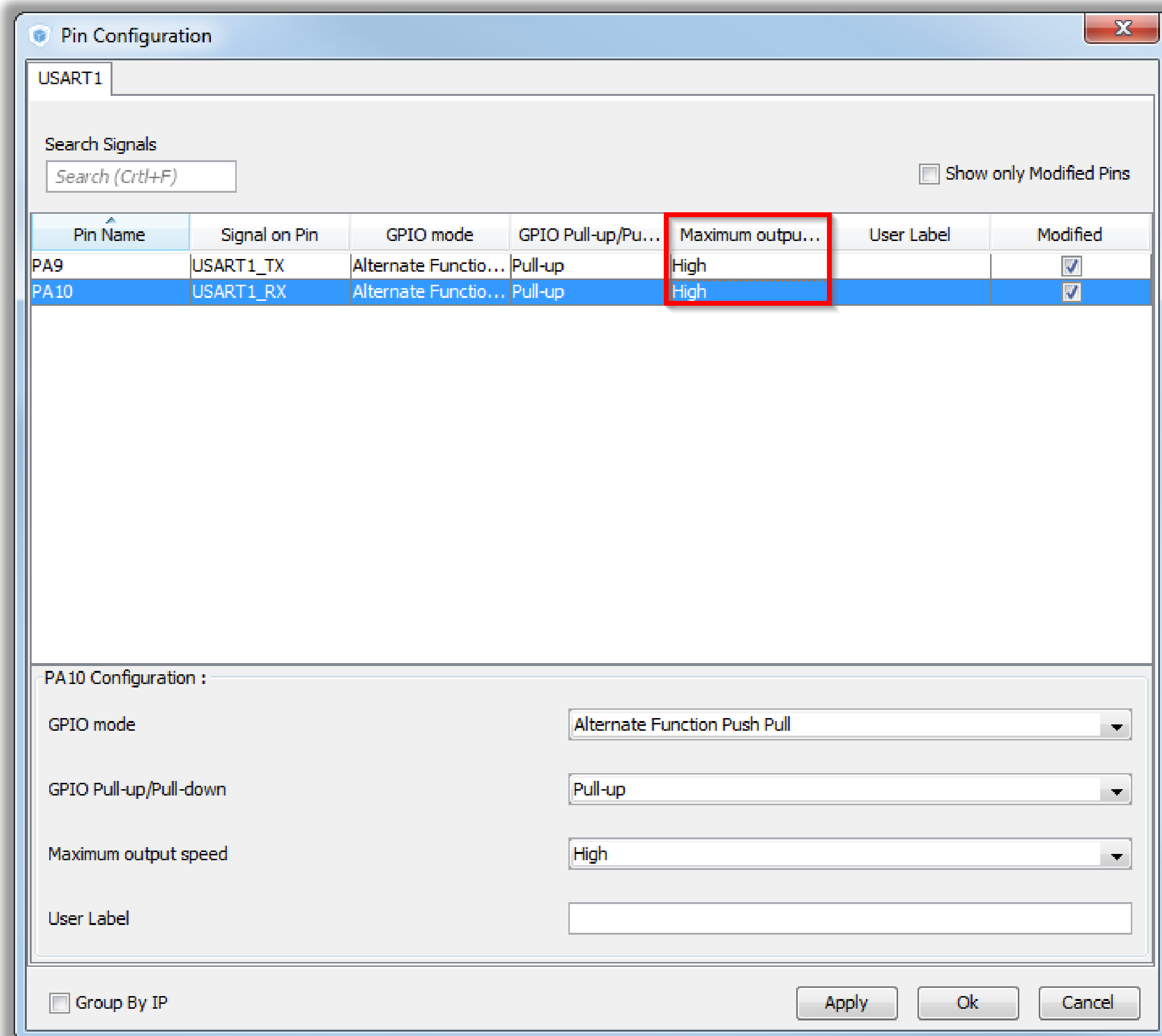
2.1.1 Simple UART communication

- CubeMX USART configuration check:
  - BaudRate
  - World length
  - Parity
  - Stop bits
  - Data direction
  - Oversampling



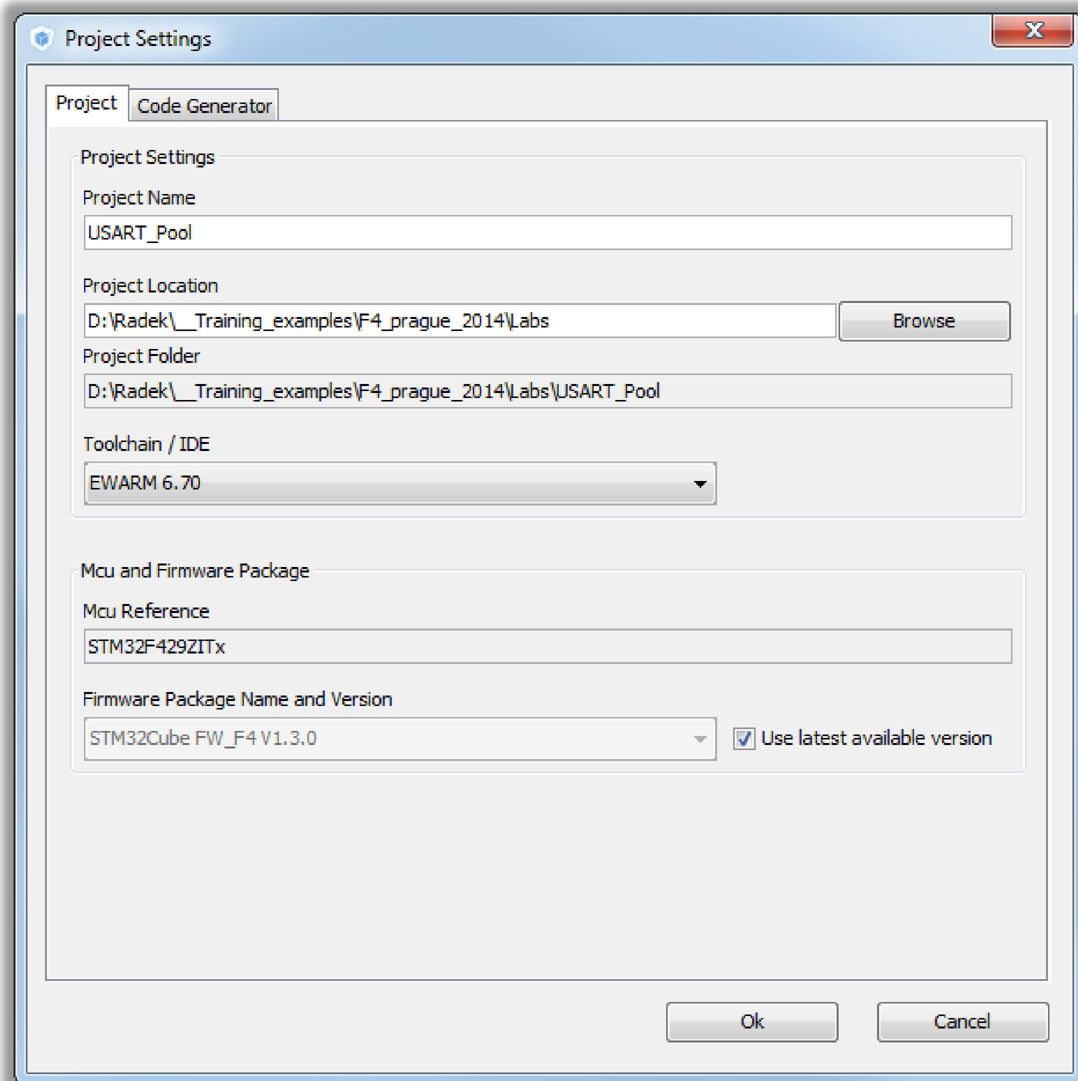
2.1.1 Simple UART communication

- 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



2.1.1 Simple UART communication

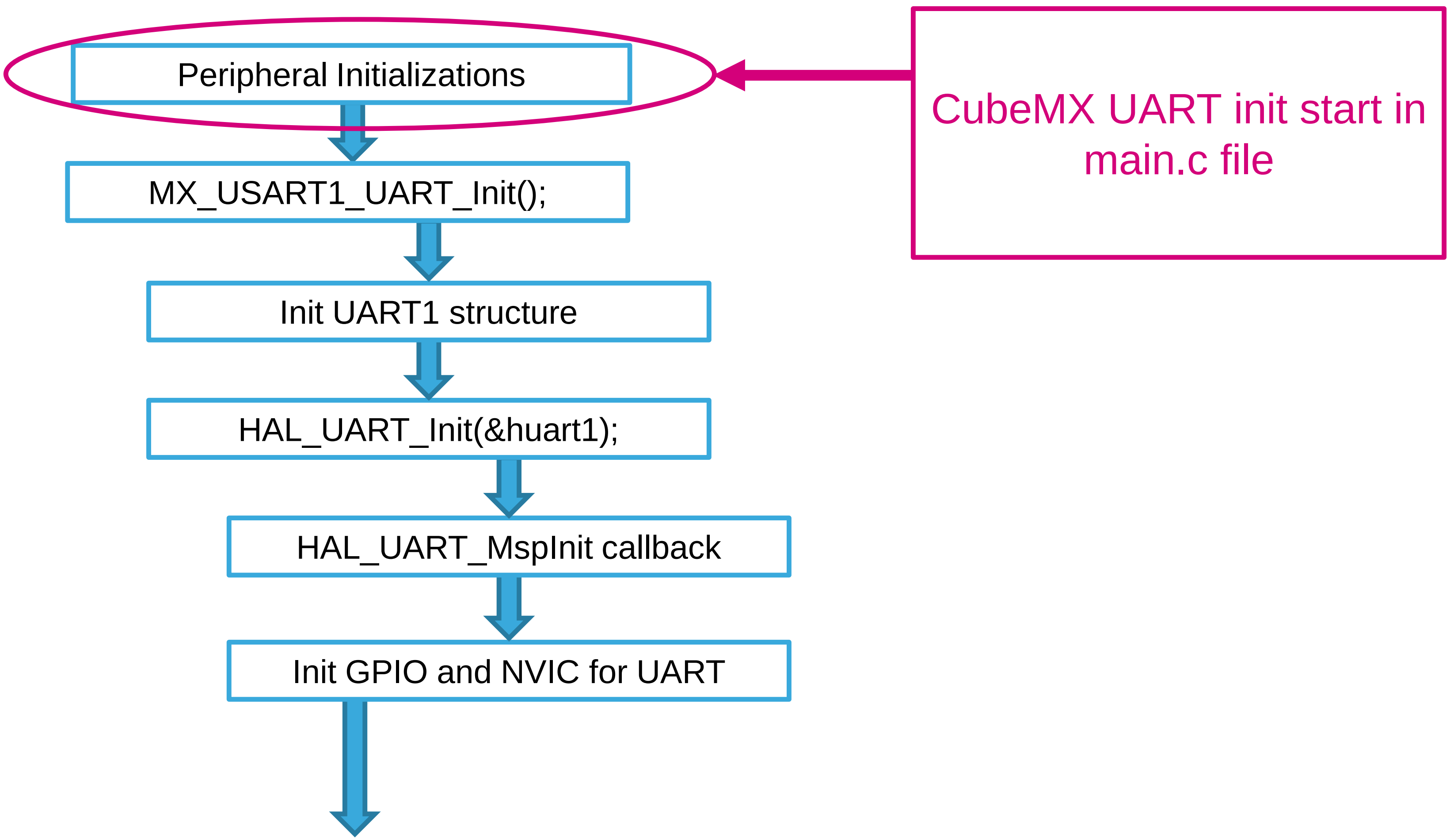
- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



life.augmented

2.1.1 Simple UART communication

HAL Library init flow



life.augmented

2.1.1 Simple UART communication

HAL Library init flow

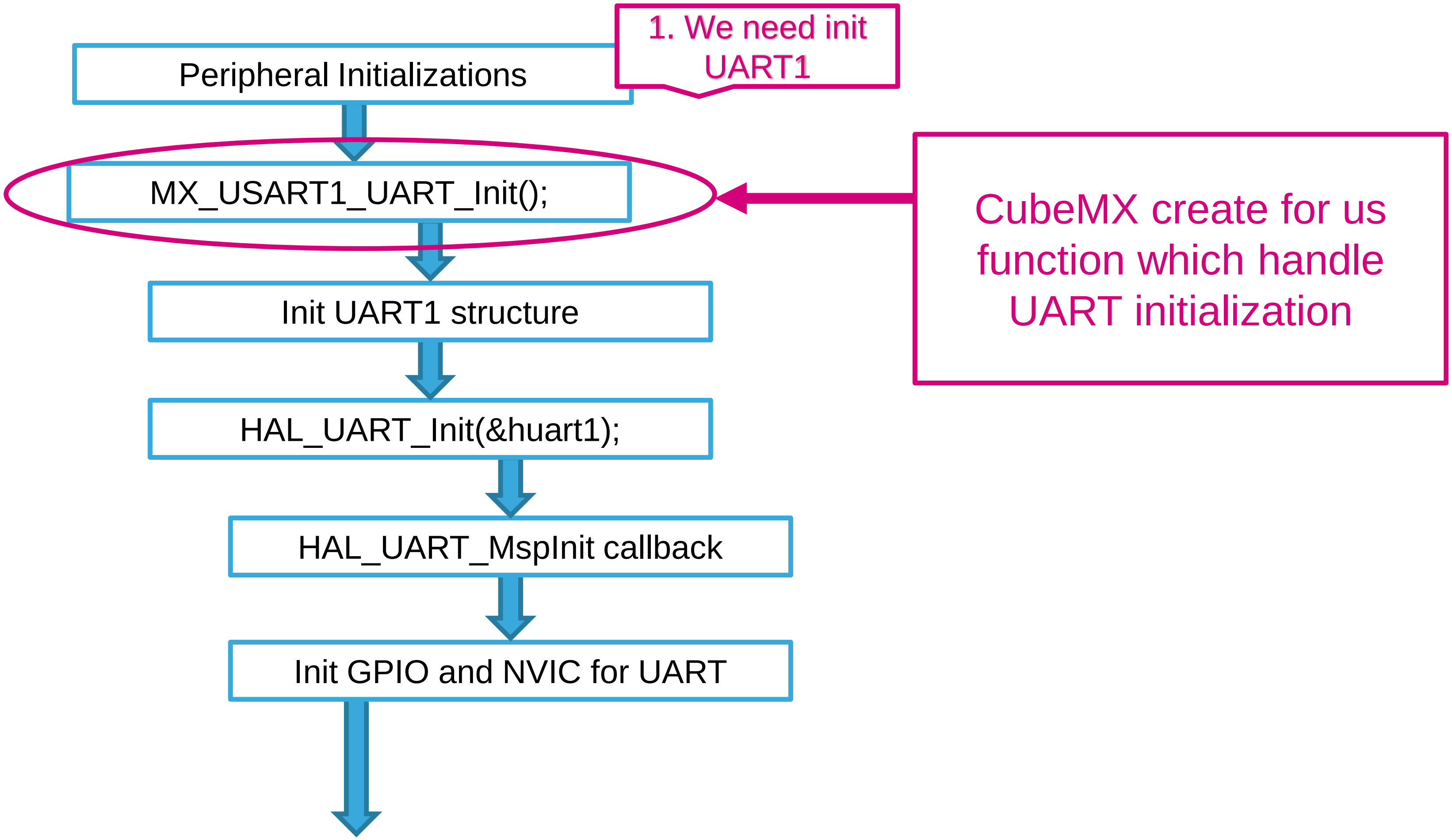
1. We need init UART1

Peripheral Initializations
↓
MX_USART1_UART_Init();
↓
Init UART1 structure
↓
HAL_UART_Init(&huart1);
↓
HAL_UART_MspInit callback
↓
Init GPIO and NVIC for UART
↓

life.augmented

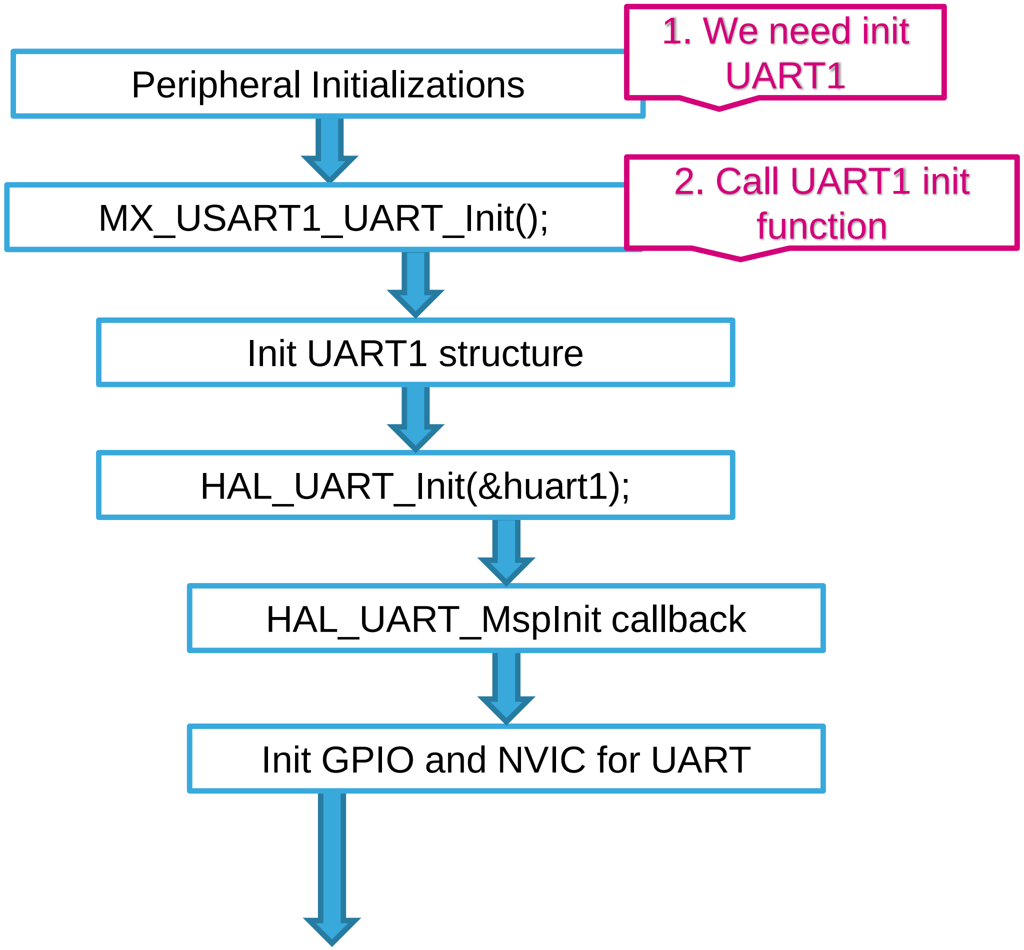
2.1.1 Simple UART communication

HAL Library init flow



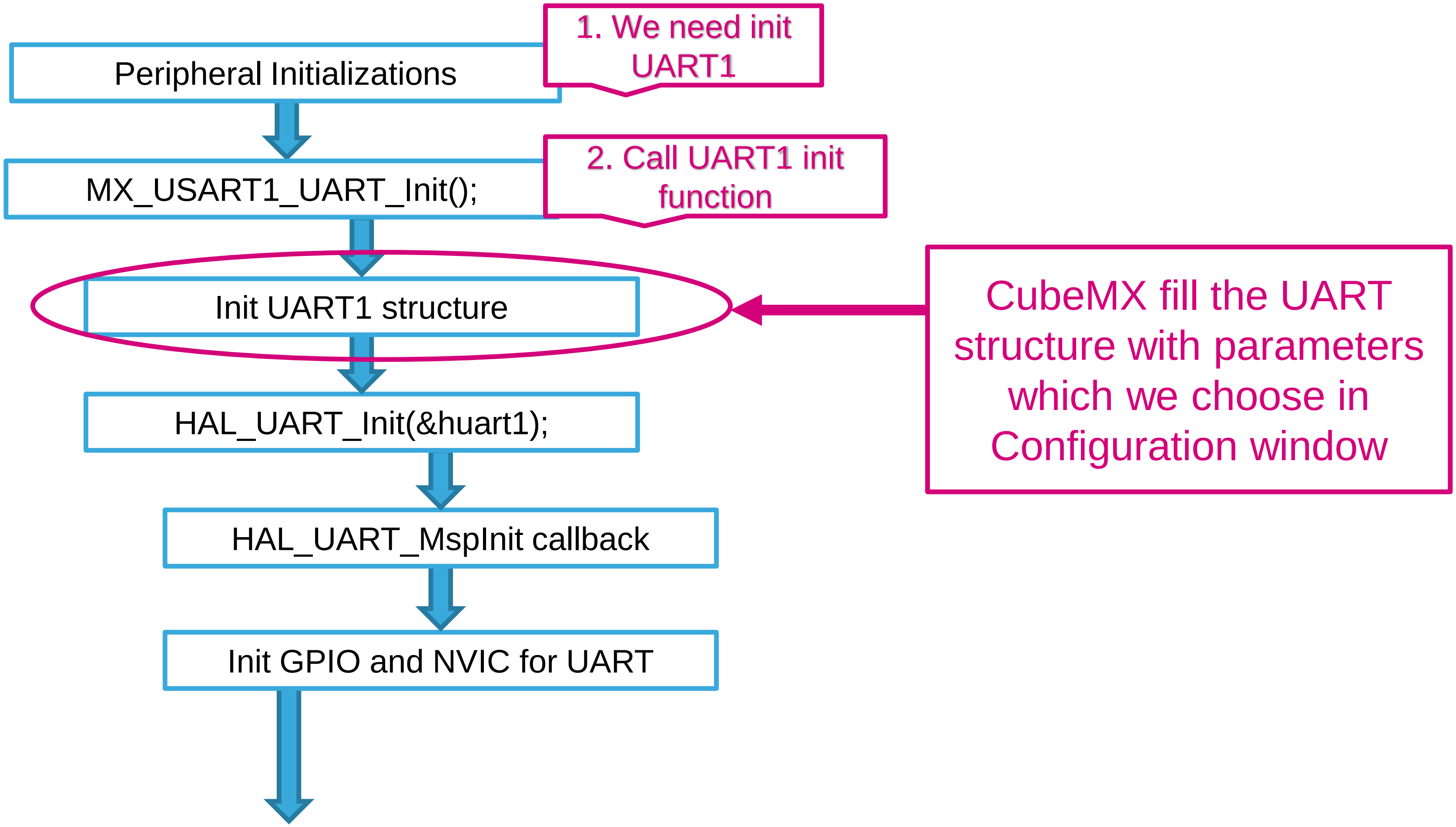
2.1.1 Simple UART communication

HAL Library init flow



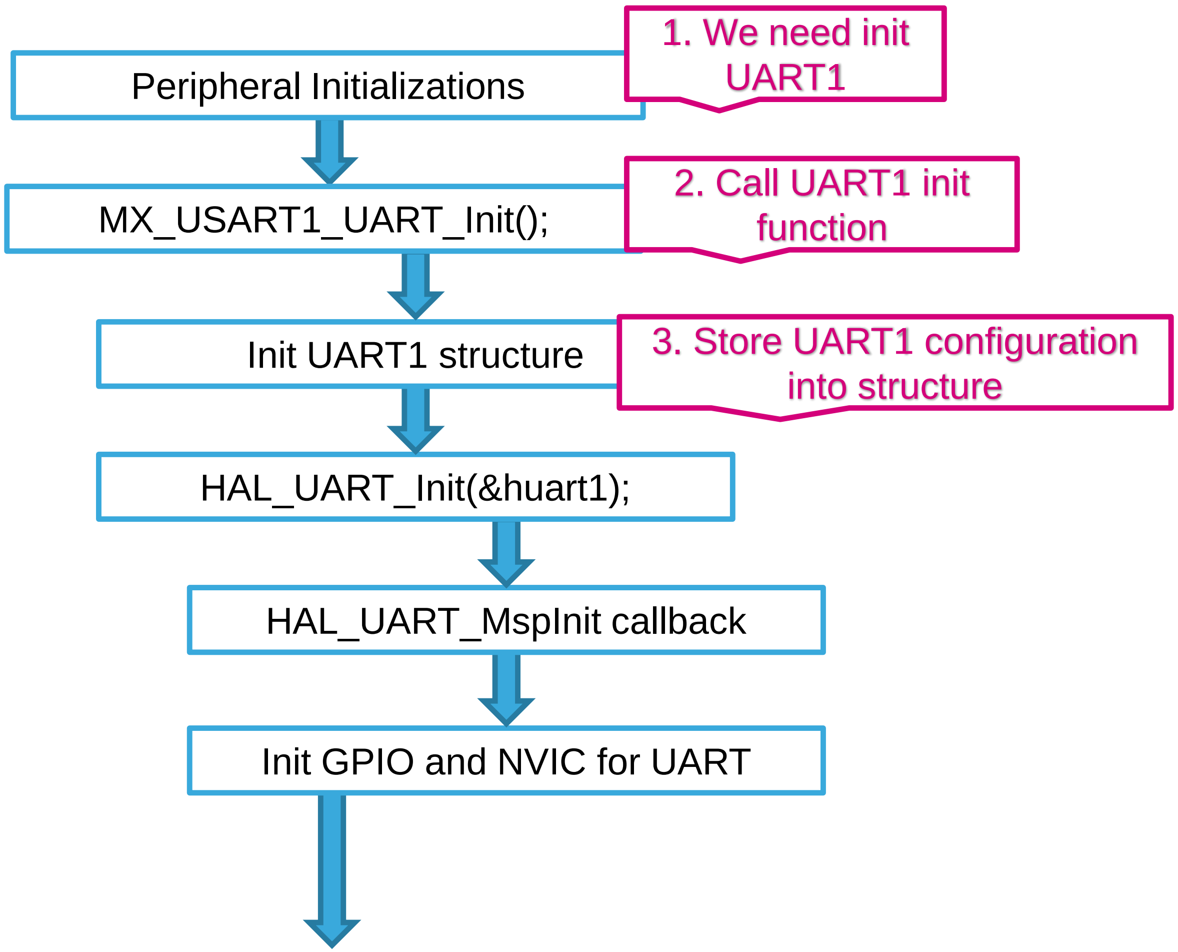
2.1.1 Simple UART communication

HAL Library init flow



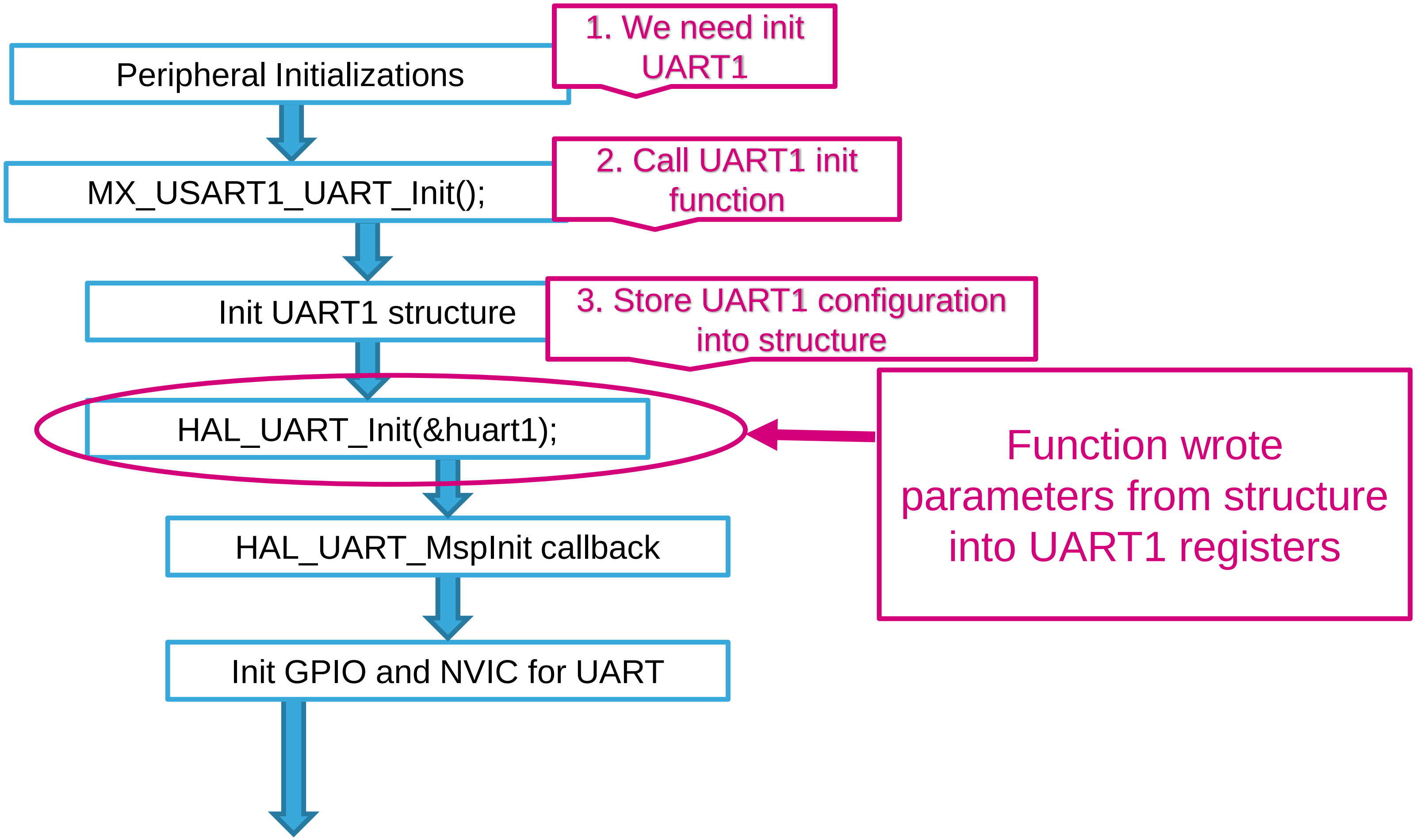
2.1.1 Simple UART communication

HAL Library init flow



2.1.1 Simple UART communication

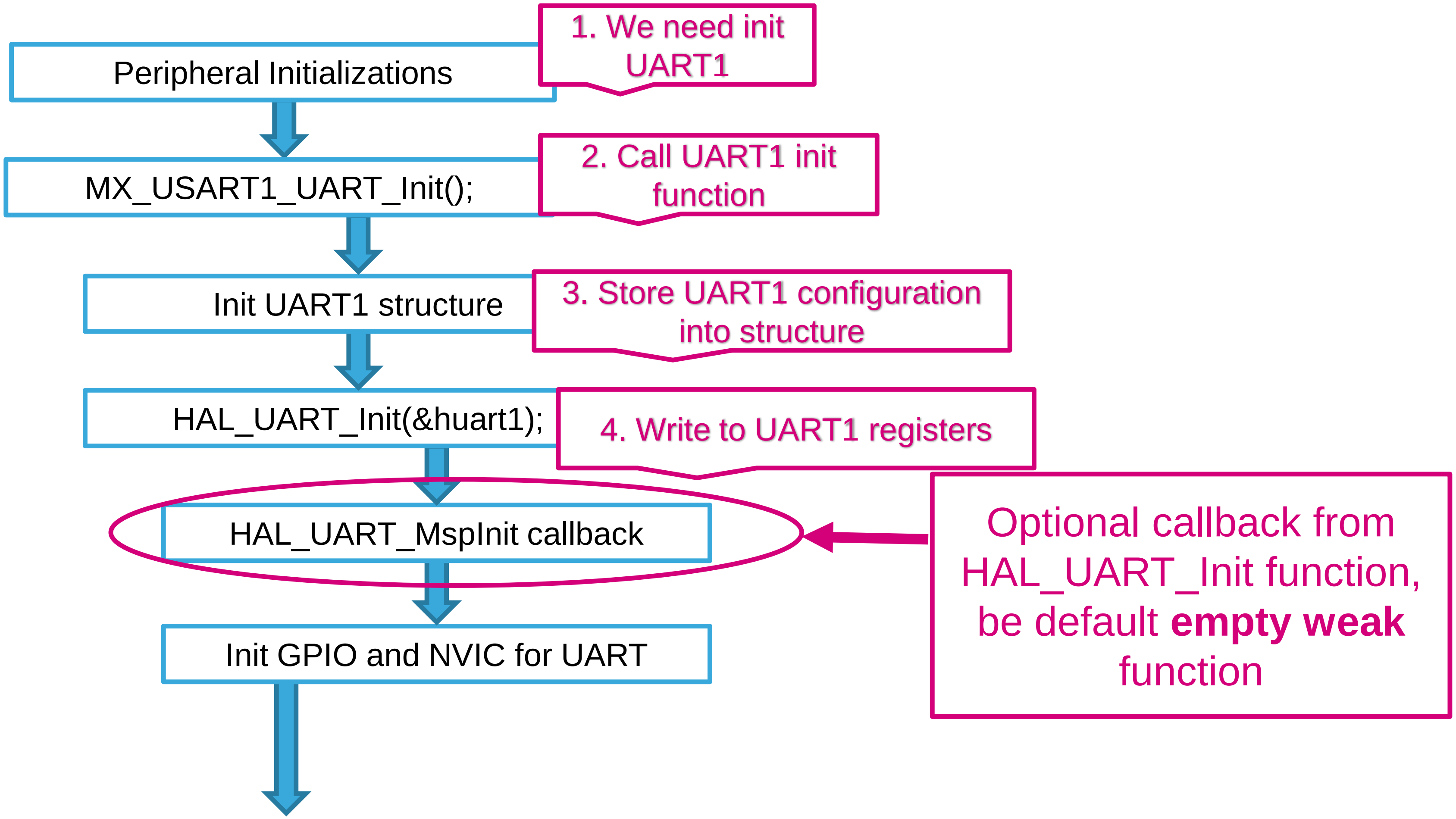
HAL Library init flow



life.augmented

2.1.1 Simple UART communication

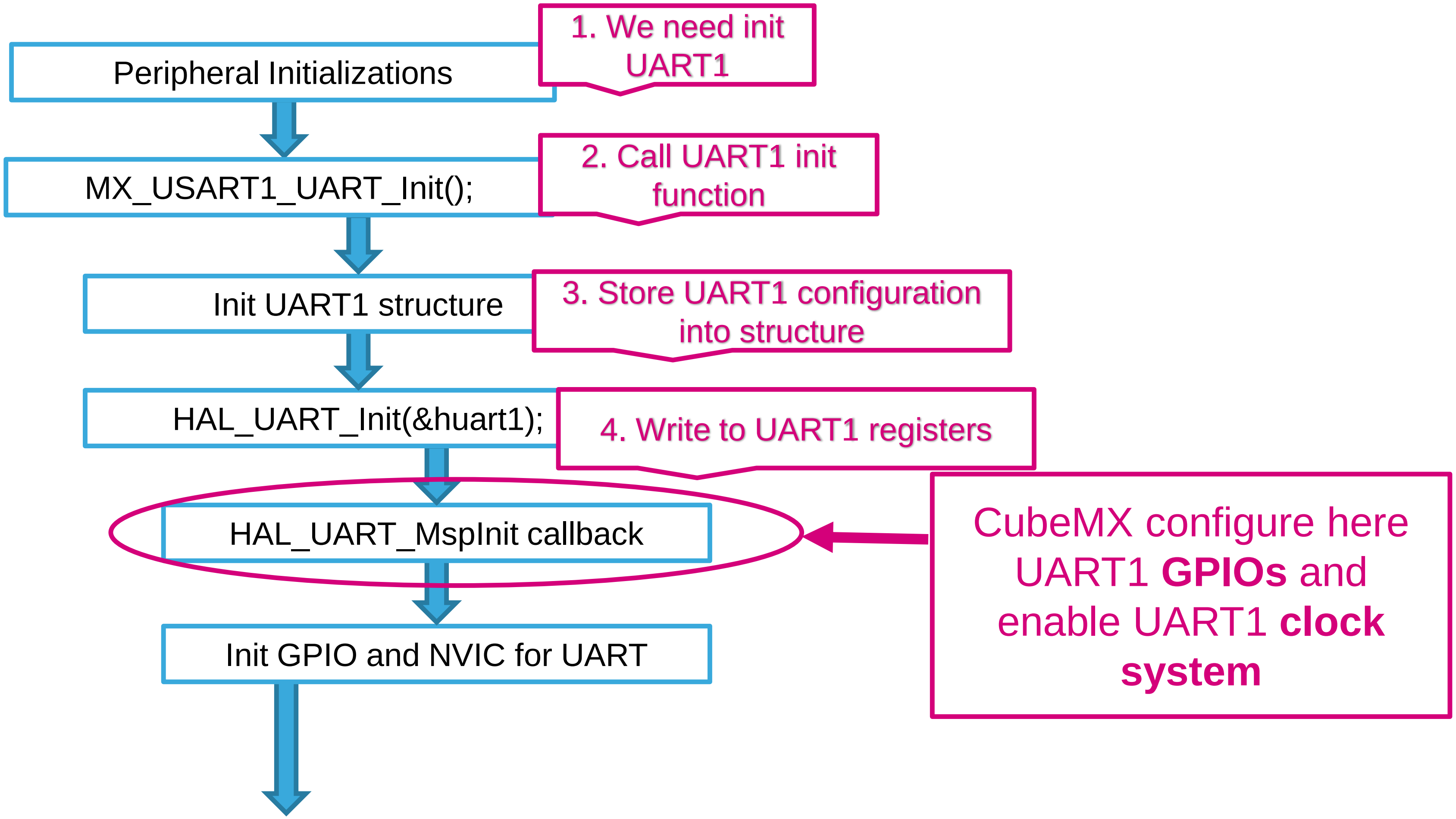
HAL Library init flow



life.augmented

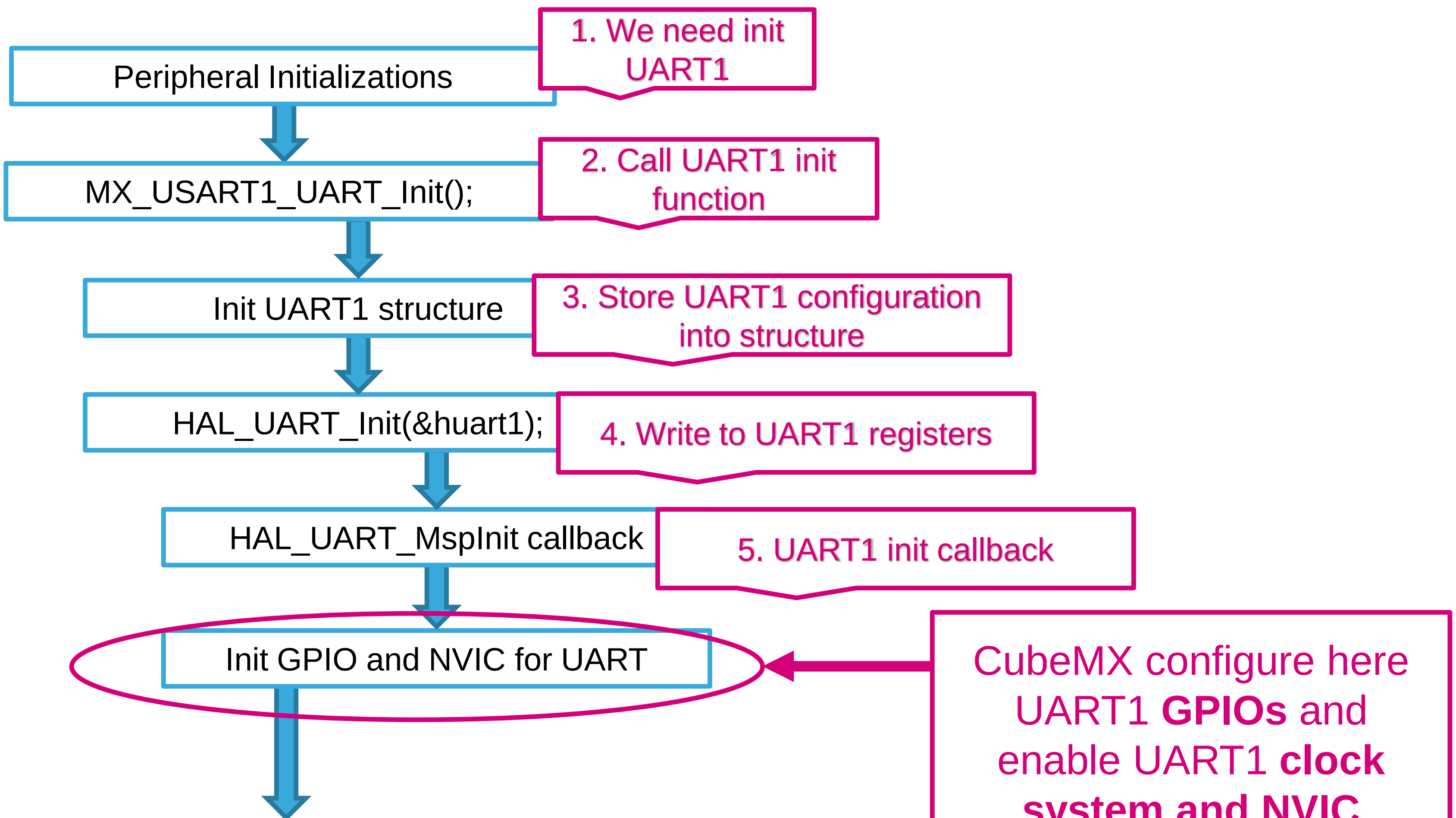
2.1.1 Simple UART communication

HAL Library init flow



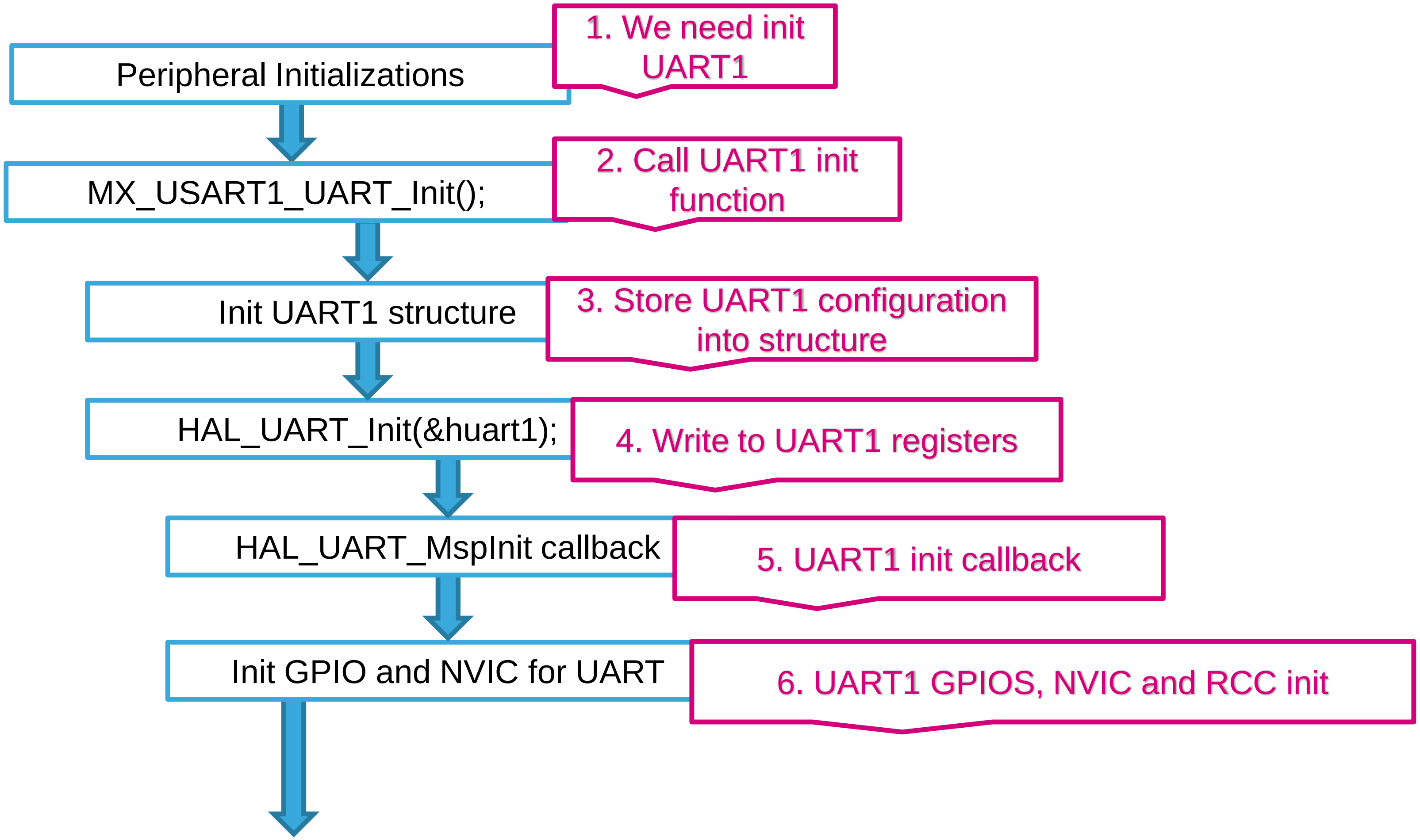
2.1.1 Simple UART communication

HAL Library init flow



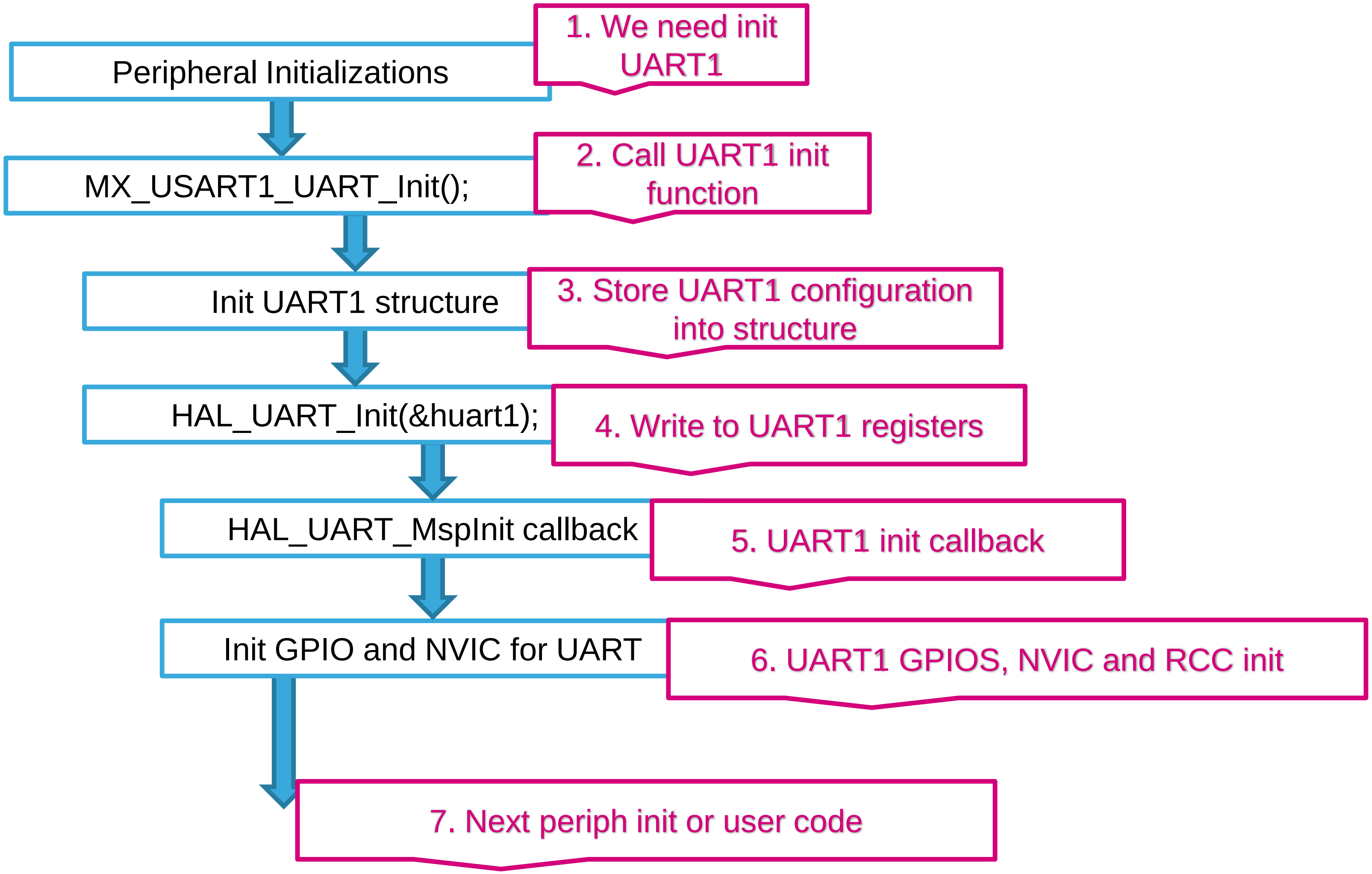
2.1.1 Simple UART communication

HAL Library init flow



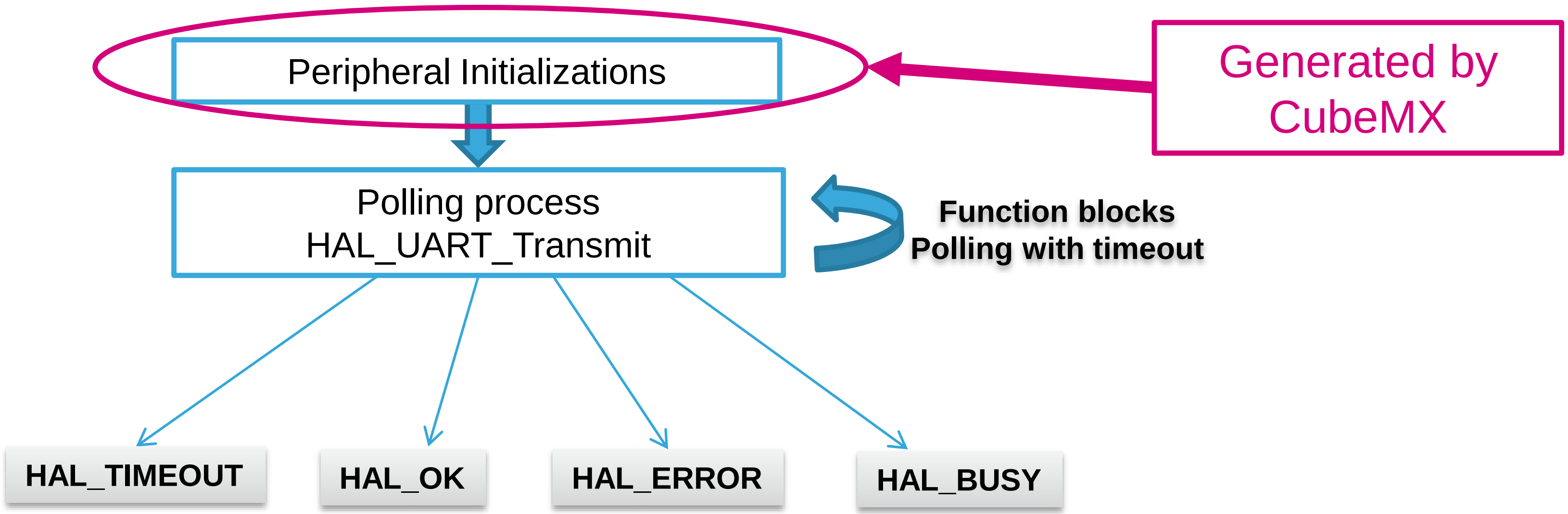
2.1.1 Simple UART communication

HAL Library init flow



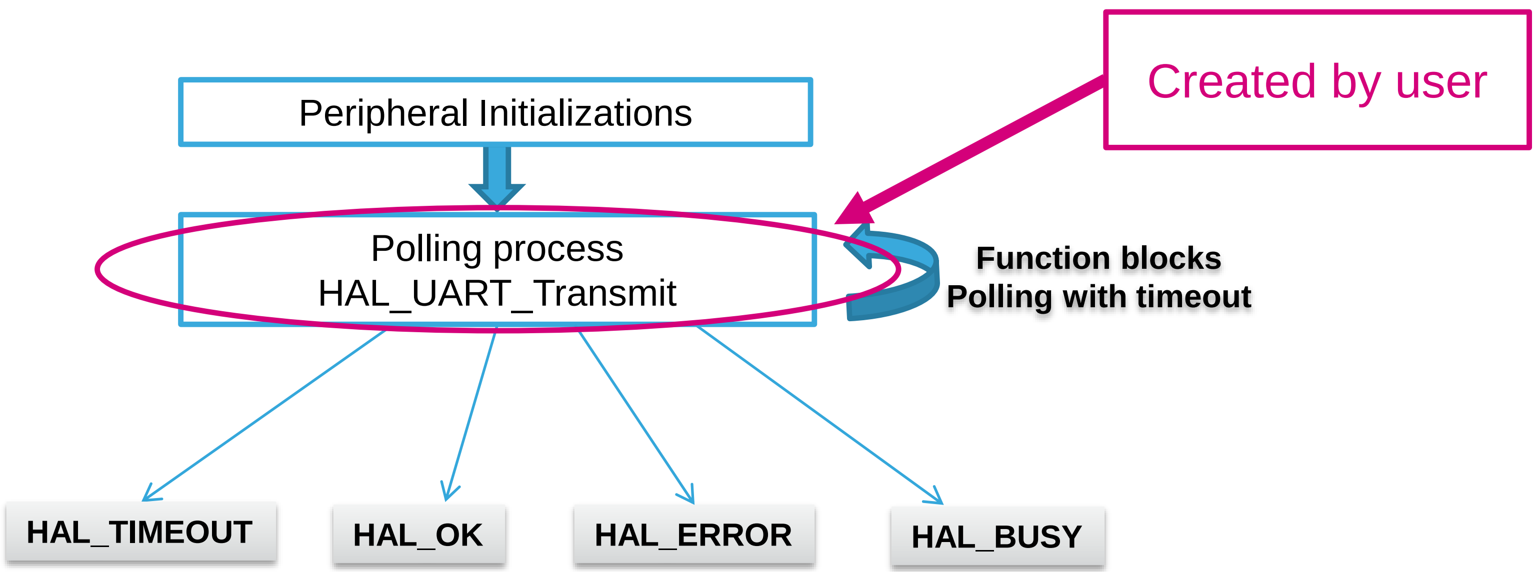
2.1.1 Simple UART communication

HAL Library transmit flow



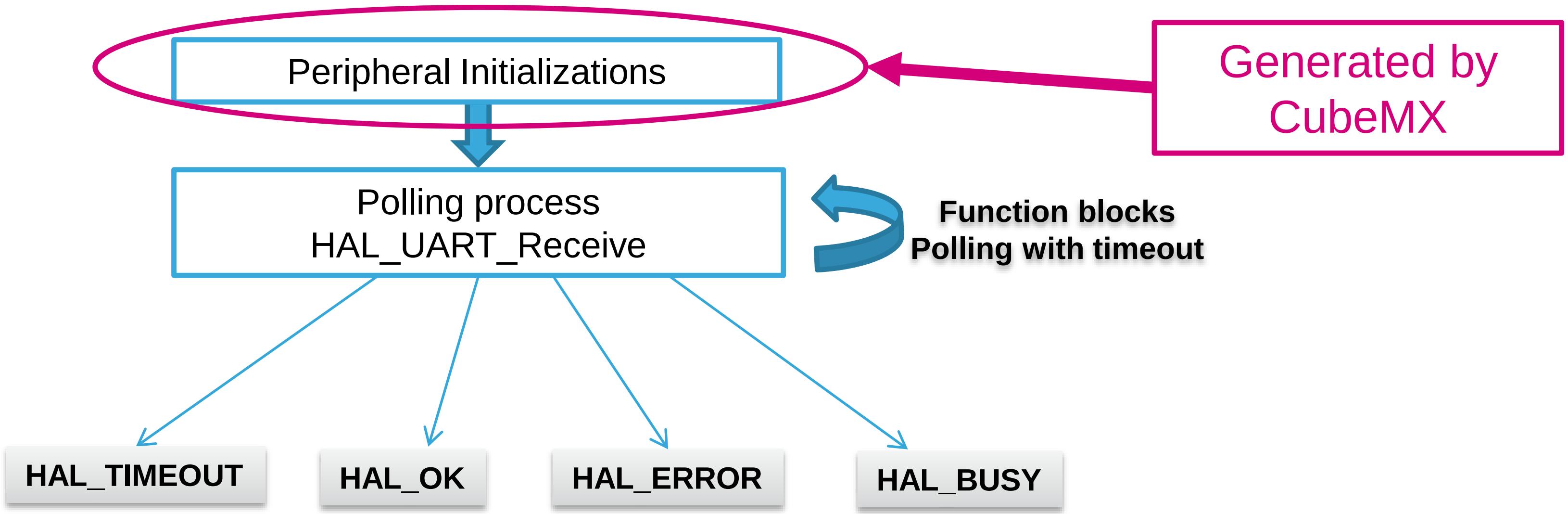
2.1.1 Simple UART communication

HAL Library transmit flow



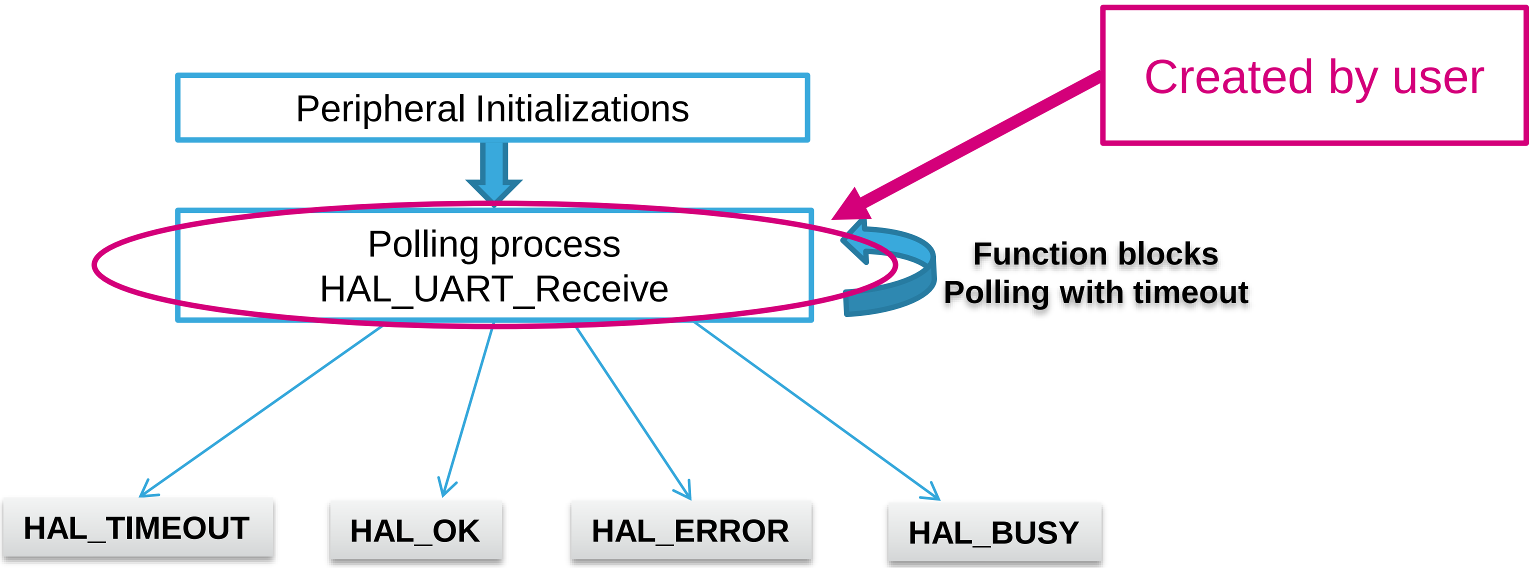
2.1.1 Simple UART communication

HAL Library receive flow



2.1.1 Simple UART communication

HAL Library receive flow



2.1.1 Simple UART communication

- 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);

2.1.1 Simple UART communication

- 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 */
```

2.1.1 Simple UART communication

- 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 */
```

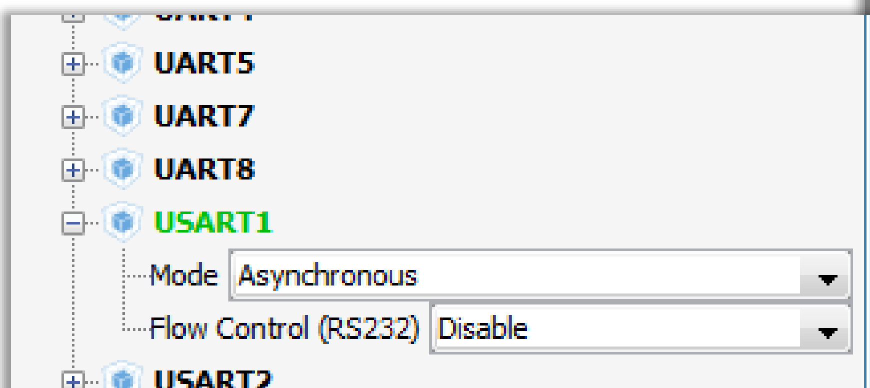
2.1.2 UART Interrupt lab

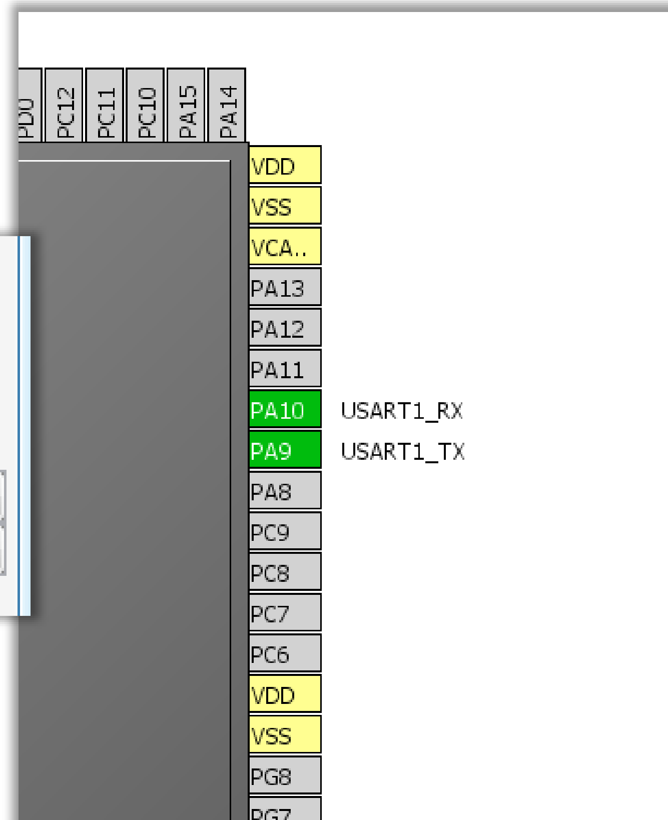
2.1.2 Use UART with interrupt

- 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

2.1.2 Use UART with interrupt

- 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





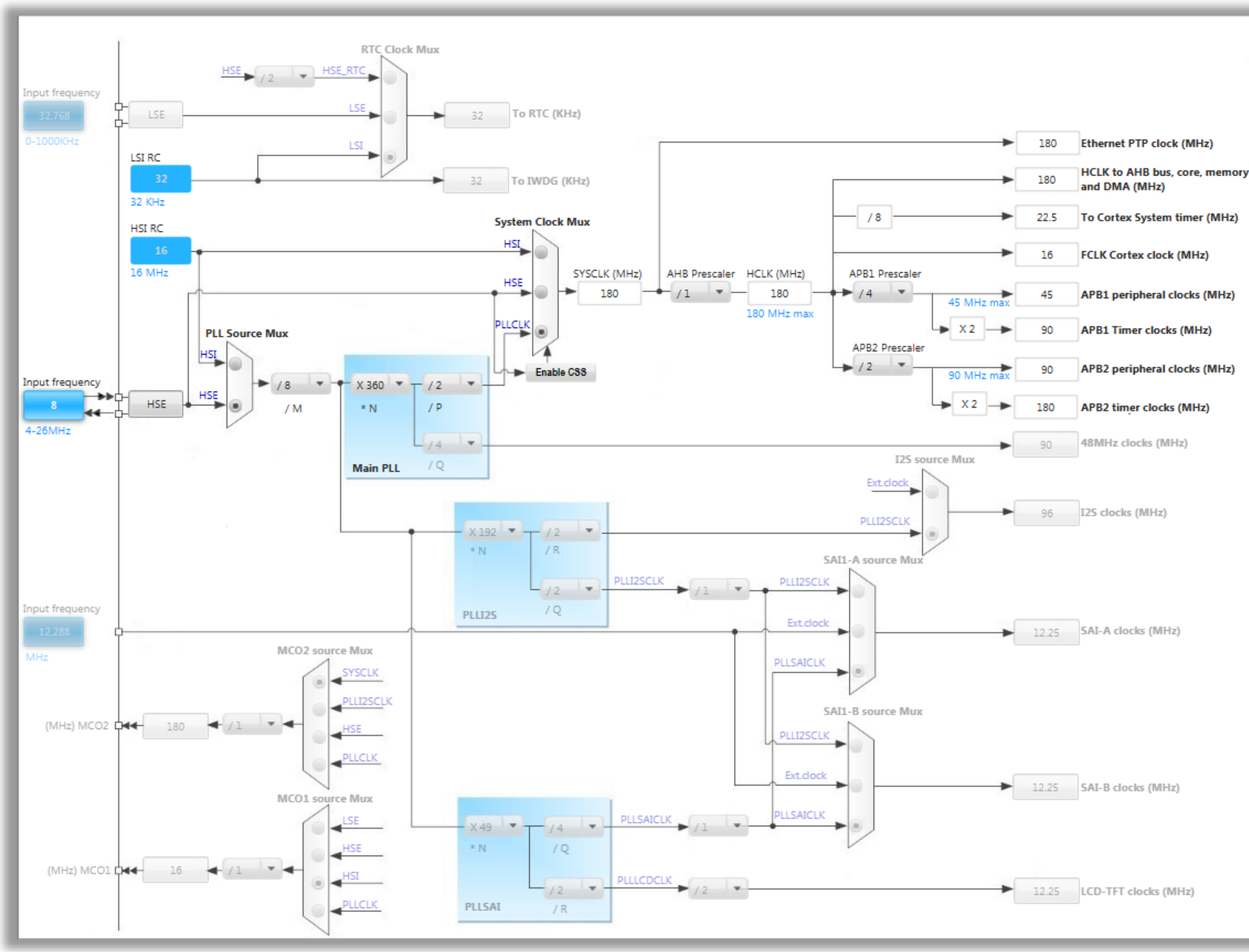
2.1.2 Use UART with interrupt

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

Hardware loopback

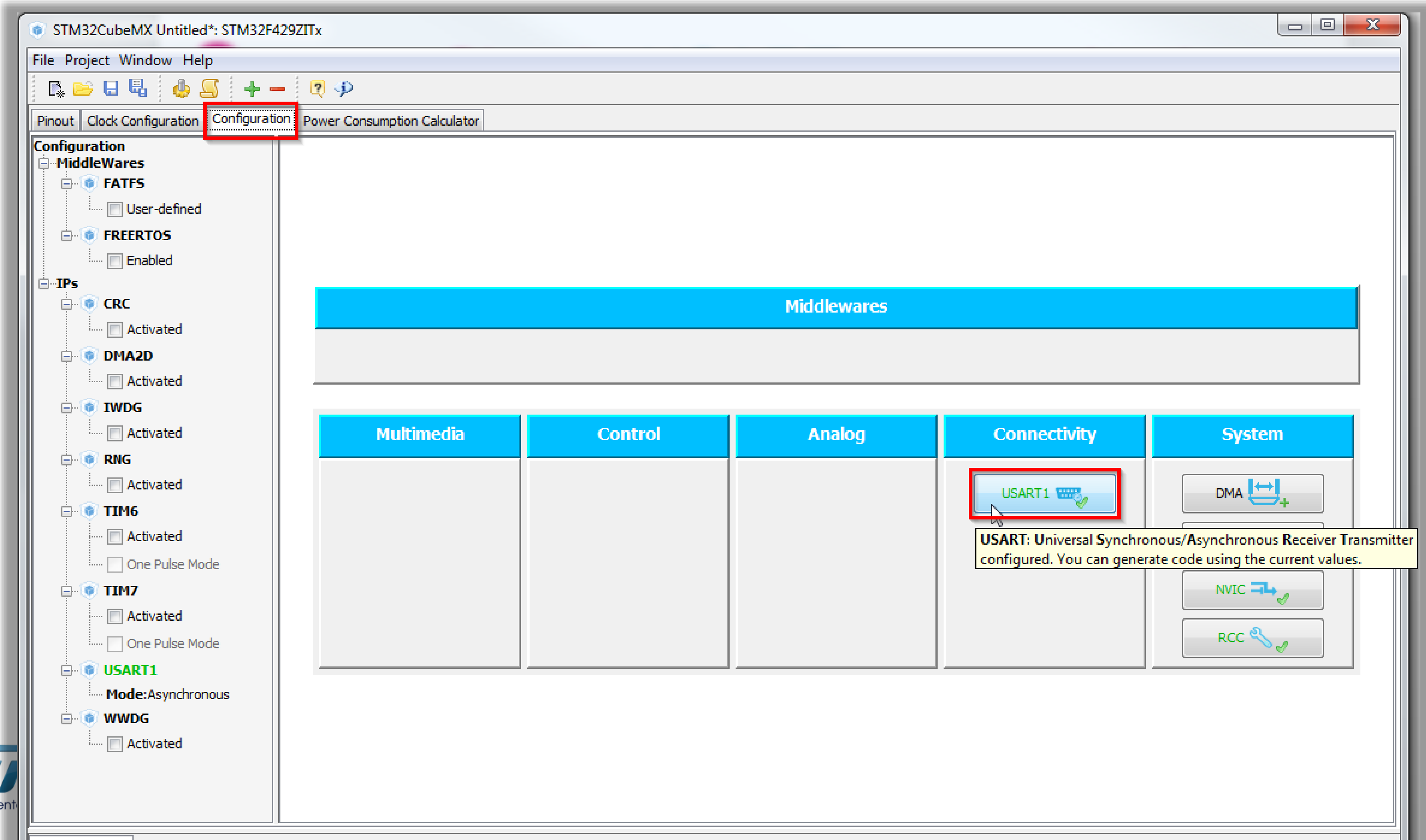
2.1.2 Use UART with interrupt

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



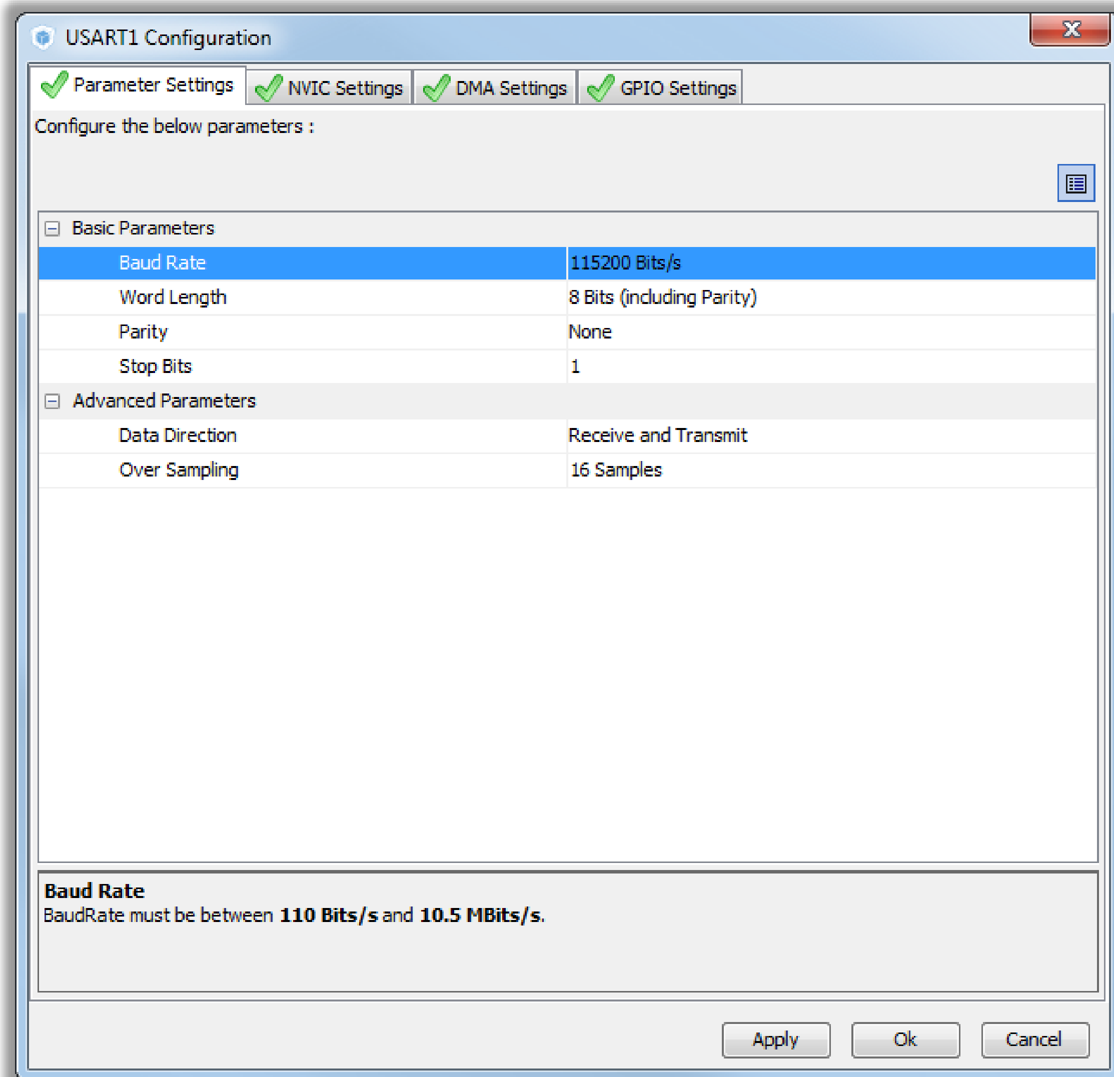
2.1.2 Use UART with interrupt

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



2.1.2 Use UART with interrupt

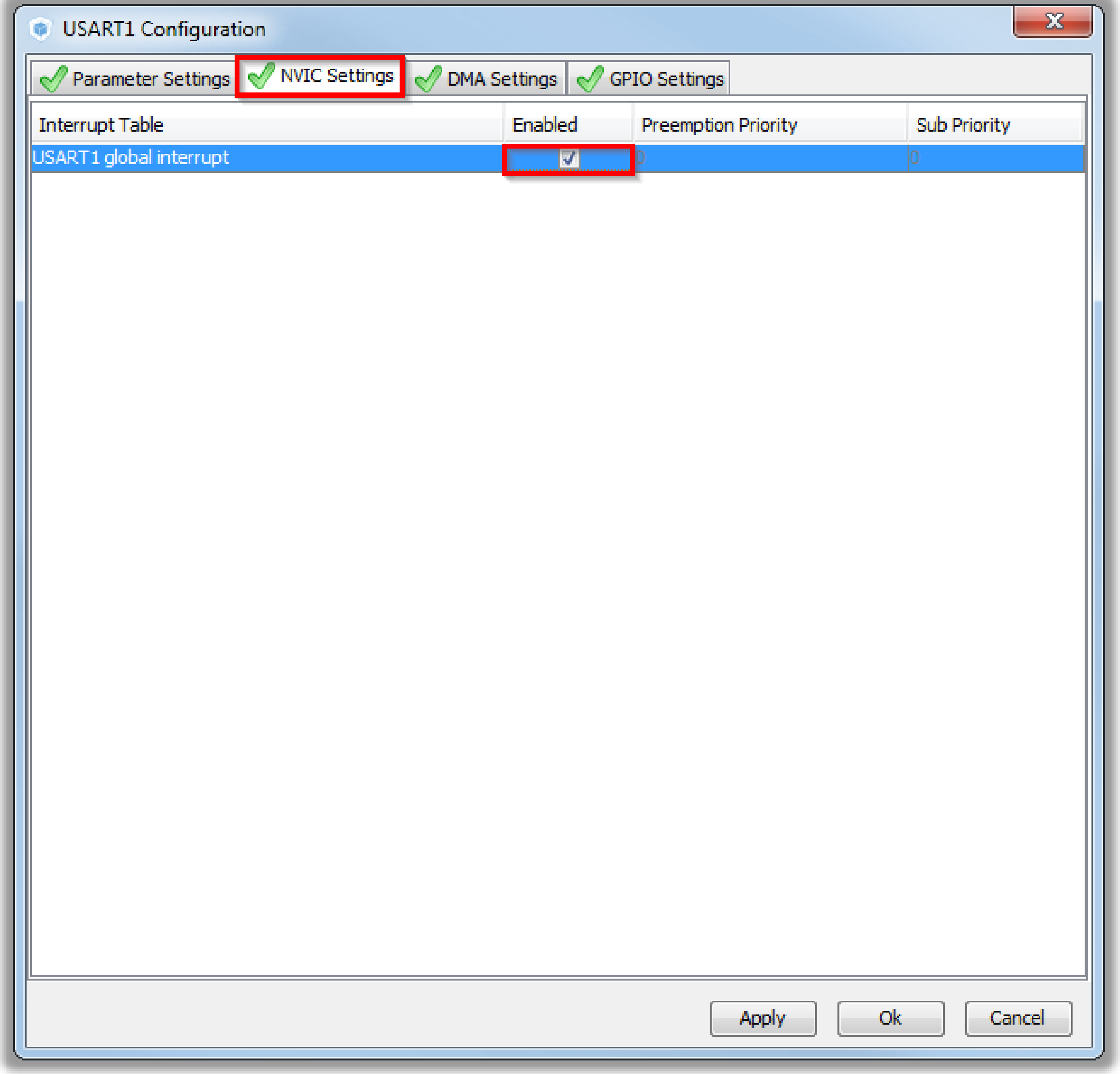
- CubeMX UART configuration check:
  - BaudRate
  - World length
  - Parity
  - Stop bits
  - Data direction
  - Oversampling



life.augmented

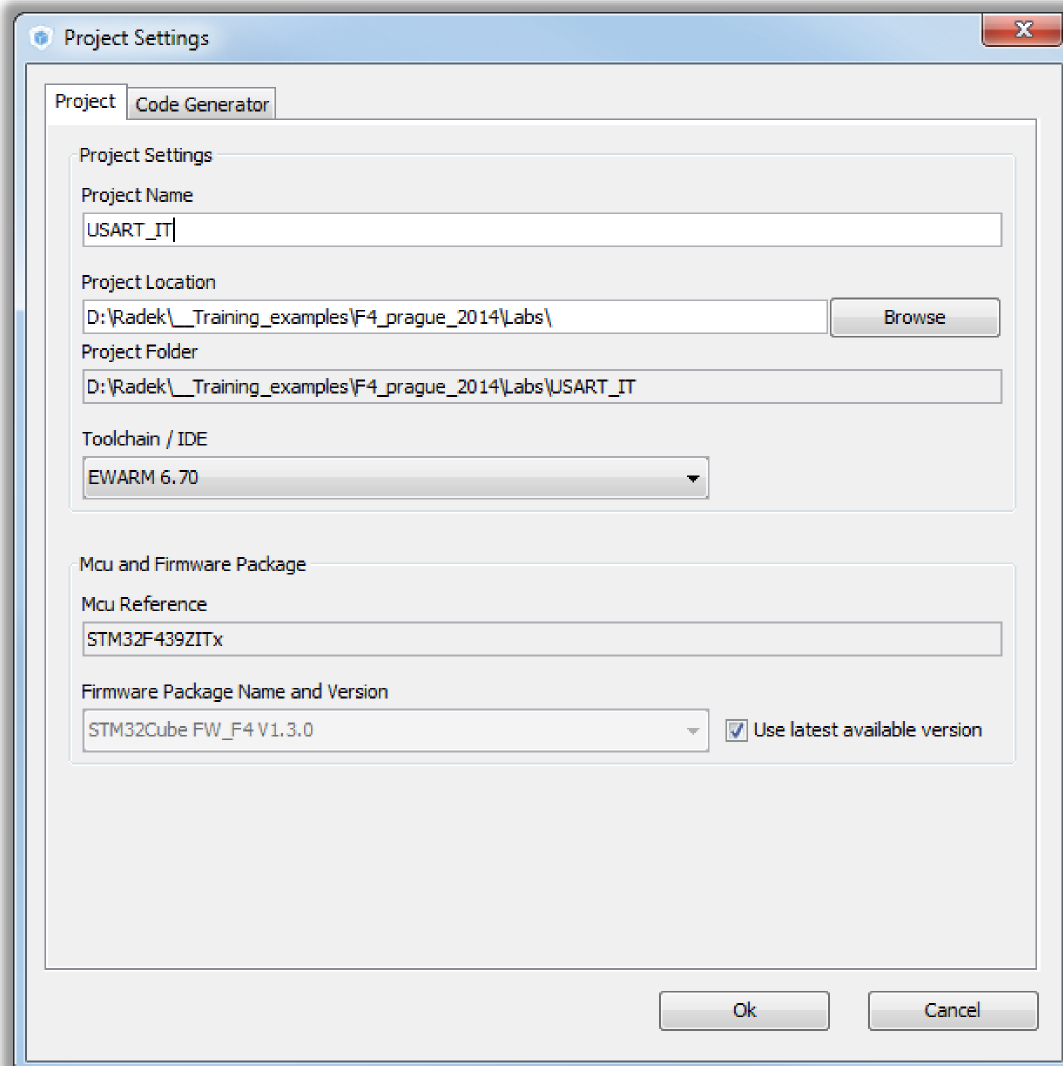
2.1.2 Use UART with interrupt

- CubeMX USART configuration NVIC settings
  - TAB>NVIC Settings
  - Enable interrupts
  - OK



2.1.2 Use UART with interrupt

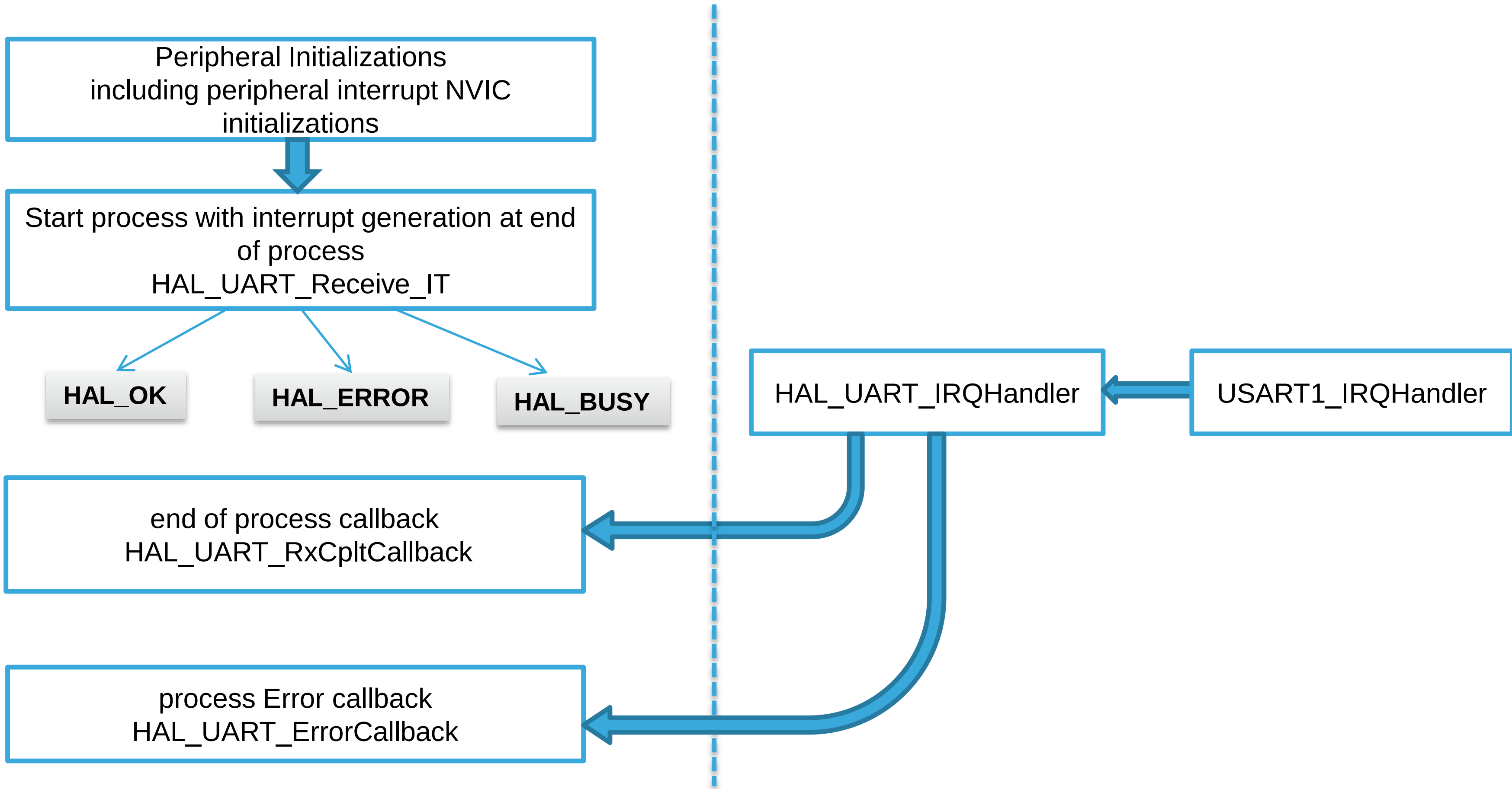
- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



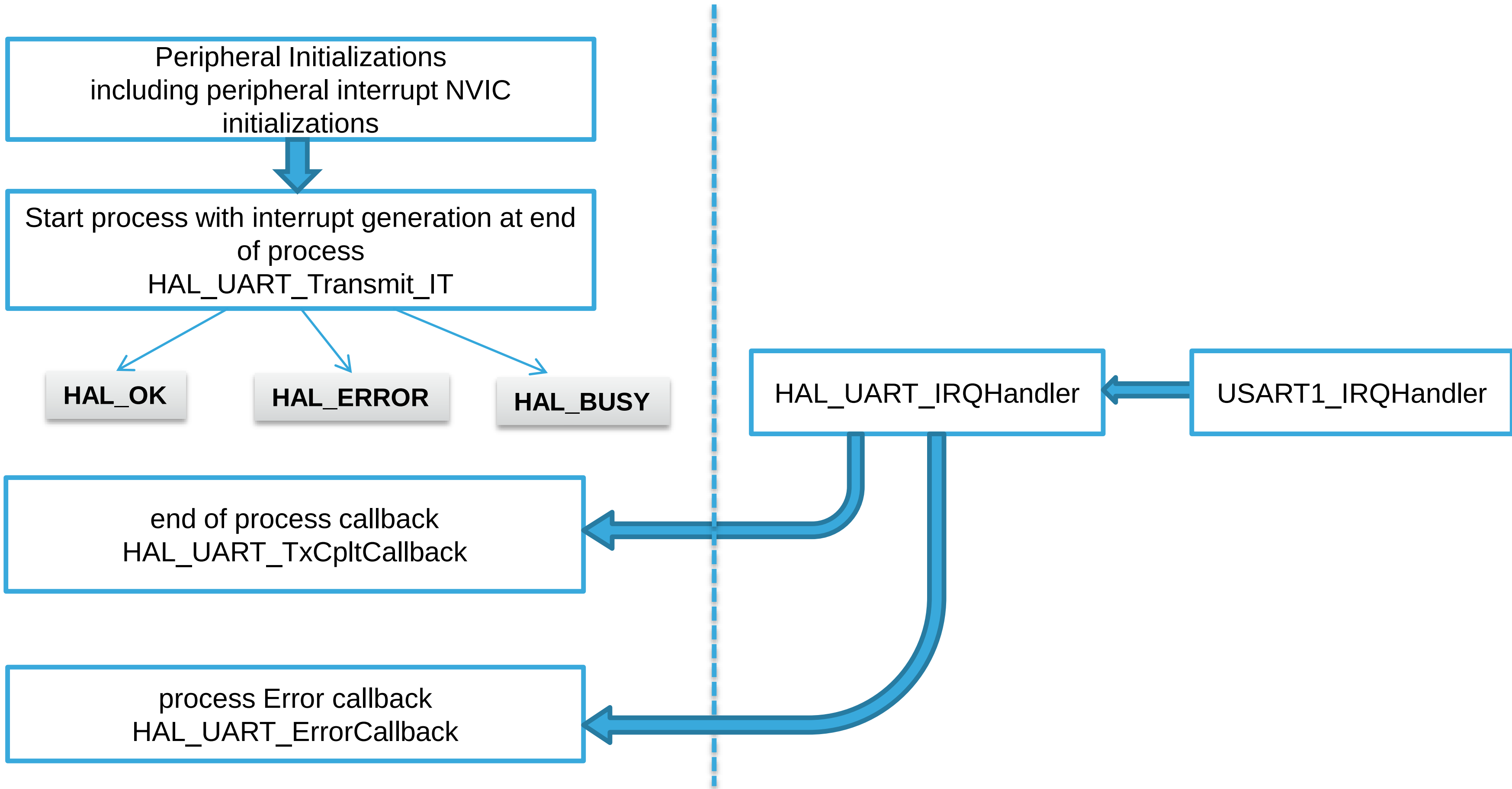
life.augmented

2.1.2 Use UART with interrupt

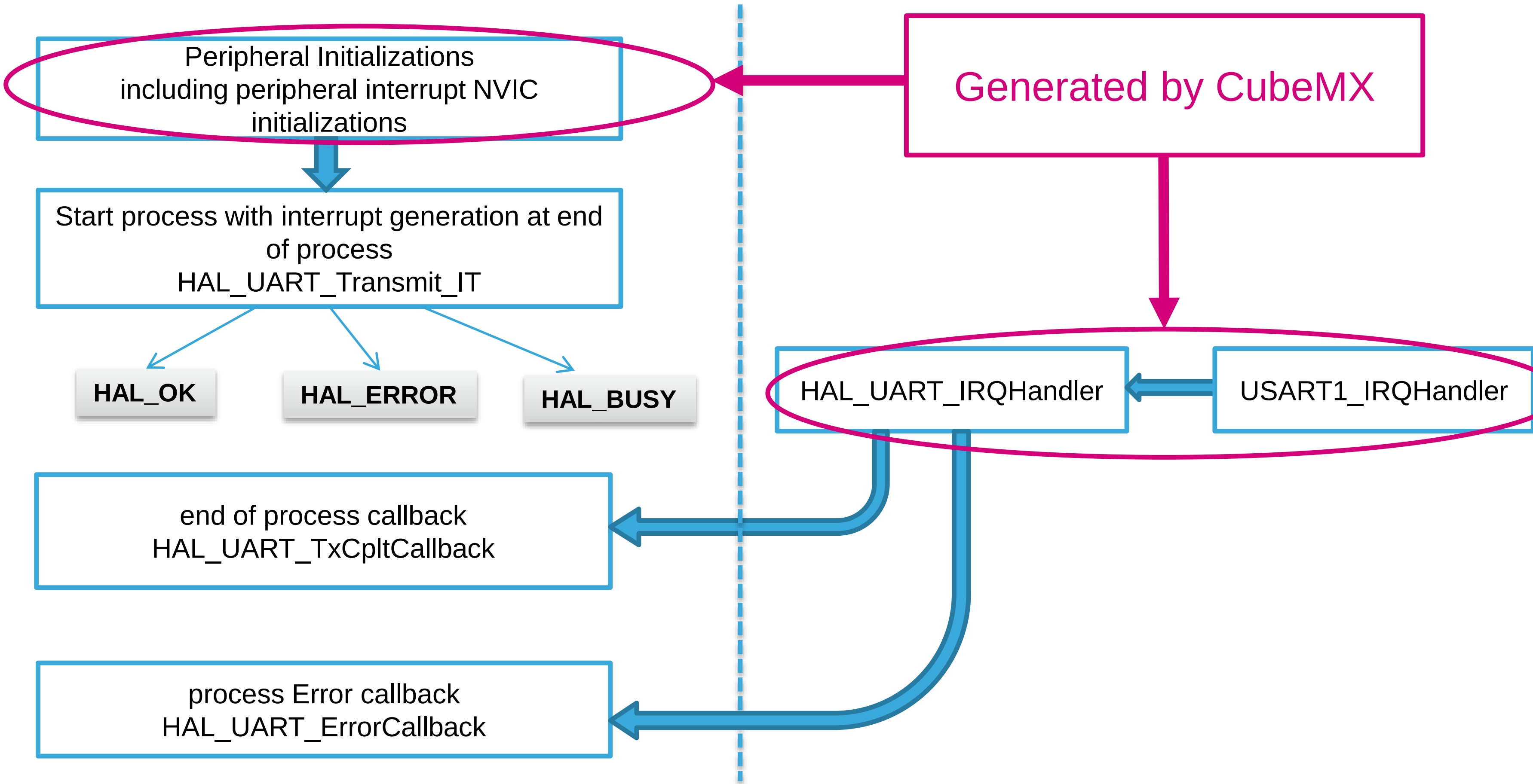
HAL Library UART with IT receive flow



HAL Library UART with IT transmit flow

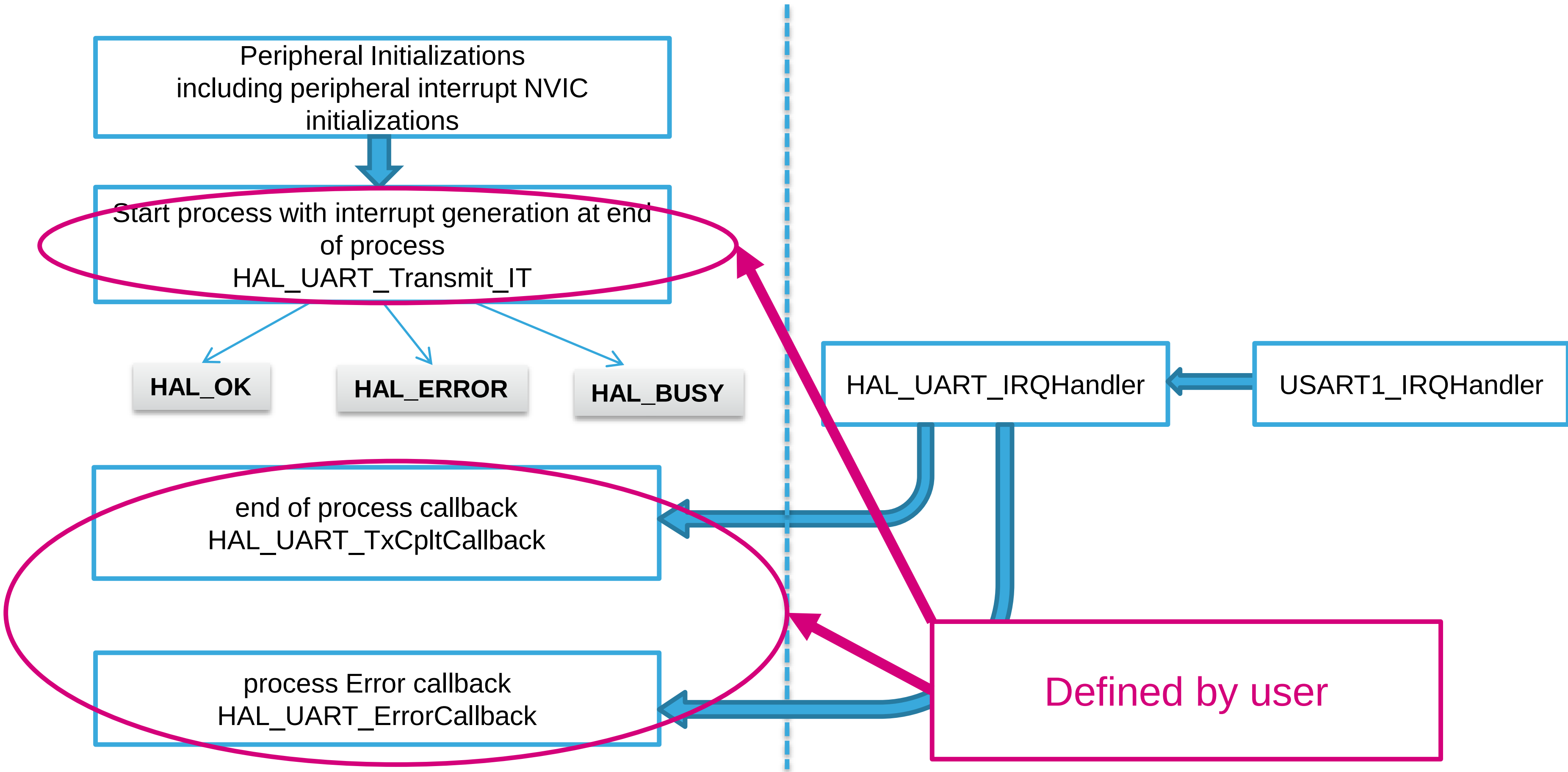


HAL Library UART with IT transmit flow



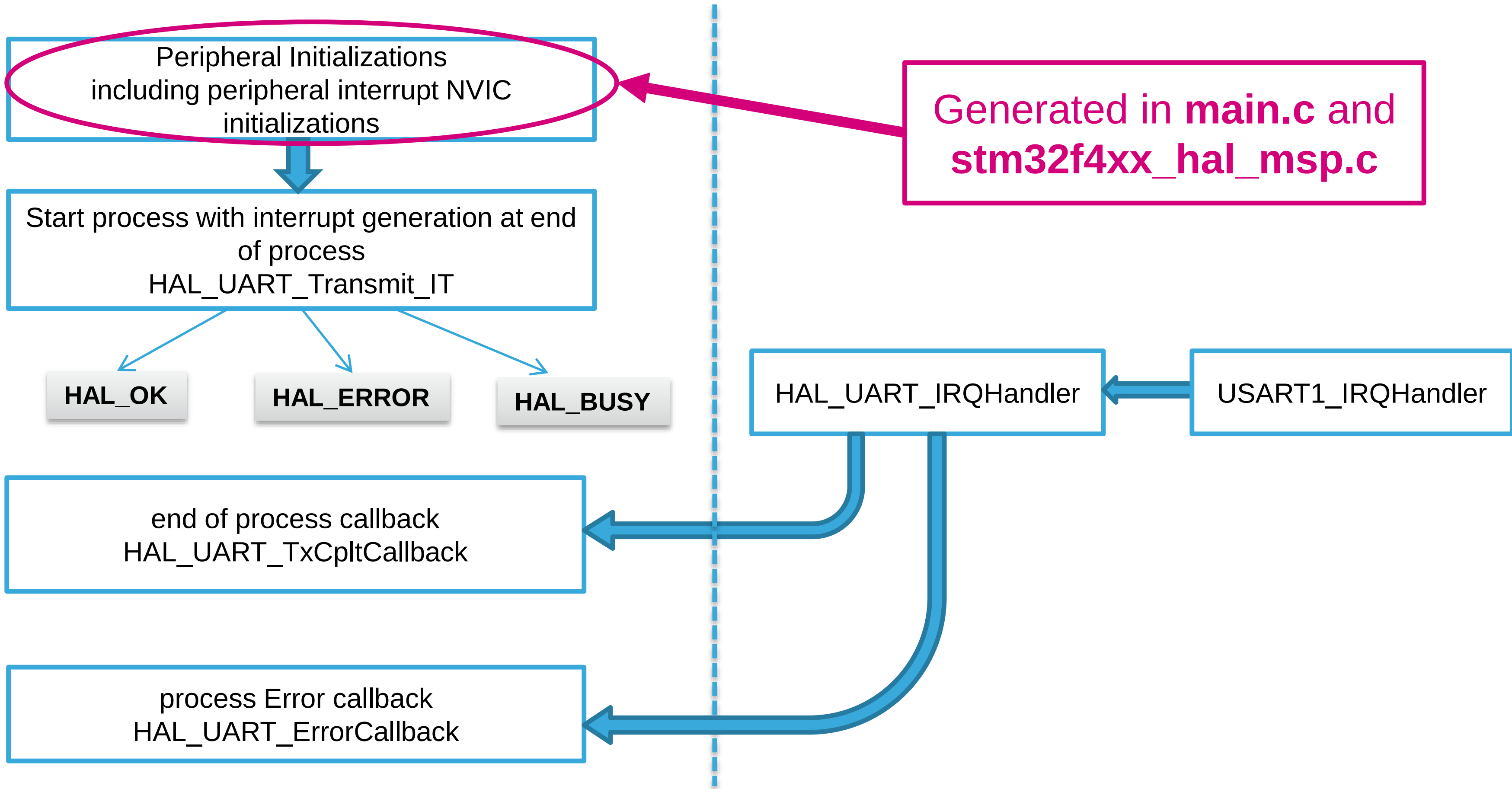
life.augmented

HAL Library UART with IT receive flow



2.1.2 Use UART with interrupt

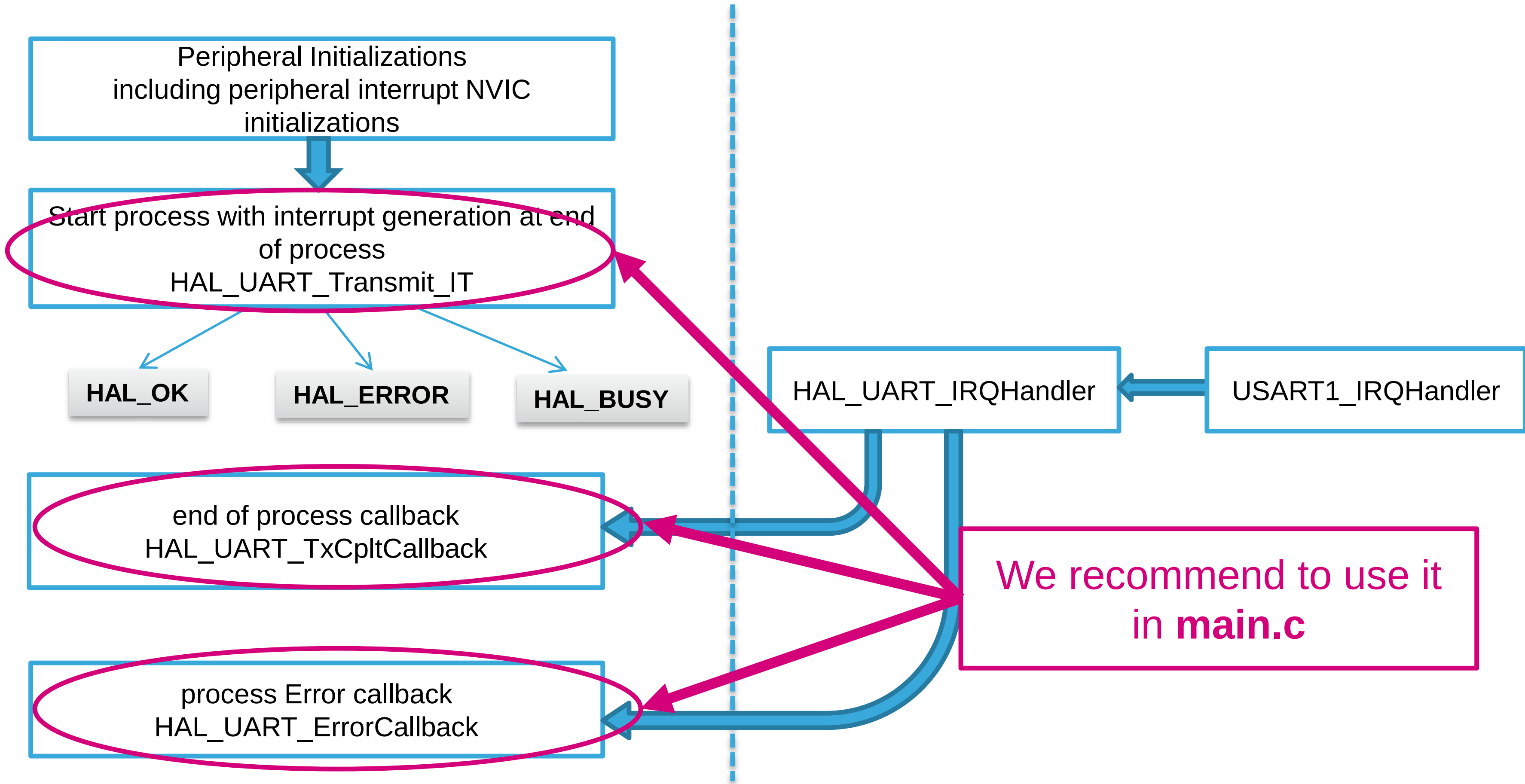
HAL Library UART with IT receive flow



life.augmented

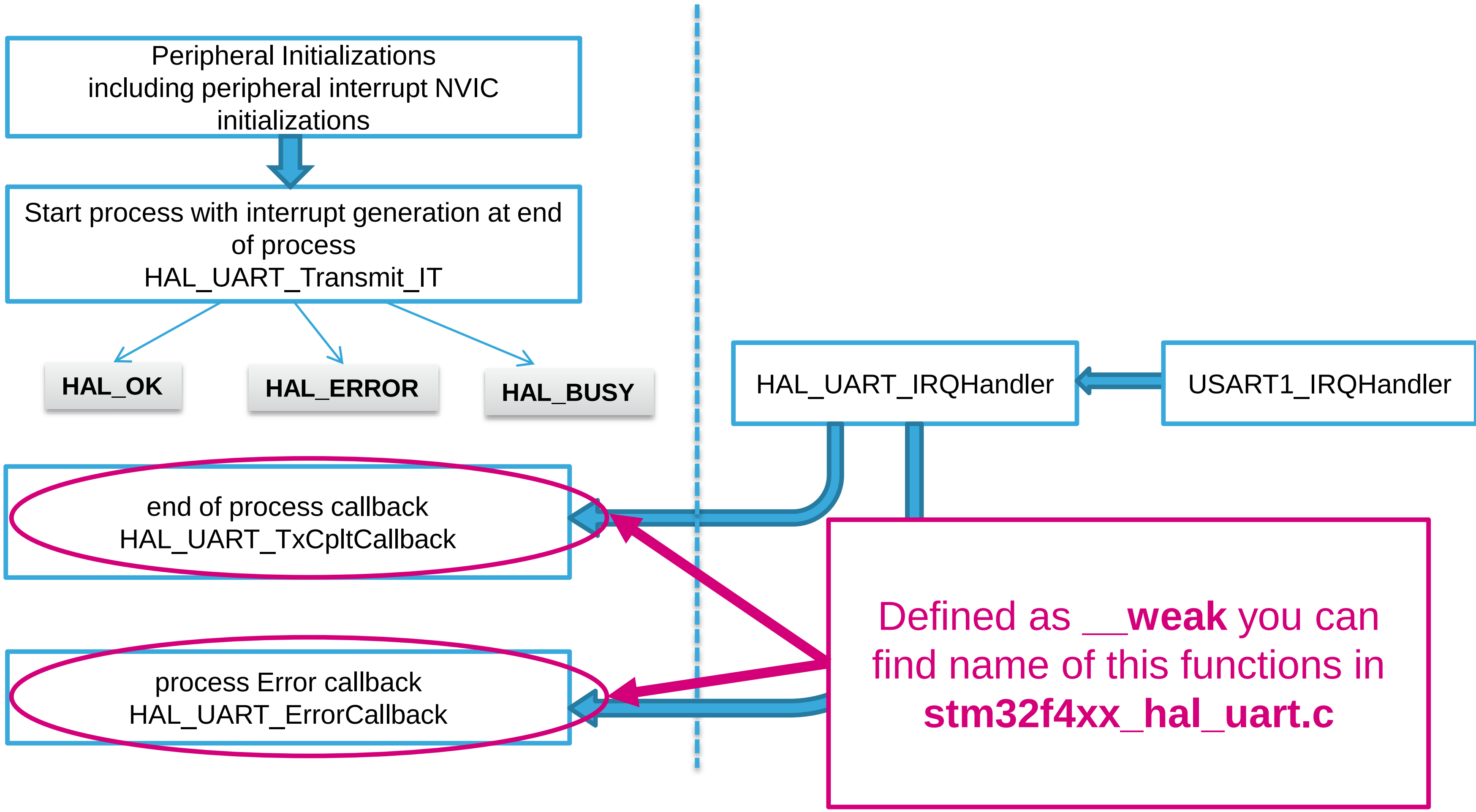
Use UART with interrupt

HAL Library UART with IT receive flow



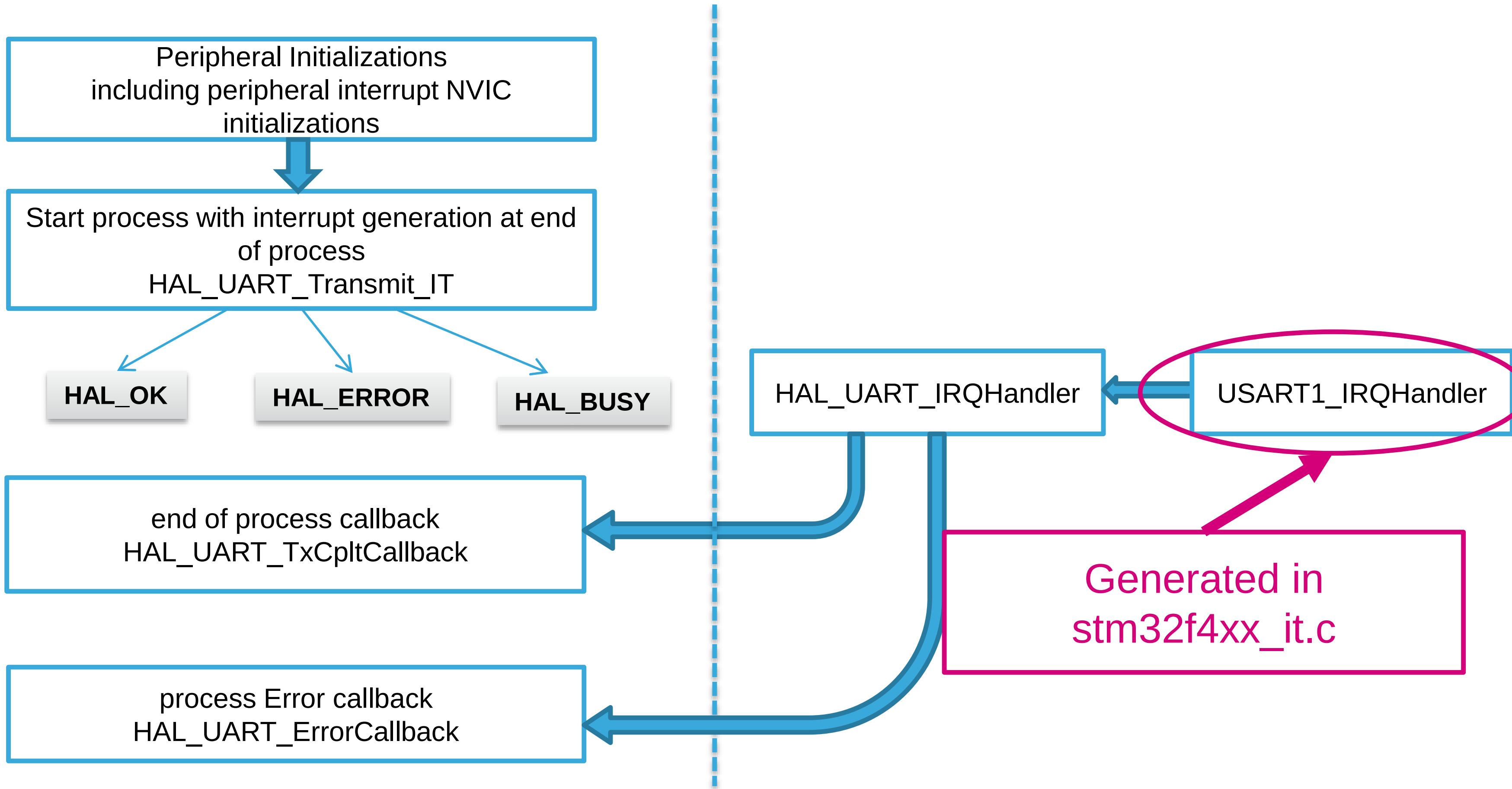
2.1.2 Use UART with interrupt

HAL Library UART with IT receive flow



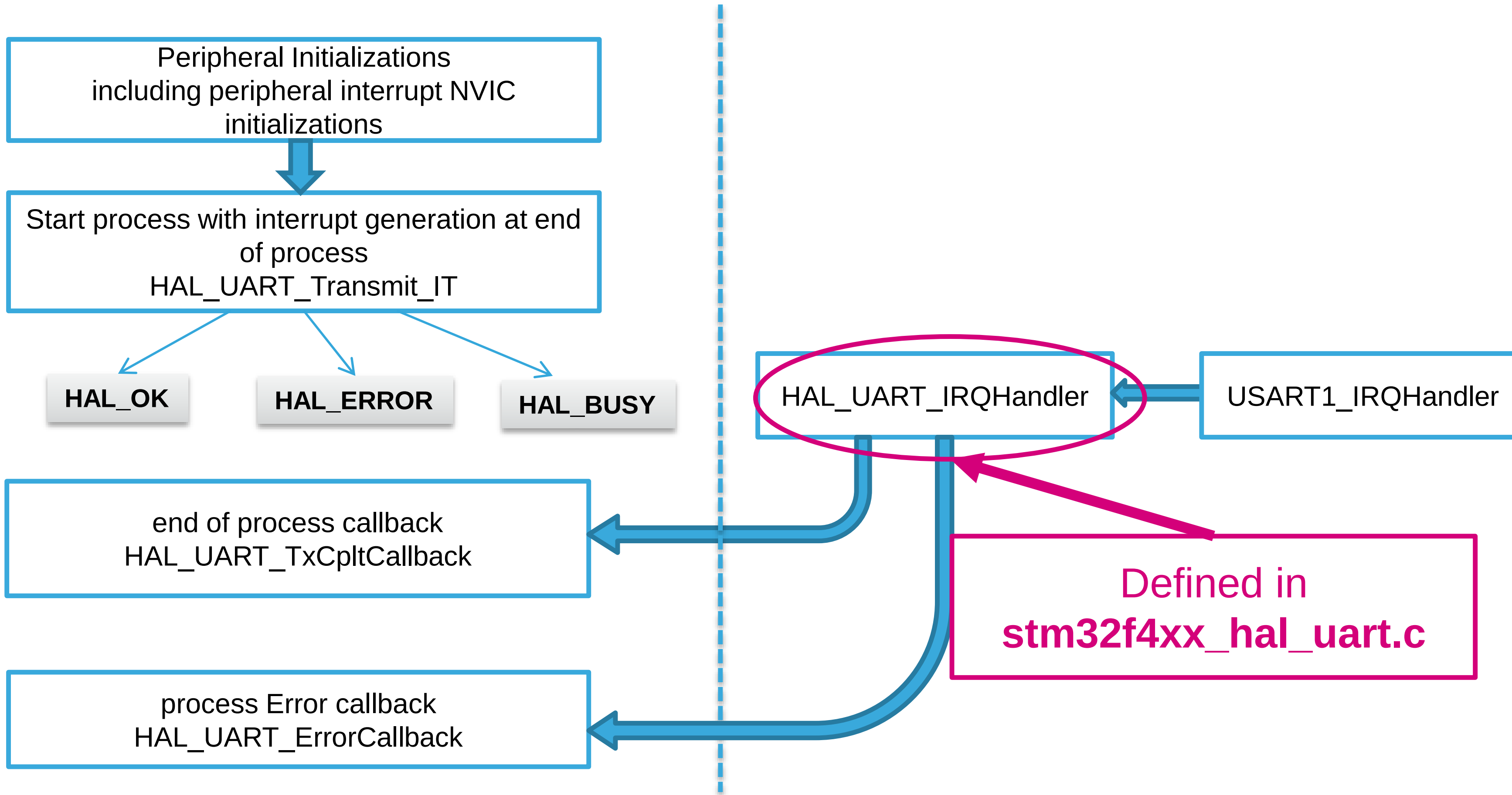
life.augmented

HAL Library UART with IT receive flow



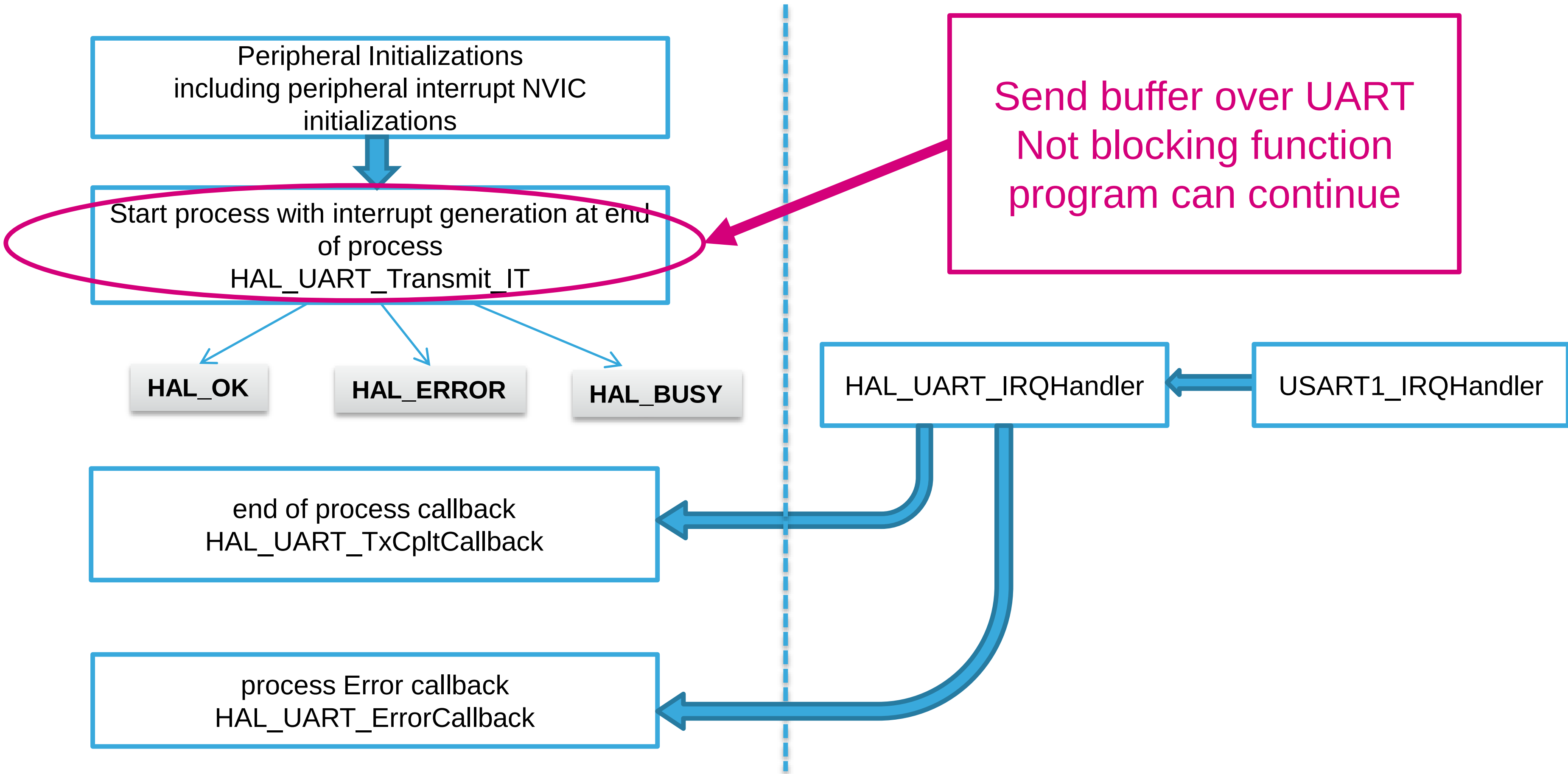
2.1.2 Use UART with interrupt

HAL Library UART with IT receive flow



2.1.2 Use UART with interrupt

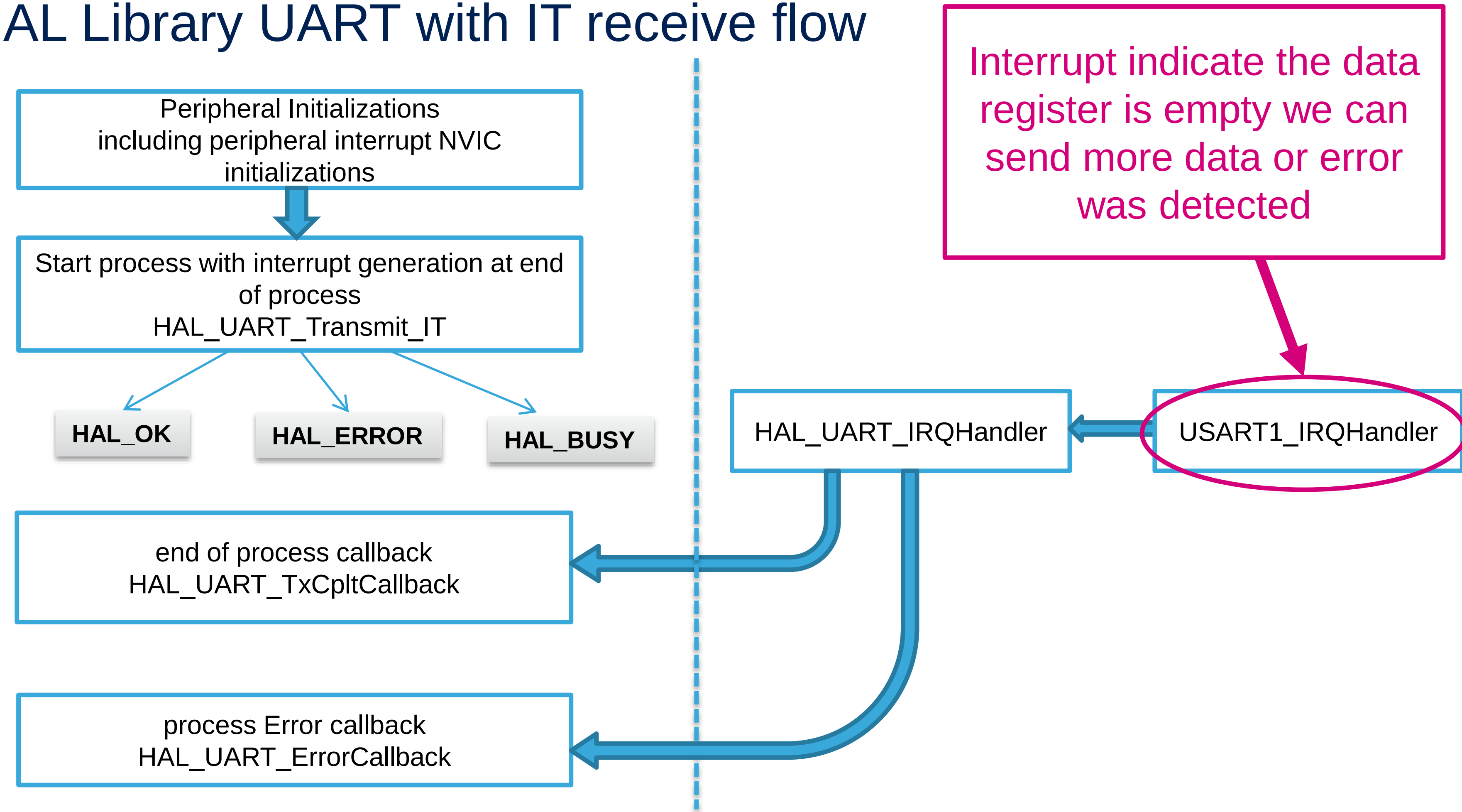
HAL Library UART with IT receive flow



life.augmented

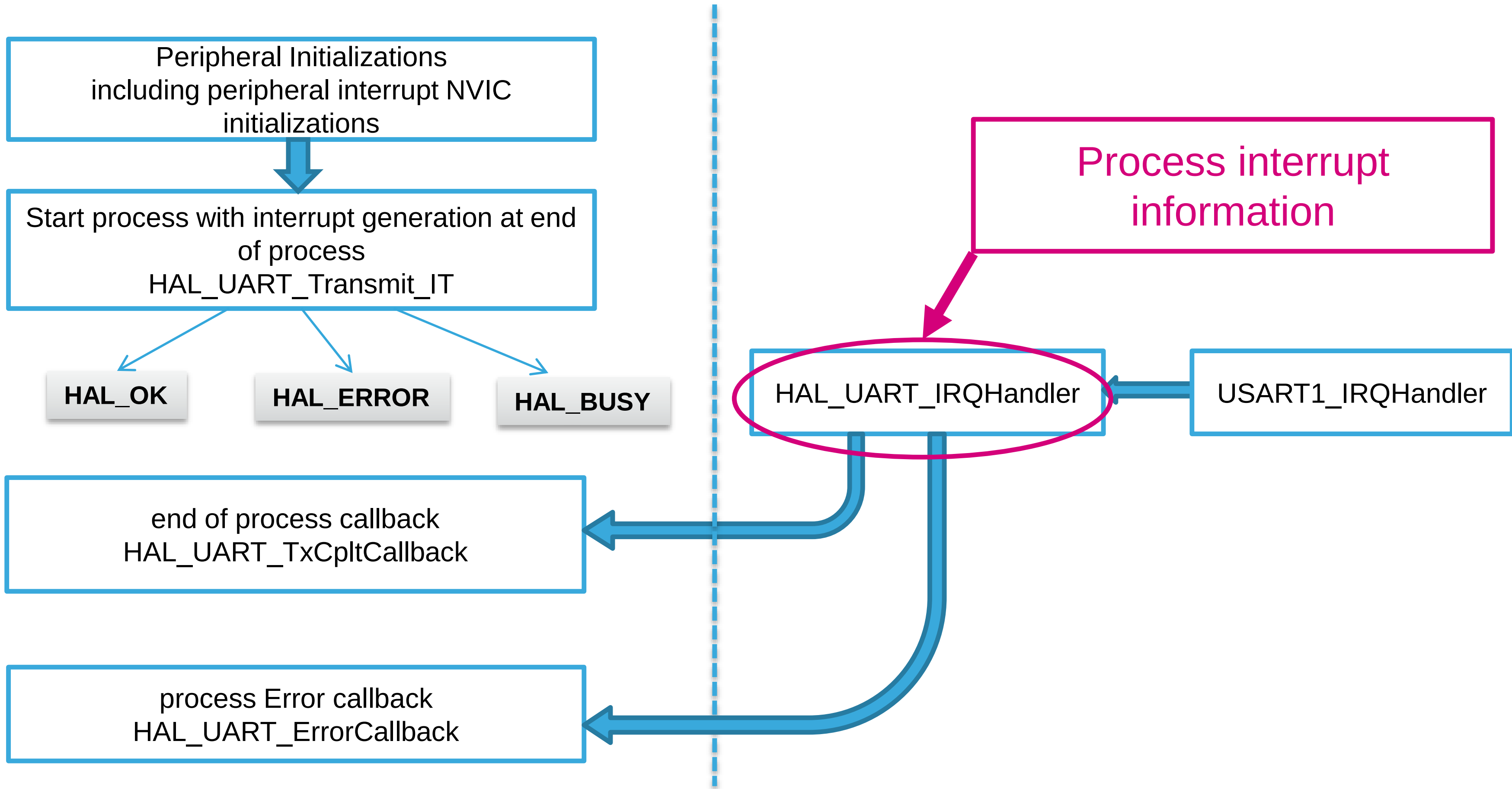
Use UART with interrupt

HAL Library UART with IT receive flow



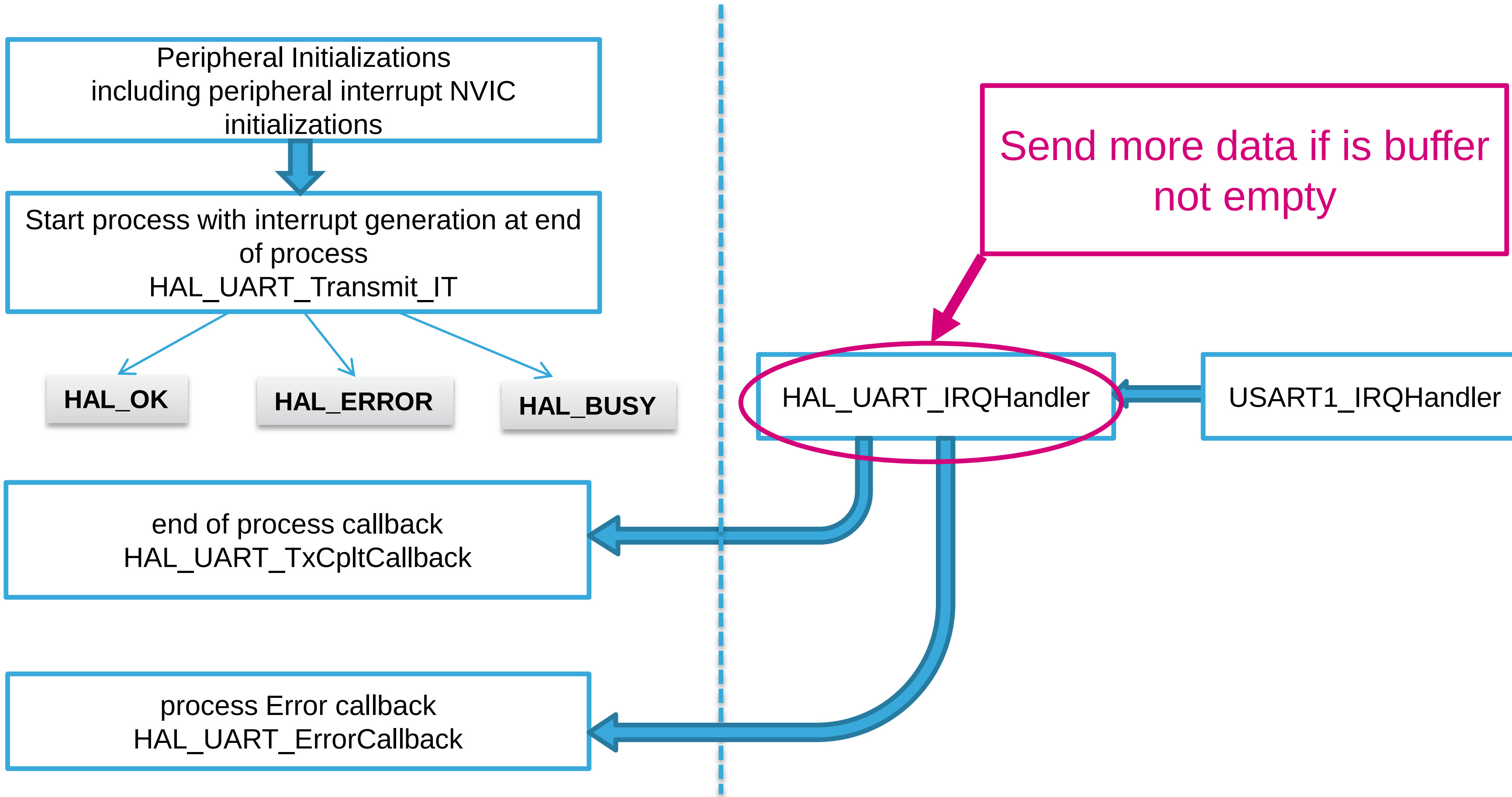
2.1.2 Use UART with interrupt

HAL Library UART with IT receive flow



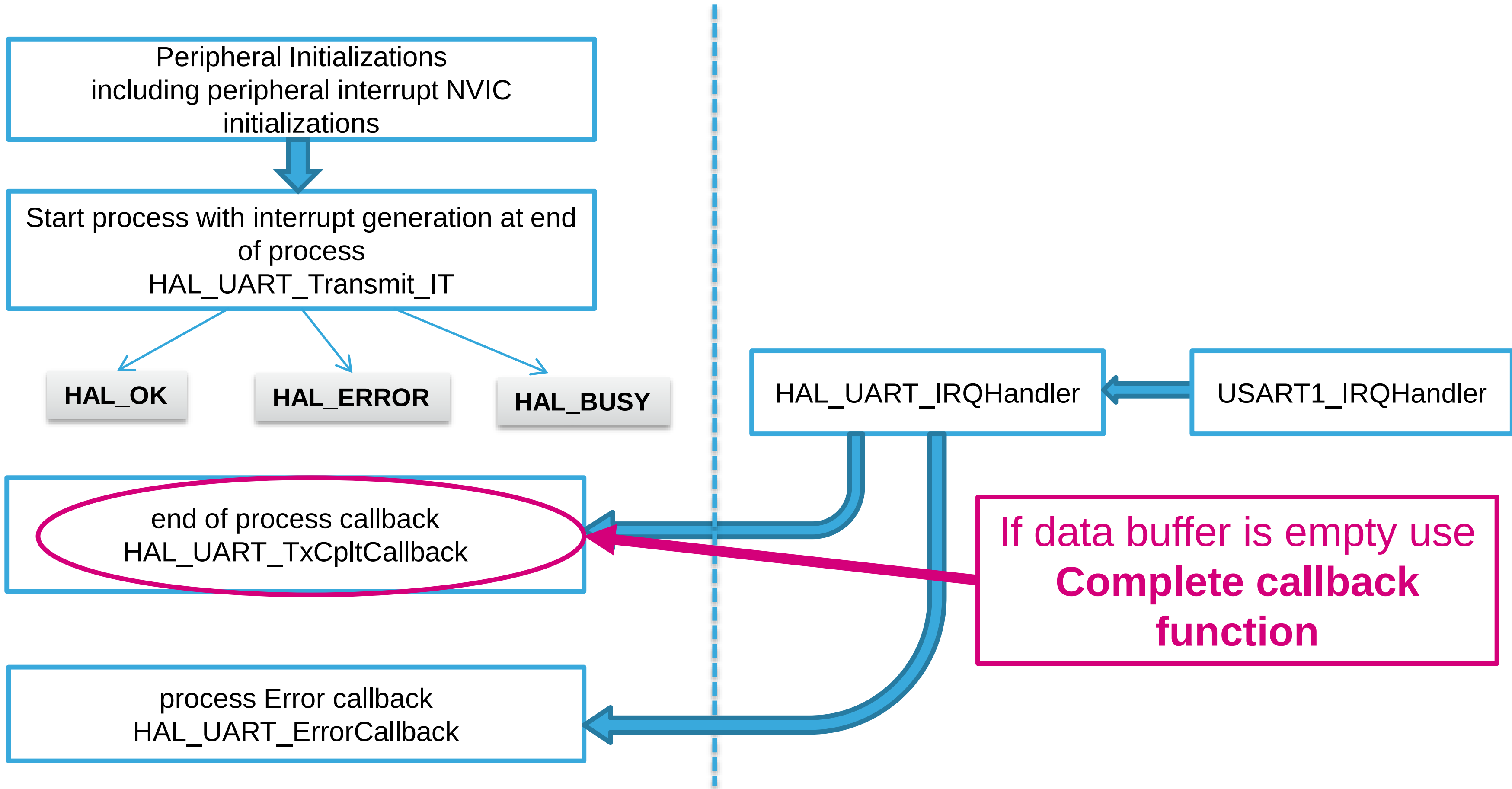
2.1.2 Use UART with interrupt

HAL Library UART with IT receive flow



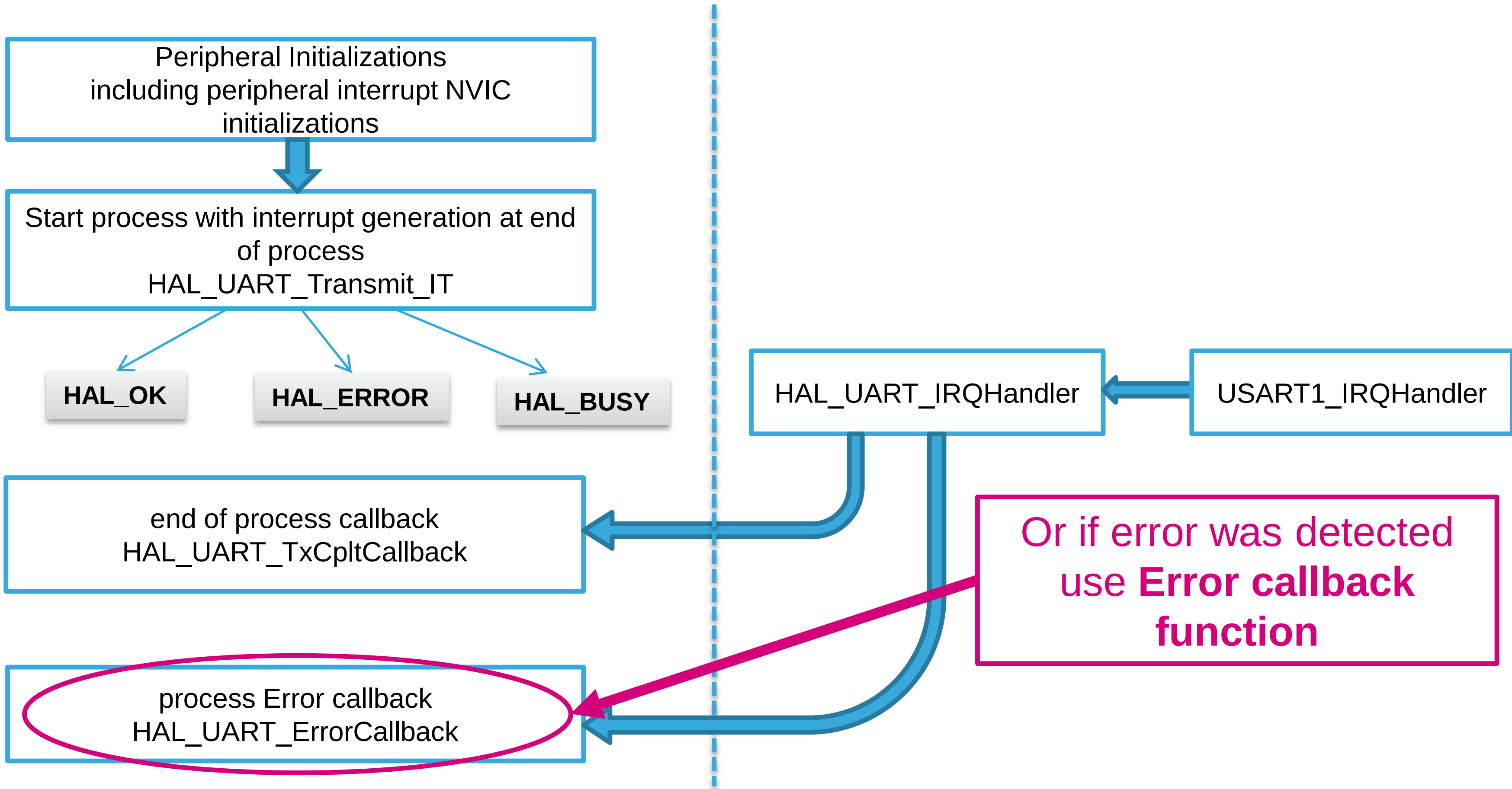
2.1.2 Use UART with interrupt

HAL Library UART with IT receive flow

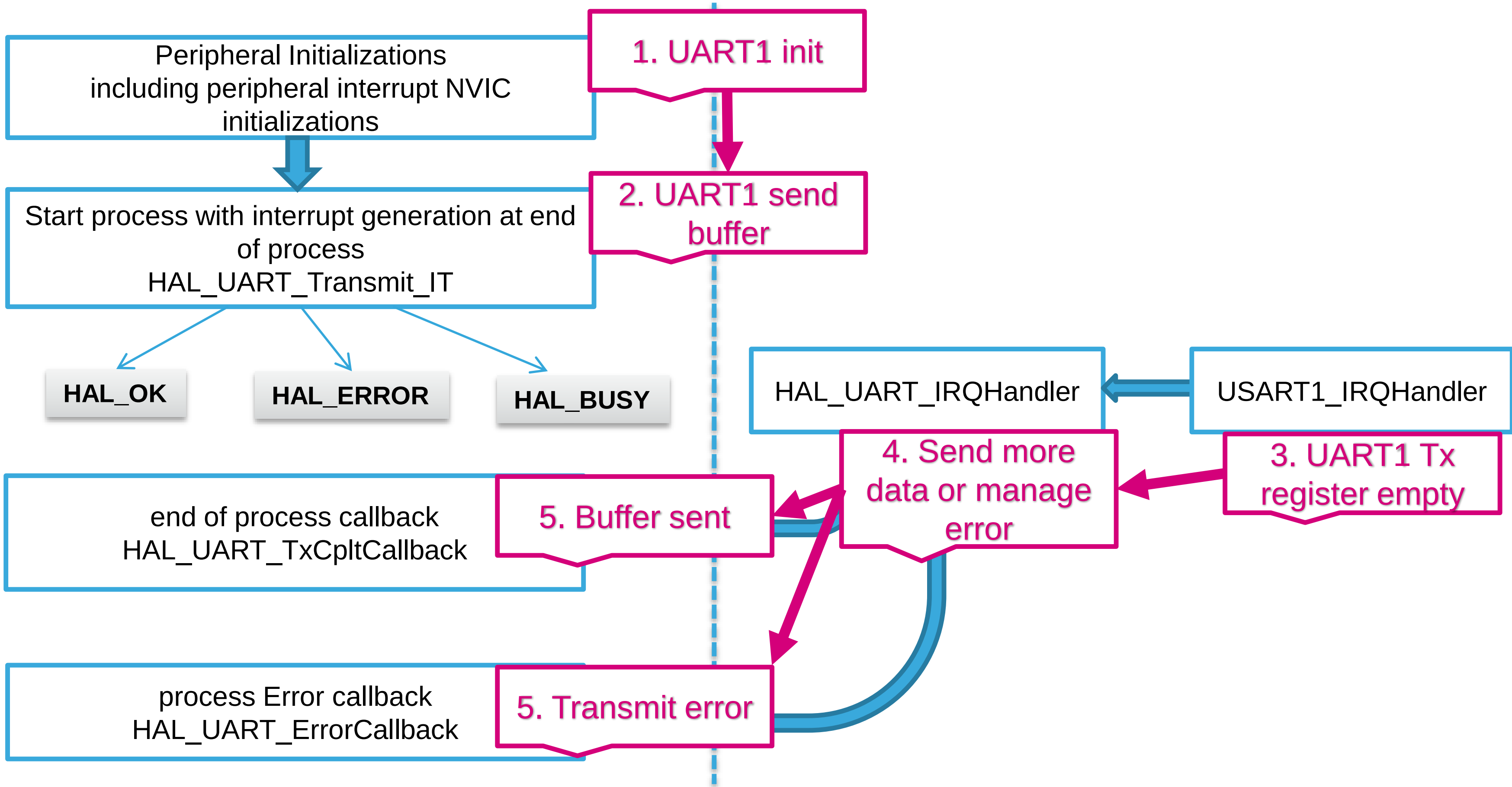


life.augmented

HAL Library UART with IT receive flow



HAL Library UART with IT receive flow



2.1.2 Use UART with interrupt

- 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);

2.1.2 Use UART with interrupt

- 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 */
```

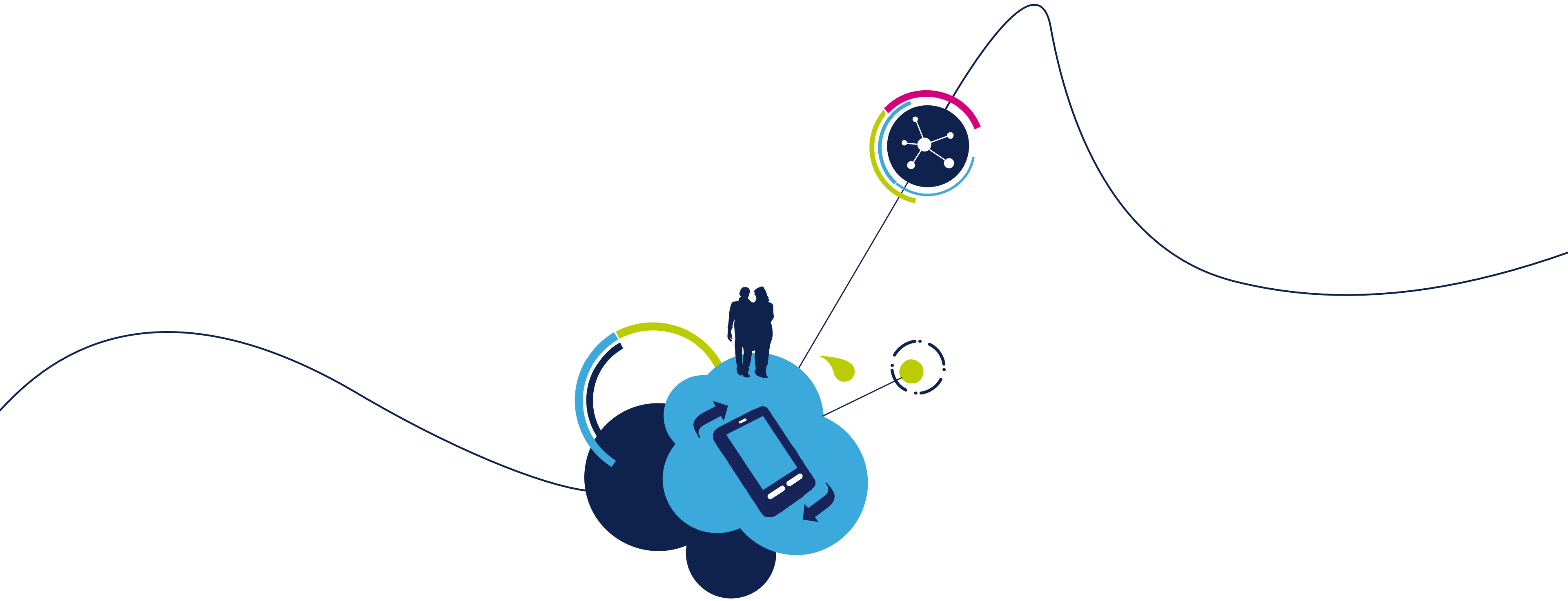
- Sending and receiving methods

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

2.1.2 Use UART with interrupt

- Complete callback check
  - We can put brakepoints 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 */
```



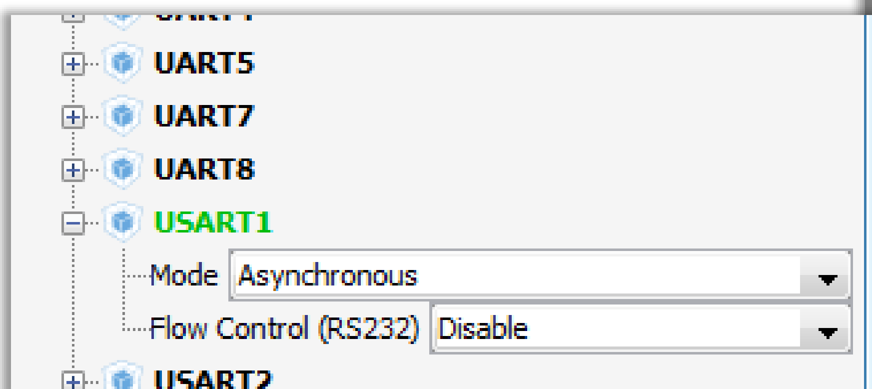
2.1.3 UART DMA lab

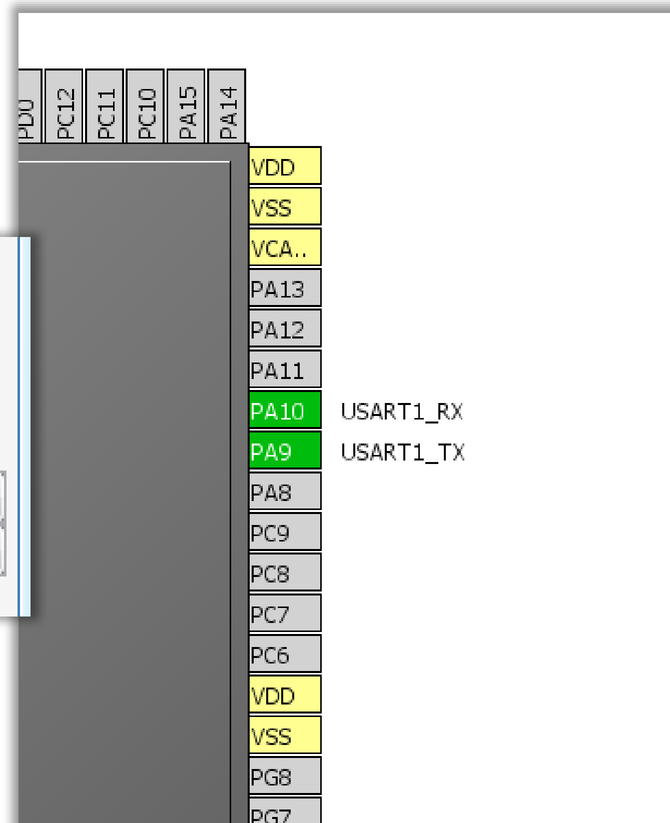
2.1.3 Use UART with DMA transfer

- 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

2.1.3 Use UART with DMA transfer

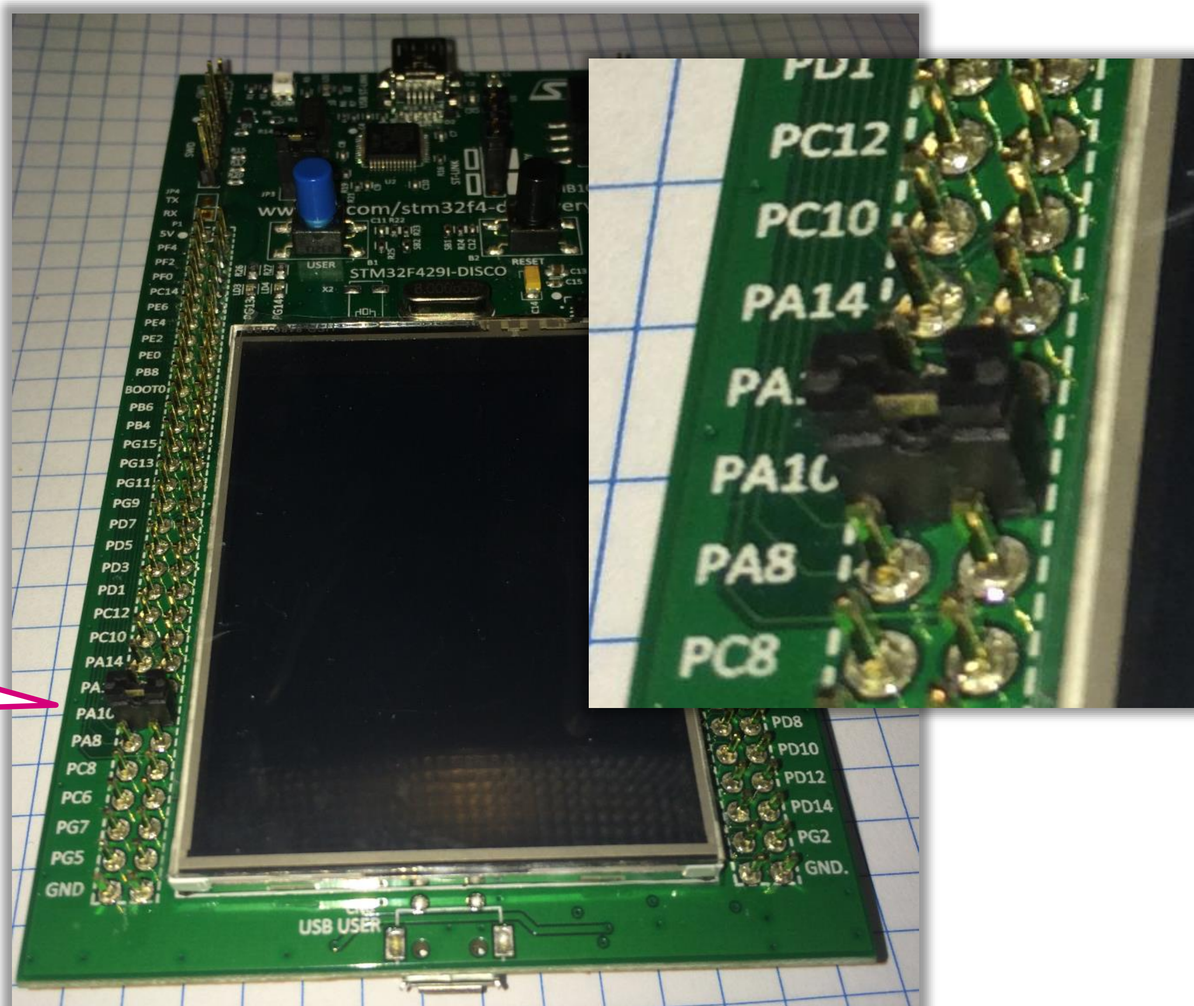
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Pin selection
  - It will be same as previous lab we use again PA9 and PA10





2.1.3 Use UART with DMA transfer

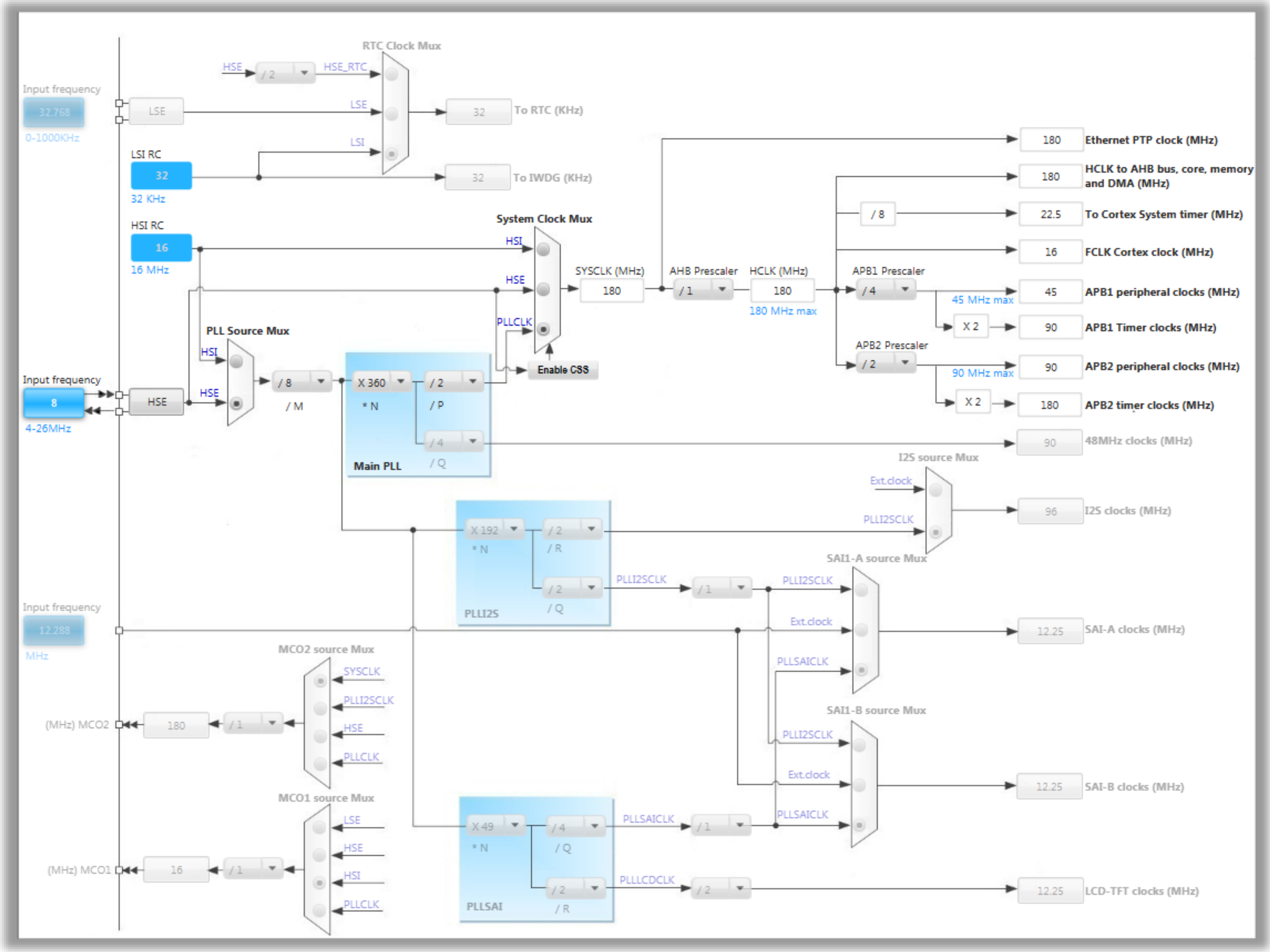
- Hardware preparation
  - We connect selected pins together by jumper, this help us to create loopback on UART



Hardware loopback

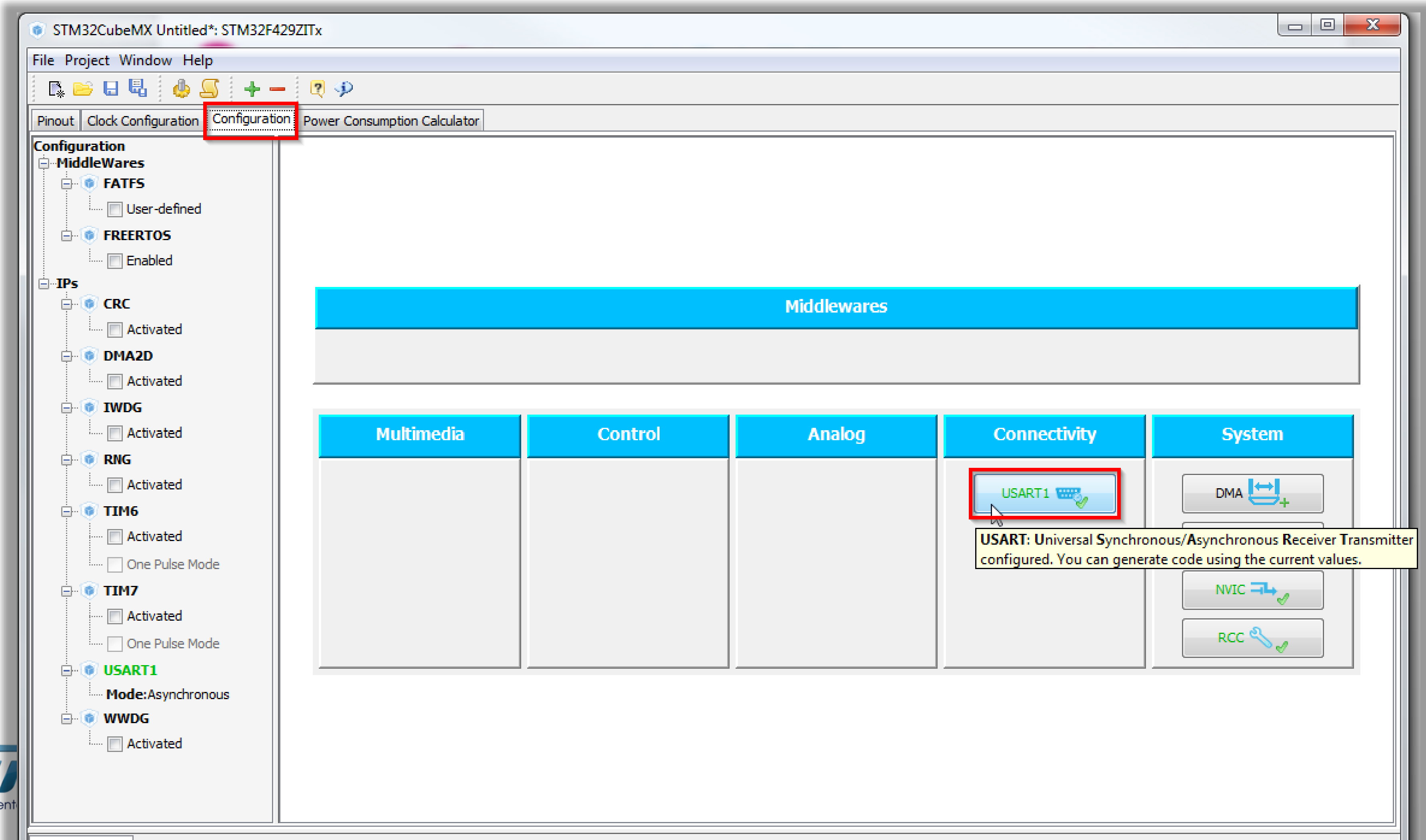
2.1.3 Use UART with DMA transfer

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



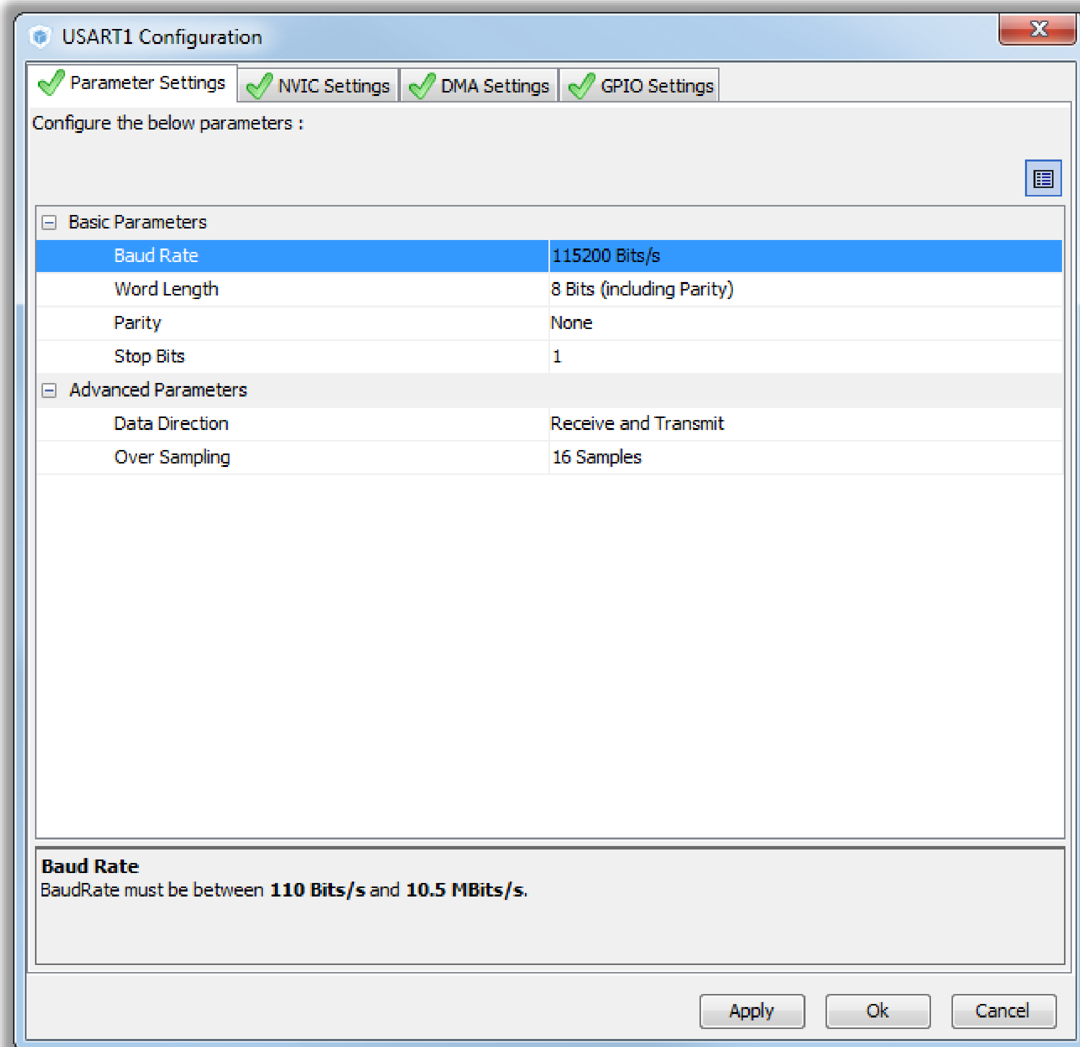
2.1.3 Use UART with DMA transfer

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



2.1.3 Use UART with DMA transfer

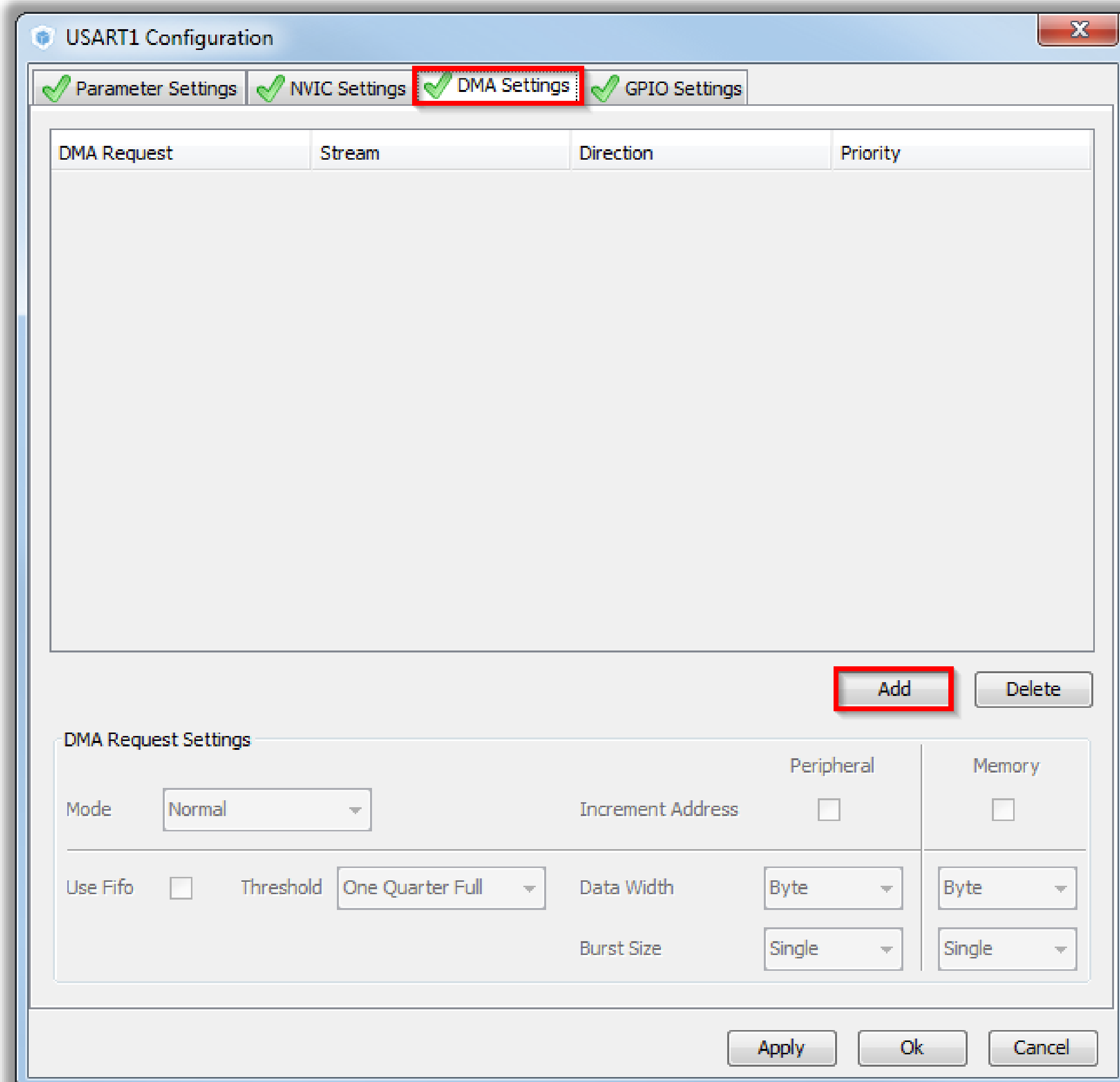
- CubeMX USART configuration check:
  - BaudRate
  - World length
  - Parity
  - Stop bits
  - Data direction
  - Oversampling



life.augmented

2.1.3 Use UART with DMA transfer

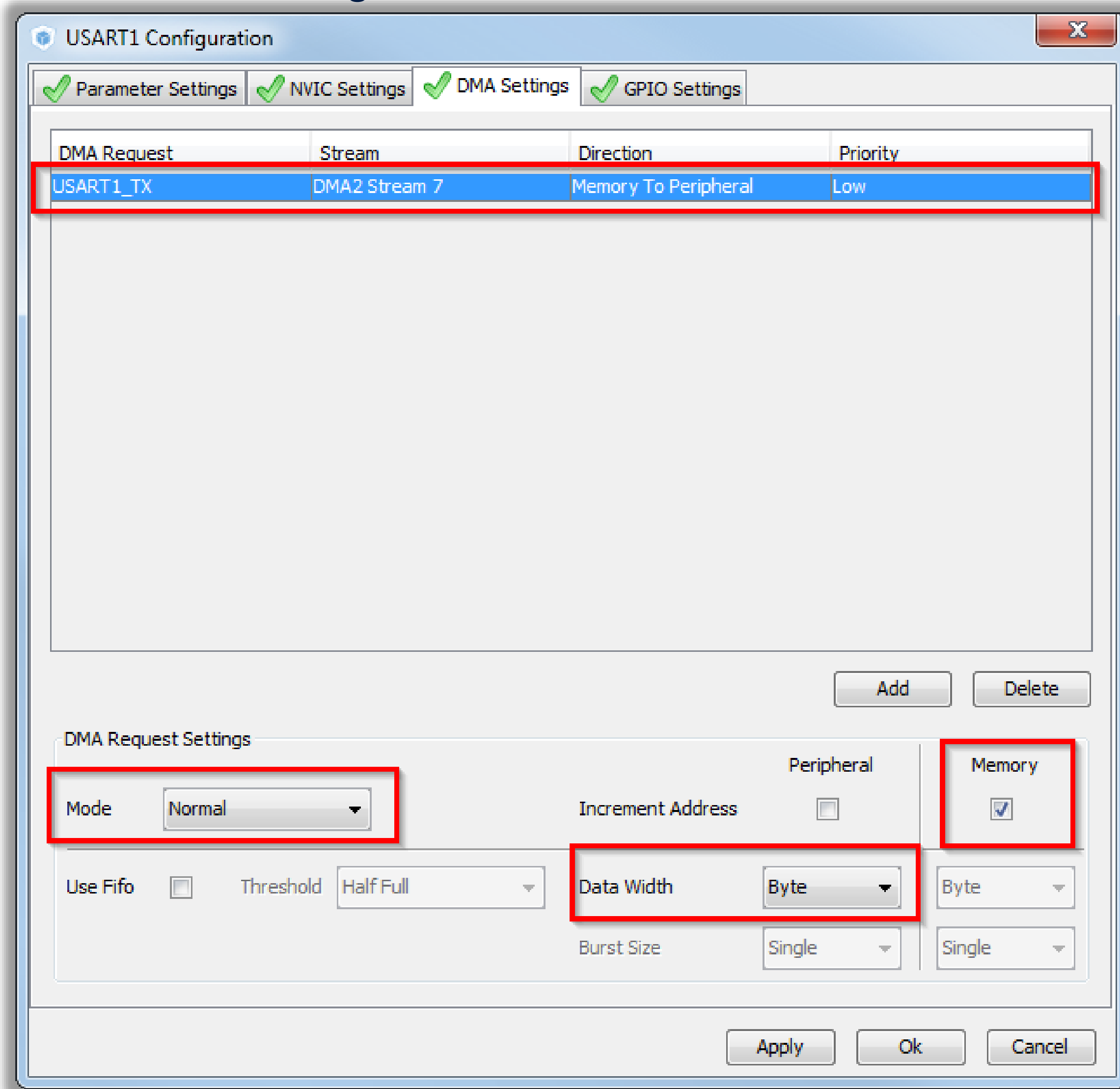
- CubeMX USART configuration DMA settings
  - TAB>DMA Settings
  - Button ADD



life.augmented

2.1.3 Use UART with DMA transfer

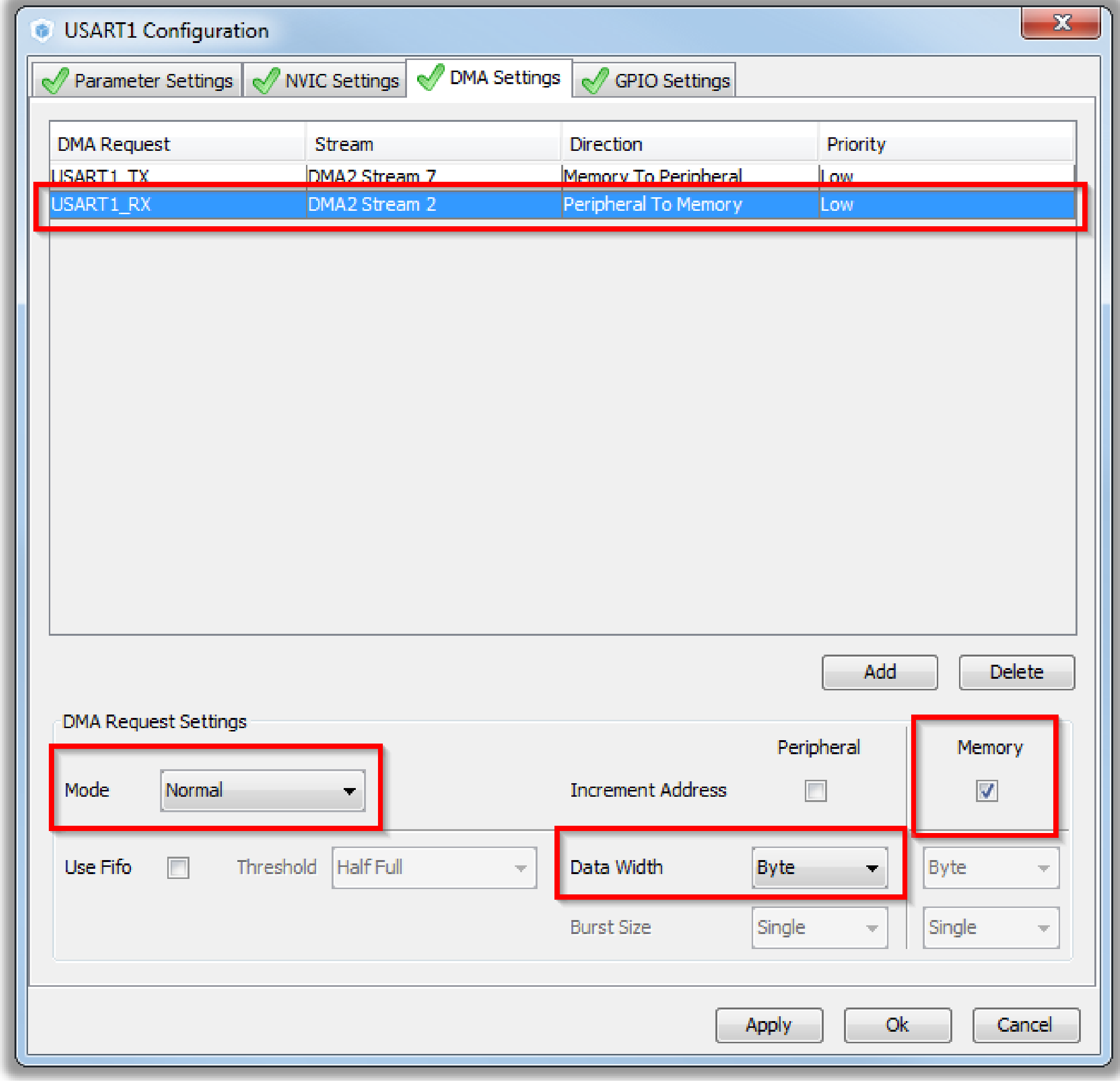
- CubeMX USART configuration DMA Tx settings
  - Set USART1_TX request
  - Memory to peripheral direction
  - Normal mode
  - Byte data width
  - Increment memory address



life.augmented

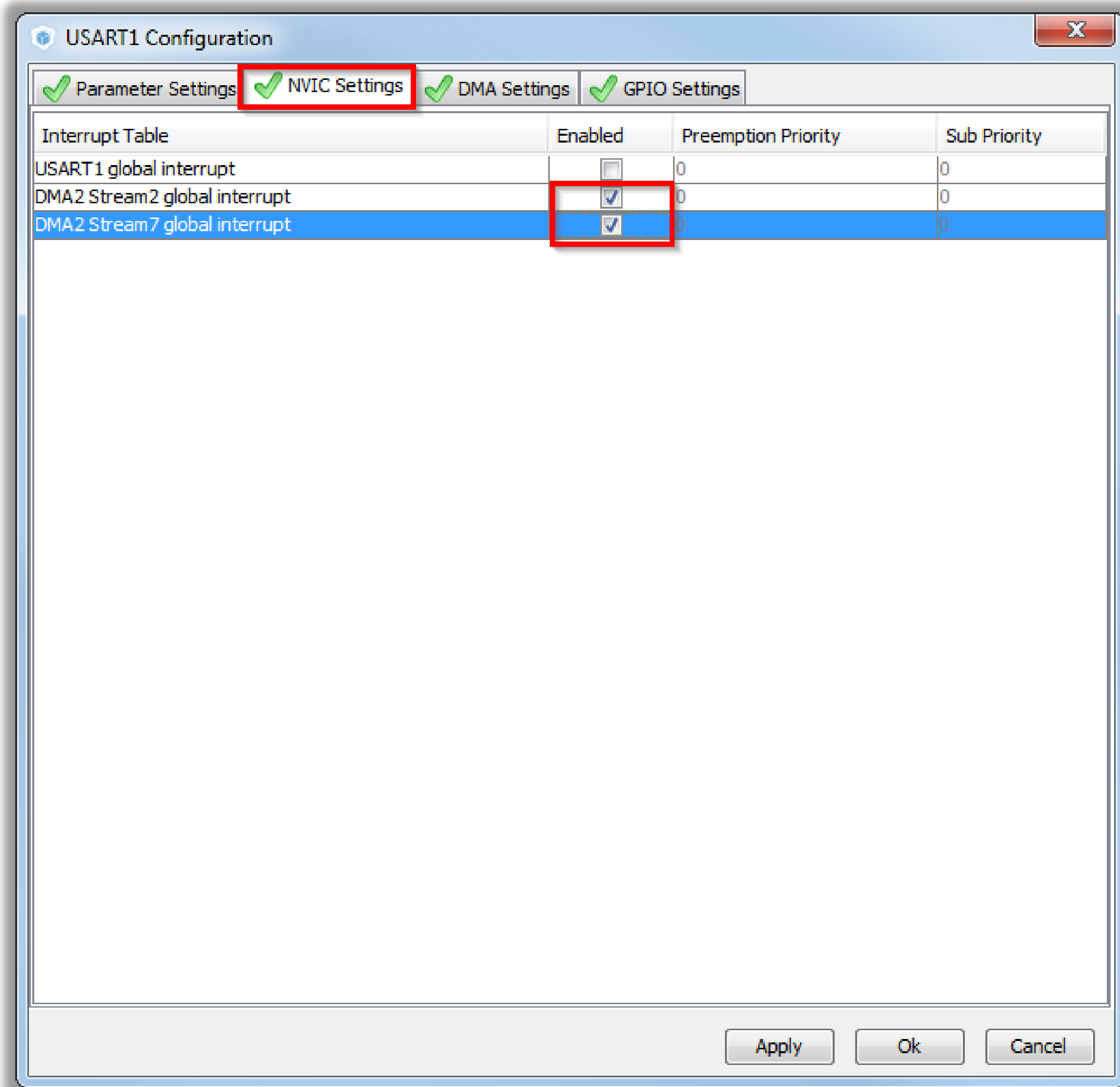
2.1.3 Use UART with DMA transfer

- CubeMX USART configuration DMA Rx settings
  - Button ADD
  - Set USART1_RX request
  - Peripheral to memory direction
  - Normal mode
  - Byte data width
  - Increment memory address



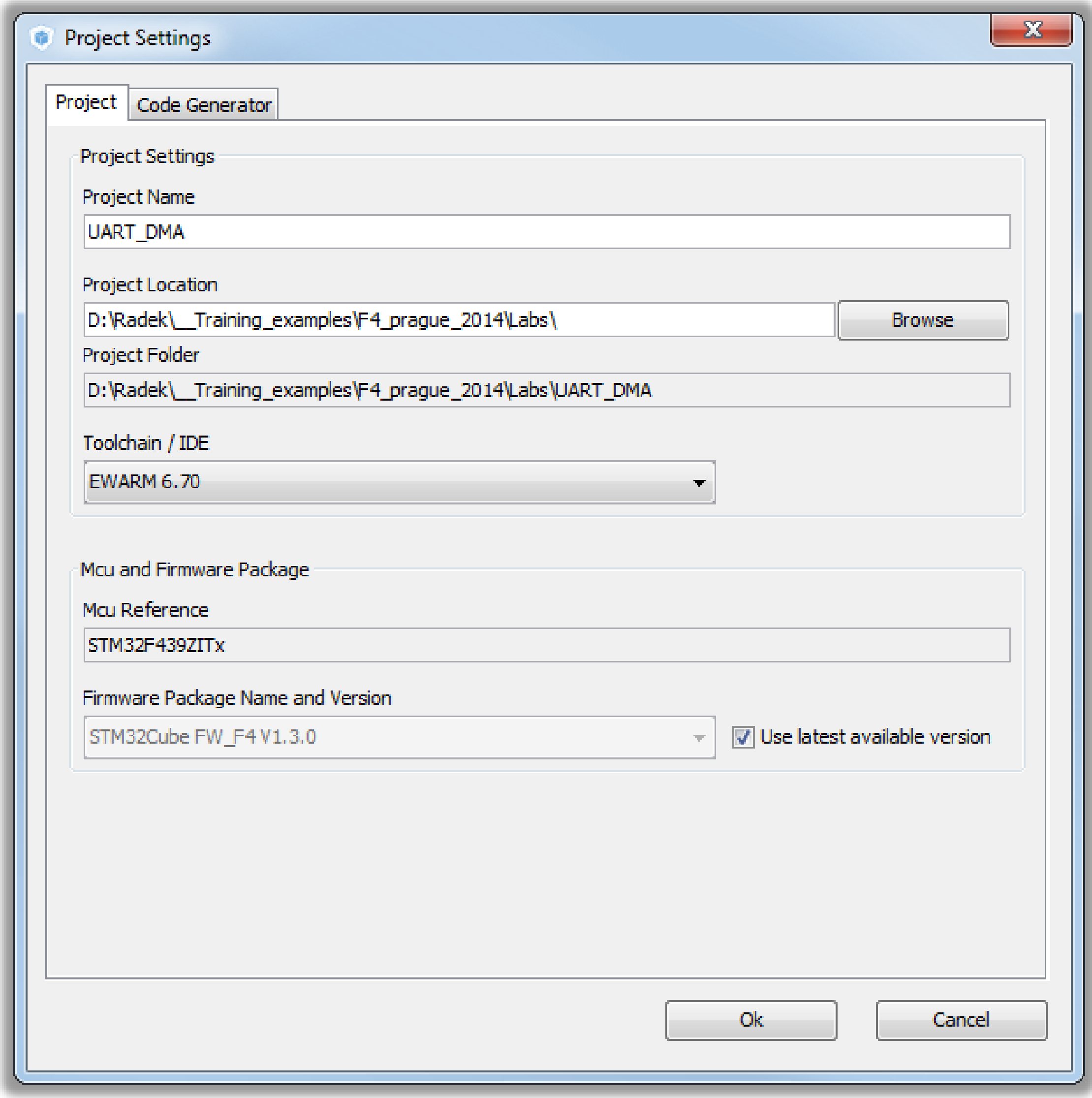
2.1.3 Use UART with DMA transfer

- CubeMX USART configuration NVIC settings
  - TAB>NVIC Settings
  - Enable DMA2 interrupts for USART1
  - Button OK

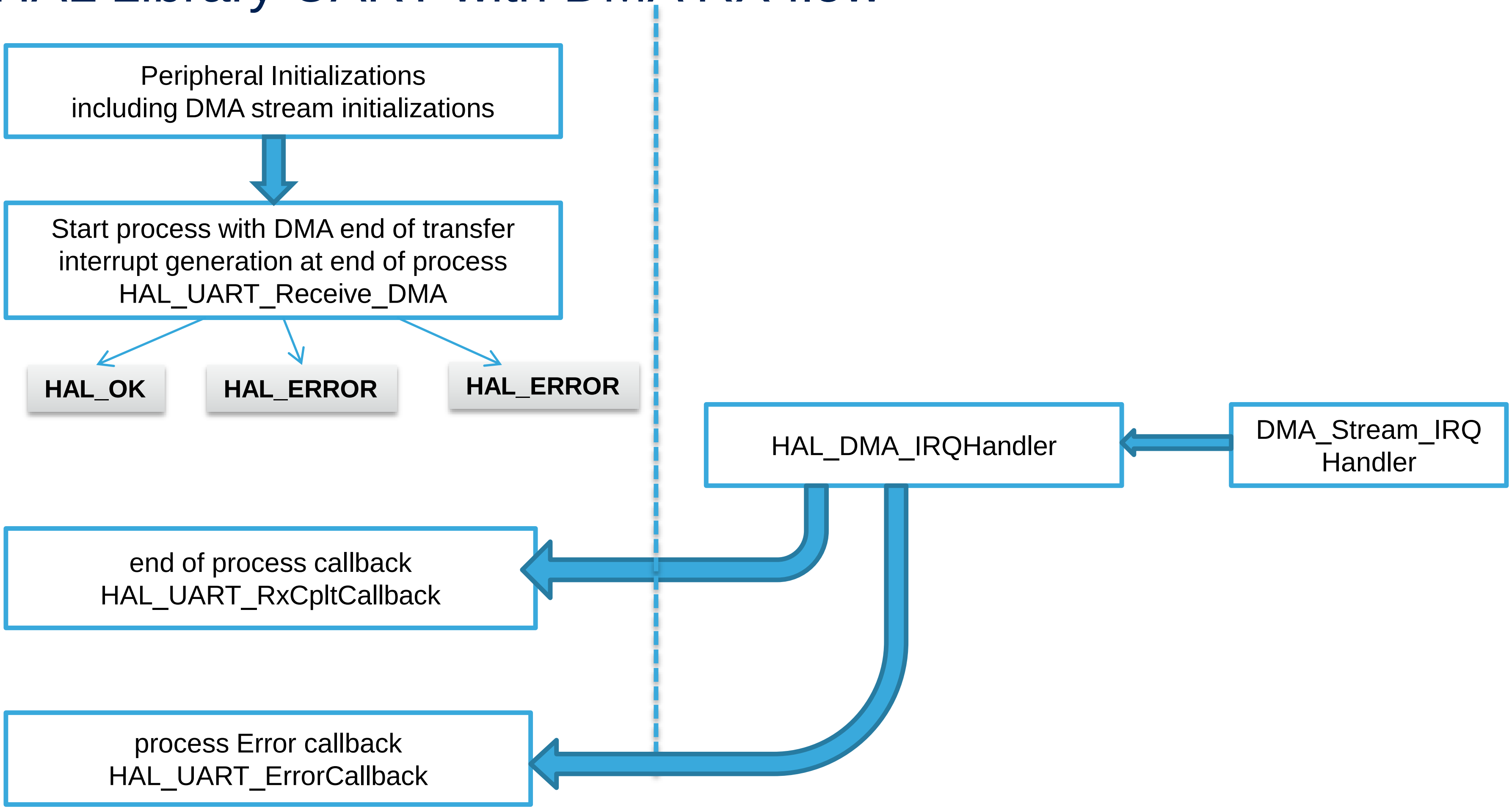


2.1.3 Use UART with DMA transfer

- 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

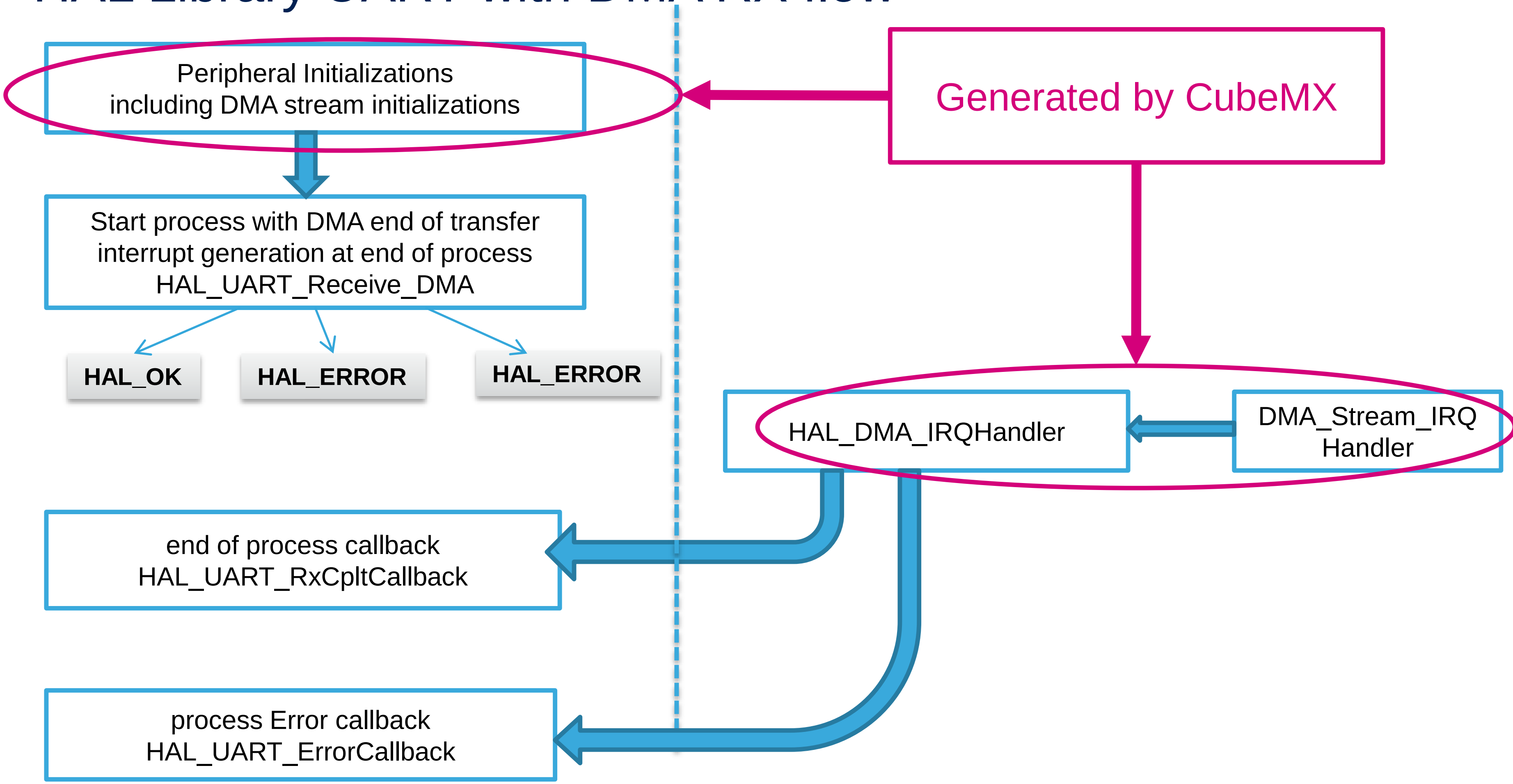


HAL Library UART with DMA RX flow



2.1.3 Use UART with DMA transfer

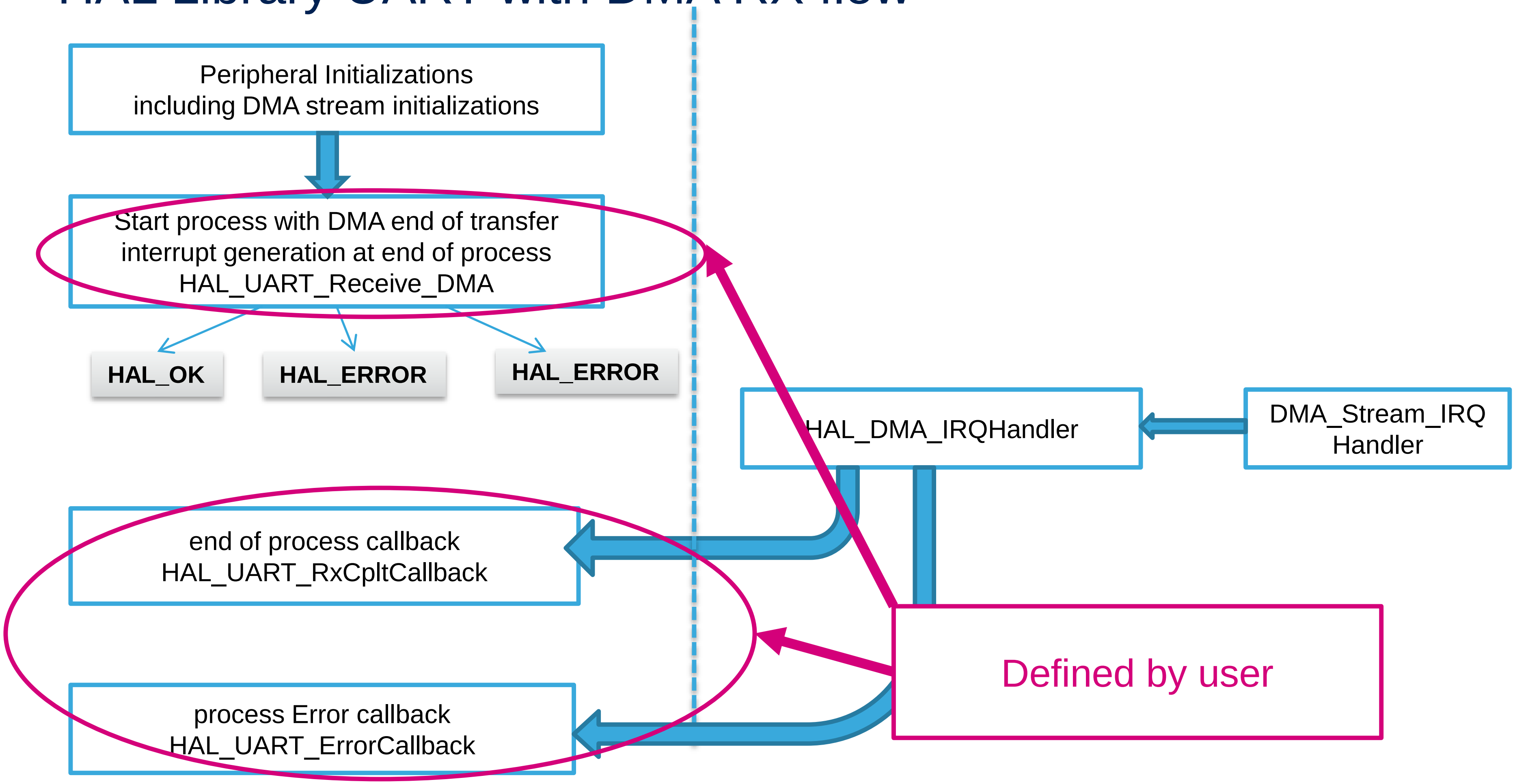
HAL Library UART with DMA RX flow



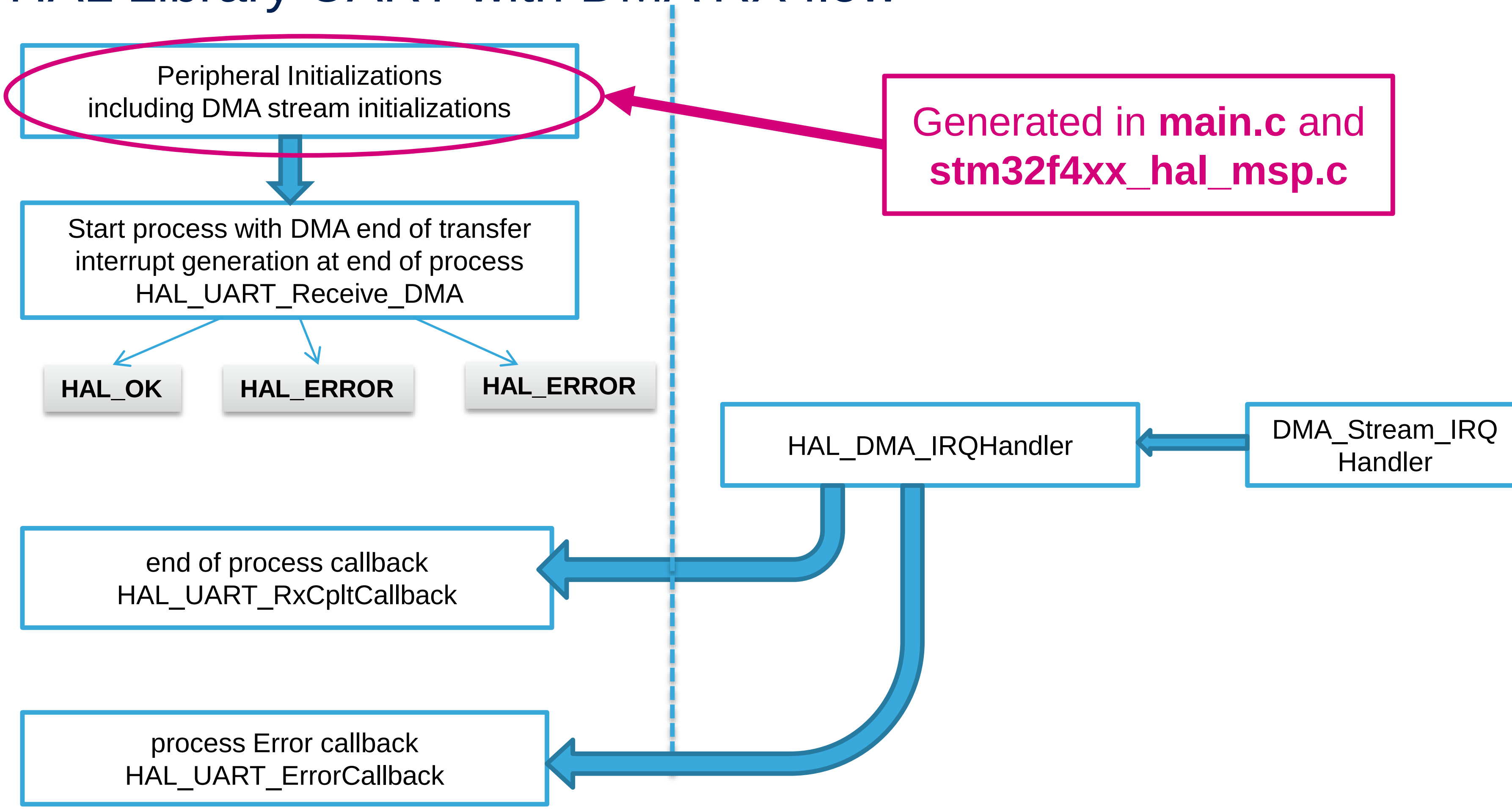
life.augmented

2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow

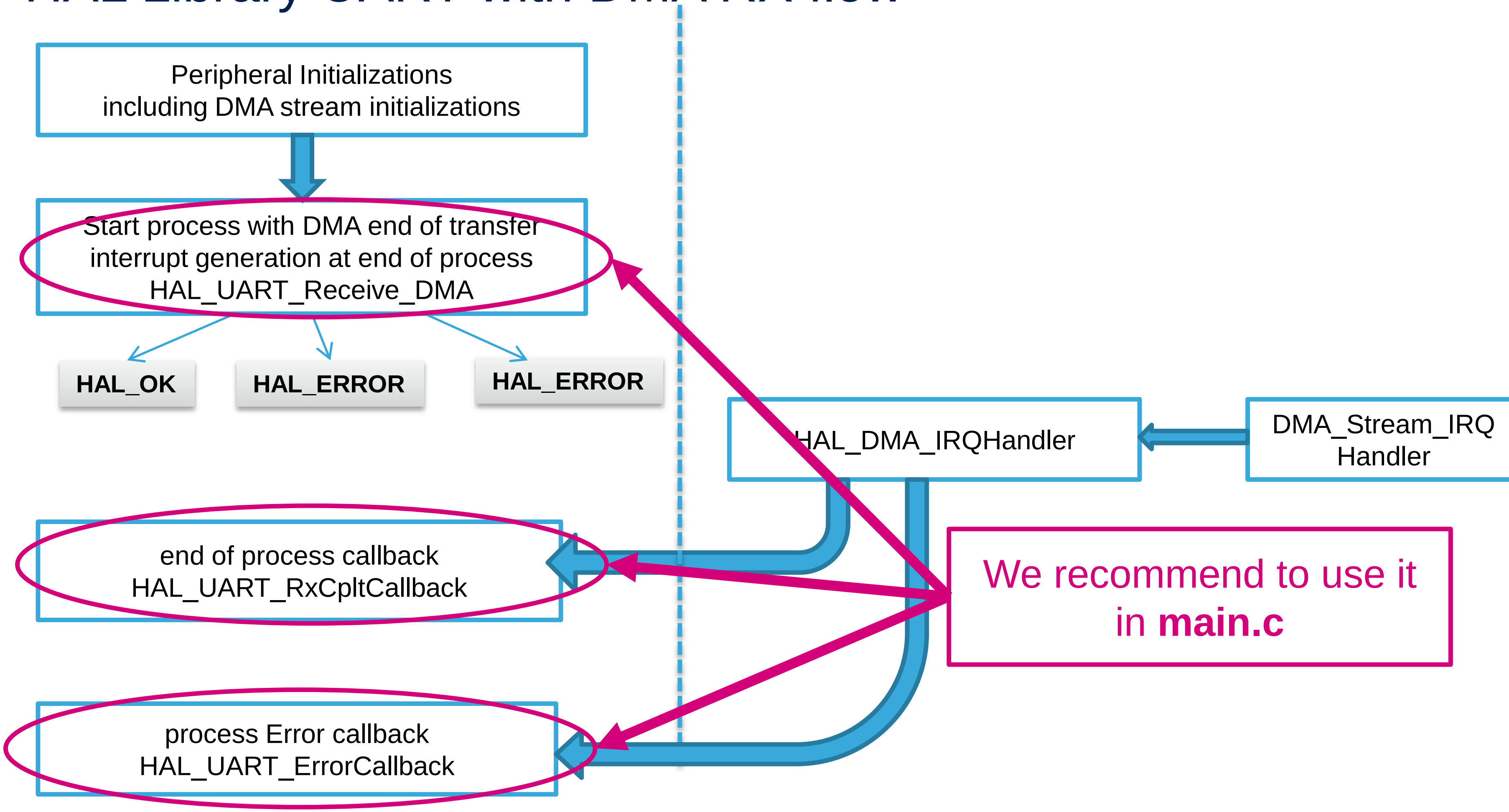


HAL Library UART with DMA RX flow



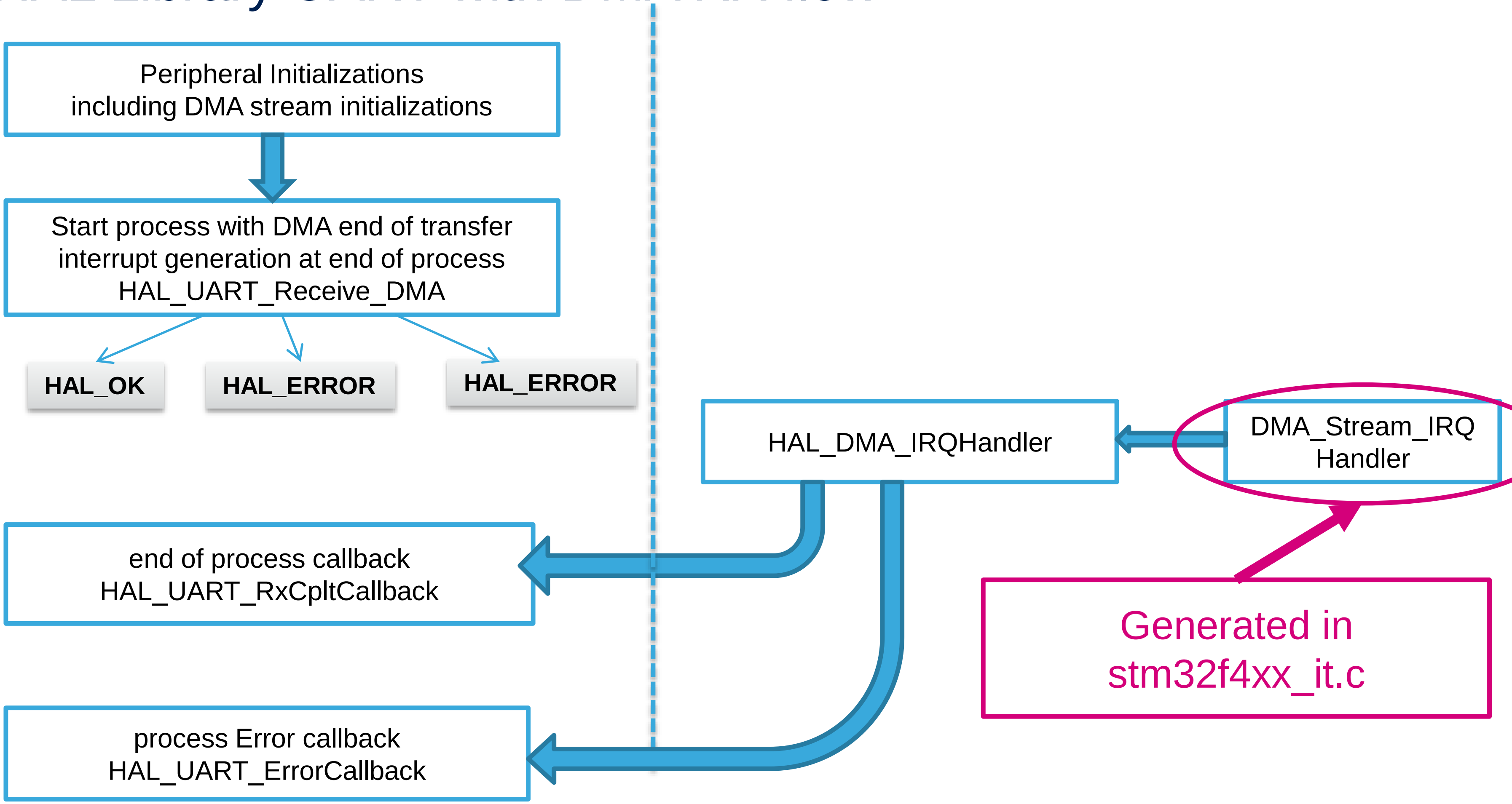
2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow



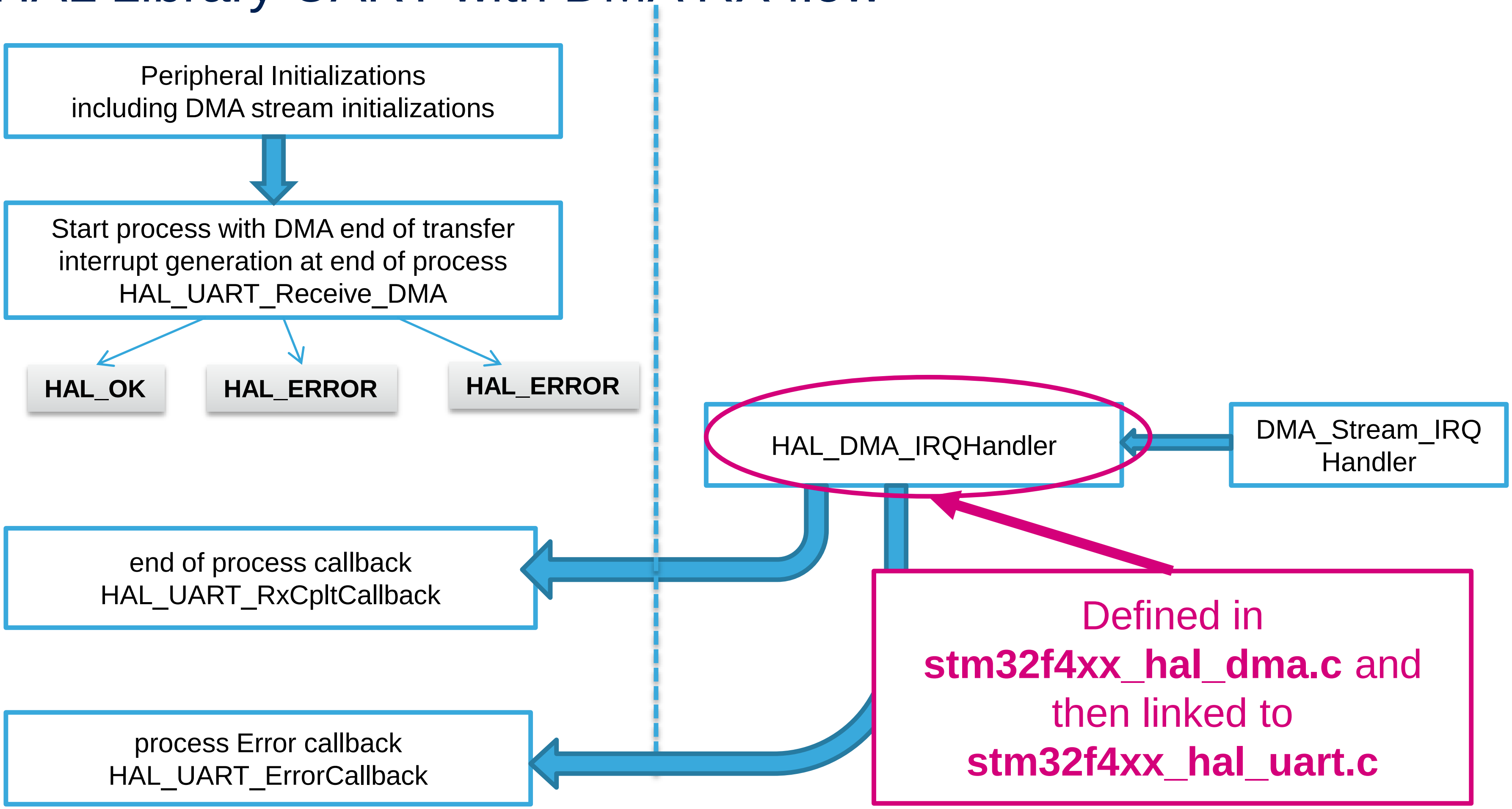
2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow

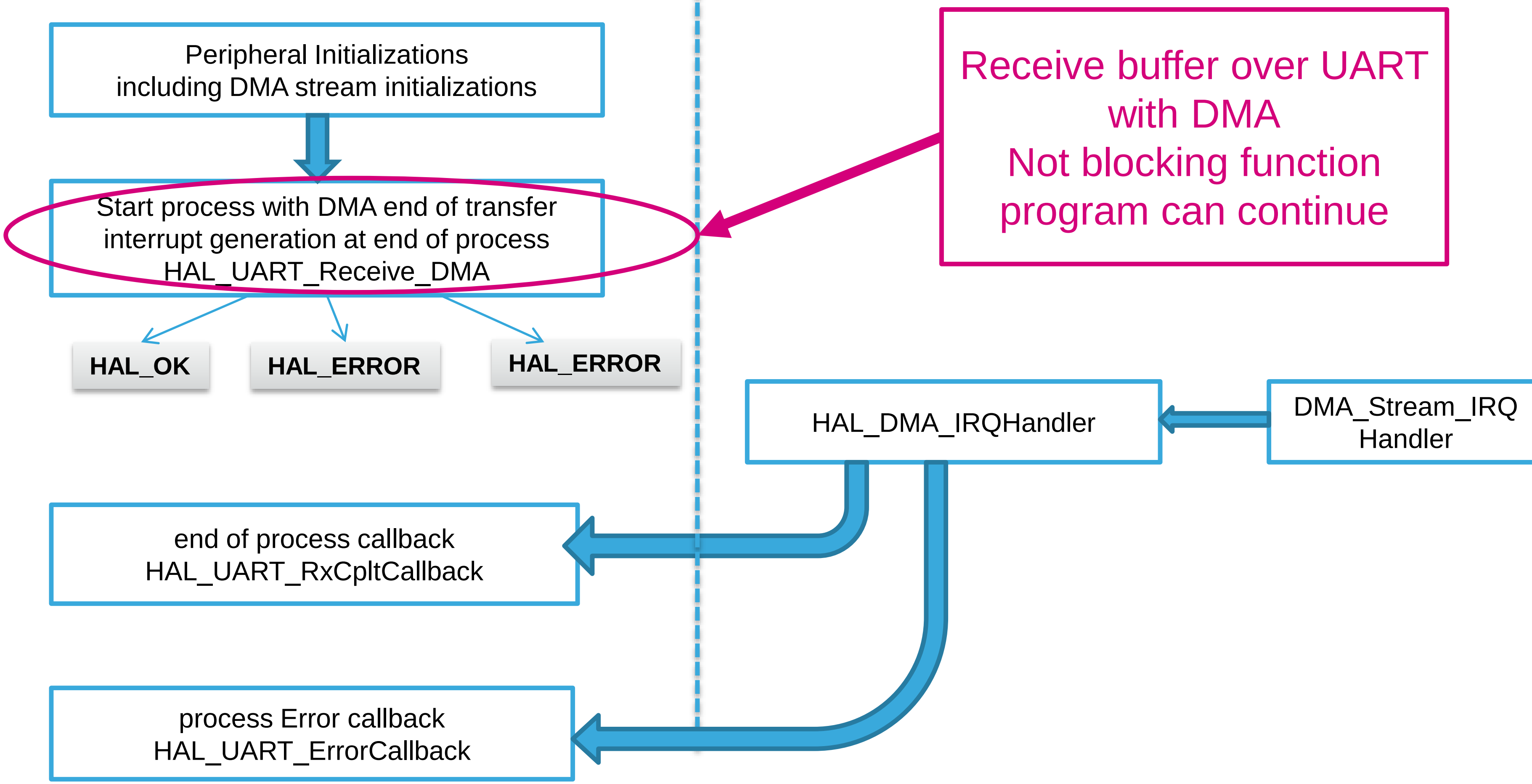


2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow

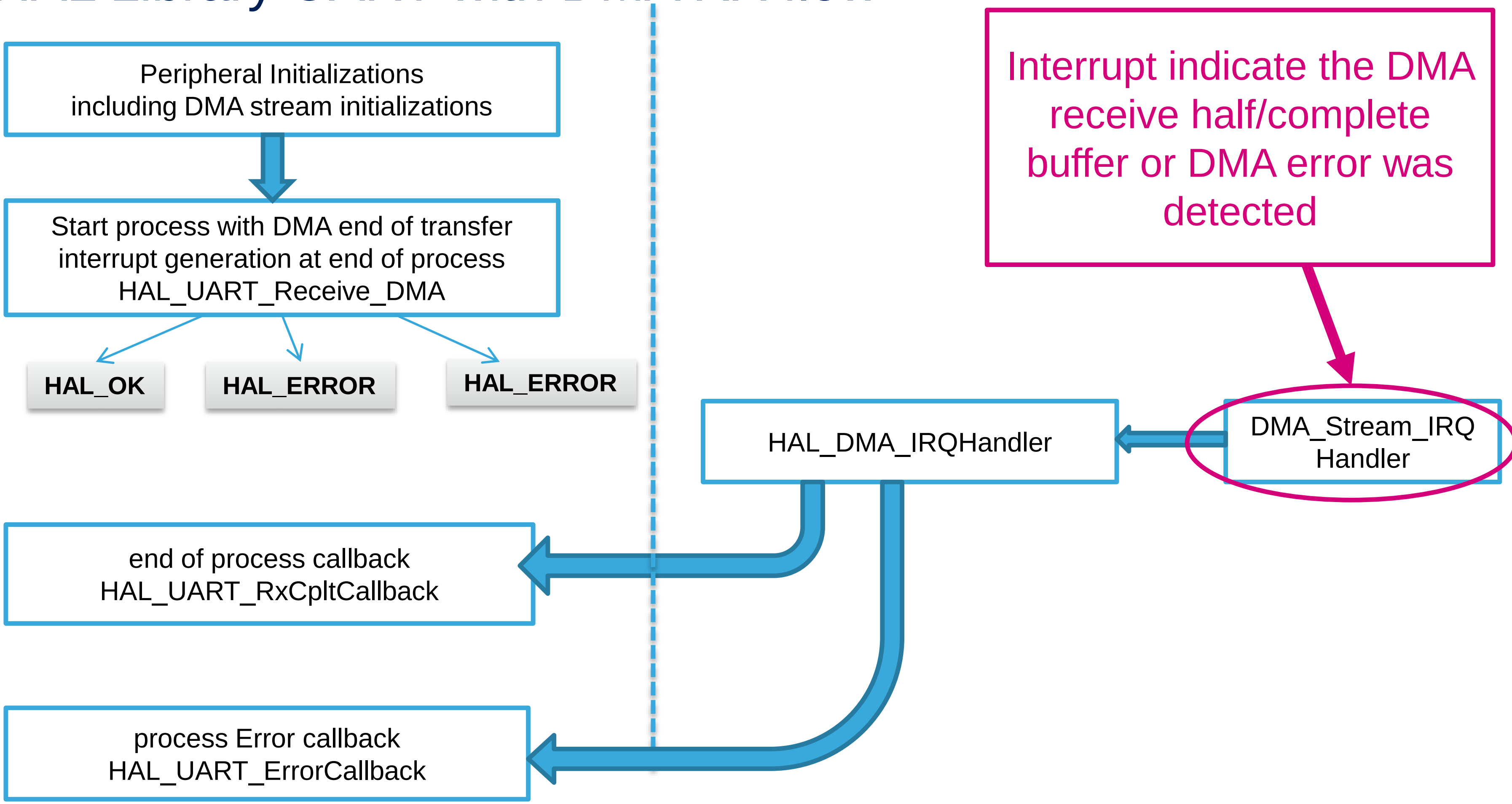


HAL Library UART with DMA RX flow



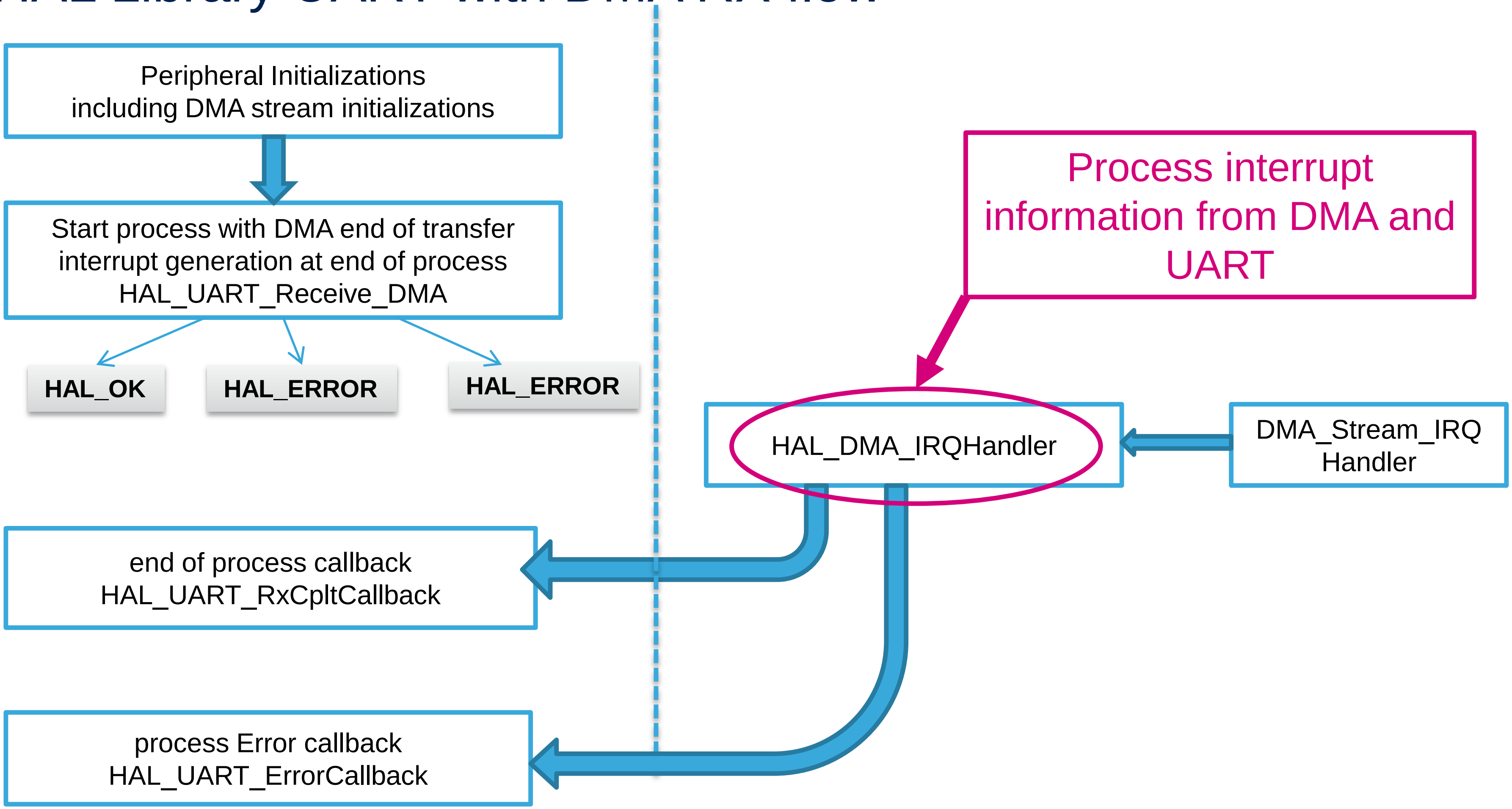
2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow



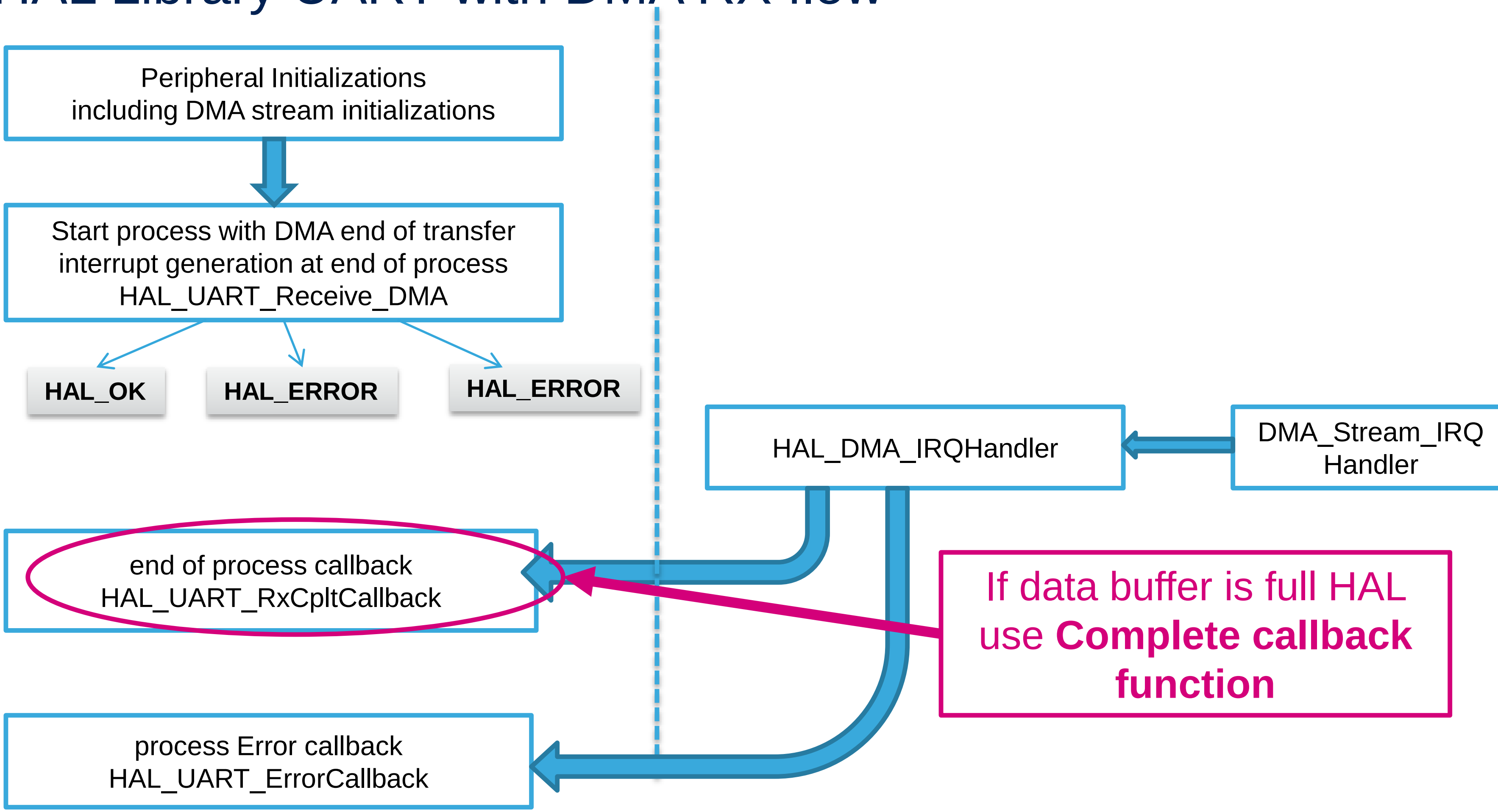
2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow



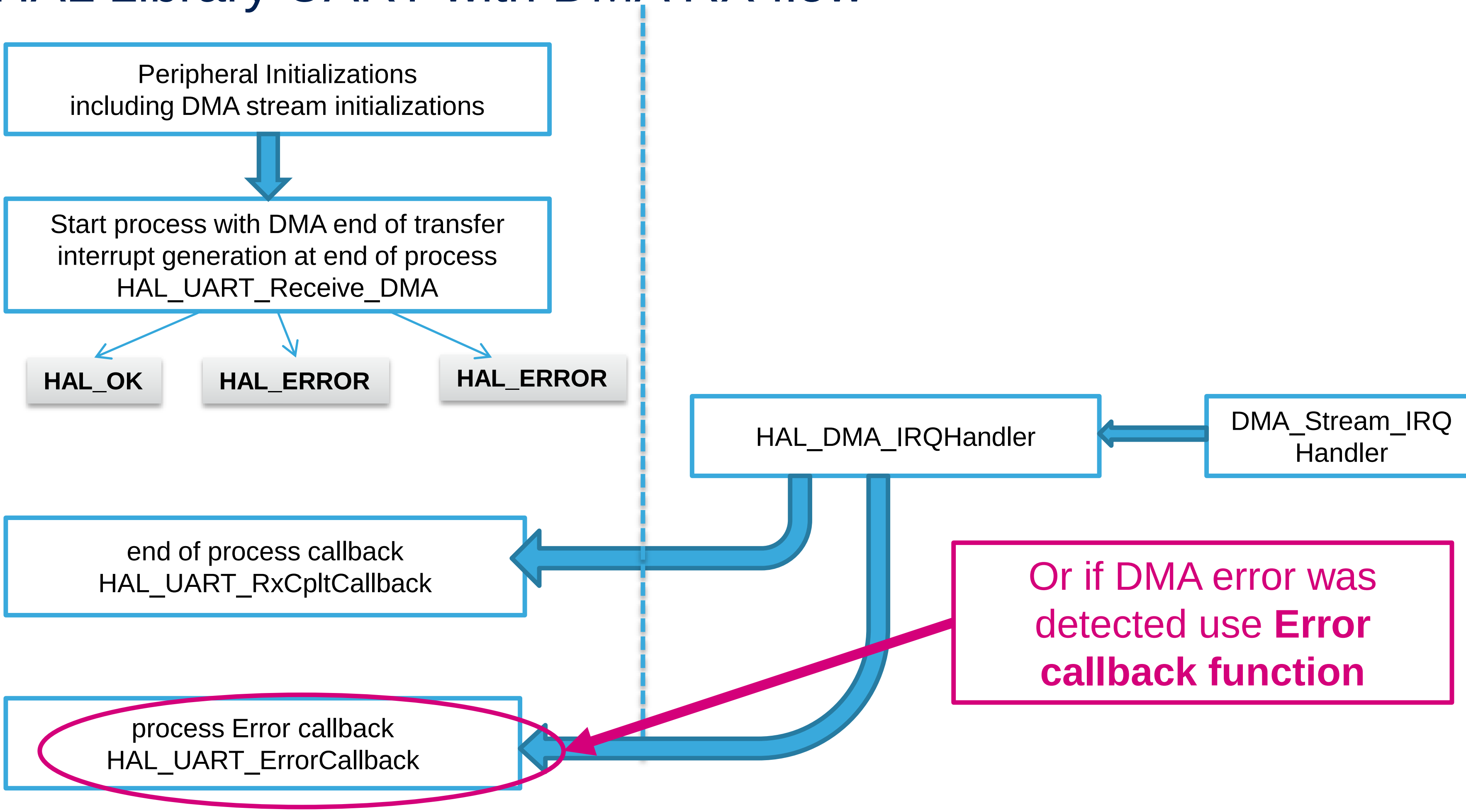
2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow



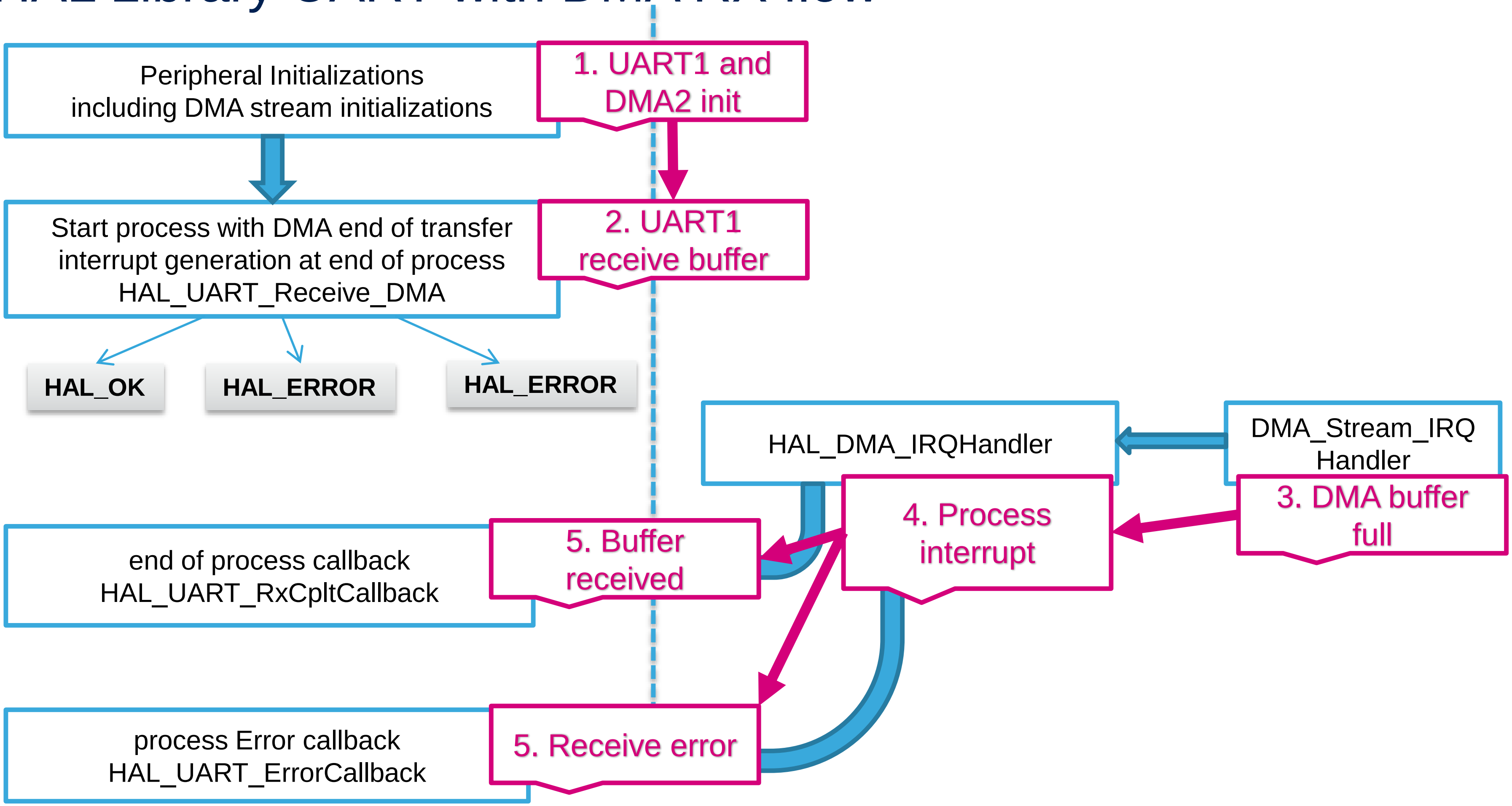
2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow



2.1.3 Use UART with DMA transfer

HAL Library UART with DMA RX flow



2.1.3 Use UART with DMA transfer

- 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);

2.1.3 Use UART with DMA transfer

- 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 */
```

2.1.3 Use UART with DMA transfer

- 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 */
```

2.2.1 SPI Poll lab

2.2.1 Simple SPI communication

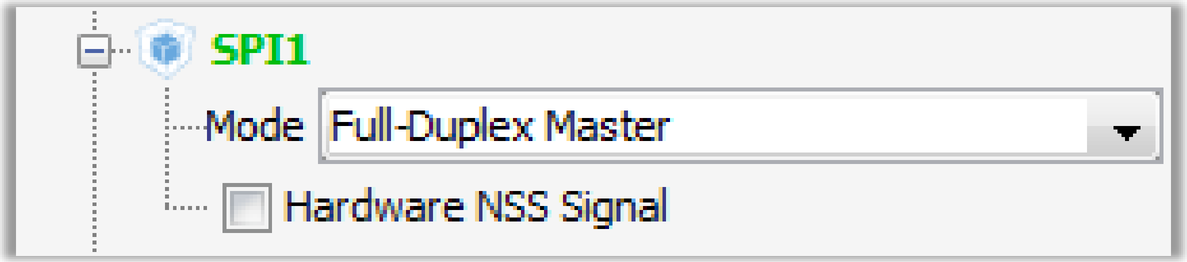
- Objective
  - Learn how to setup SPI in CubeMX
  - How to Generate Code in CubeMX and use HAL functions
- Goal
  - Configure SPI in CubeMX and Generate Code
  - Learn how to send and receive data over SPI without interrupts
  - Verify the correct functionality

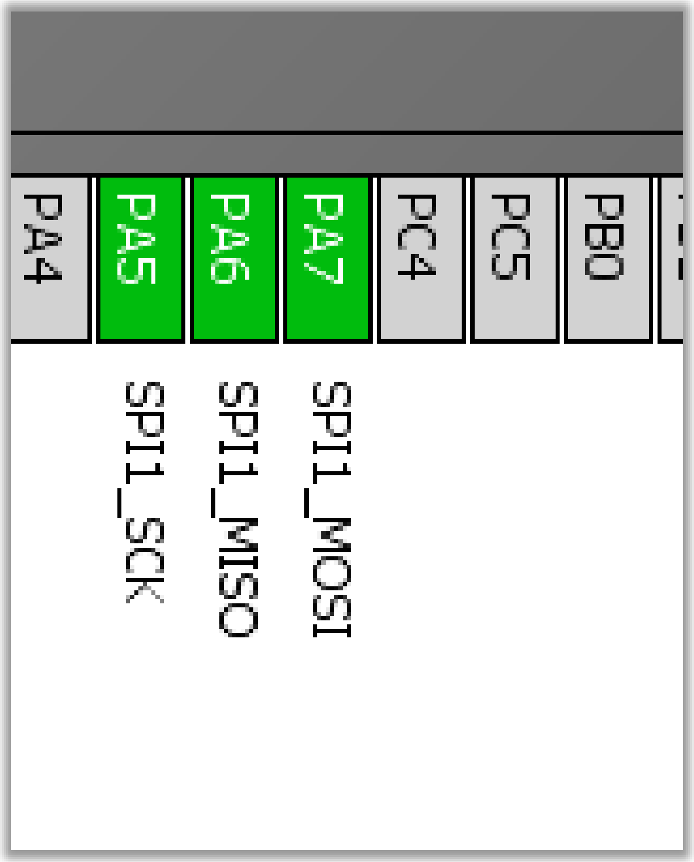
2.2.1 Simple SPI communication

- 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

2.2.1 Simple SPI communication

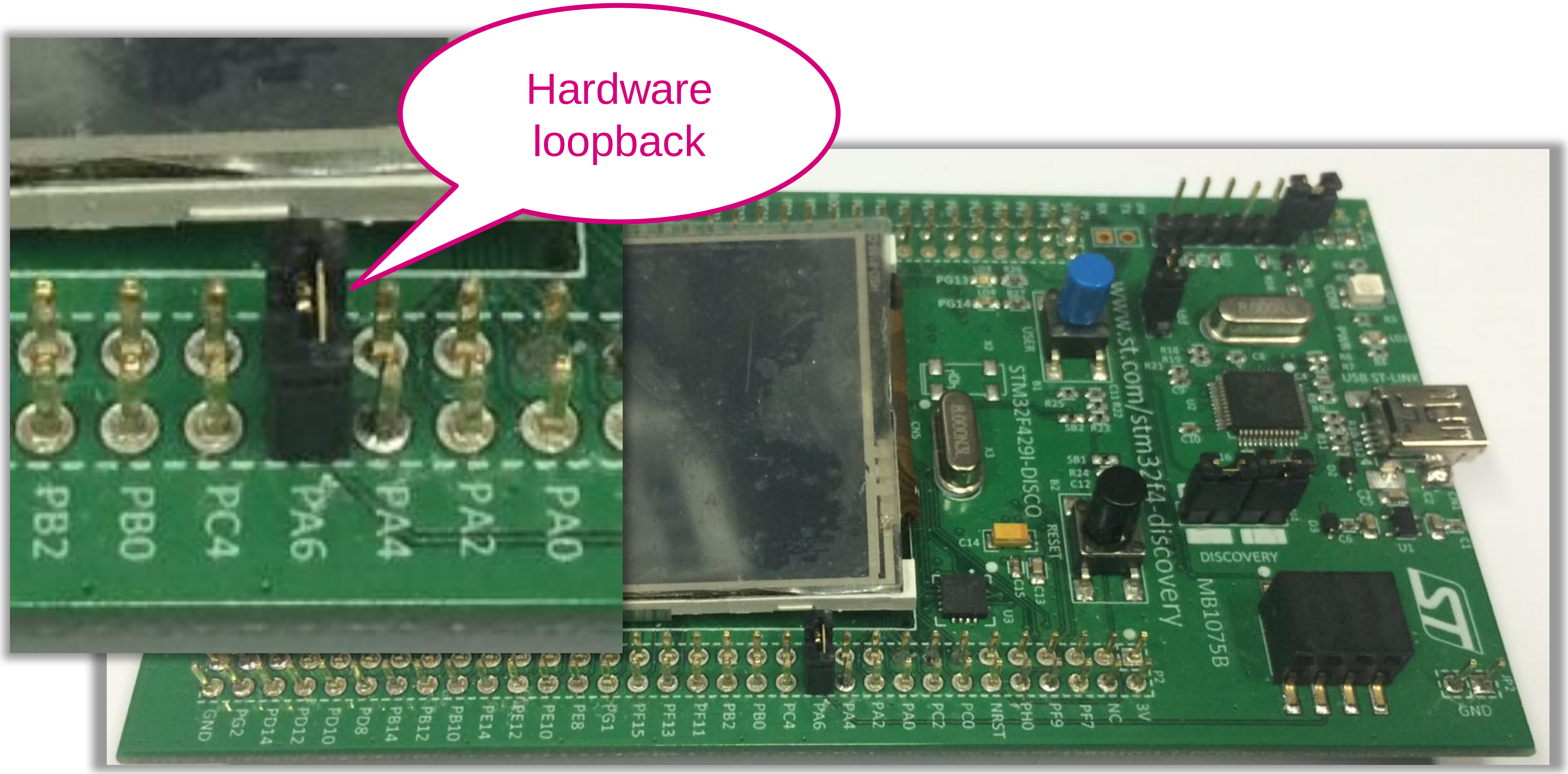
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX SPI selection
  - Select SPI1 Full-Duplex Master
  - Select PA5, PA6, PA7 for SPI1 if weren't selected





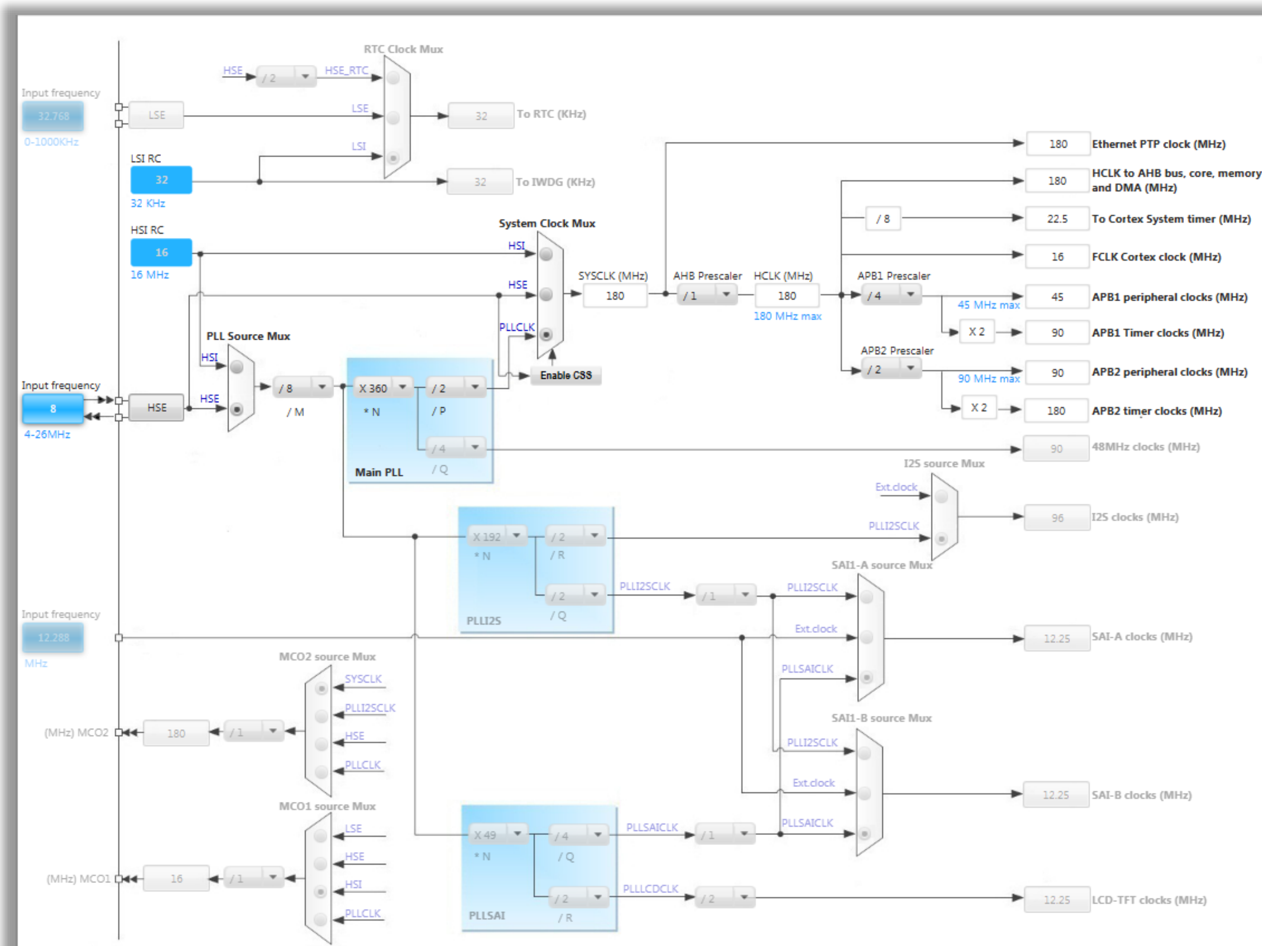
2.2.1 Simple SPI communication

- Hardware preparation
  - Connect PA6 and PA7 together with jumper



2.2.1 Simple SPI communication

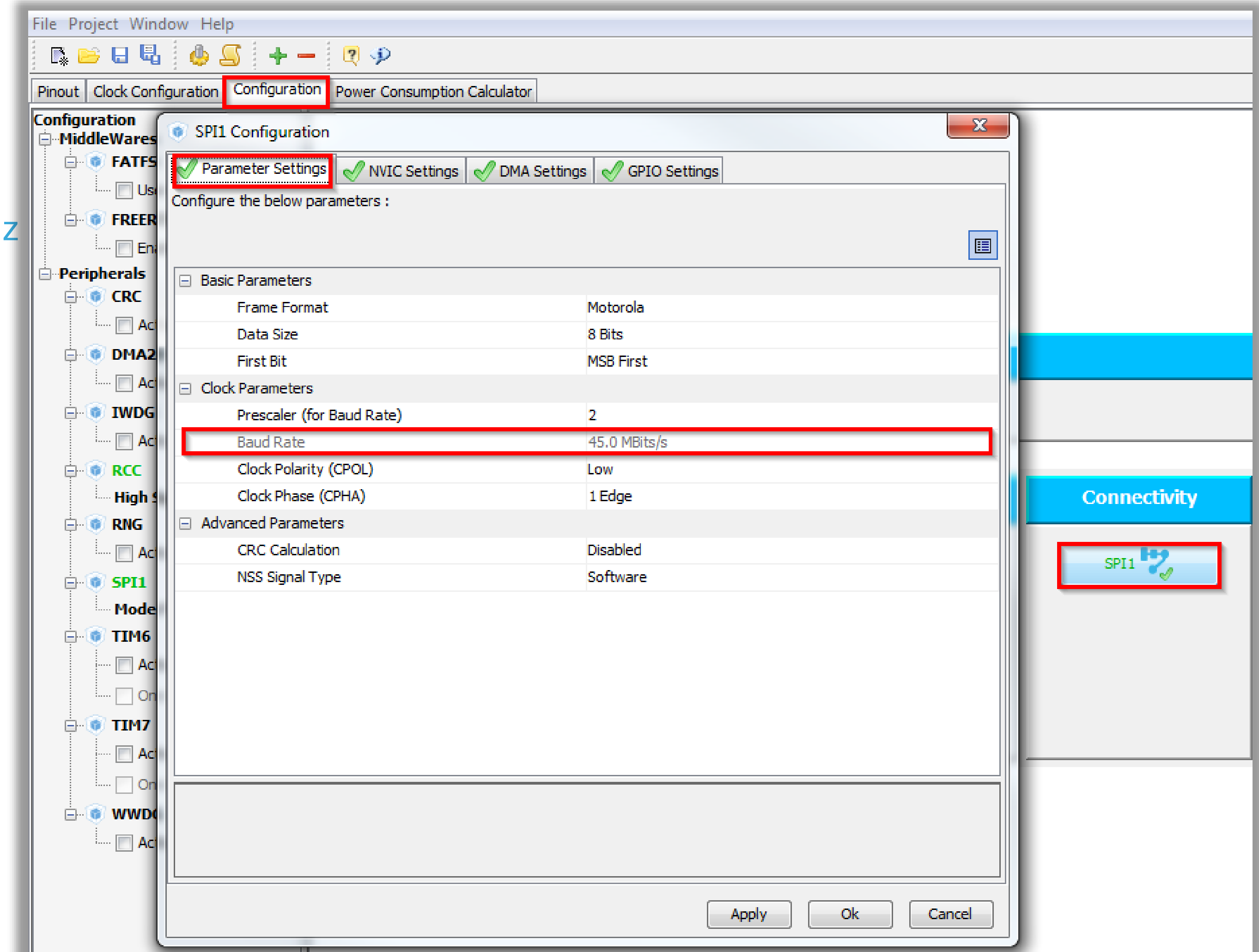
- In order to run on maximum frequency, setup clock system
- Details in lab 0



life.augmented

2.2.1 Simple SPI communication

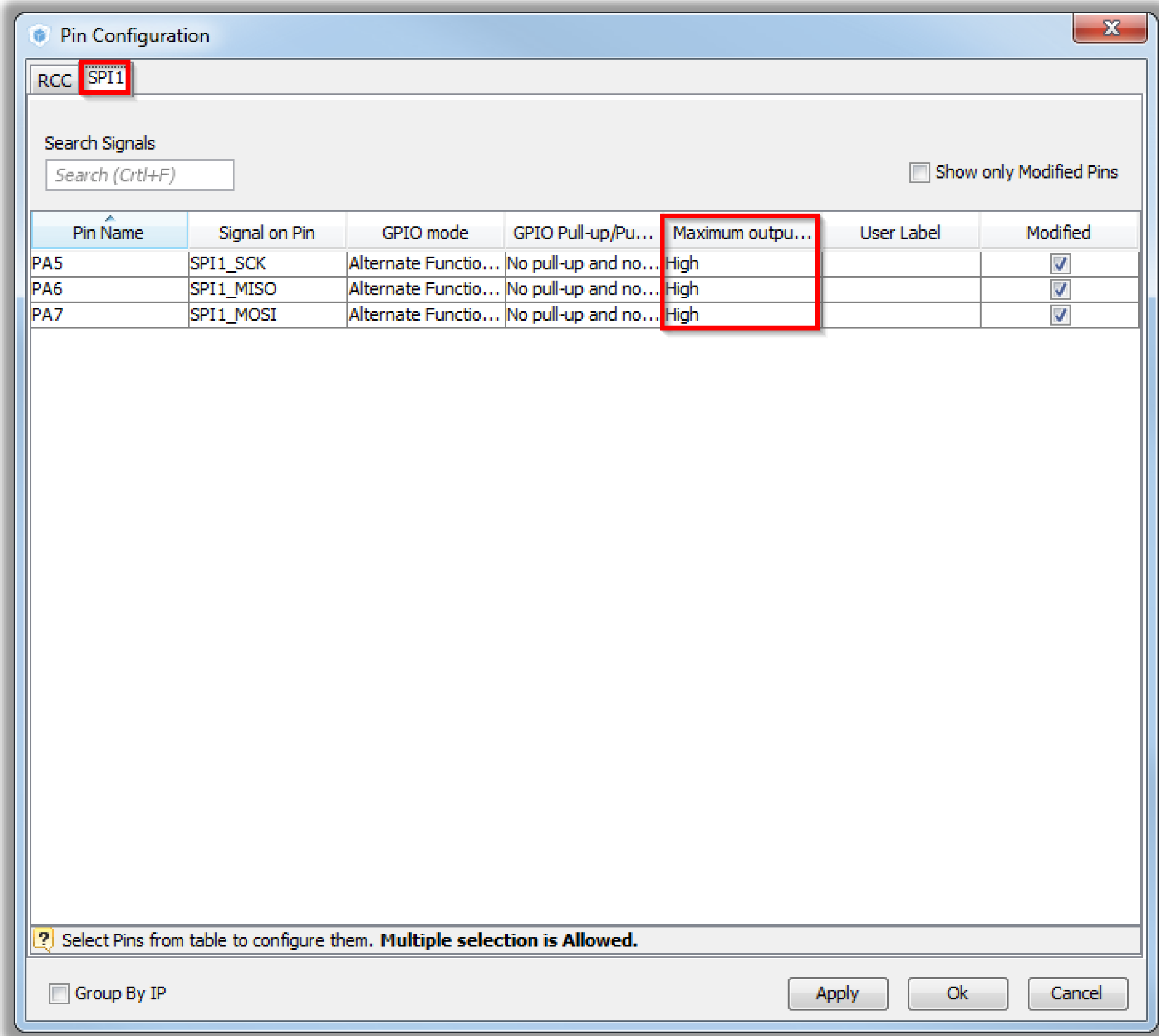
- CubeMX SPI configuration
  - Tab>Configuration>Connectivity>SPI1
  - Check the settings
  - Button OK
  - The CLK frequency with core on 180MHz is now 45MHz
  - For this clock use HIGH GPIO speed



life.augmented

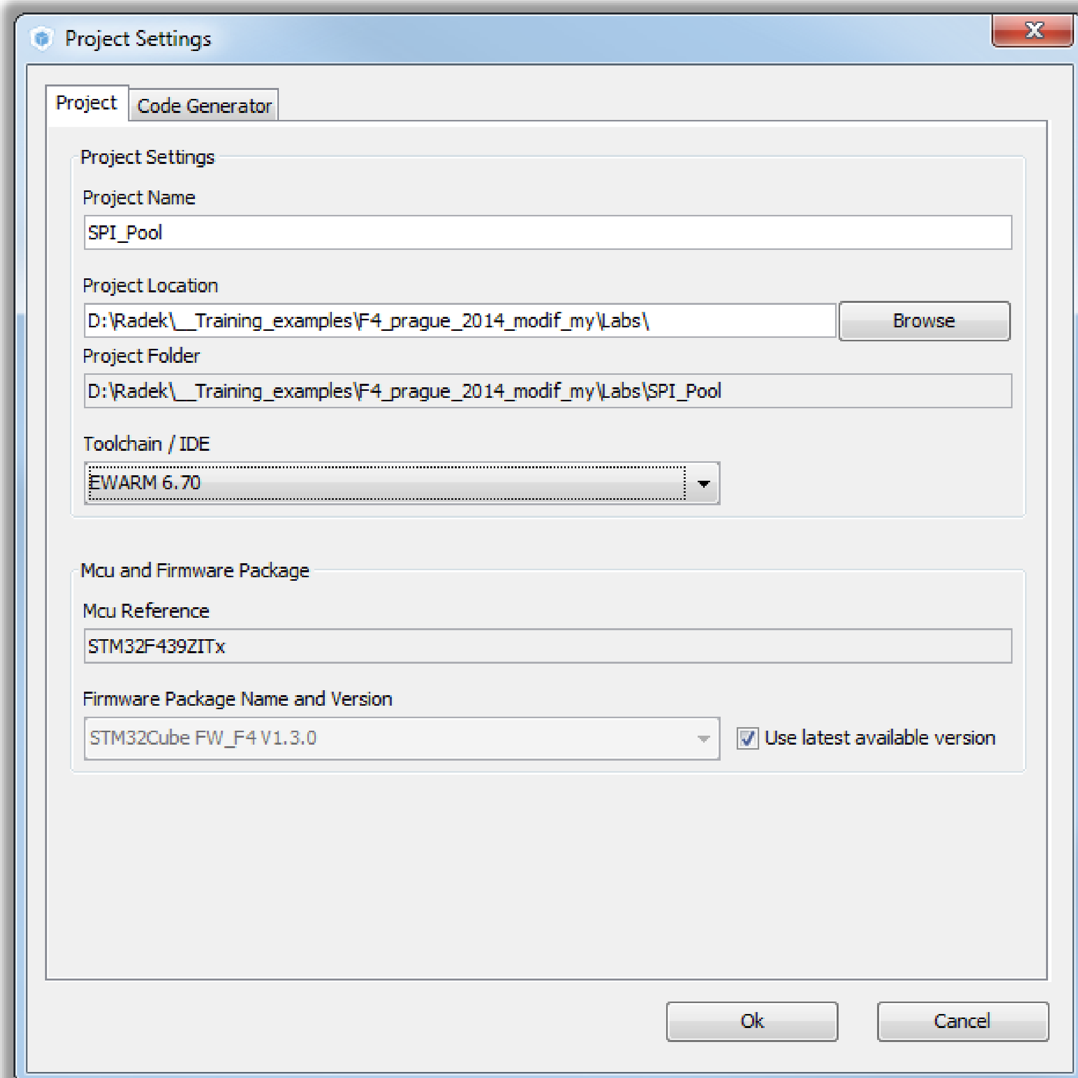
2.2.1 Simple SPI communication

- 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
  - Seth High output speed
  - Button OK



2.2.1 Simple SPI communication

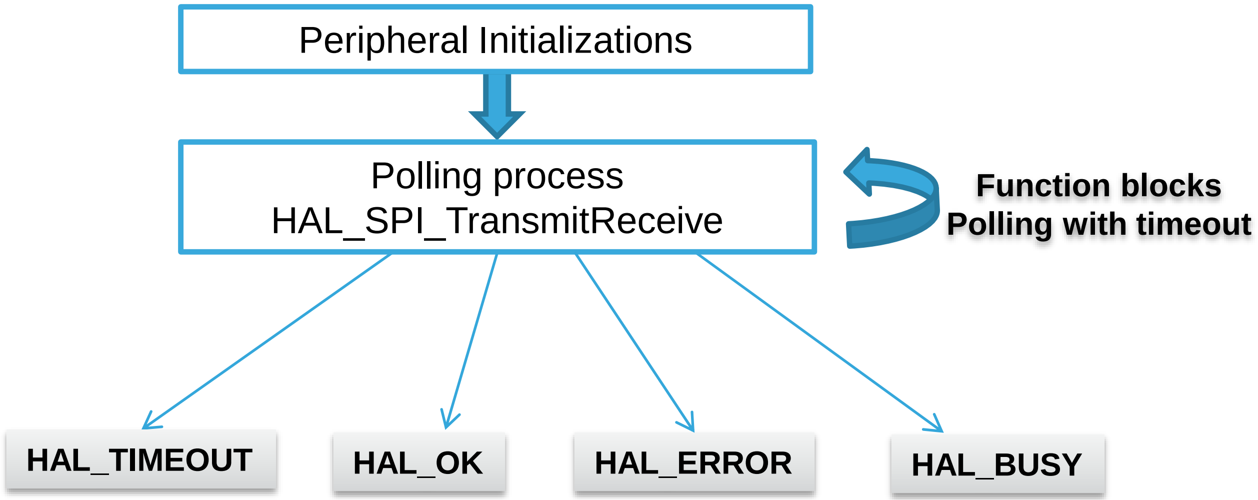
- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



life.augmented

2.2.1 Simple SPI communication

HAL Library transmit receive flow



2.2.1 Simple SPI communication

- 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)

2.2.1 Simple SPI communication

- 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 */
```

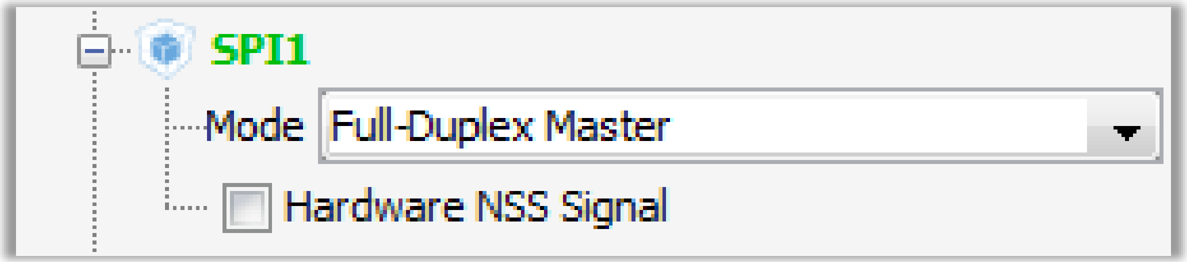
2.2.2 SPI Interrupt lab

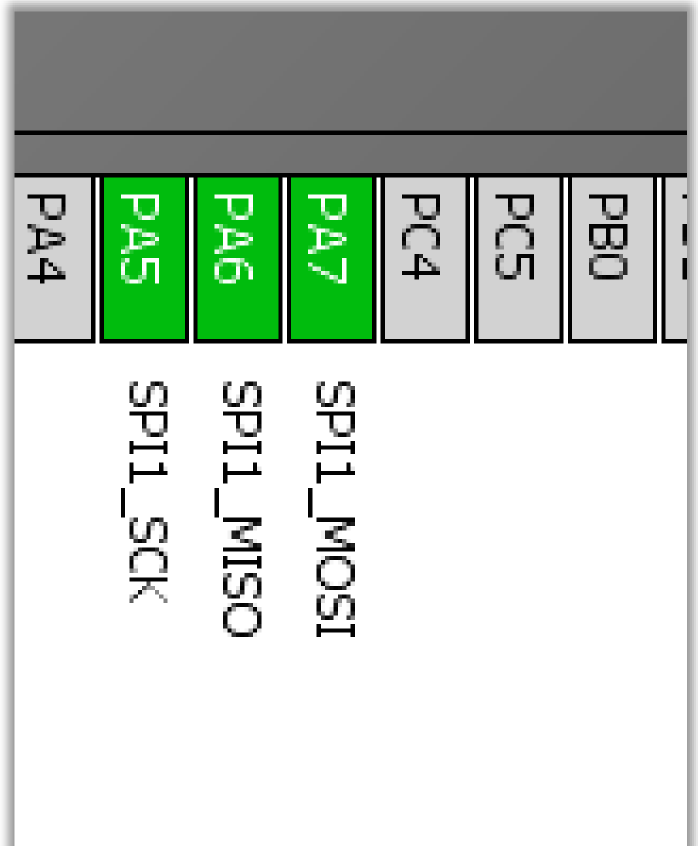
2.2.2 Use SPI with interrupt

- 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

2.2.2 Use SPI with interrupt

- 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

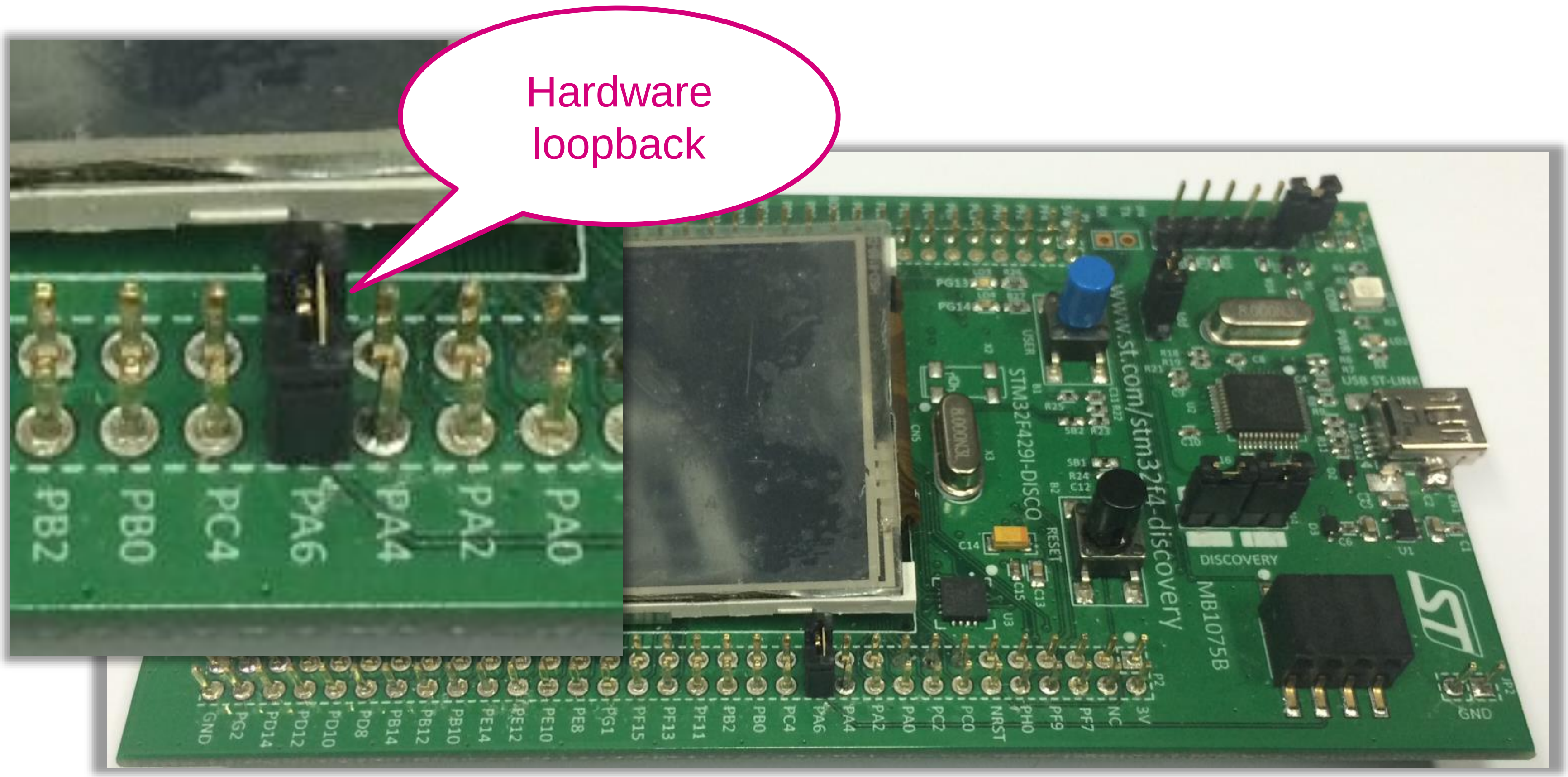




life.augmented

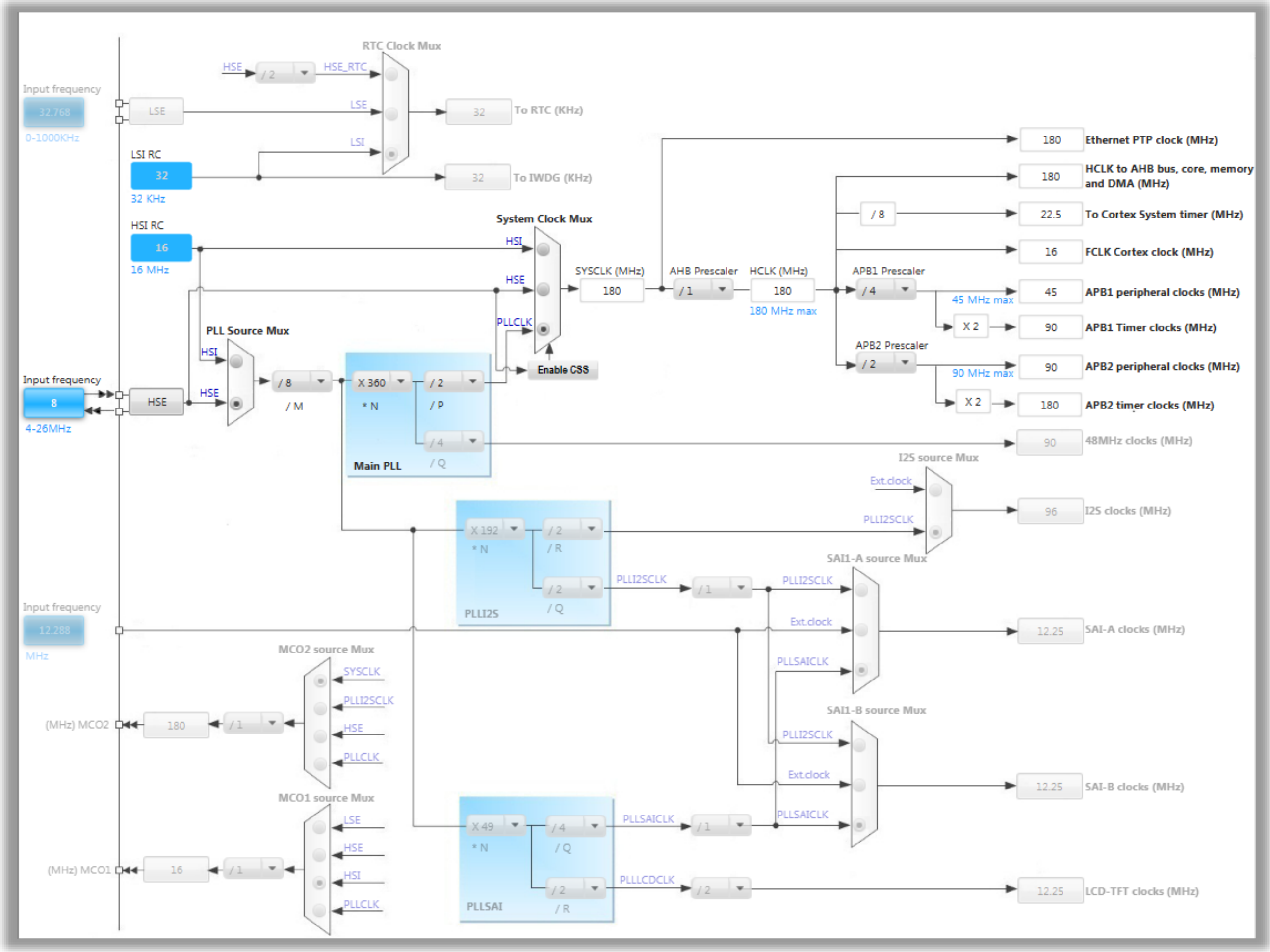
2.2.2 Use SPI with interrupt

- Hardware preparation
  - Connect PA6 and PA7 together with jumper



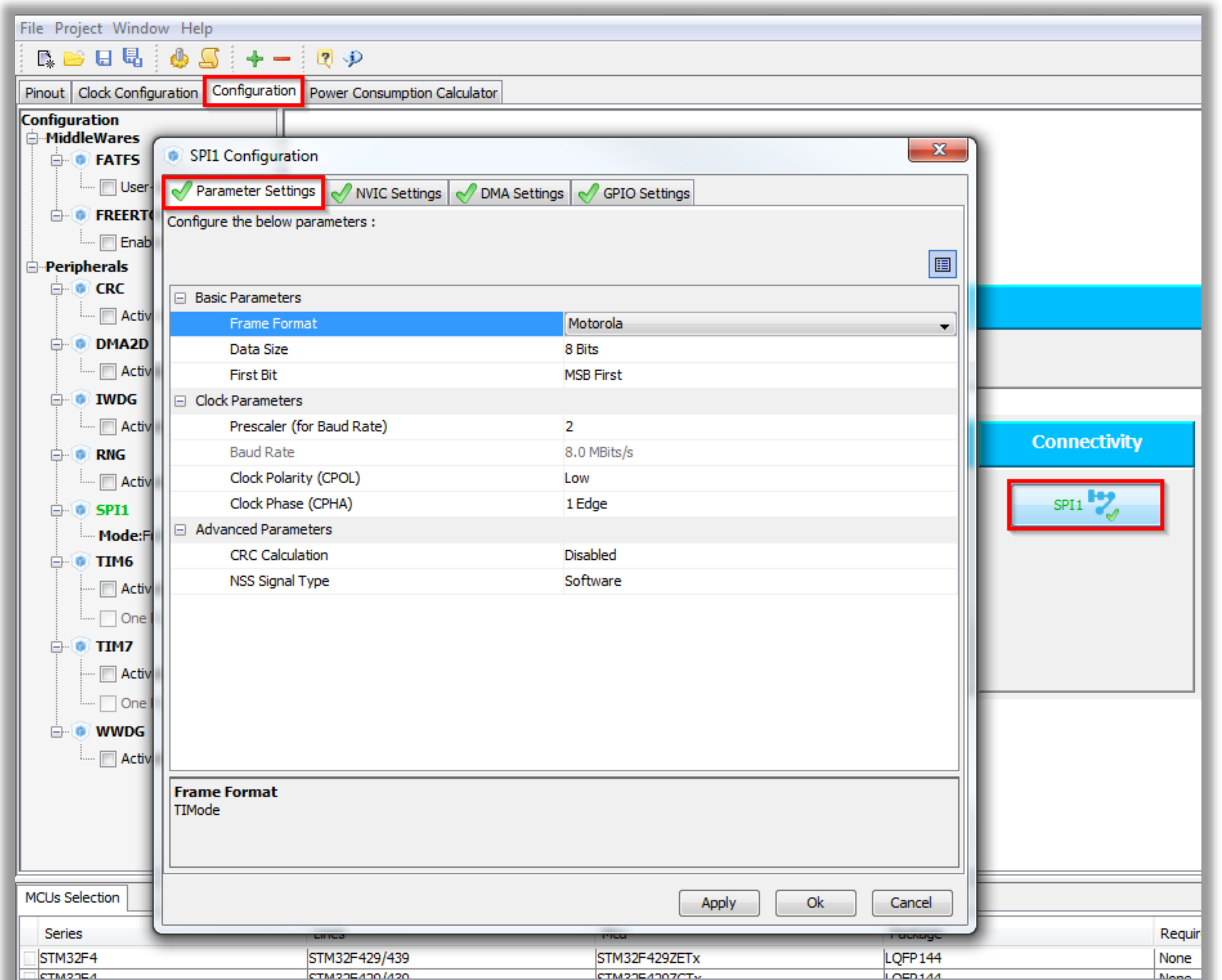
2.2.2 Use SPI with interrupt

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



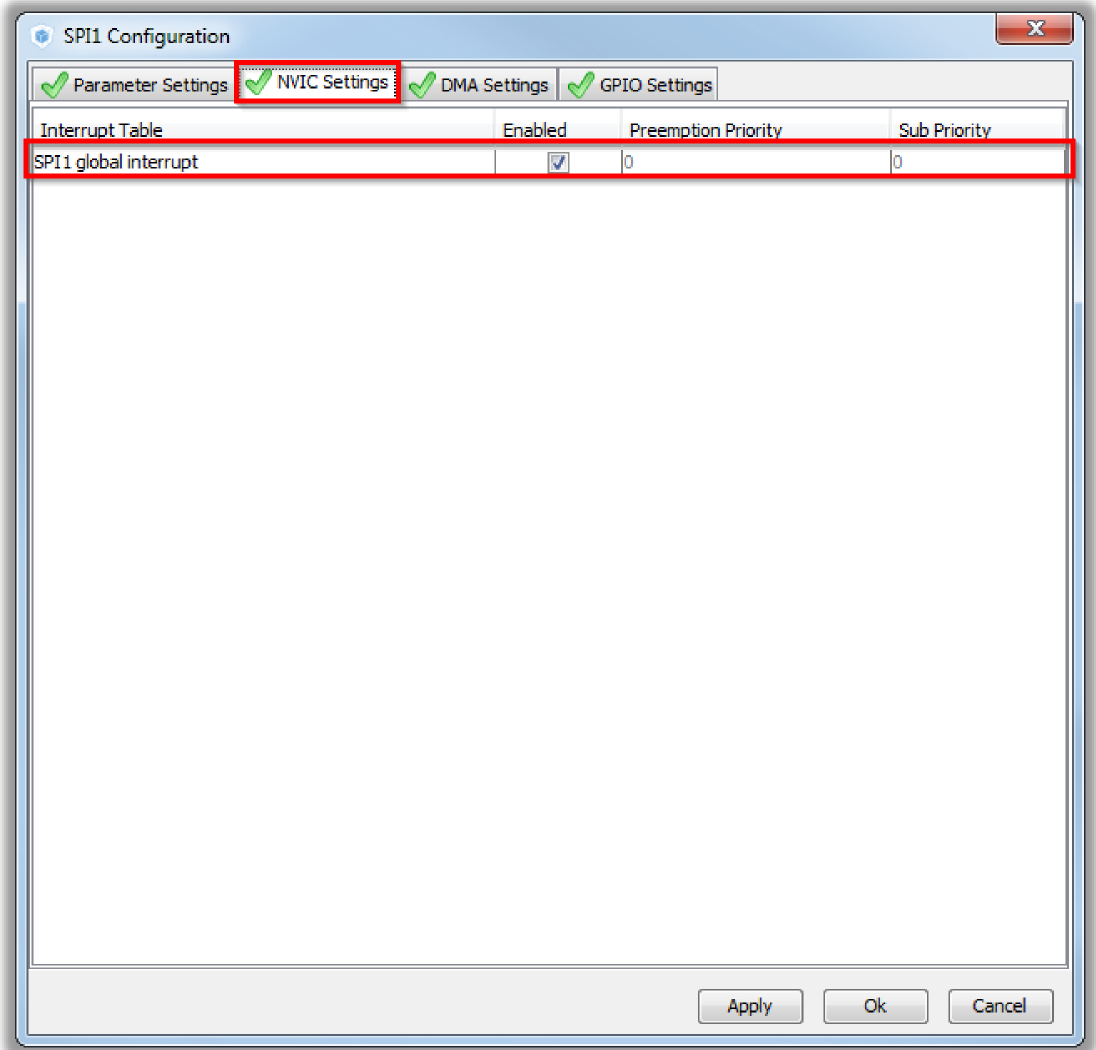
2.2.2 Use SPI with interrupt

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



2.2.2 Use SPI with interrupt

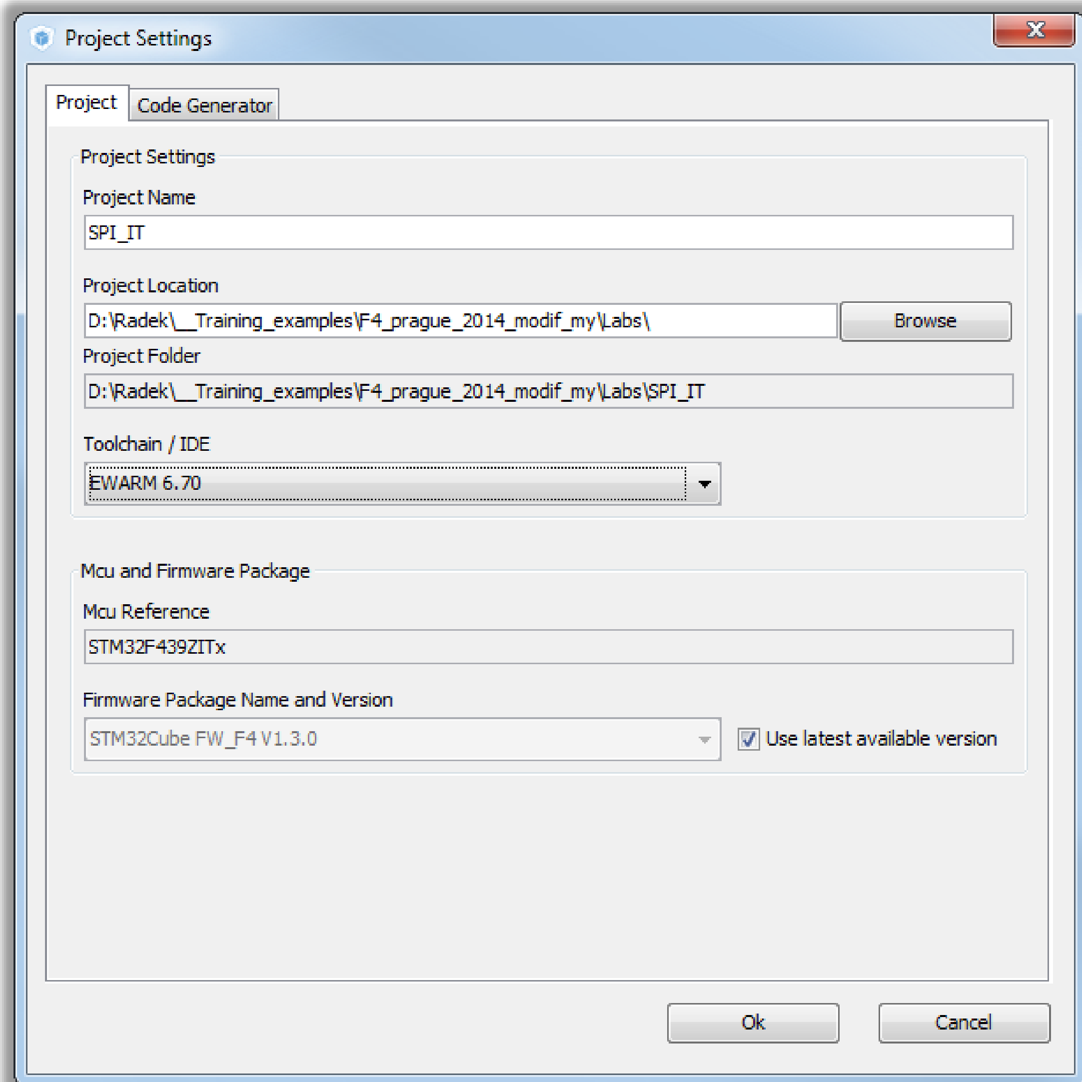
- CubeMX SPI configuration
  - TAB>NVIC Settings
  - Enable SPI interrupt
  - Button OK



life.augmented

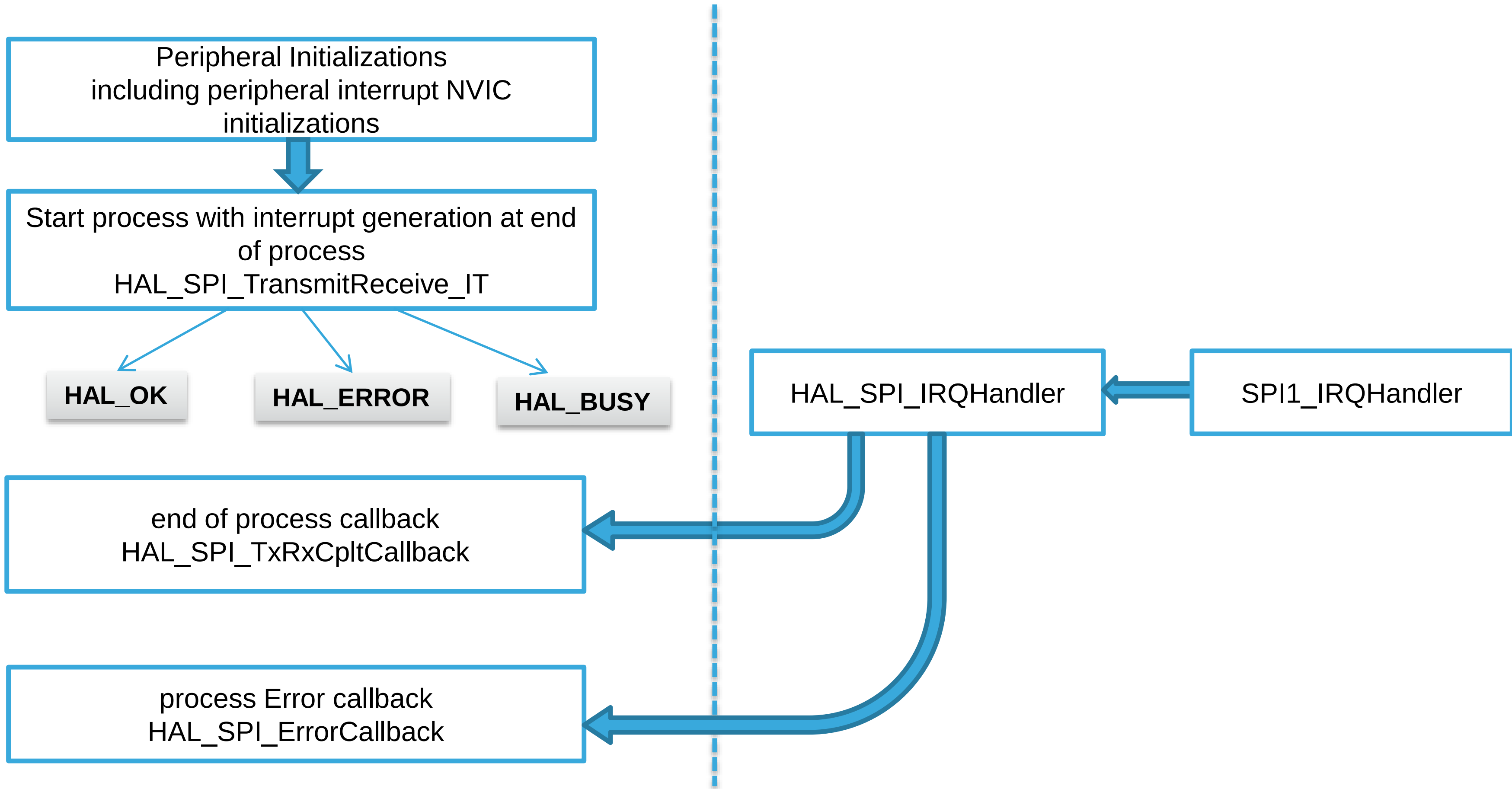
2.2.2 Use SPI with interrupt

- 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



2.2.2 Use SPI with interrupt

HAL Library SPI with IT transmit receive flow



2.2.2 Use SPI with interrupt

- 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)

2.2.2 Use SPI with interrupt

- 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 */
```

2.2.2 Use SPI with interrupt

- 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 */
```

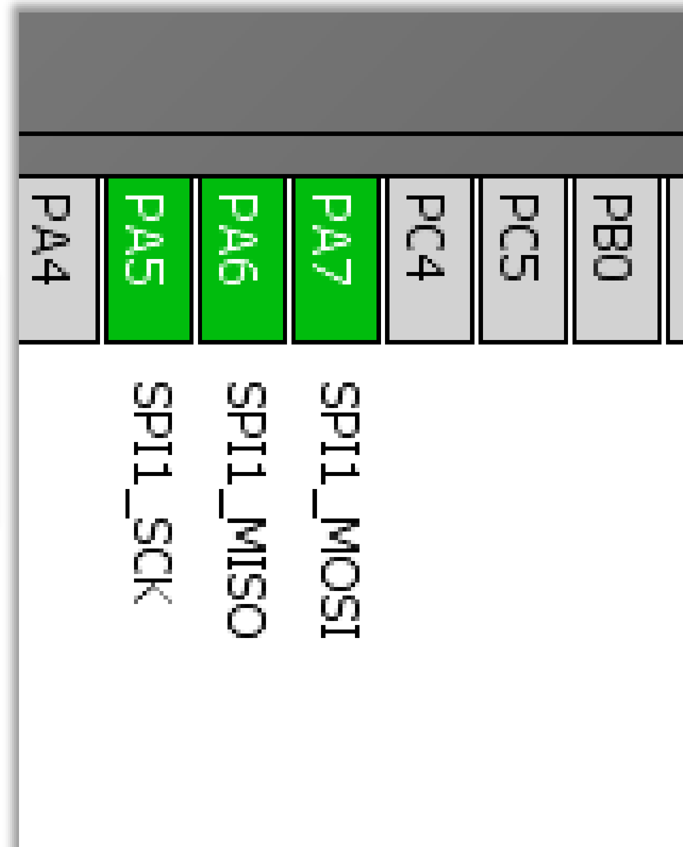
2.2.3 SPI DMA lab

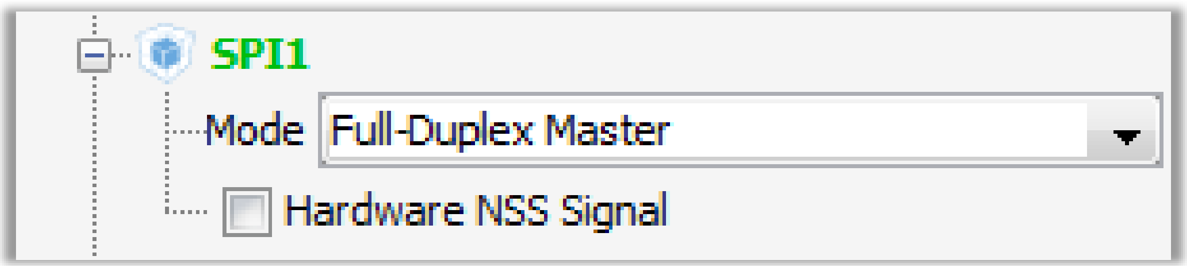
2.2.3 Use SPI with DMA transfer

- 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

2.2.3 Use SPI with DMA transfer

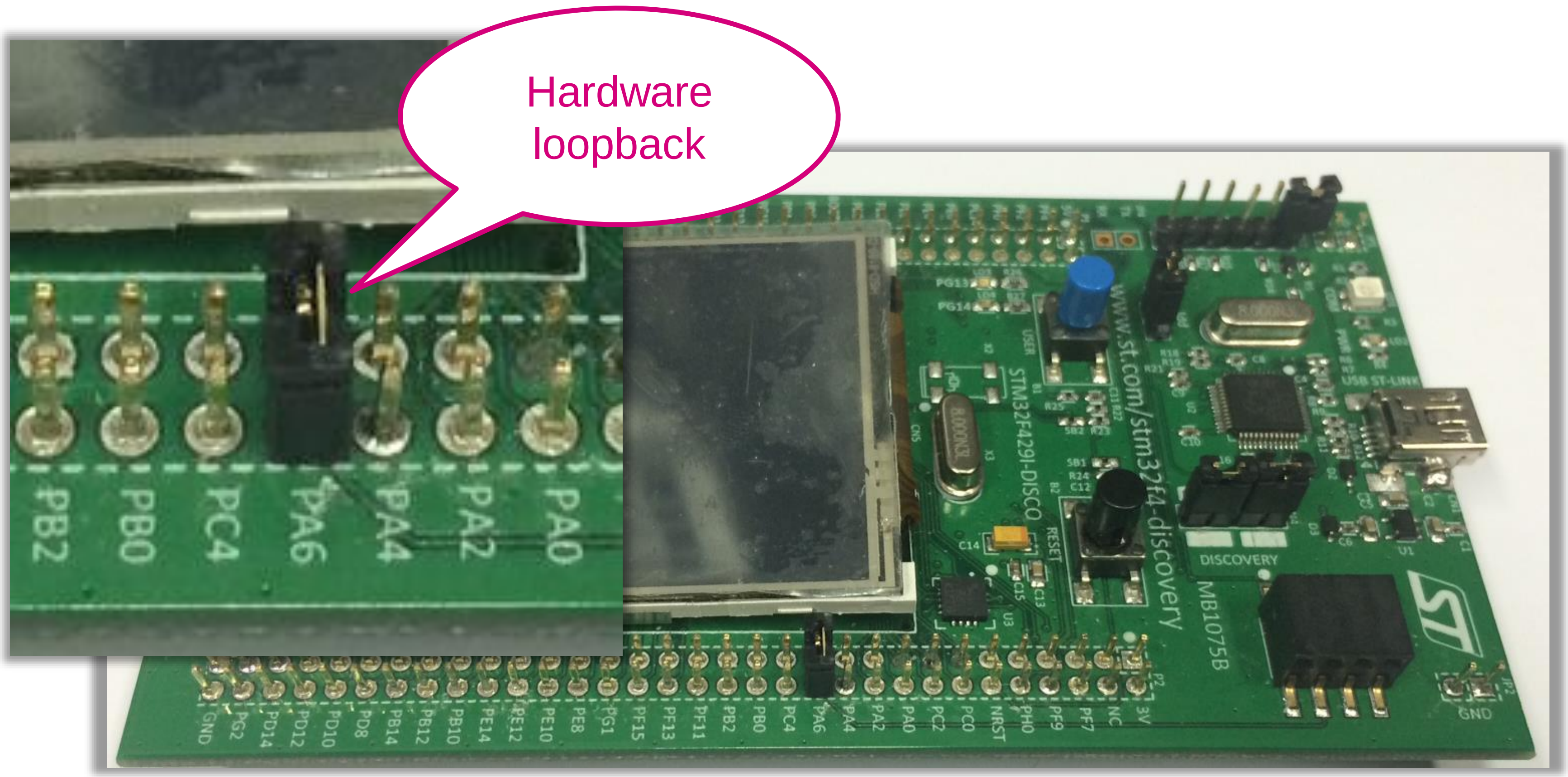
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX SPI selection
  - Select SPI1 Full-Duplex Master
  - Select PA5, PA6, PA7 for SPI1 if weren't selected





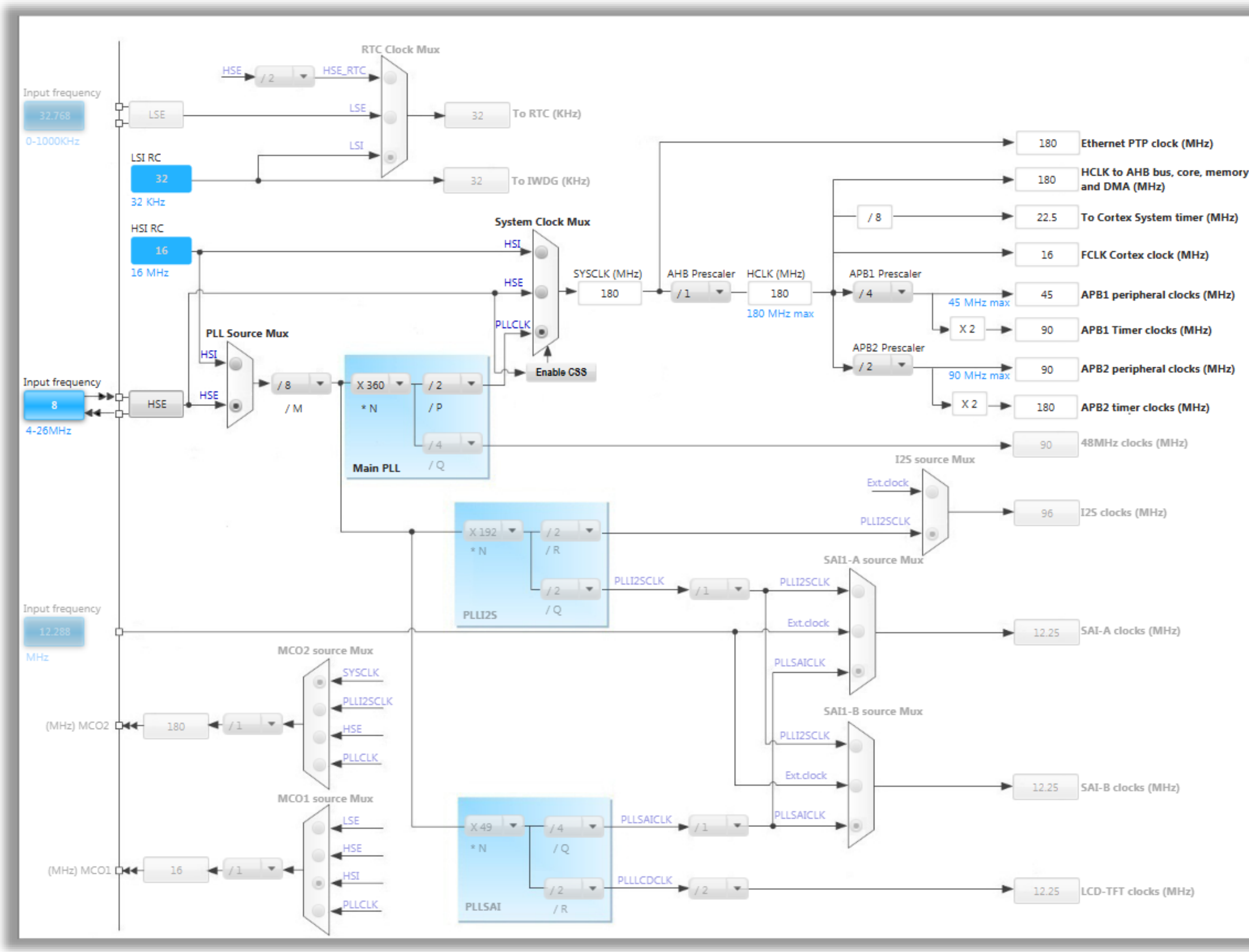
2.2.3 Use SPI with DMA transfer

- Hardware preparation
    - Connect PA6 and PA7 together with jumper



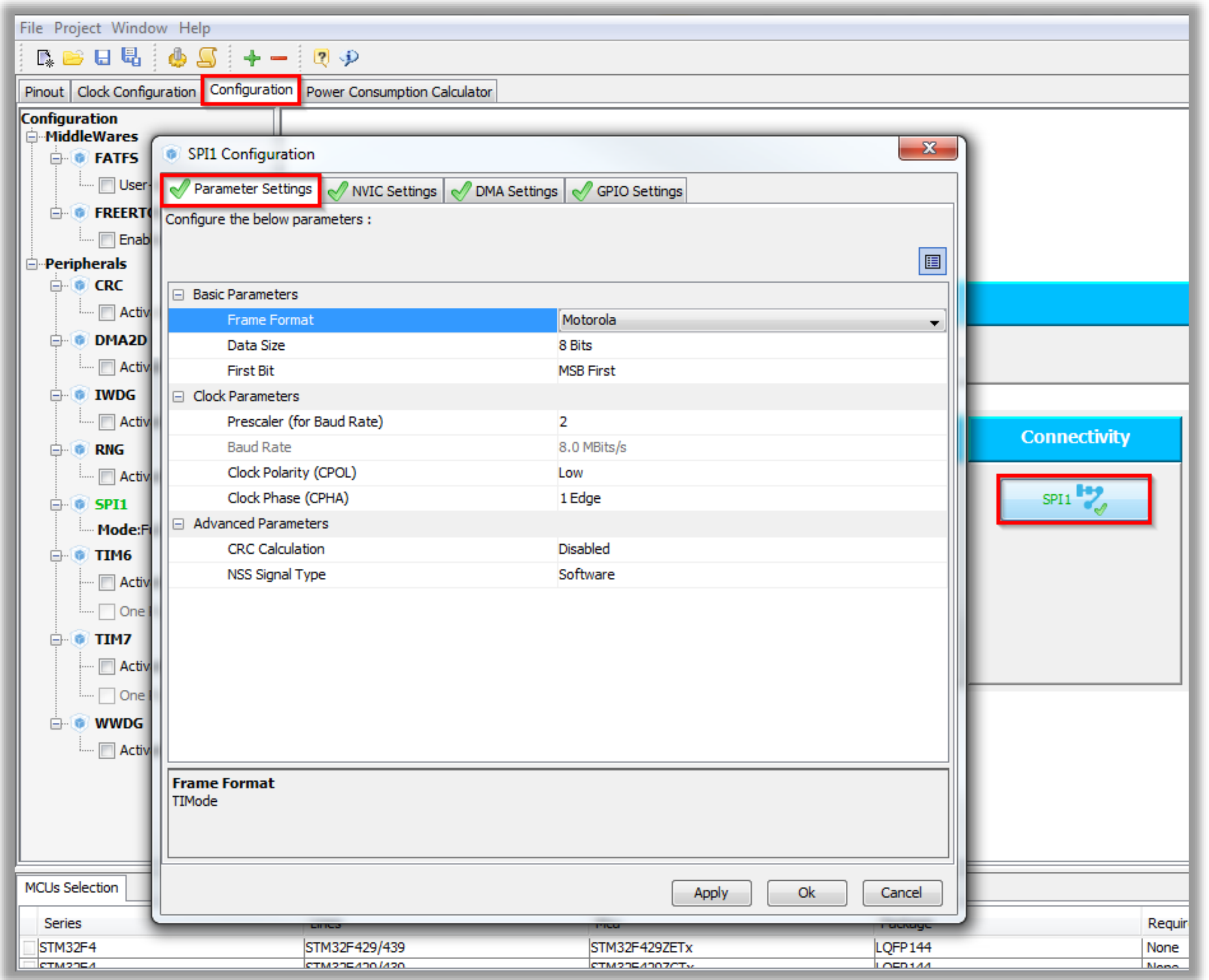
2.2.3 Use SPI with DMA transfer

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



2.2.3 Use SPI with DMA transfer

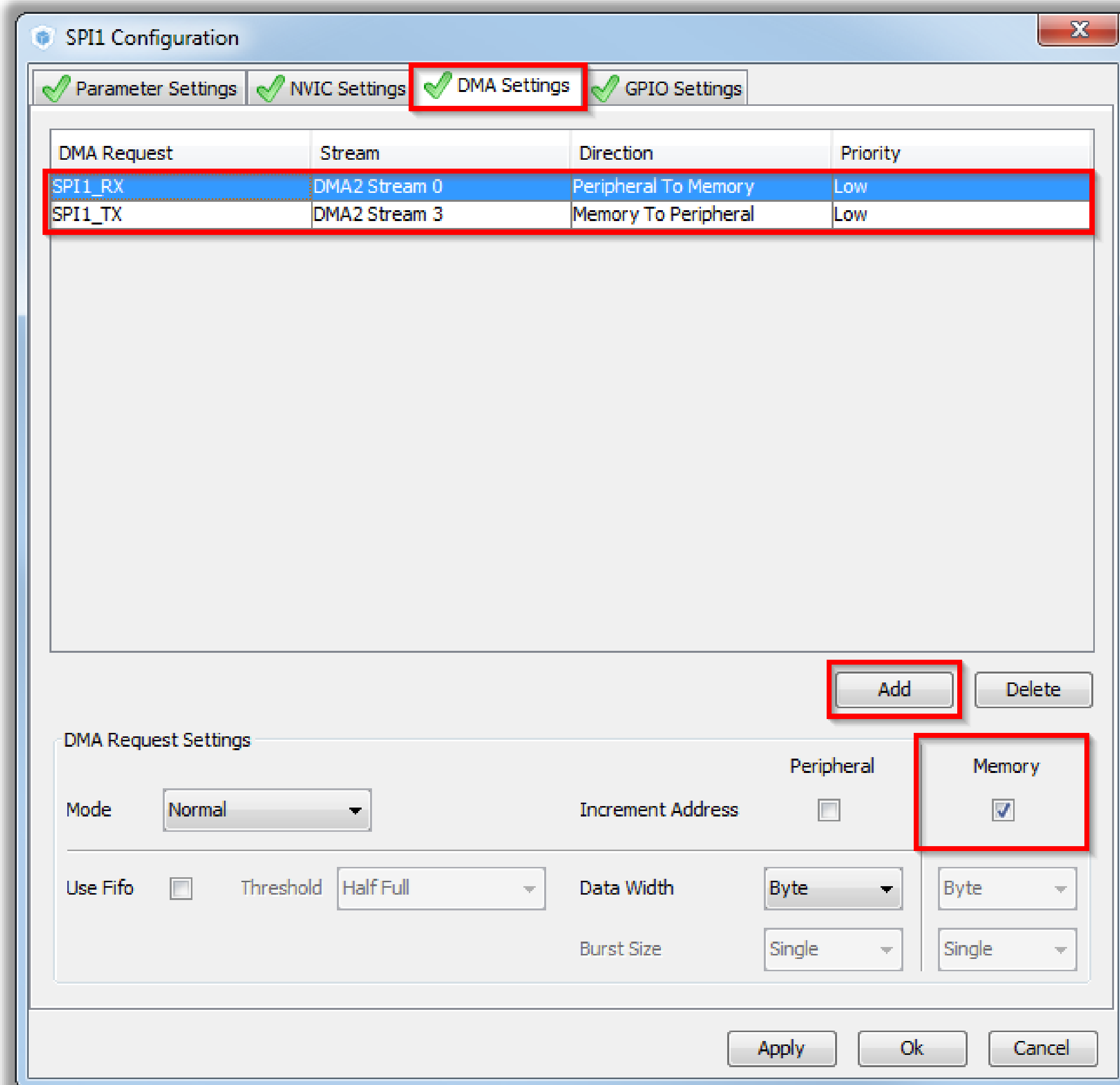
- CubeMX SPI configuration
  - Tab>Configuration>Connectivity>SPI1
  - Check the settings
  - Button OK



life.augmented

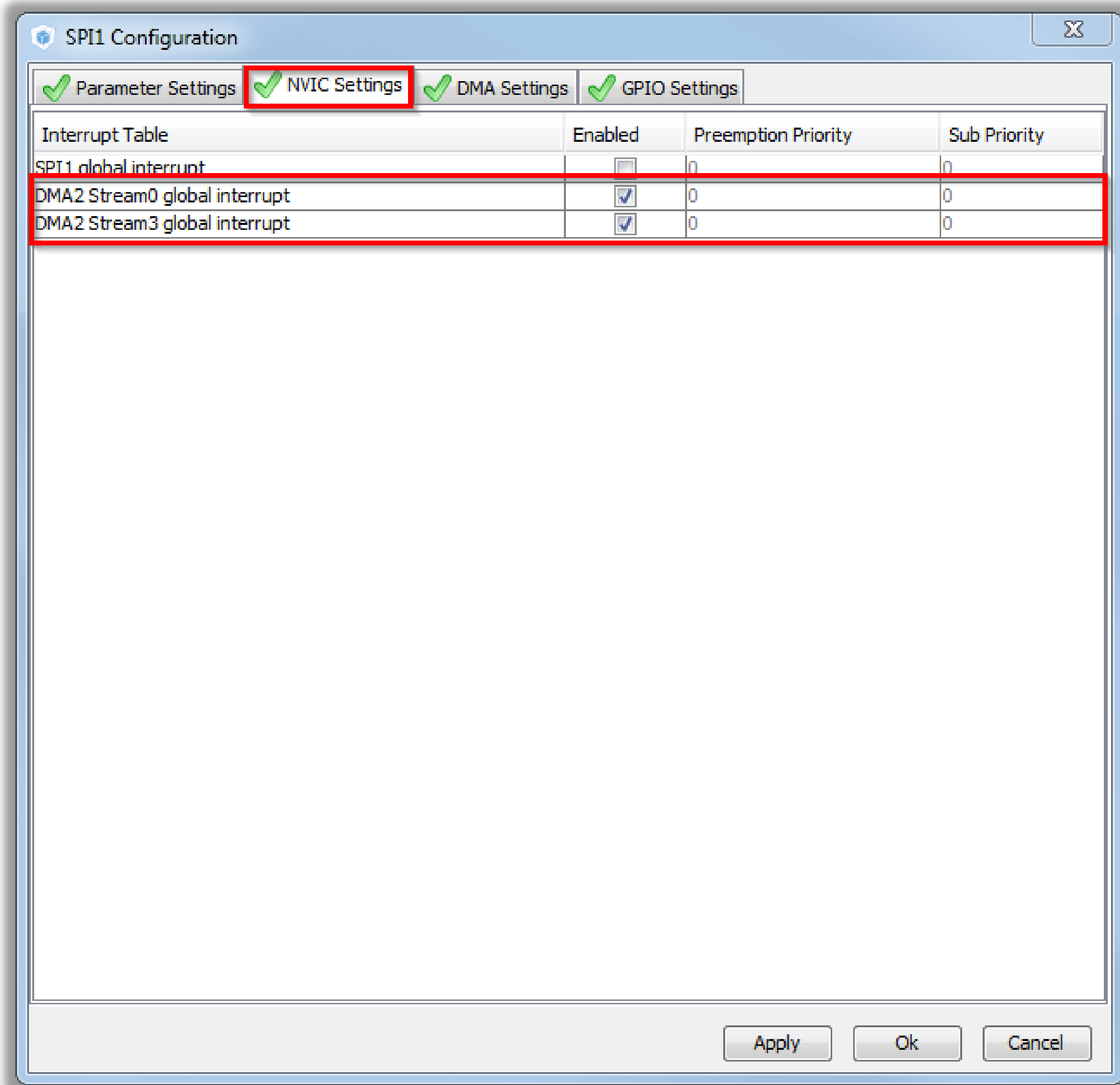
2.2.3 Use SPI with DMA transfer

- CubeMX SPI configuration DMA settings
  - TAB>DMA Settings
  - Button ADD
  - SPI1_RX
  - Memory increment
  - Button ADD
  - SPI1_Tx
  - Memory increment
  - Button OK



2.2.3 Use SPI with DMA transfer

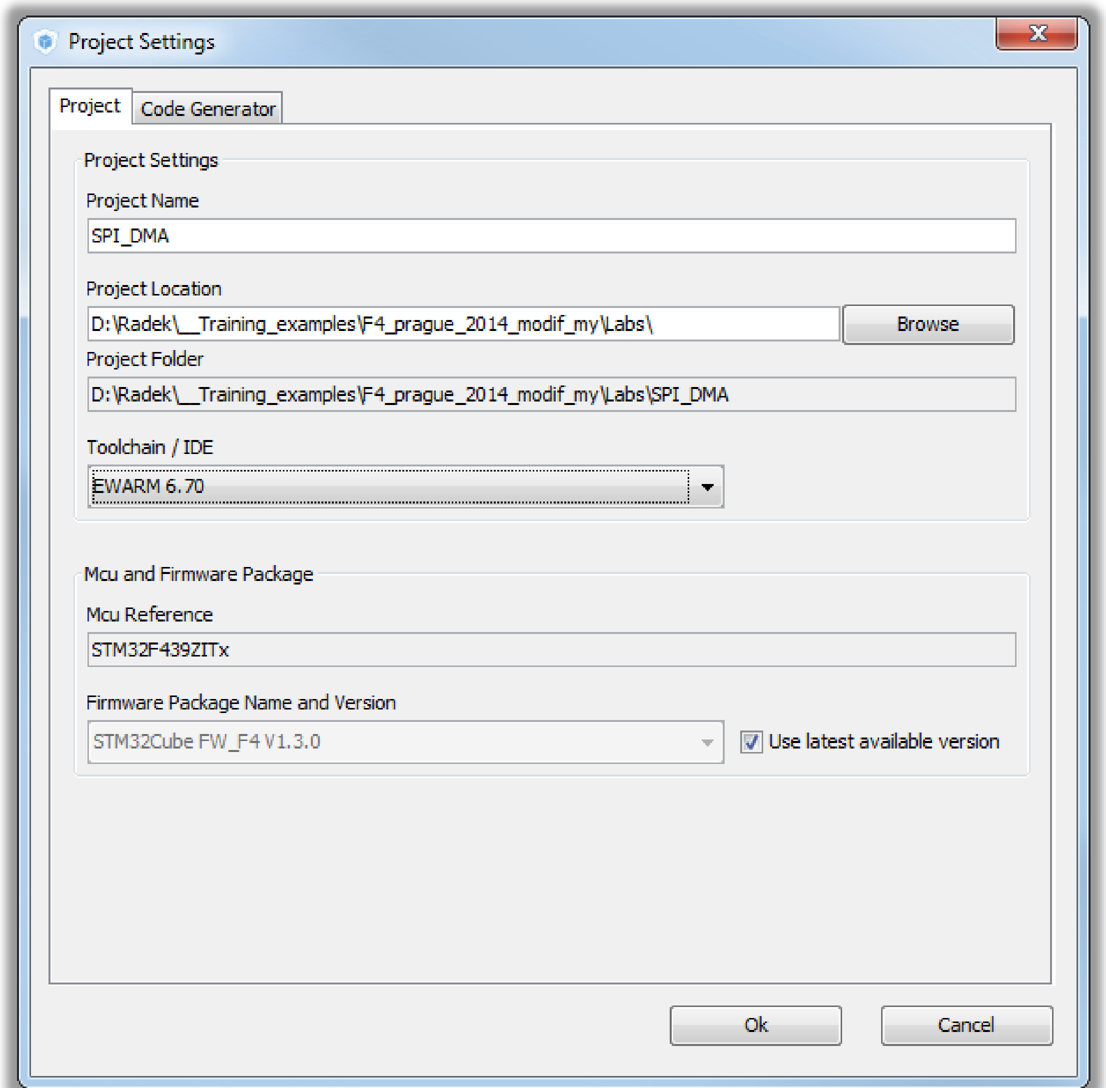
- CubeMX SPI configuration NVIC settings
  - TAB>NVIC Settings
  - Enable DMA2 interrupts for SPI1
  - Button OK



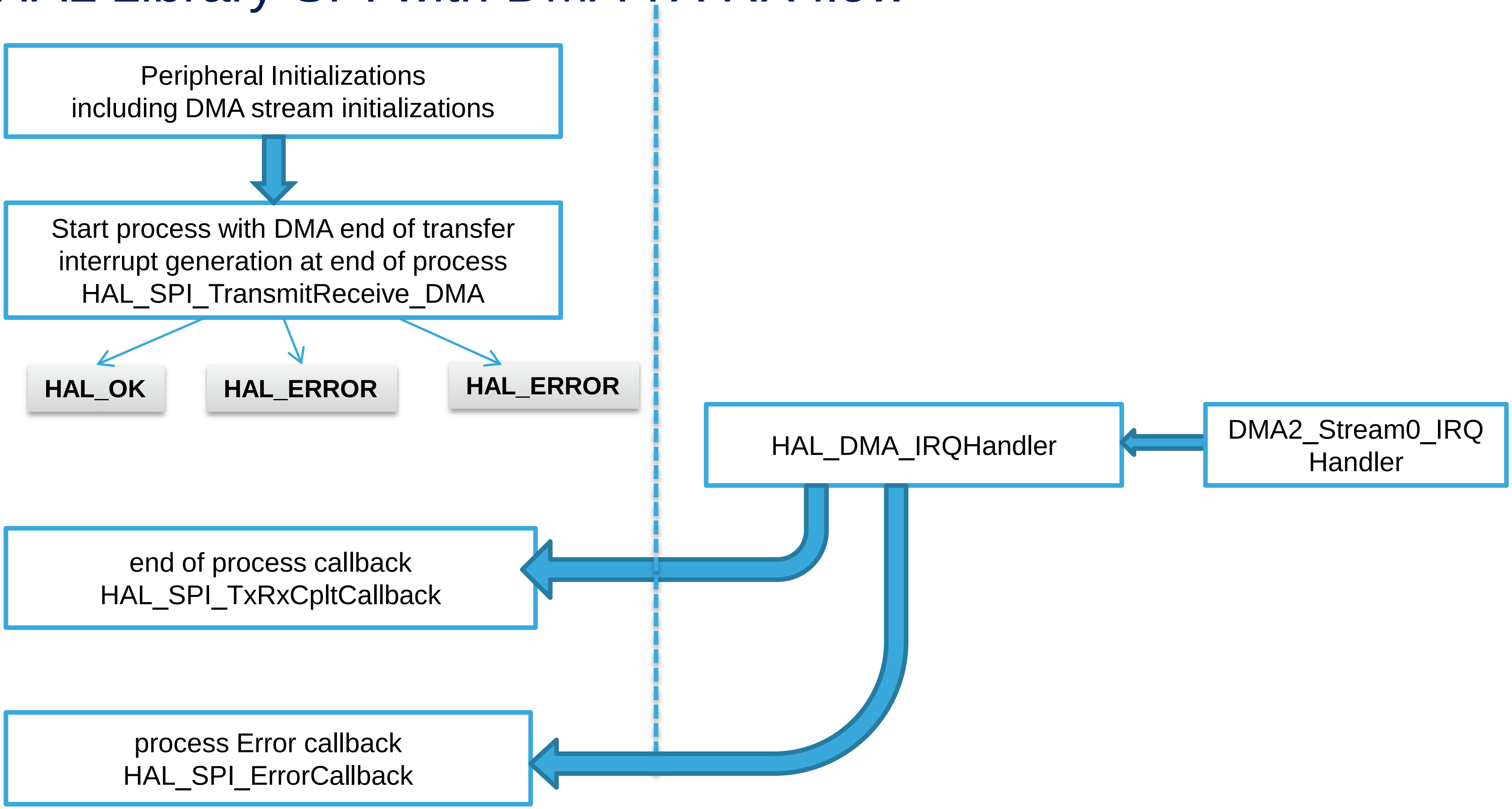
life.augmented

2.2.3 Use SPI with DMA 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



HAL Library SPI with DMA TX RX flow



life.augmented

2.2.3 Use SPI with DMA transfer

- 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)

2.2.3 Use SPI with DMA transfer

- 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 */
```

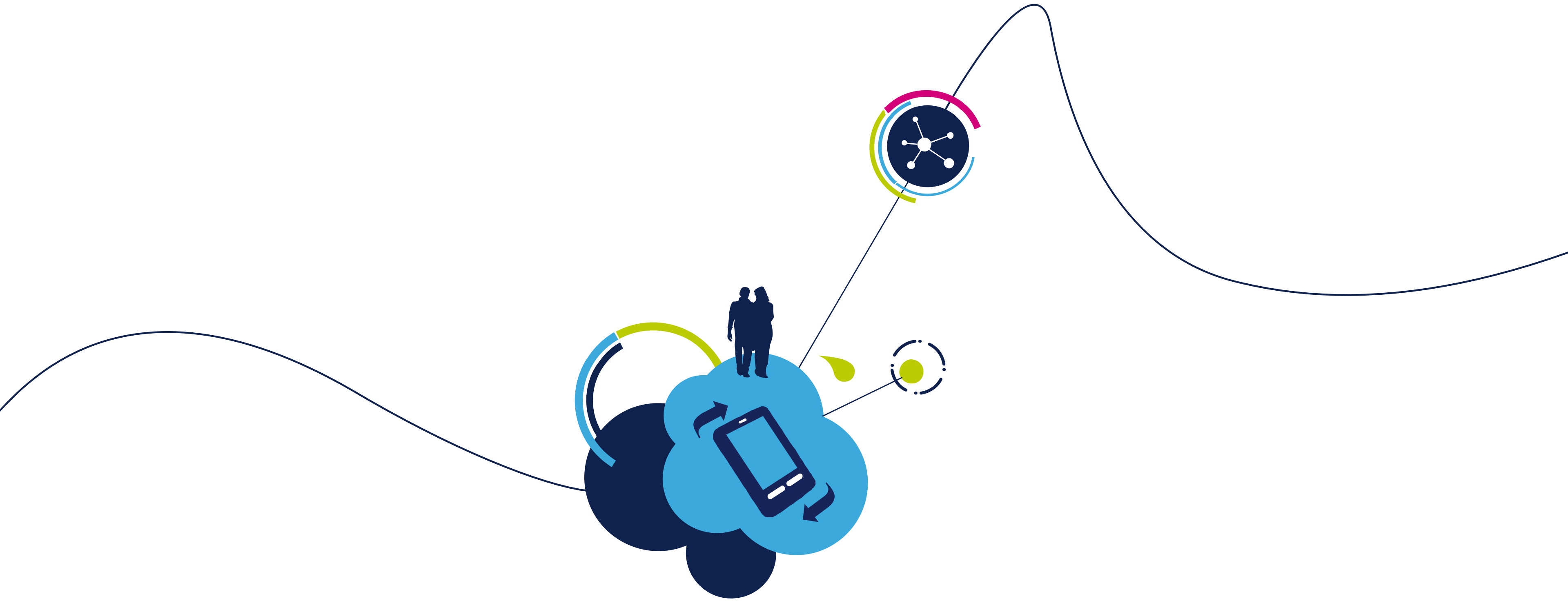
- Sending and receiving methods

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

2.2.3 Use SPI with DMA transfer

- 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 */
```



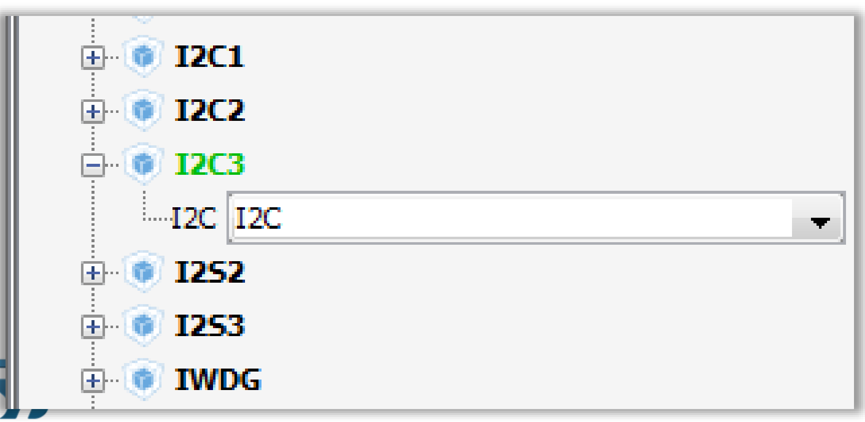
2.3.1 I2C Poll lab

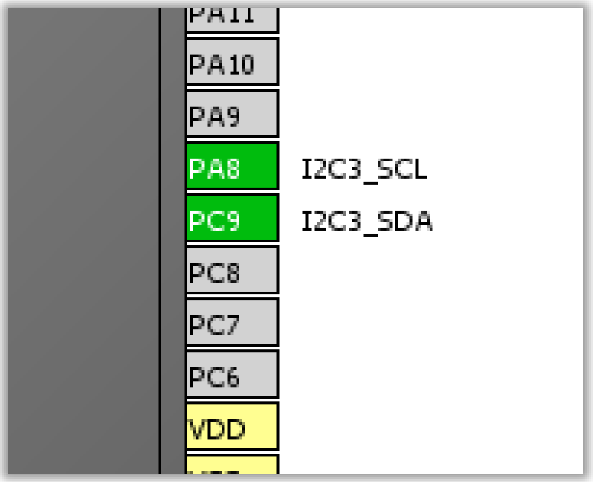
2.3.1 Simple I2C communication

- Objective
  - Learn how to setup i2C in CubeMX
  - How to Generate Code in CubeMX and use HAL functions
  - Work in pairs, one will create I2C transmitter and second I2C receiver
- Goal
  - Configure I2C in CubeMX and Generate Code
  - Learn how to send and receive data over I2C without interrupts
  - Verify the correct functionality

2.3.1 Simple I2C communication

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX SPI selection
  - Select I2C3 Periphery
  - Select PA8, PC9 for I2C3 if weren't selected

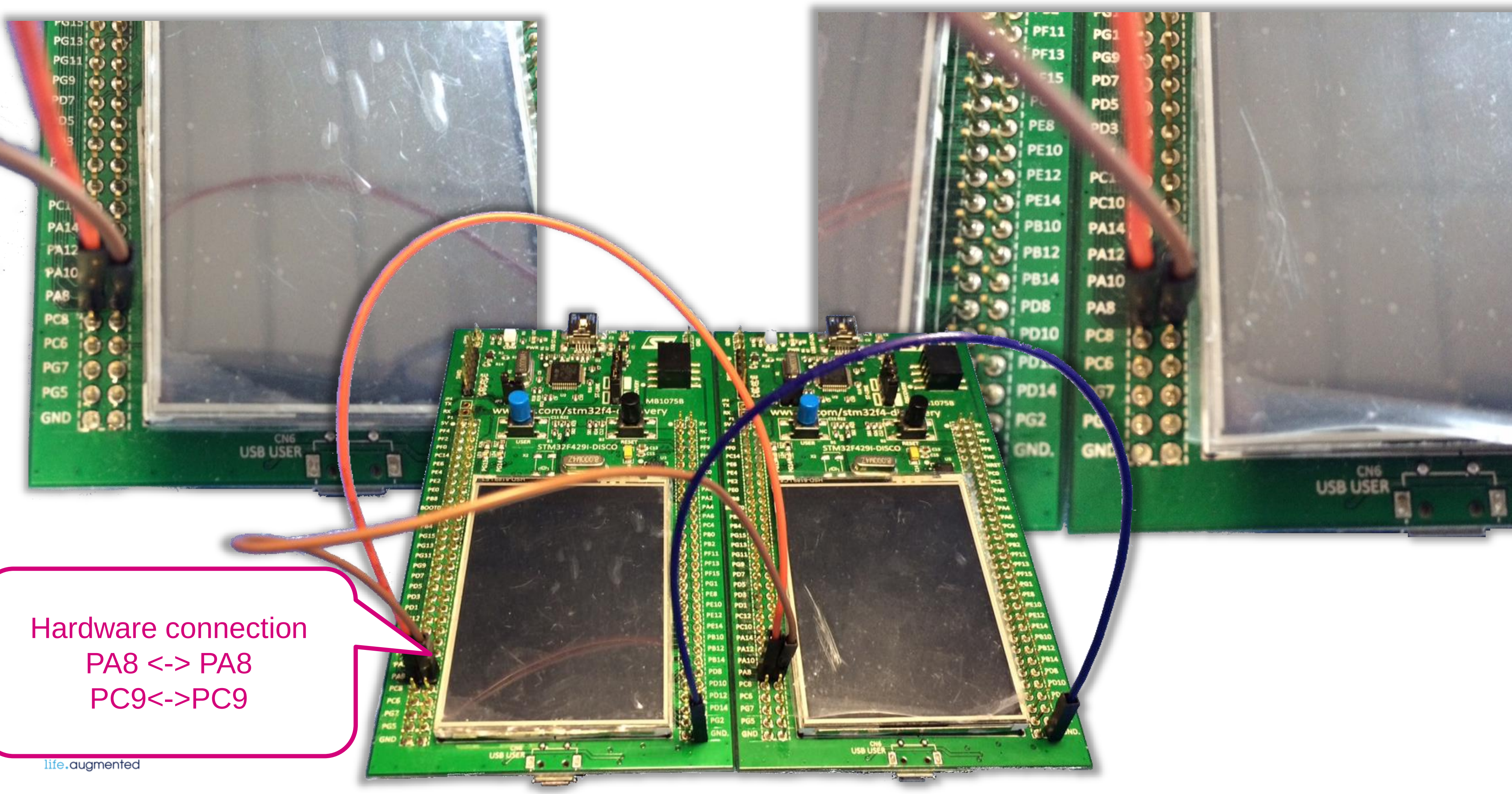




life.augmented

2.3.1 Simple I2C communication

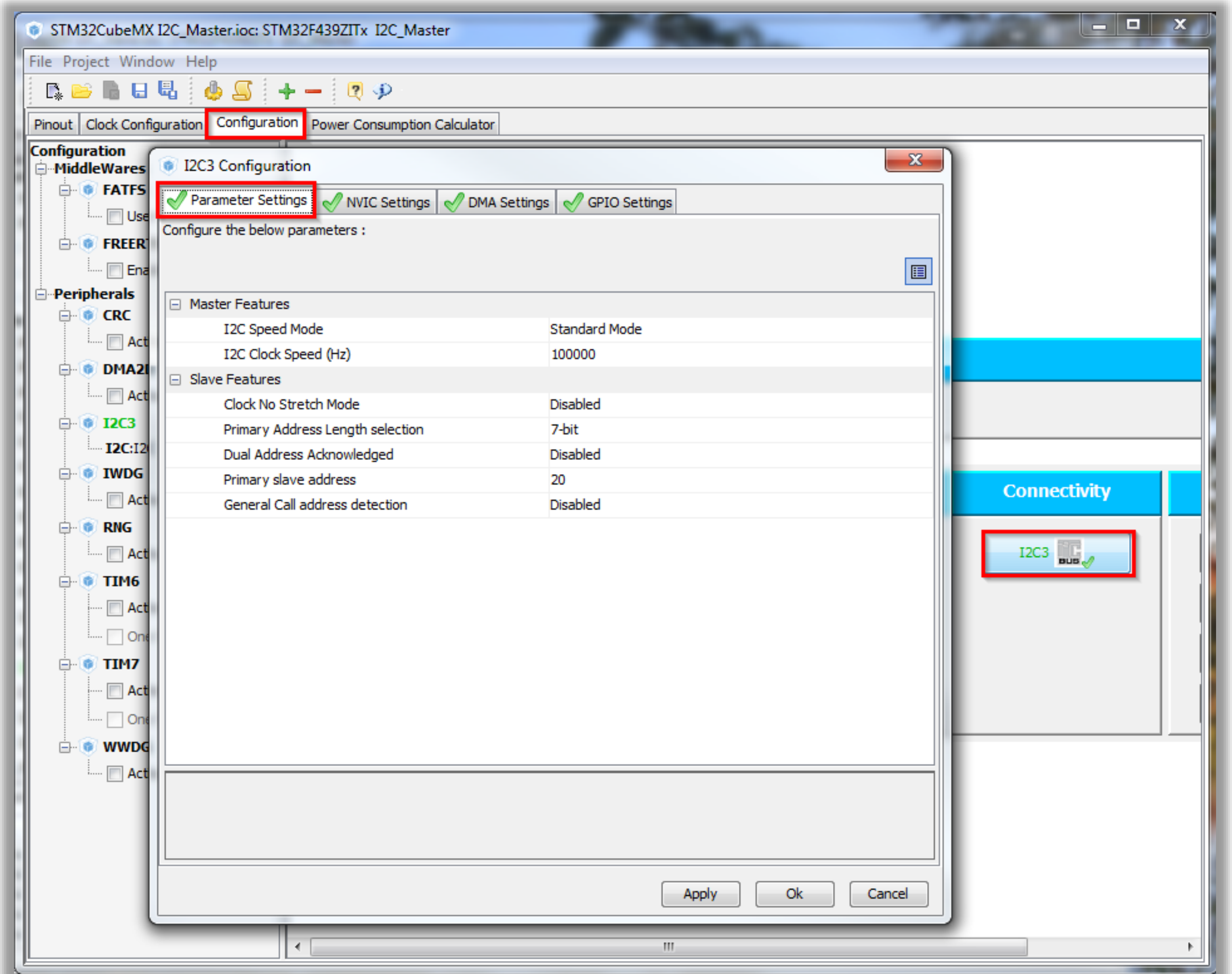
- Hardware preparation
  - Connect together PA8 and PC9 pins from both boards, and GND



Hardware connection
PA8 <-> PA8
PC9<->PC9

2.3.1 Simple I2C communication

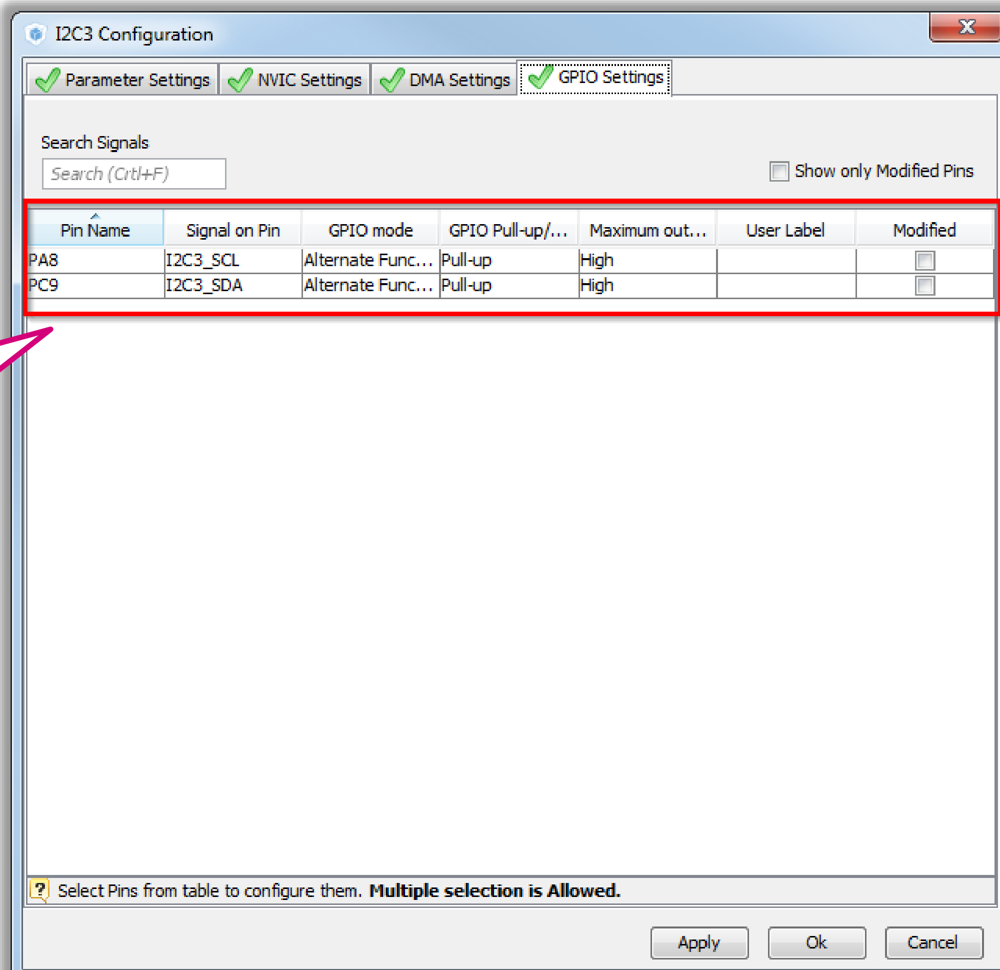
- CubeMX I2C configuration
  - Tab>Configuration>Connectivity>I2C3
  - Check the settings
  - Define Slave address to 20
  - Button OK



life.augmented

2.3.1 Simple I2C communication

- CubeMX I2C configuration
  - Tab>GPIO Settings
  - Check Pull-UP
  - Check GPIO speed

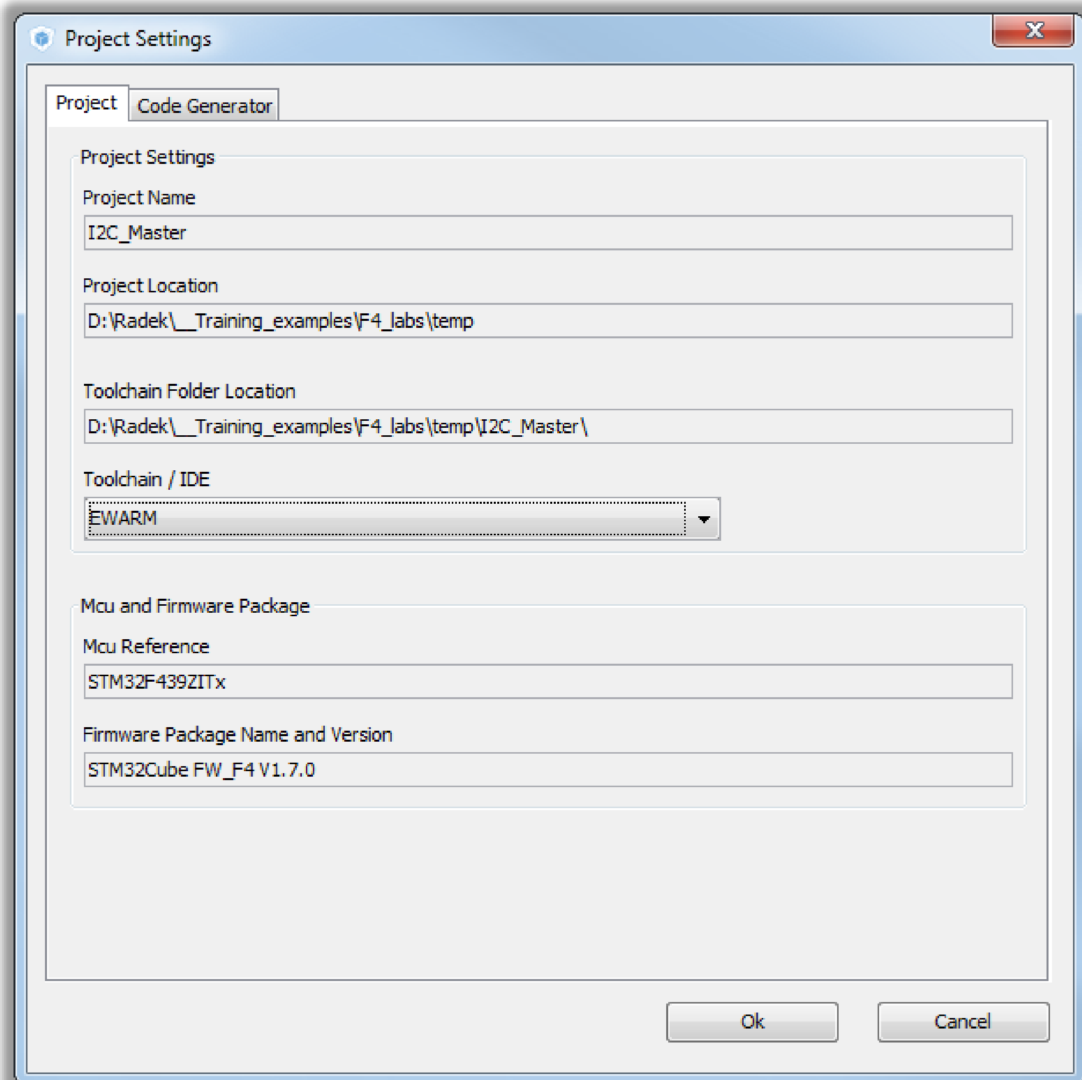


Check if the Pull-UP resistor are necessary in case you are using external one
Check the GPIO speed
This two parameters can limit I2C maximum speed

life.augmented

2.3.1 Simple I2C communication

- 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



life.augmented

2.3.1 Simple I2C communication

- Buffer definition

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

- Master simple send and receive functions

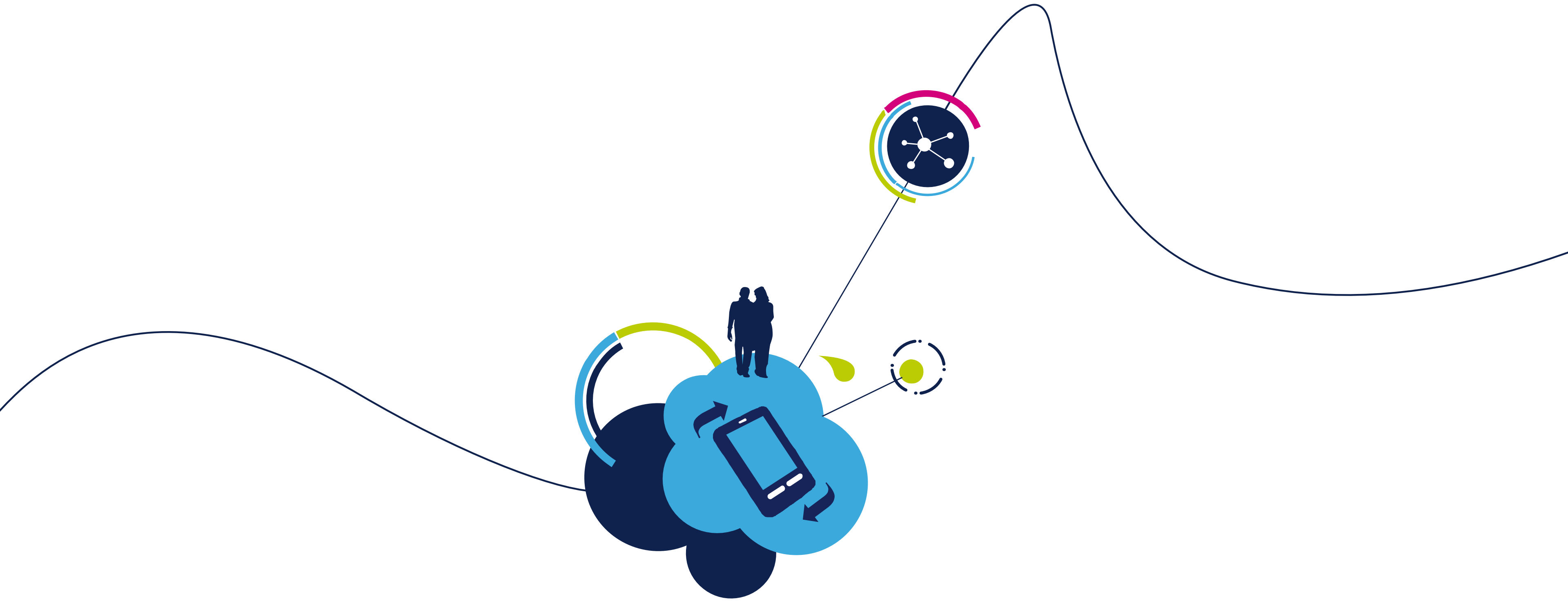
```
/* USER CODE BEGIN 2 */
 HAL_Delay(100);
 HAL_I2C_Master_Transmit(&hi2c3,20,data_tx,10,1000);
 HAL_Delay(2);
 HAL_I2C_Master_Receive(&hi2c3,20,data_rx,10,1000);
 /* USER CODE END 2 */
```

2.3.1 Simple I2C communication

- Slave simple send and receive functions

```
 /* USER CODE BEGIN 2 */
status1=HAL_I2C_Slave_Receive(&hi2c3,data_rx,10,10000);
status2=HAL_I2C_Slave_Transmit(&hi2c3,data_rx,10,10000);
  /* USER CODE END 2 */
```

- During this example the SLAVE must begin the code execution first to be prepared on MASTER Receive/Send requests



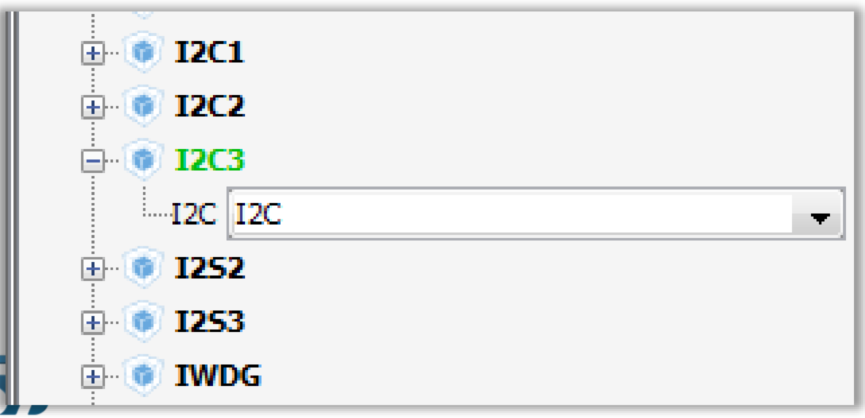
2.3.2 I2C Interrupt lab

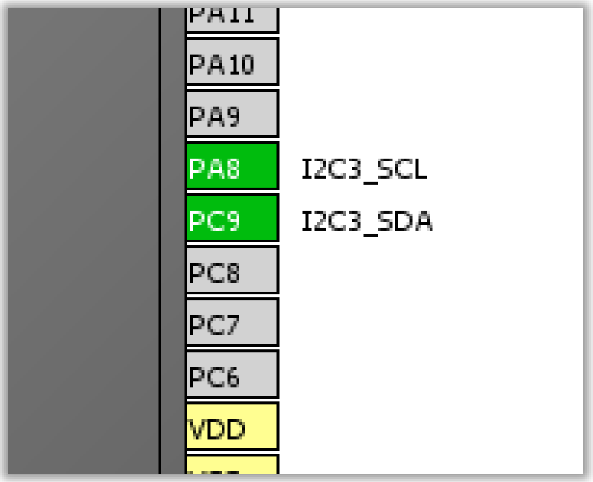
2.3.2 Use I2C with interrupt

- Objective
  - Learn how to setup I2C with interrupts in CubeMX
  - How to Generate Code in CubeMX and use HAL functions
  - Create simple loopback example with interrupts
- Goal
  - Configure I2C in CubeMX and Generate Code
  - Learn how to send and receive data over I2C with interrupts
  - Verify the correct functionality

2.3.2 Use I2C with interrupt

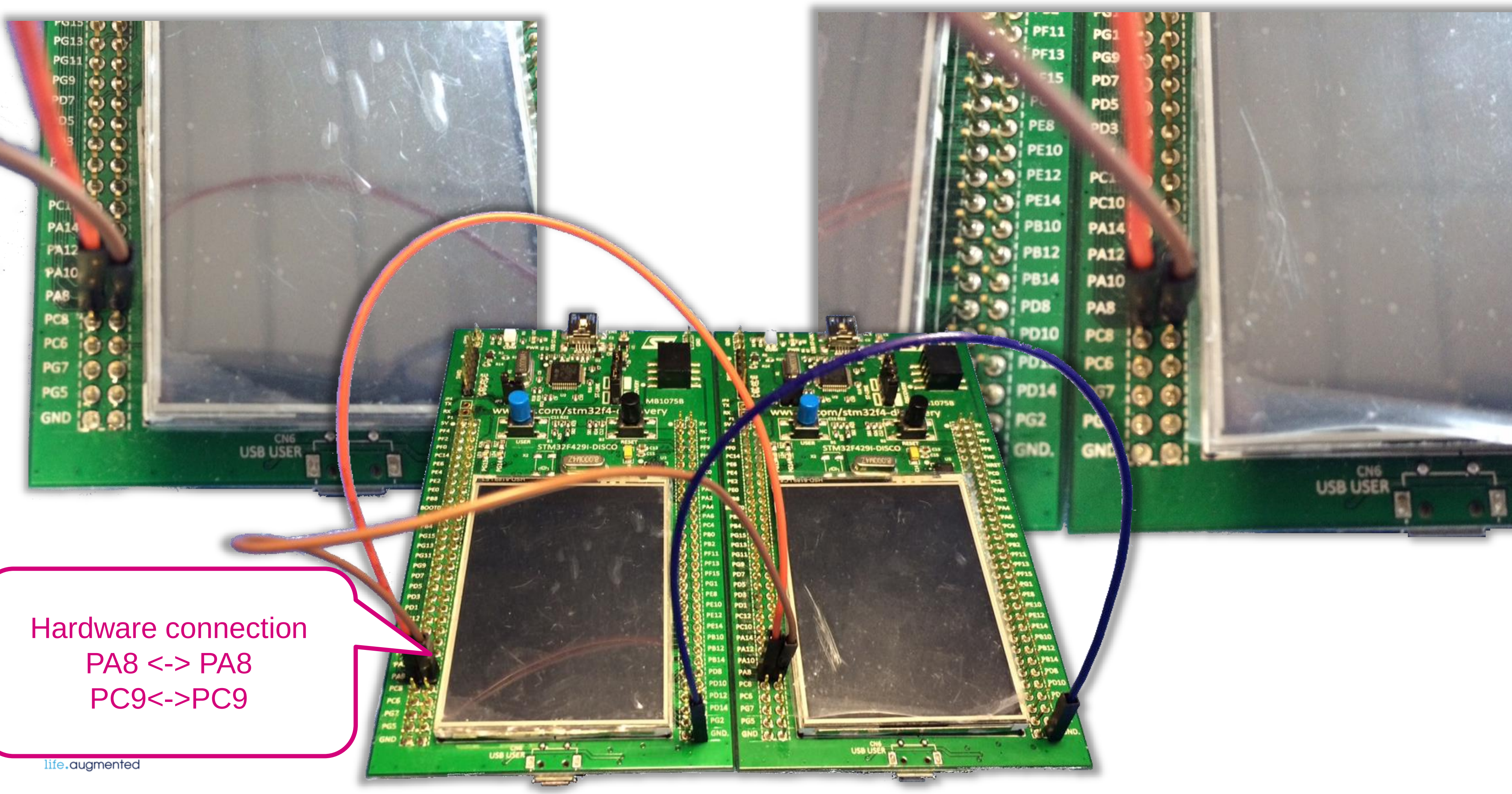
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX SPI selection
  - Select I2C3 Periphery
  - Select PA8, PC9 for I2C3 if weren't selected





2.3.2 Use I2C with interrupt

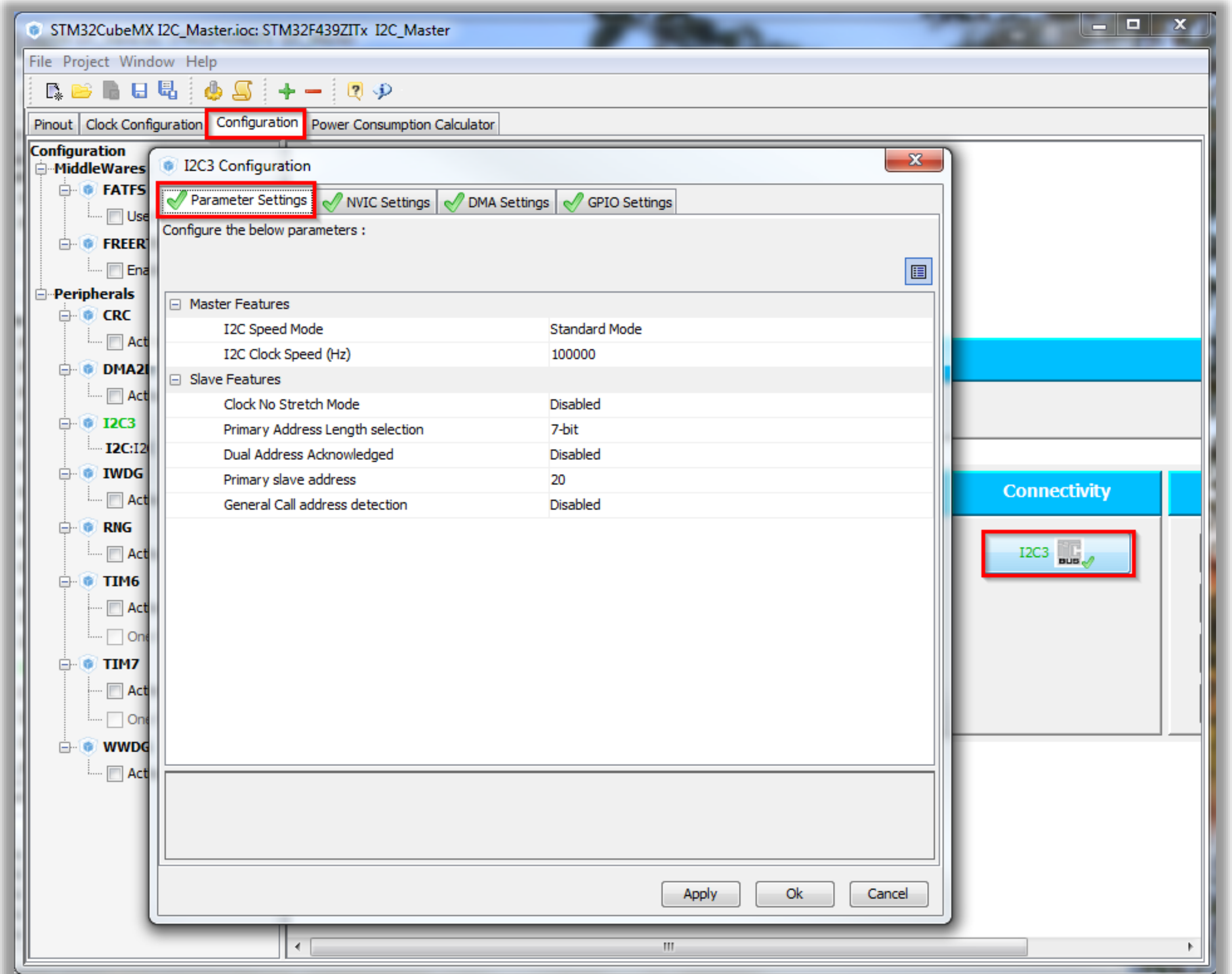
- Hardware preparation
  - Connect together PA8 and PC9 pins from both boards, and GND



Hardware connection
PA8 <-> PA8
PC9<->PC9

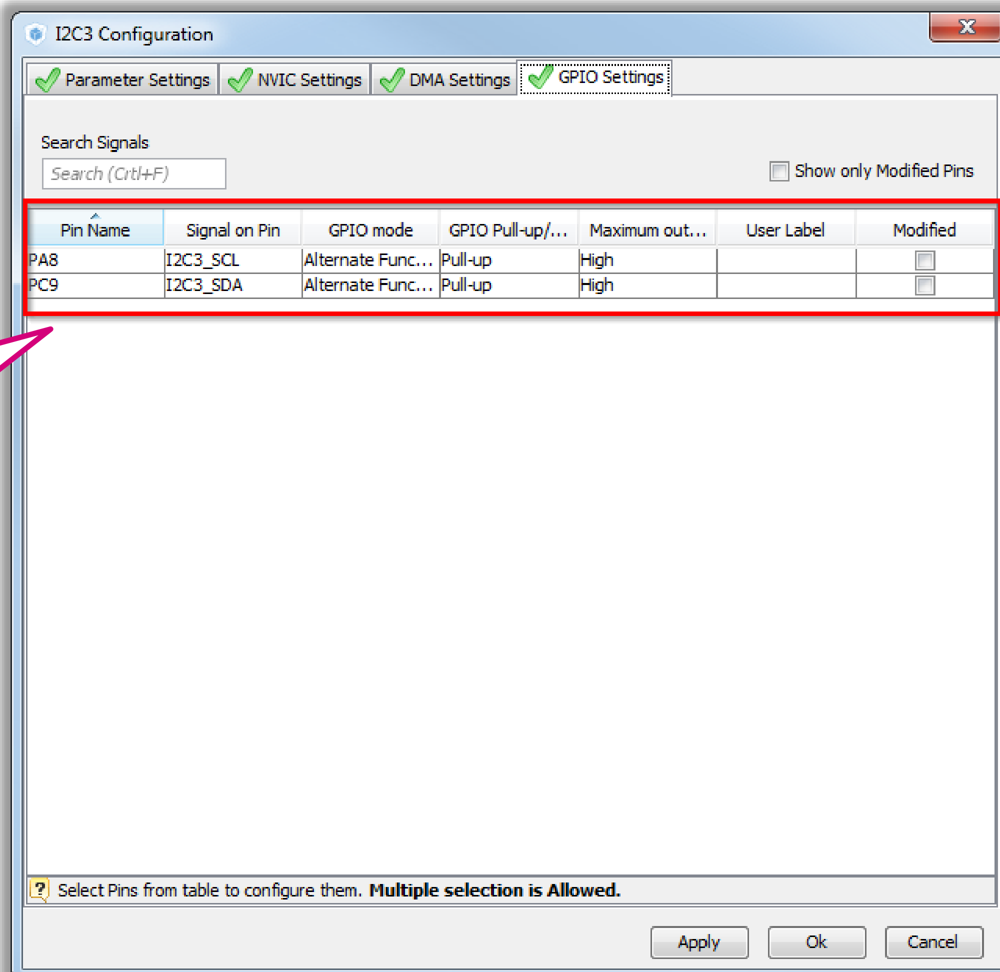
2.3.2 Use I2C with interrupt

- CubeMX I2C configuration
  - Tab>Configuration>Connectivity>I2C3
  - Check the settings
  - Define Slave address to 20
  - Button OK



2.3.2 Use I2C with interrupt

- CubeMX I2C configuration
  - Tab>GPIO Settings
  - Check Pull-UP
  - Check GPIO speed

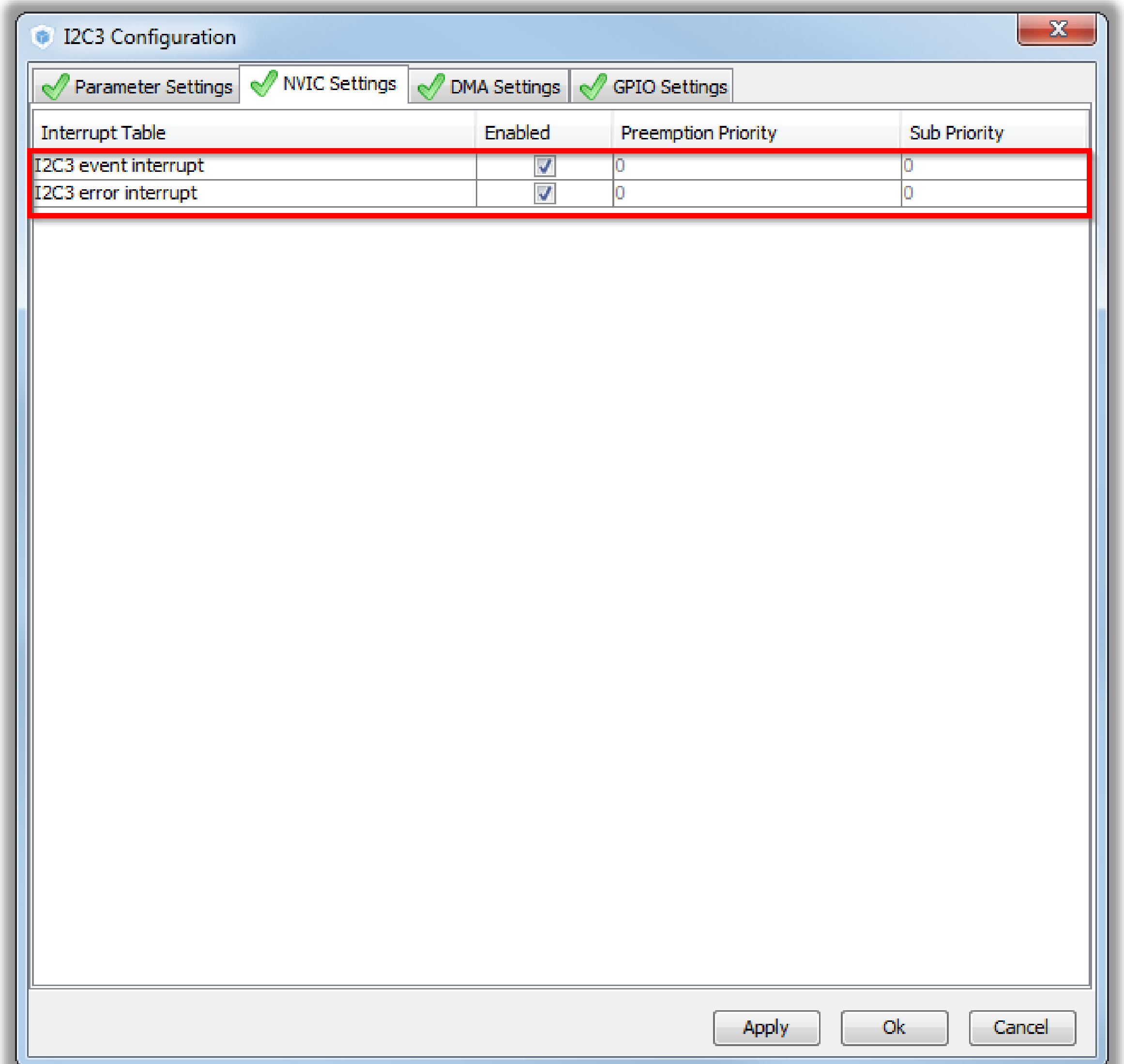


Check if the Pull-UP resistor are necessary in case you are using external one
Check the GPIO speed
This two parameters can limit I2C maximum speed

life.augmented

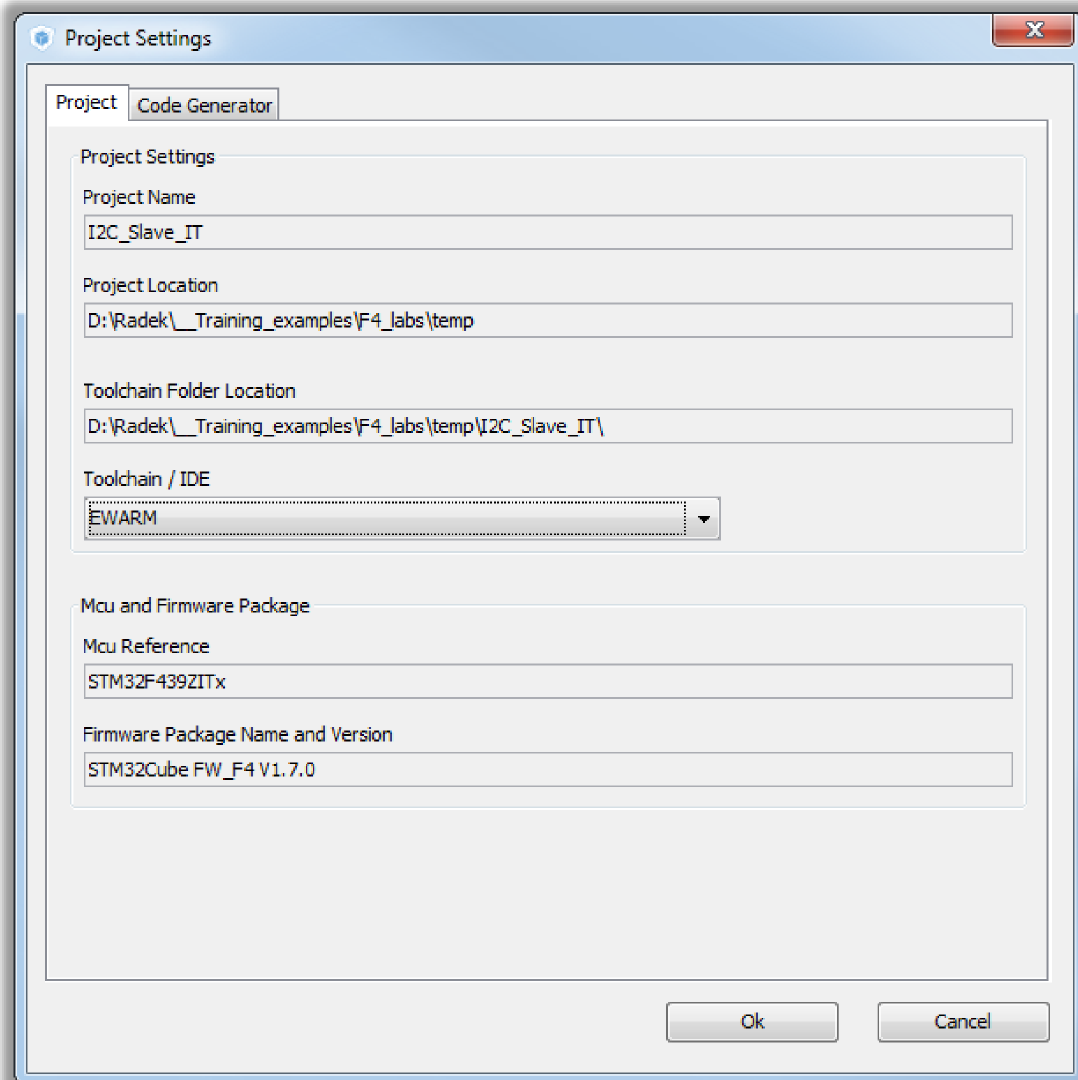
2.3.2 Use I2C with interrupt

- CubeMX I2C configuration
  - Tab>NVIC settings
  - Enable interrupts
  - Button OK



2.3.2 Use I2C 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



life.augmented

2.3.2 Use I2C with interrupt

- Buffer definition

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

- Master simple send and receive functions with interrupts

```
 /* USER CODE BEGIN 2 */
HAL_I2C_Master_Transmit_IT(&hi2c3,20,data_tx,10);
HAL_Delay(2);
HAL_I2C_Master_Receive_IT(&hi2c3,20,data_rx,10);
   /* USER CODE END 2 */
```

- Master callbacks

```
/* USER CODE BEGIN 4 */
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c){}
void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c){}
/* USER CODE END 4 */
```

2.3.2 Use I2C with interrupt

- Slave receive

```
/* USER CODE BEGIN 2 */
HAL_I2C_Slave_Receive_IT(&hi2c3,data_rx,10);
/* USER CODE END 2 */
```

- Slave transmit in callback

```
/* USER CODE BEGIN 4 */
void HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef *hi2c){
}

void HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c){
  HAL_I2C_Slave_Transmit_IT(&hi2c3,data_rx,10);
}
/* USER CODE END 4 */
```

2.3.3 I2C DMA lab

2.3.3 Use I2C with DMA transfer

- 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

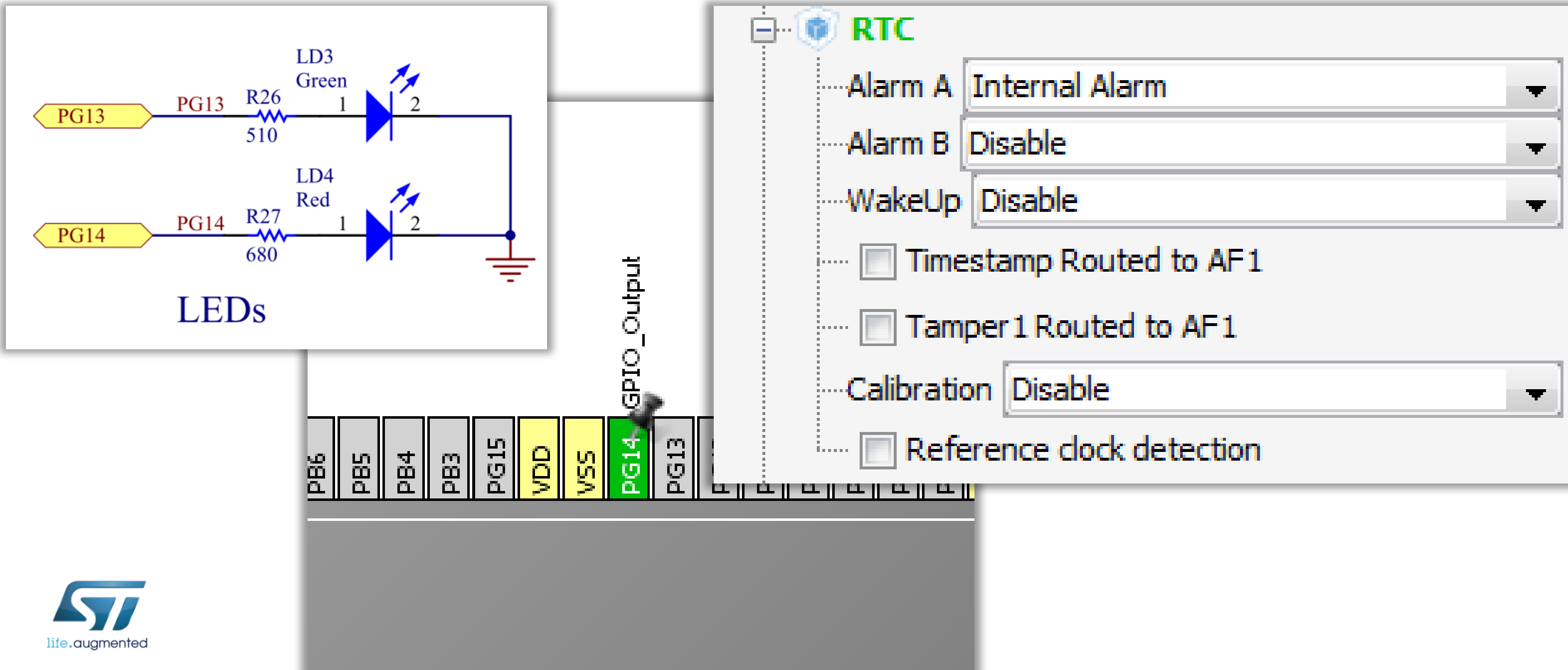
3.1.1 Use RTC Alarm lab

3.1.1 Use RTC and Alarm with interrupt

- 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

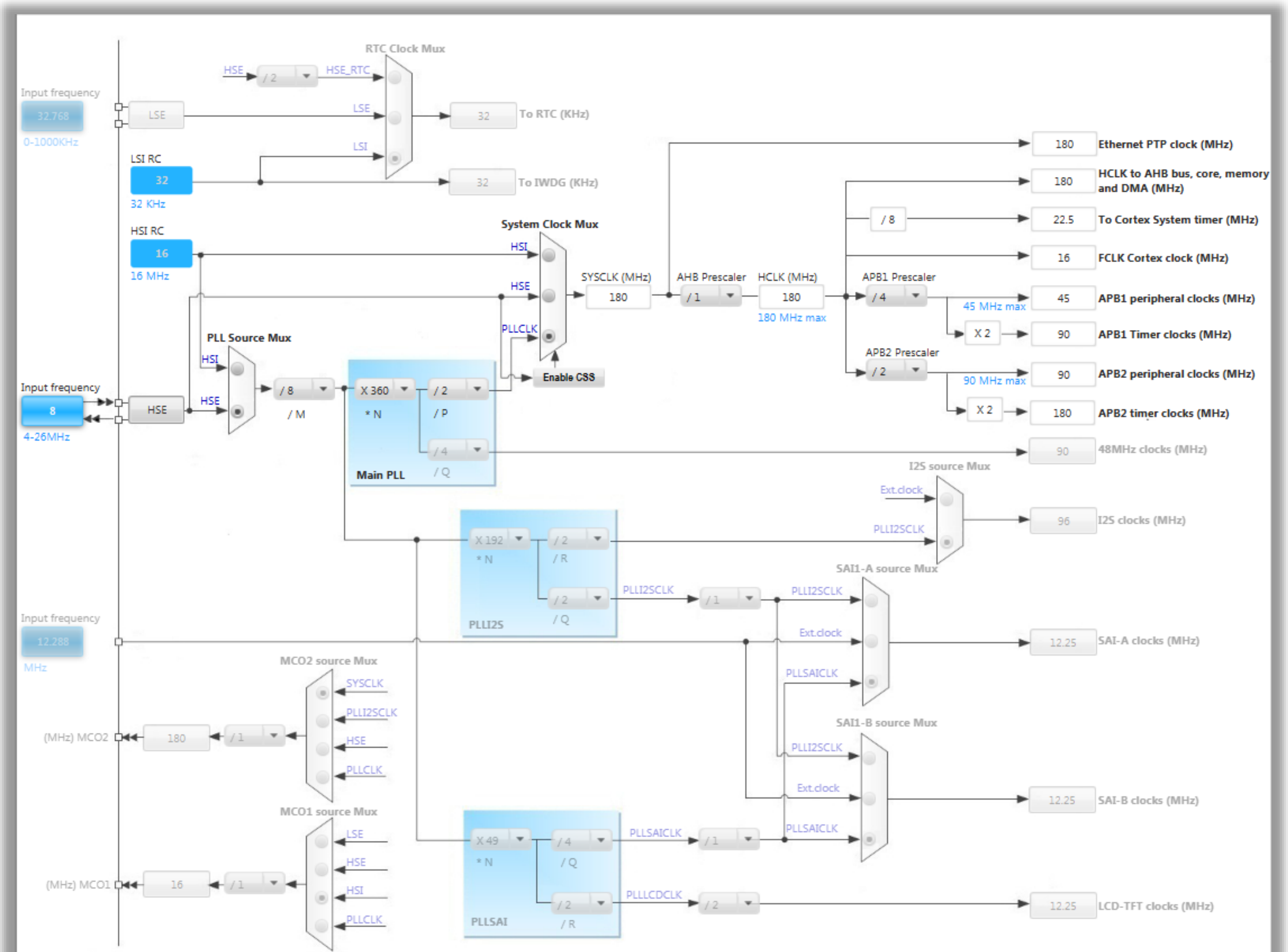
3.1.1 Use RTC and Alarm with interrupt

- 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



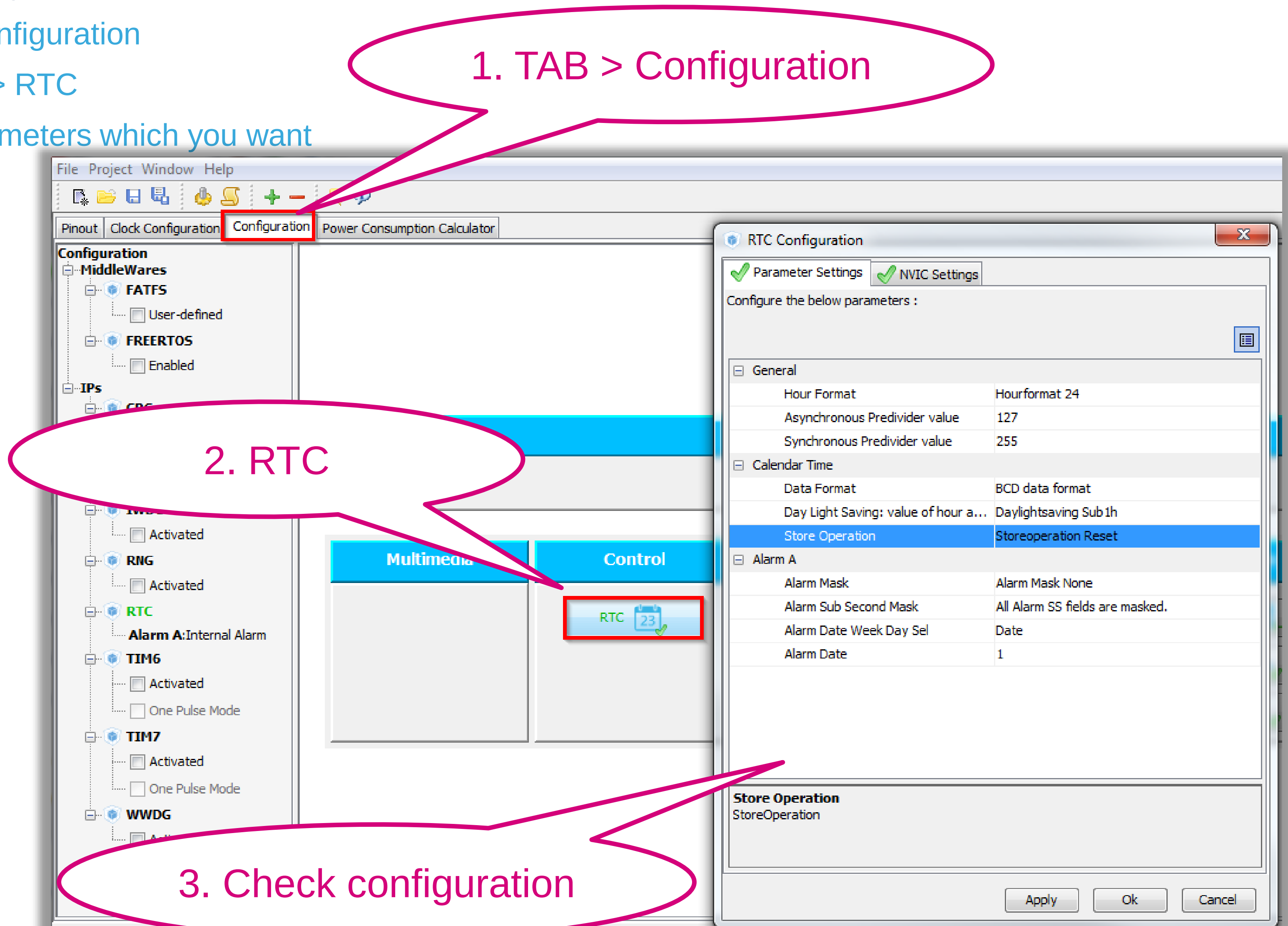
3.1.1 Use RTC and Alarm with interrupt

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



3.1.1 Use RTC and Alarm with interrupt

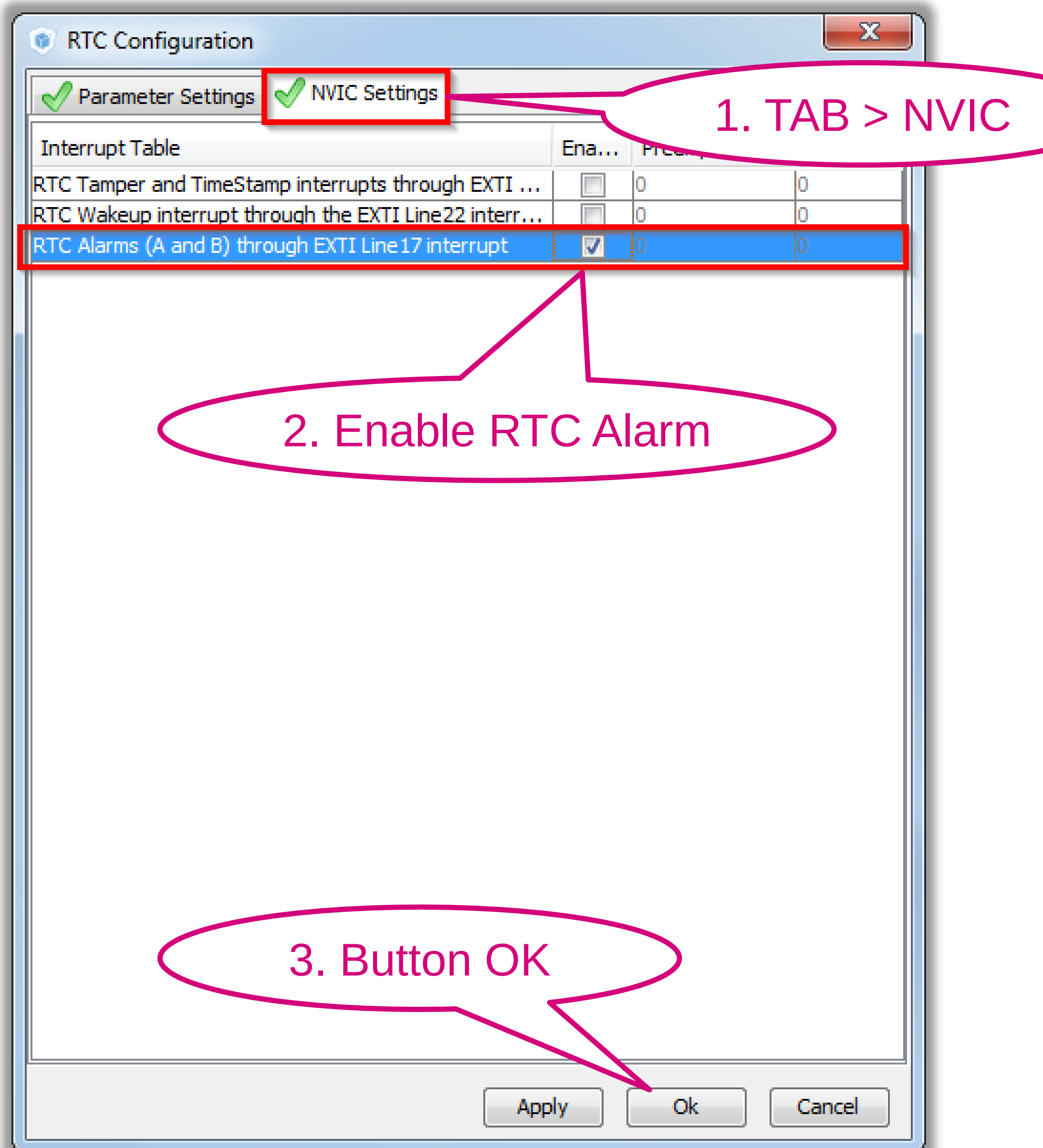
- RTC Configuration
  - TAB>Configuration
  - Control > RTC
  - Set parameters which you want



life.augmented

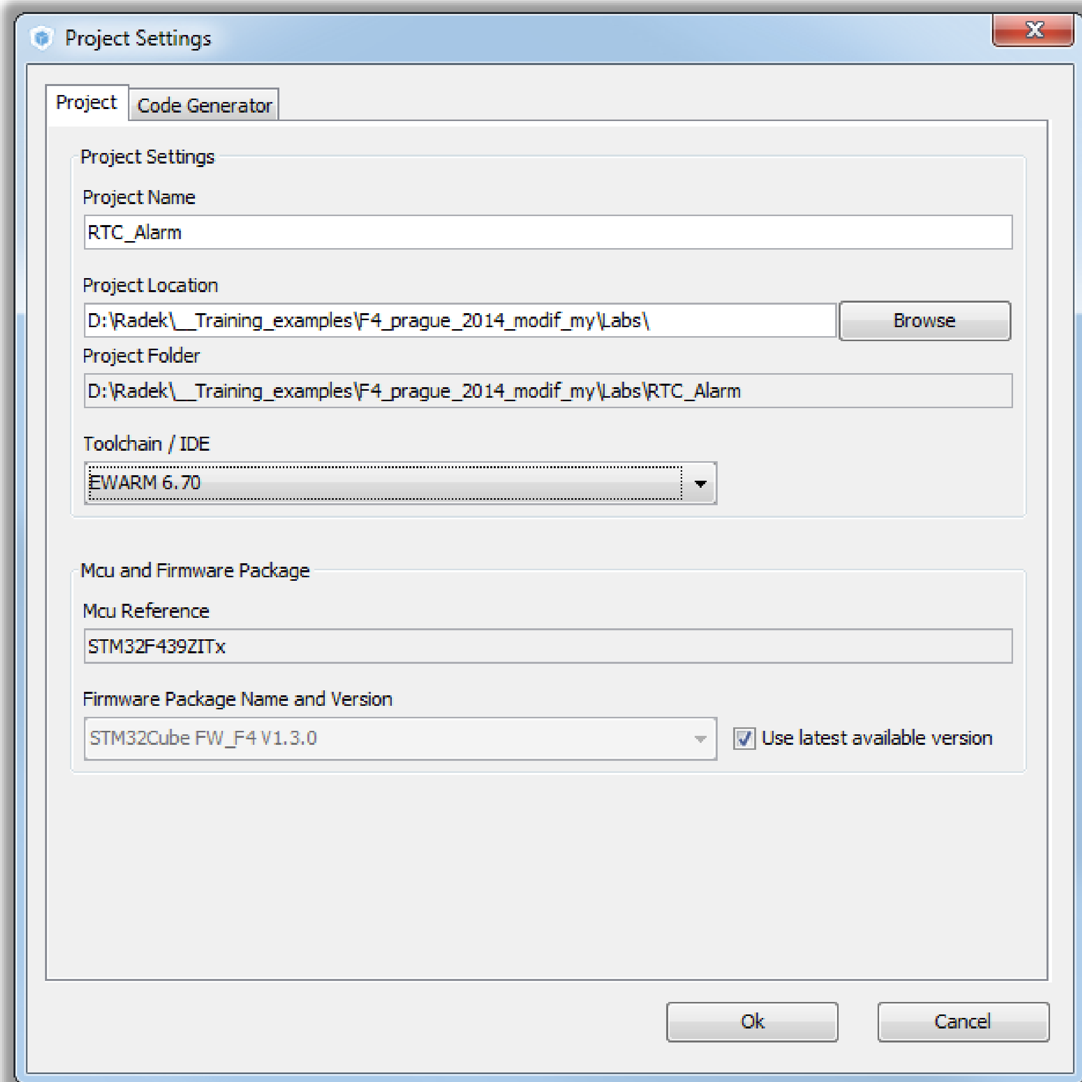
3.1.1 Use RTC and Alarm with interrupt

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



3.1.1 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



life.augmented

3.1.1 Use RTC and Alarm with interrupt

- 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)

3.1.1 Use RTC and Alarm with interrupt

- 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;
```

3.1.1 Use RTC and Alarm with interrupt

- 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 */
```

3.1.1 Use RTC and Alarm with interrupt

- 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_ALARMSUBSECONDMASK_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

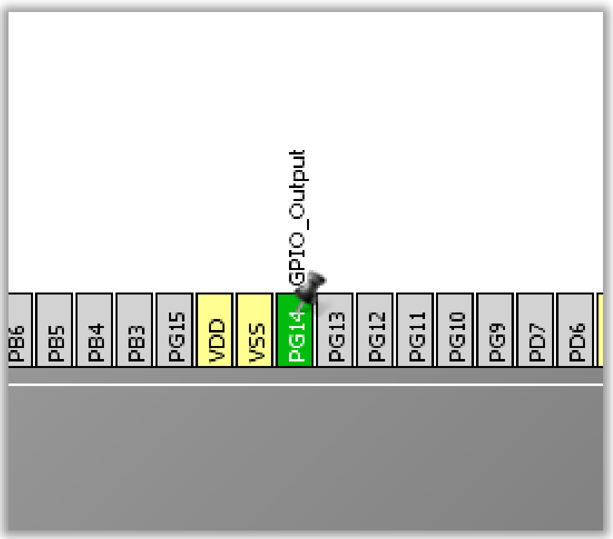
3.2.1 TIM with interrupt lab

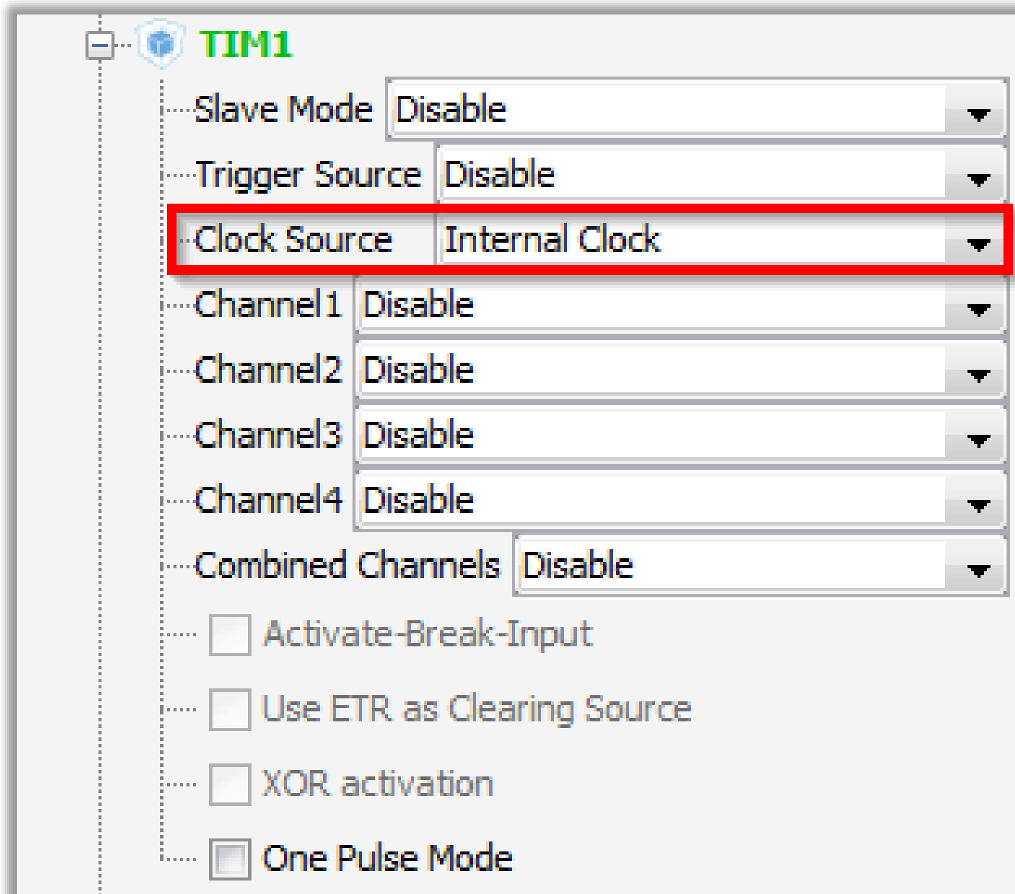
3.2.1 Use TIM with interrupt

- 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

3.2.1 Use TIM with interrupt

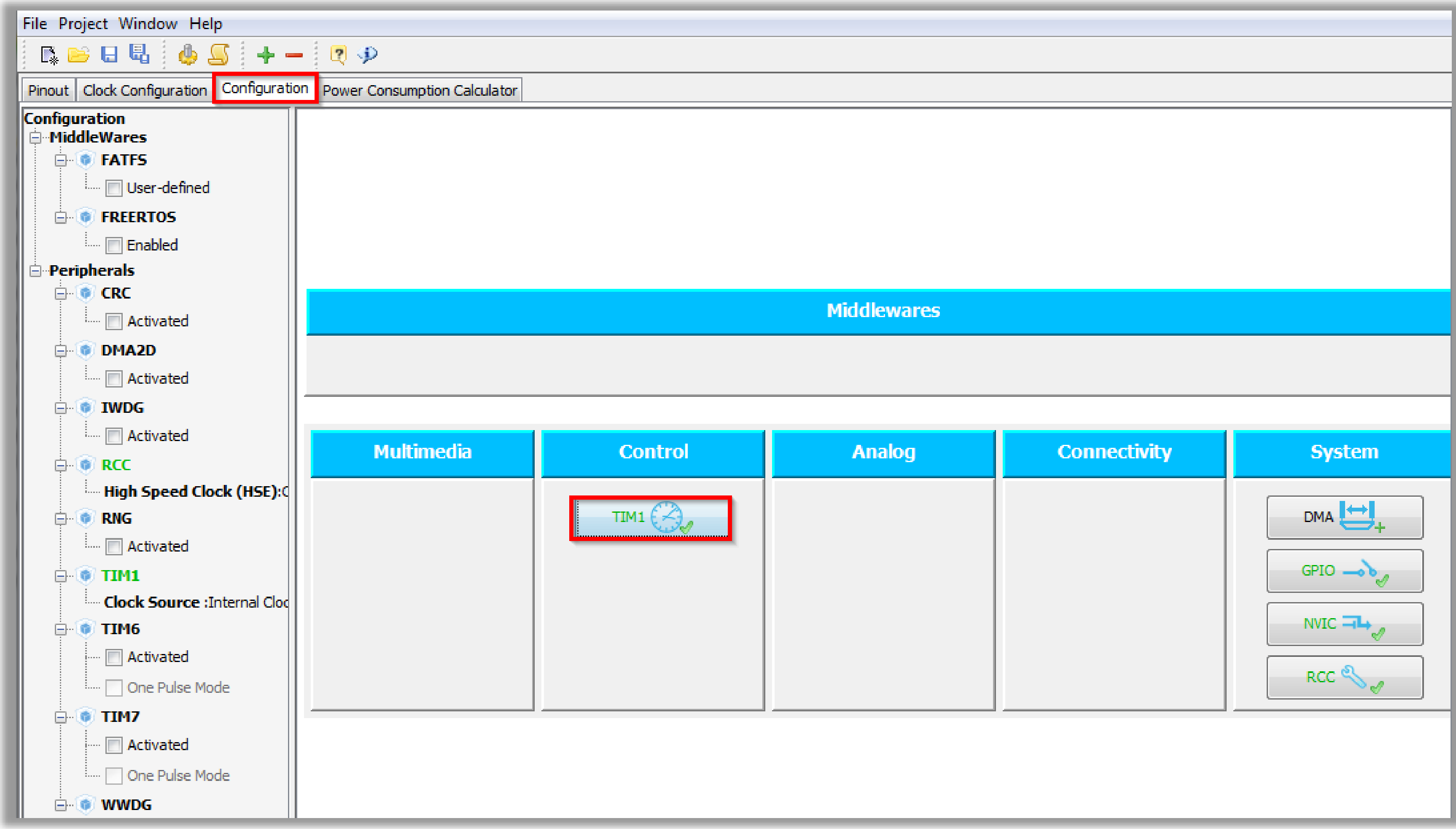
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX TIM selection
  - Select TIM clock source Internal clock
  - Enable GPIO for LED PG14





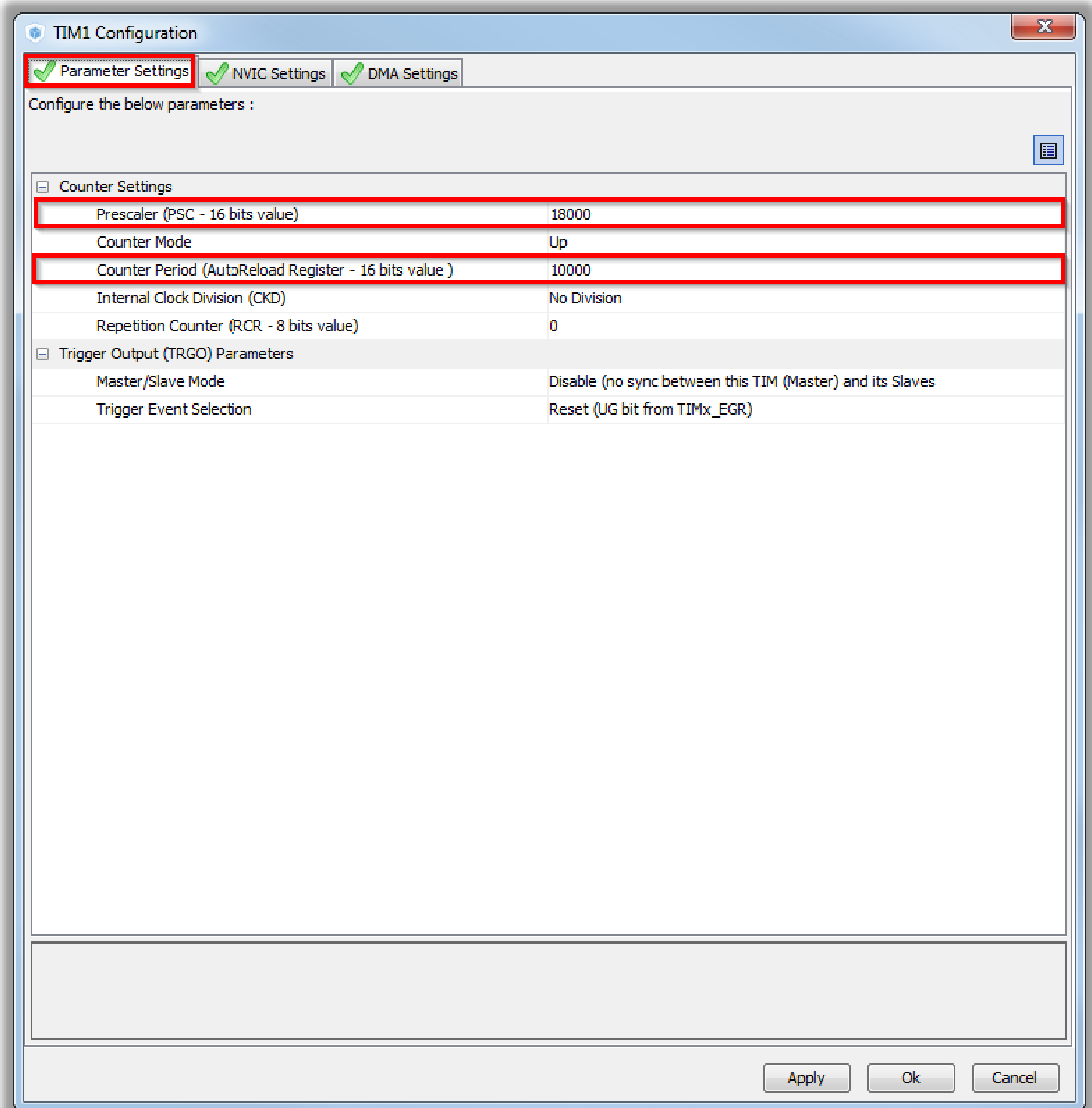
3.2.1 Use TIM with interrupt

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



3.2.1 Use TIM with interrupt

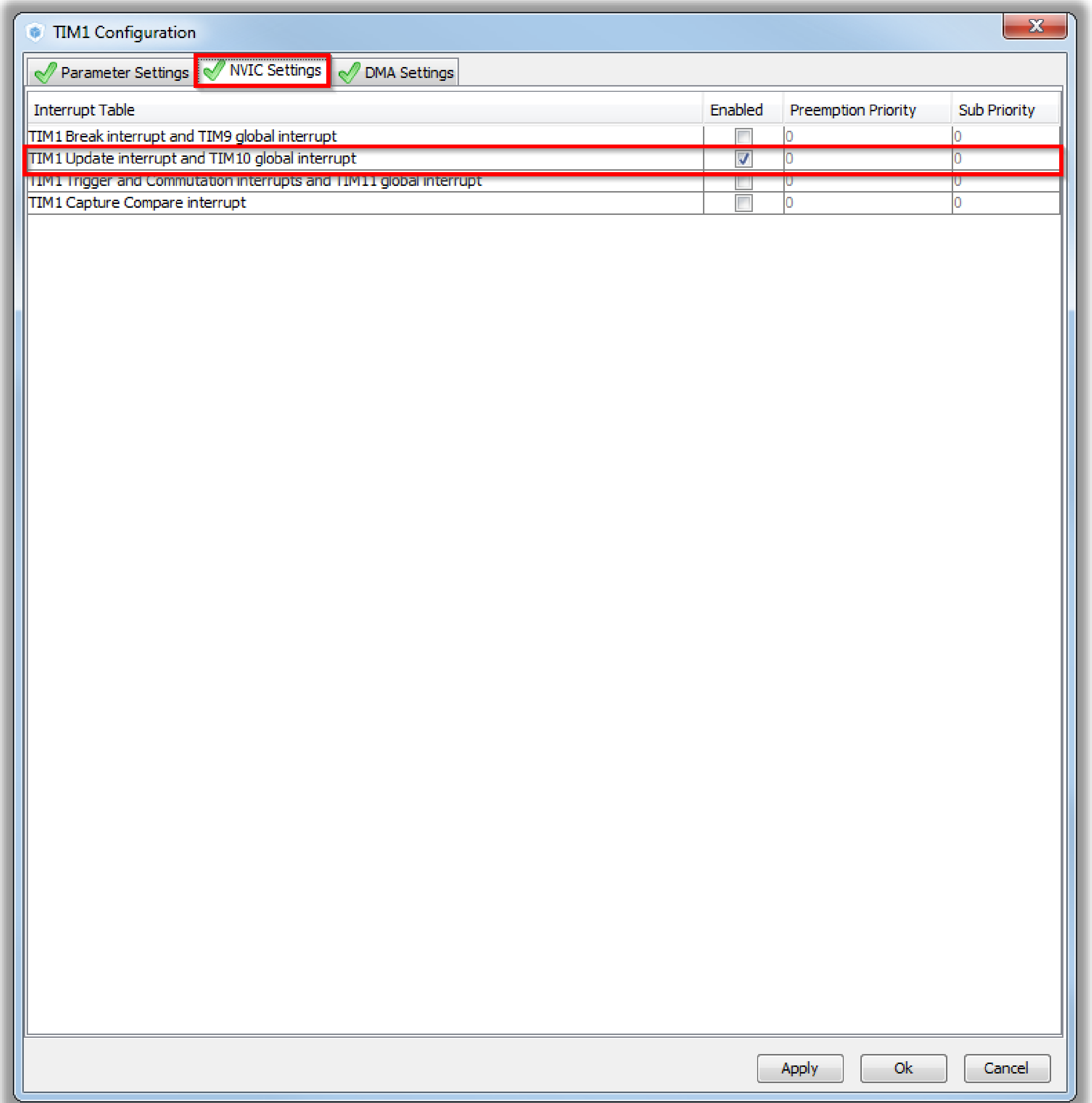
- CubeMX TIM configuration
  - Tab>Parameter Settings
  - Prescaler to 18000
  - Counter period to 10000
  - Together with 180MHz TIMER1 clock we get period 1Hz



life.augmented

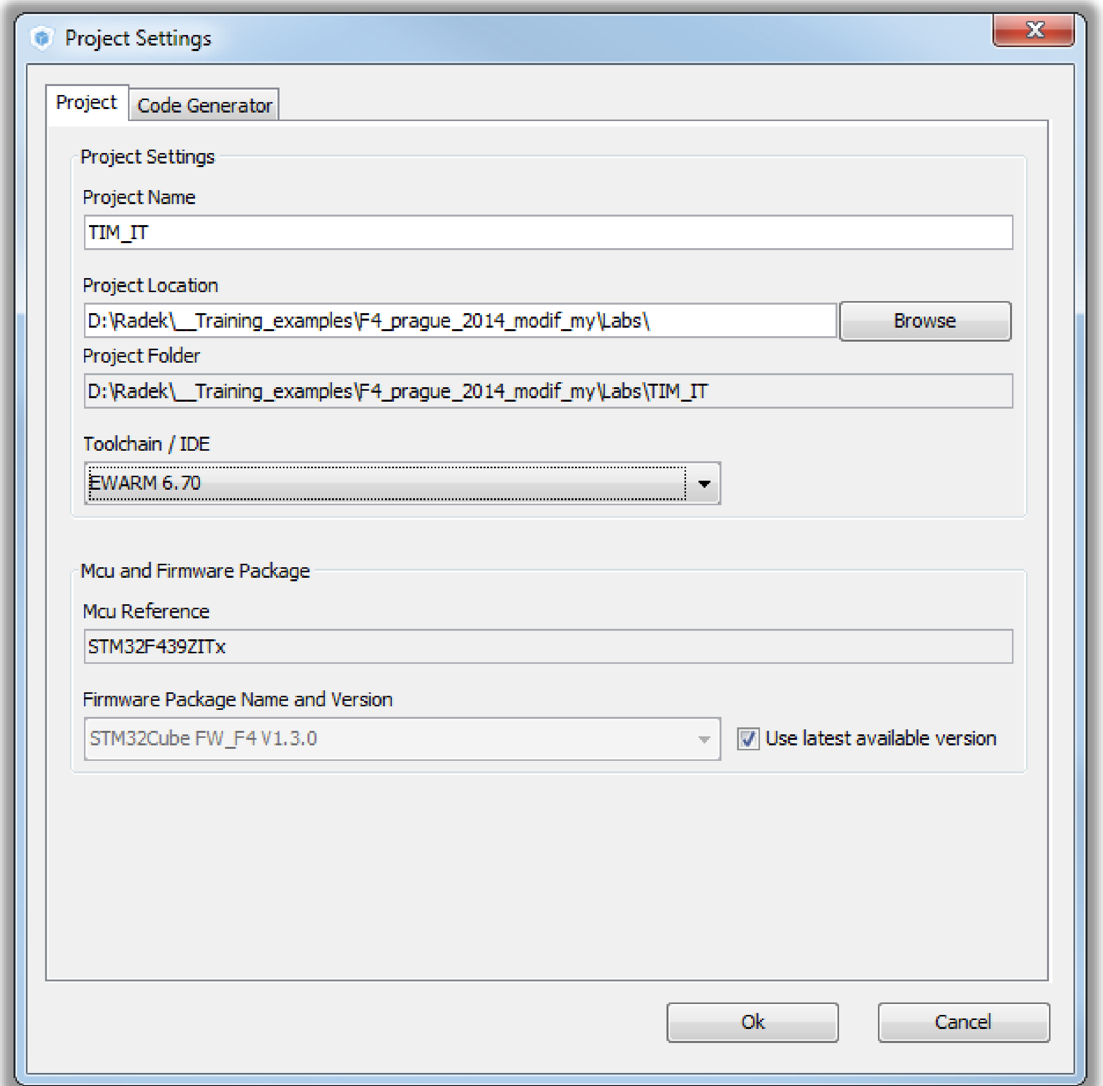
3.2.1 Use TIM with interrupt

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



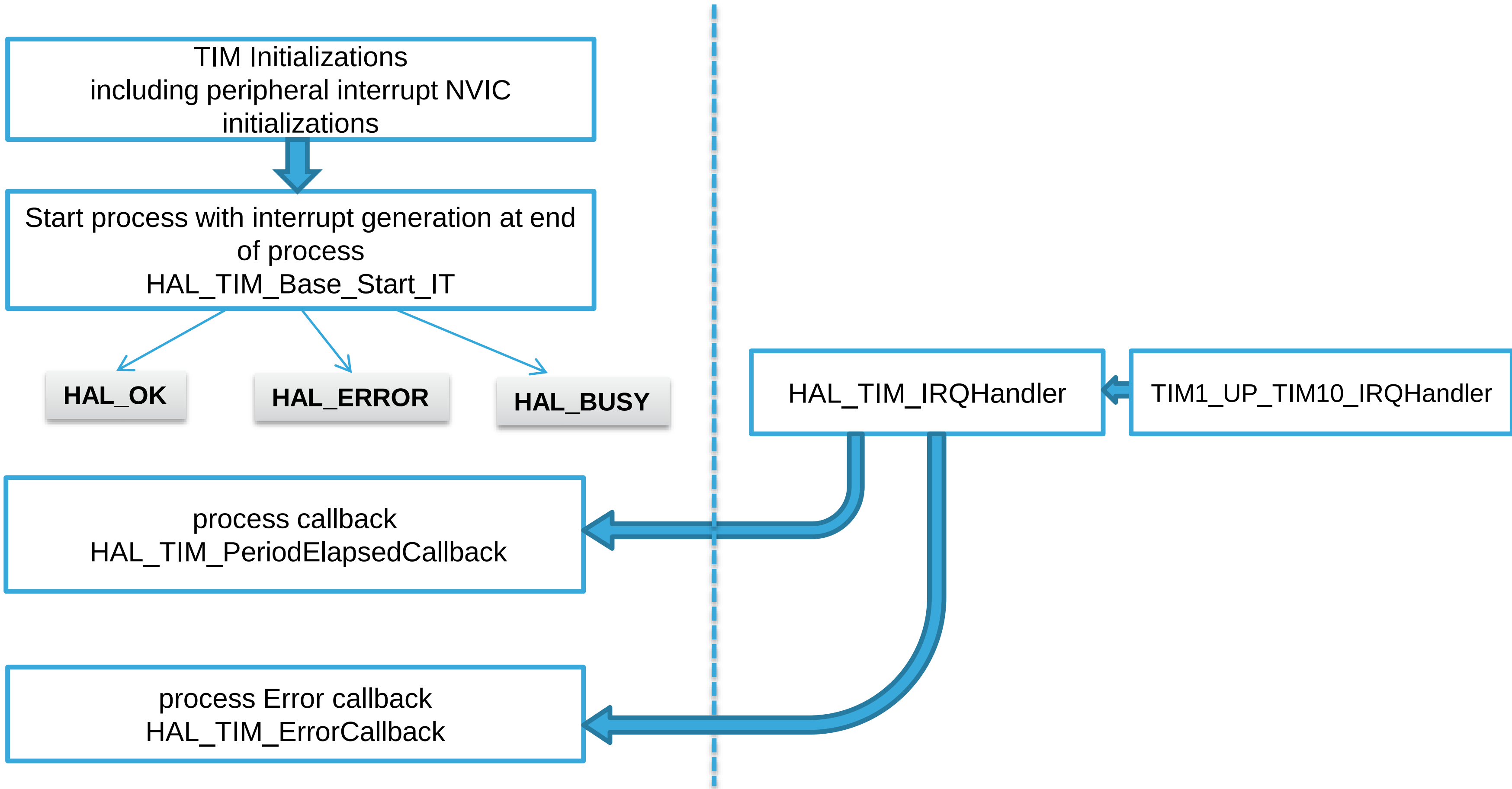
3.2.1 Use TIM with interrupt

- 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



life.augmented

HAL Library TIM with IT flow



3.2.1 Use TIM with interrupt

- 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)

3.2.1 Use TIM with interrupt

- 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 */
```

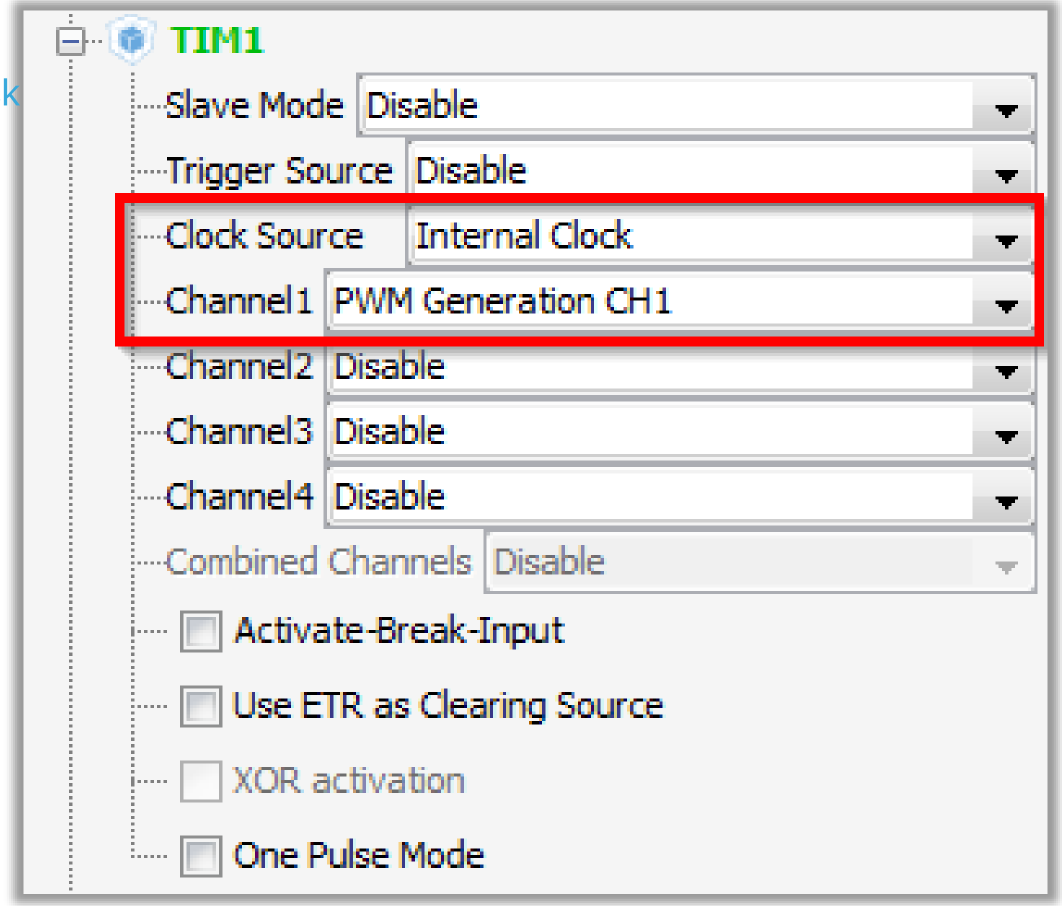
3.2.2 TIM with PWM output lab

3.2.2 Use TIM with PWM output

- 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

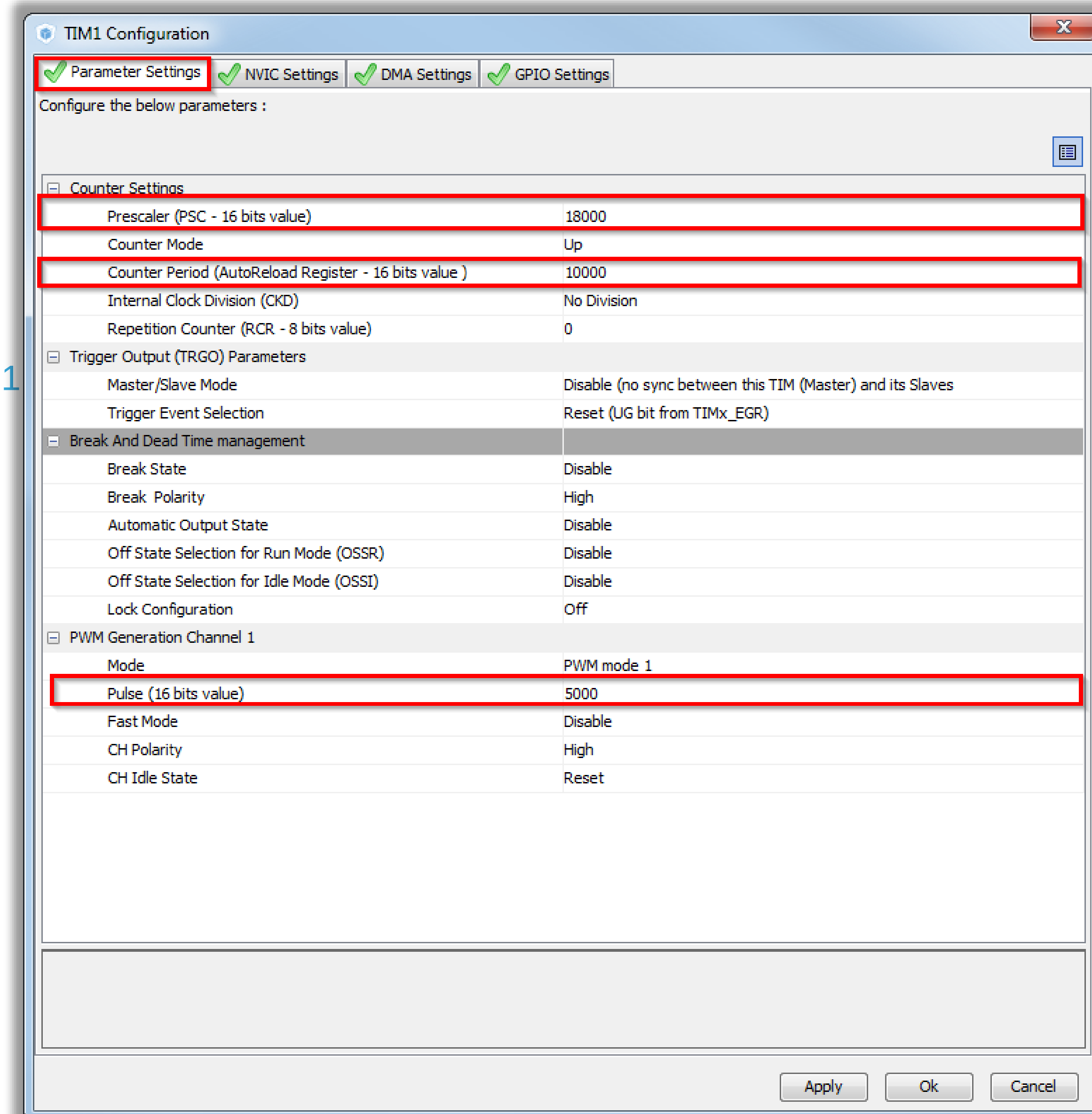
3.2.2 Use TIM with PWM output

- 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



3.2.2 Use TIM with PWM output

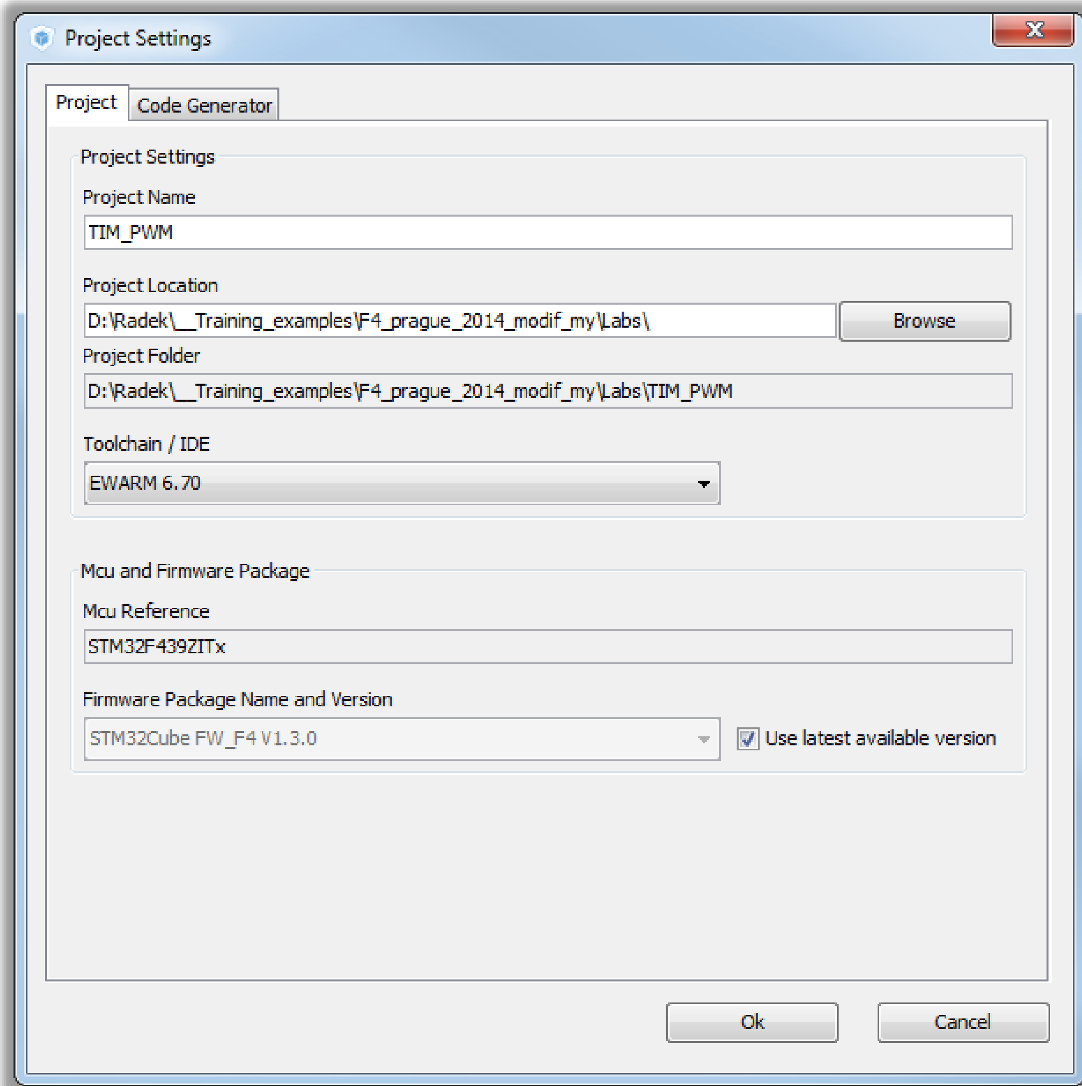
- CubeMX TIM configuration
  - TAB>Configuration >Control>TIM1
  - TAB>Parameter settings
  - Prescaler to 18000
  - Counter period to 10000
  - Together with 180MHz TIMER1 clock we get period 1Hz
  - PWM pulse to 5000 this give us 1Hz blinking frequency



life.augmented

3.2.2 Use TIM with PWM output

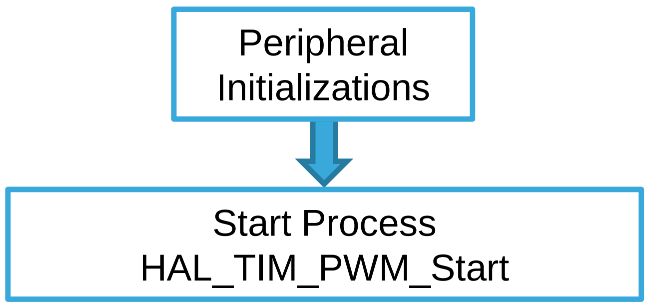
- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



life.augmented

3.2.2 Use TIM with PWM output

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



life.augmented

3.2.2 Use TIM with PWM output

- 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

3.2.2 Use TIM with PWM output

- 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

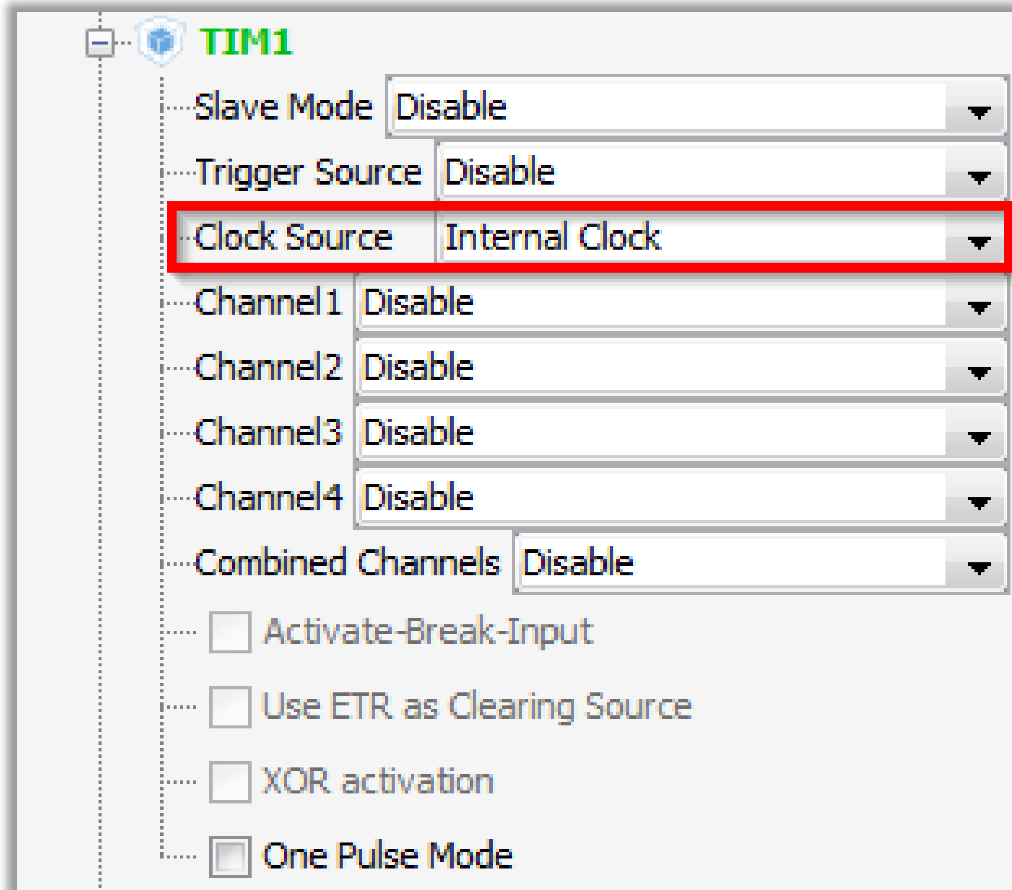
3.2.3 TIM with DMA lab

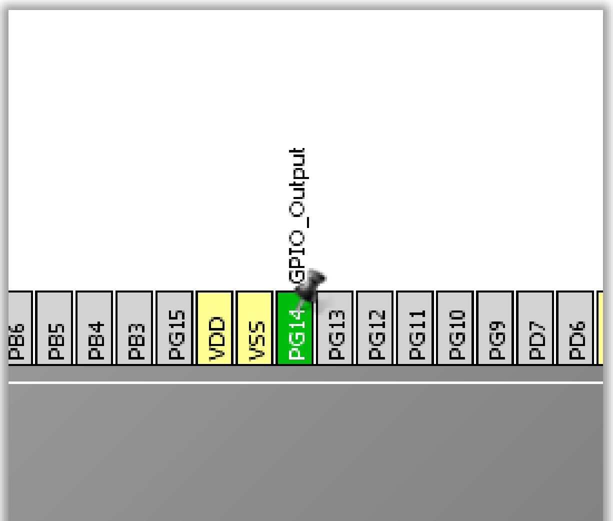
3.2.3 Use TIM with DMA transfer

- 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

3.2.3 Use TIM with DMA transfer

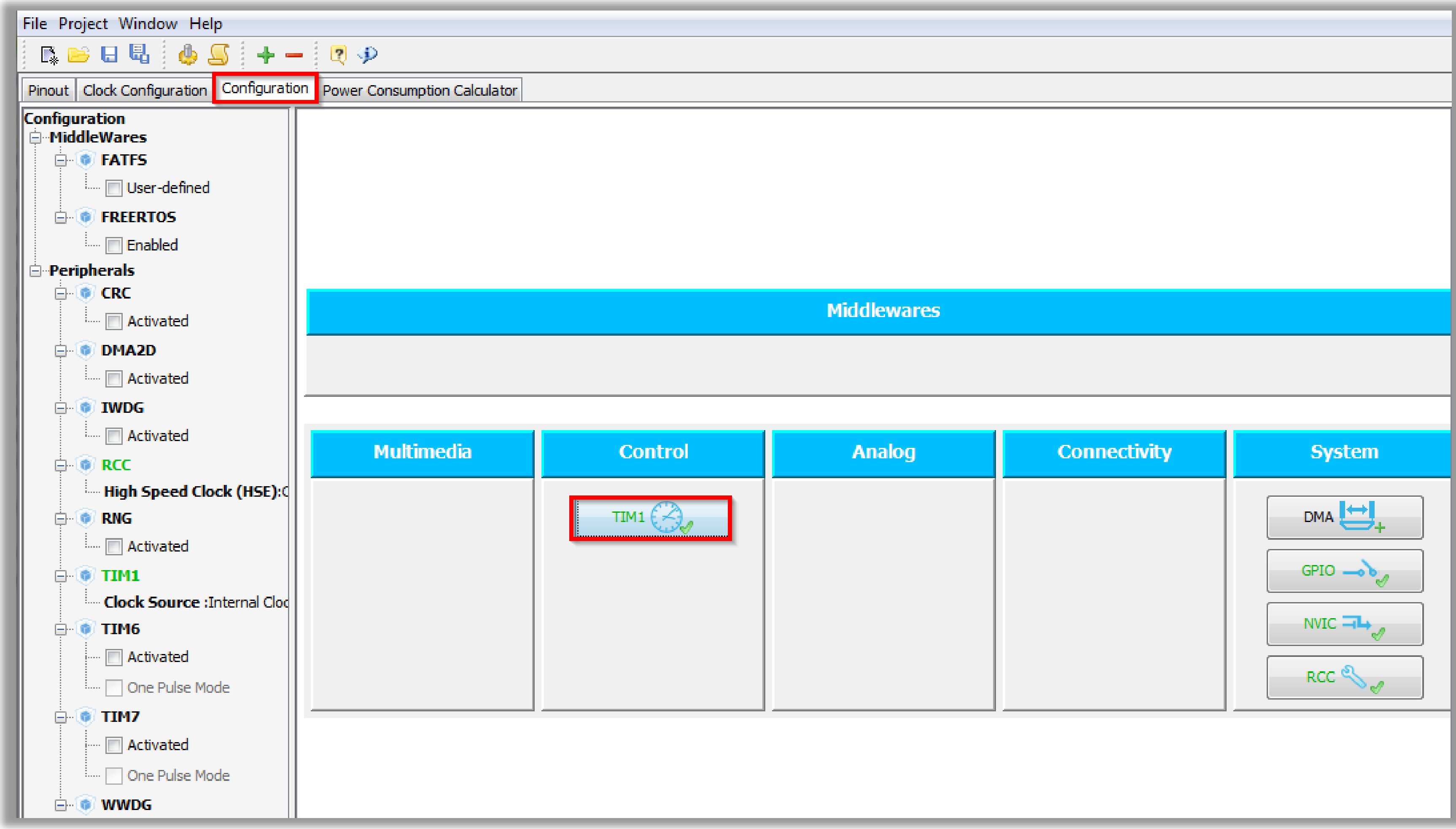
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX TIM selection
  - Select TIM clock source Internal clock
  - Enable GPIO for LED PG14





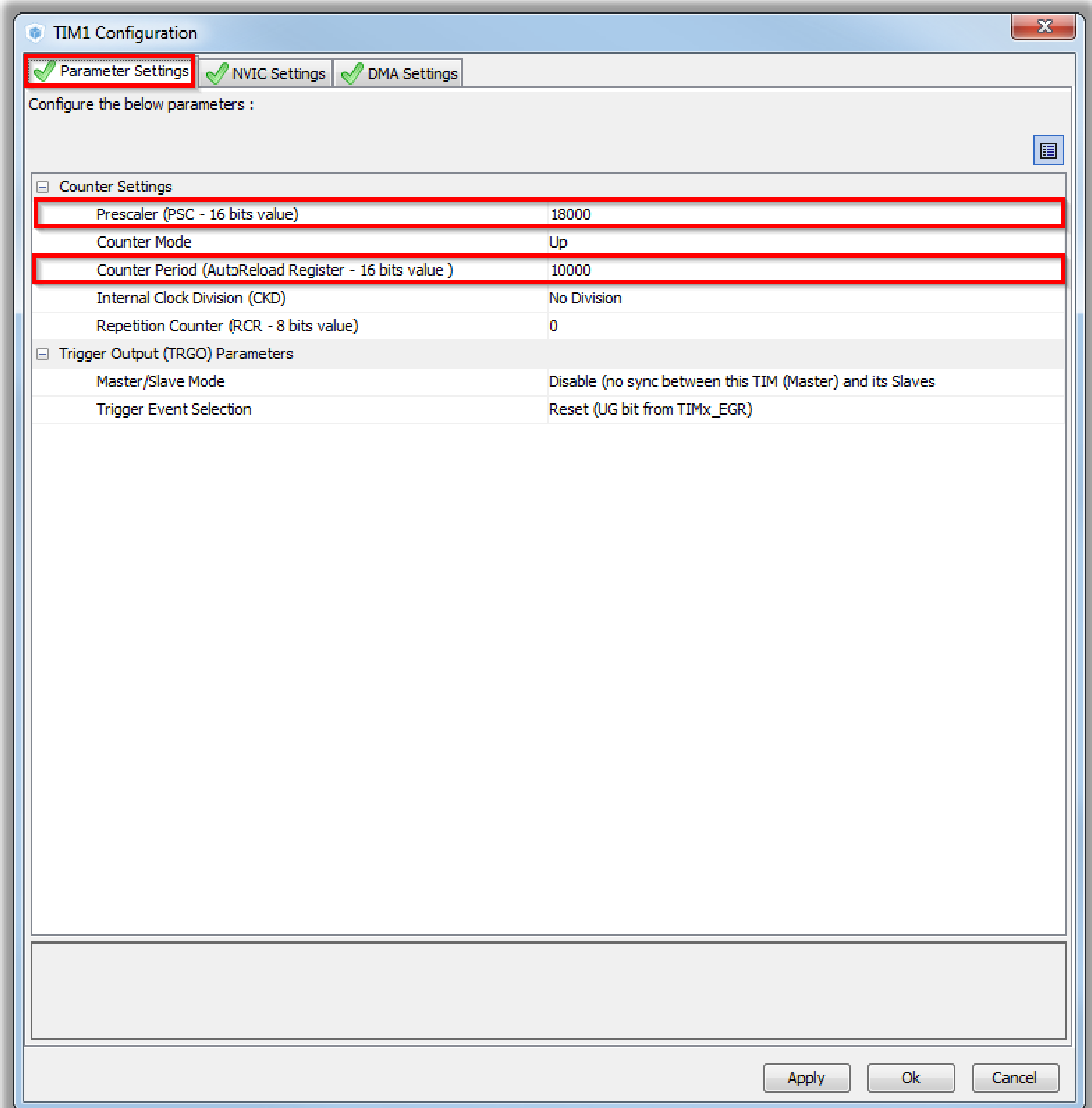
3.2.3 Use TIM with DMA transfer

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



3.2.3 Use TIM with DMA transfer

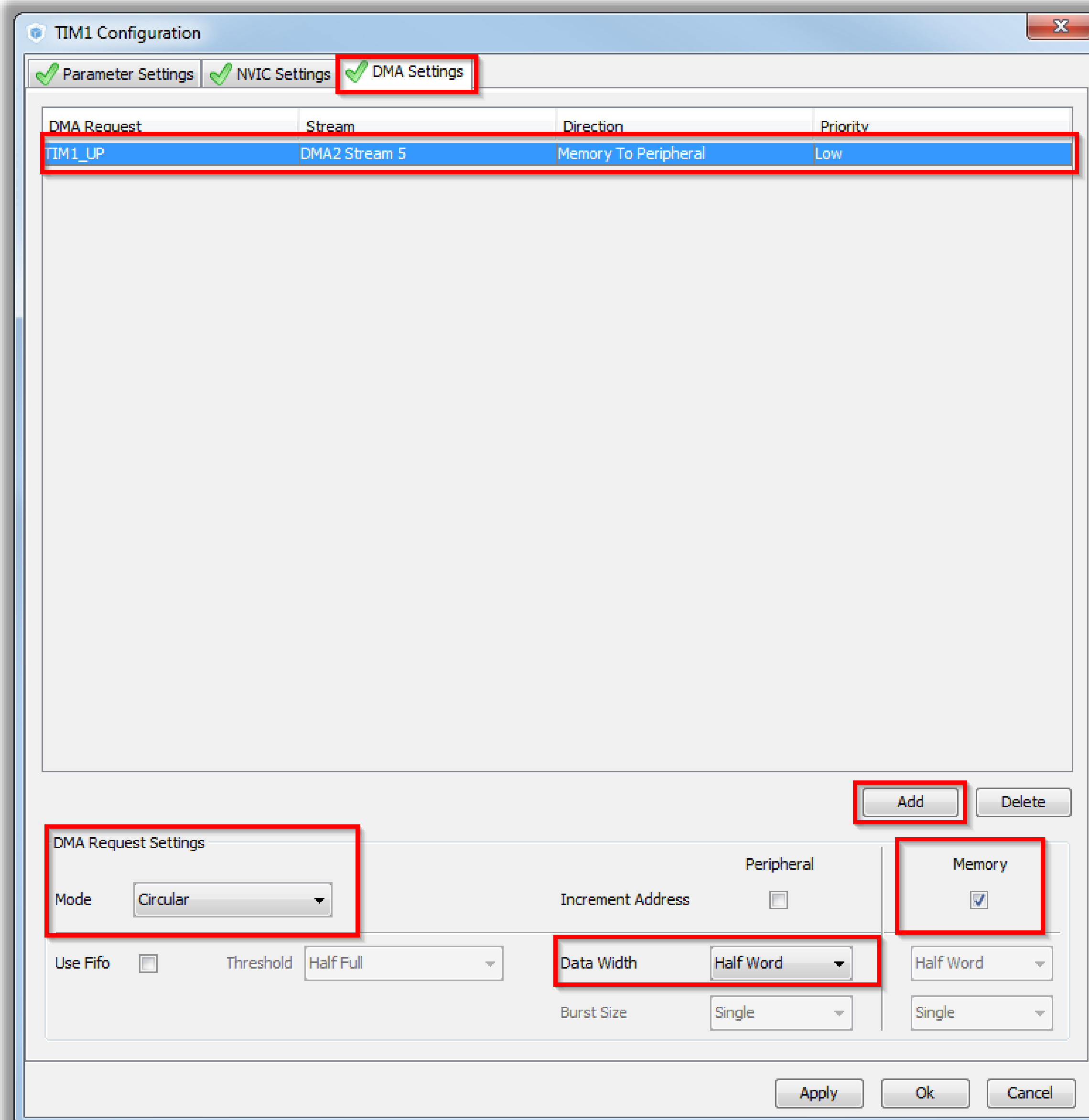
- CubeMX TIM configuration
  - Tab>Parameter Settings
  - Prescaler to 18000
  - Counter period to 10000
  - Together with 180MHz TIMER1 clock we get period 1Hz



life.augmented

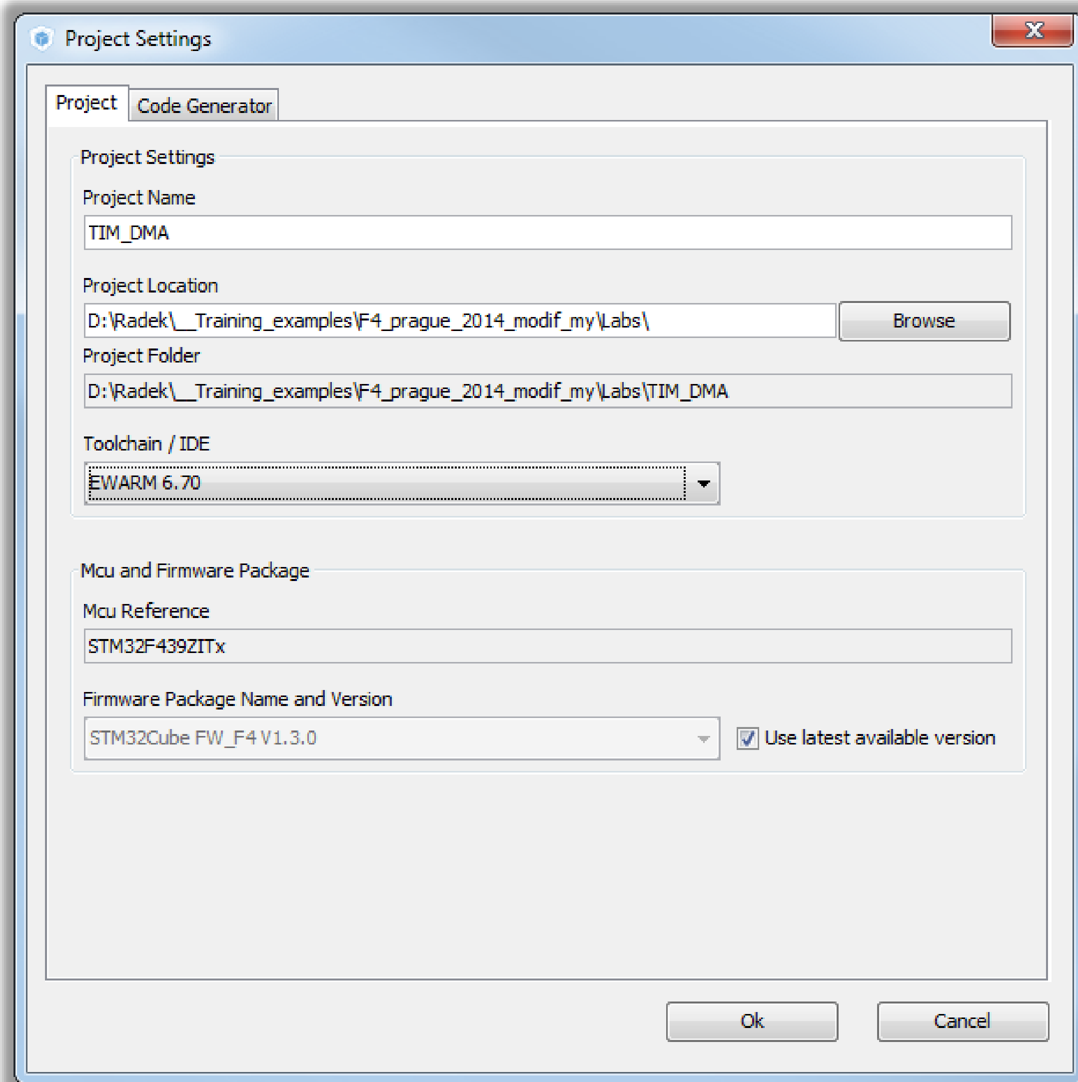
3.2.3 Use TIM with DMA transfer

- 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



3.2.3 Use TIM with DMA 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



life.augmented

3.2.3 Use TIM with DMA transfer

HAL Library TIM with DMA flow

Peripheral Initializations
including DMA stream initializations

Start process with DMA
HAL_DMA_Start

HAL_OK | **HAL_ERROR** | **HAL_ERROR**

Start process with TIM
HAL_TIM_Base_Start

HAL_OK | **HAL_ERROR** | **HAL_ERROR**

3.2.3 Use TIM with DMA transfer

- 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)

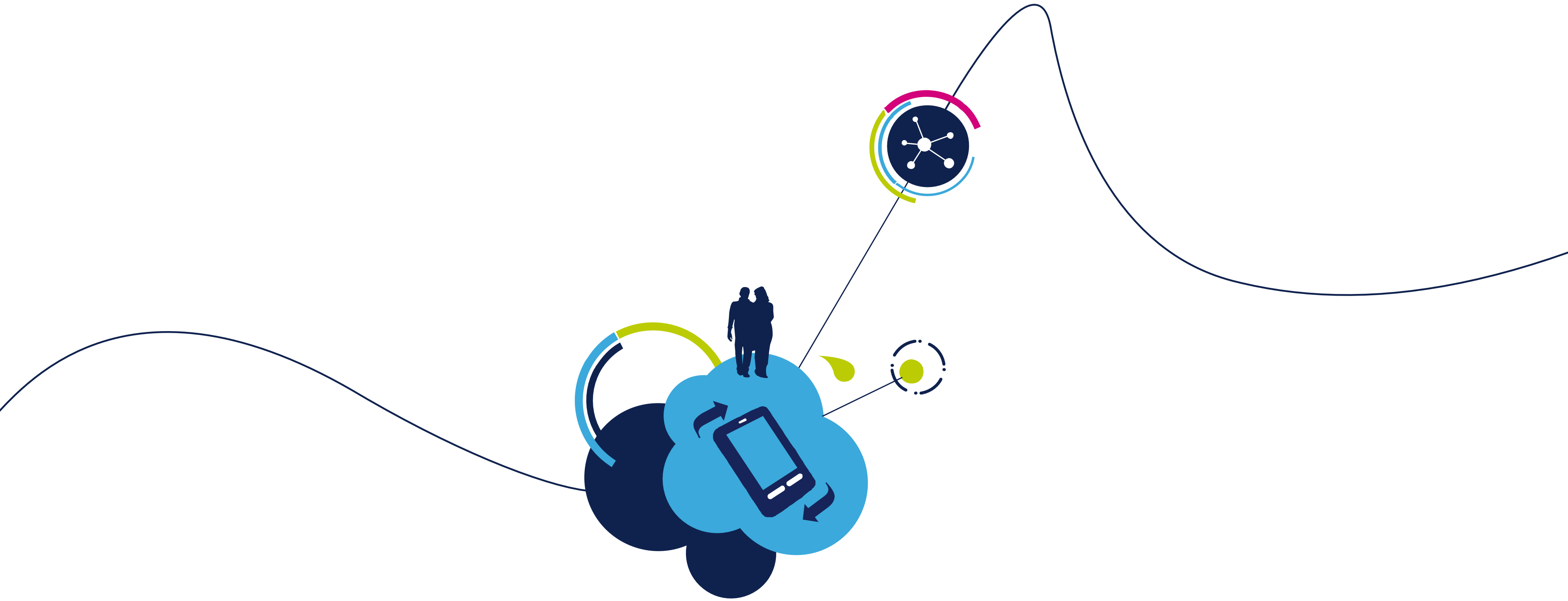
3.2.3 Use TIM with DMA transfer

- 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 */
```



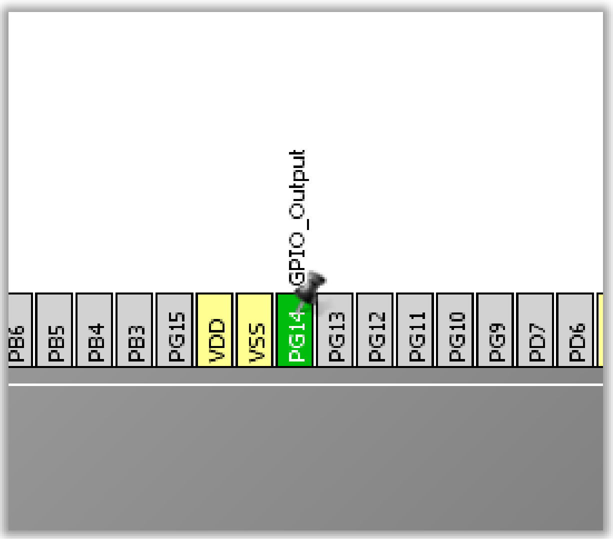
3.2.4 TIM as counter lab

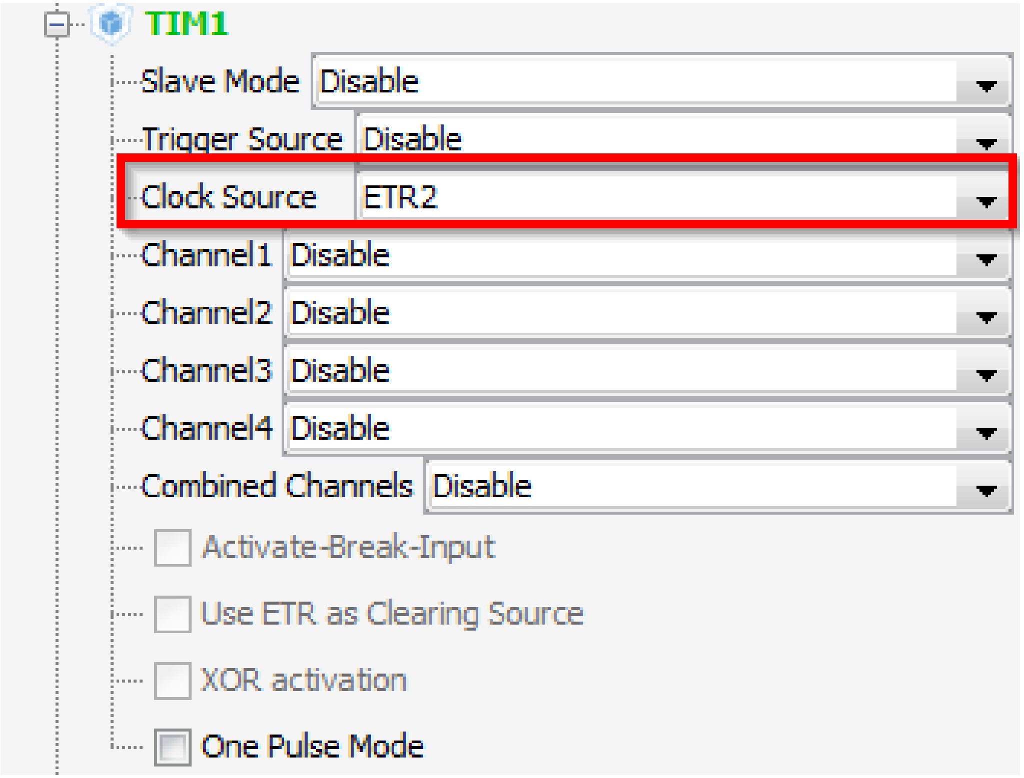
3.2.4 Use TIM as pulse counter

- 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

3.2.4 Use TIM as pulse counter

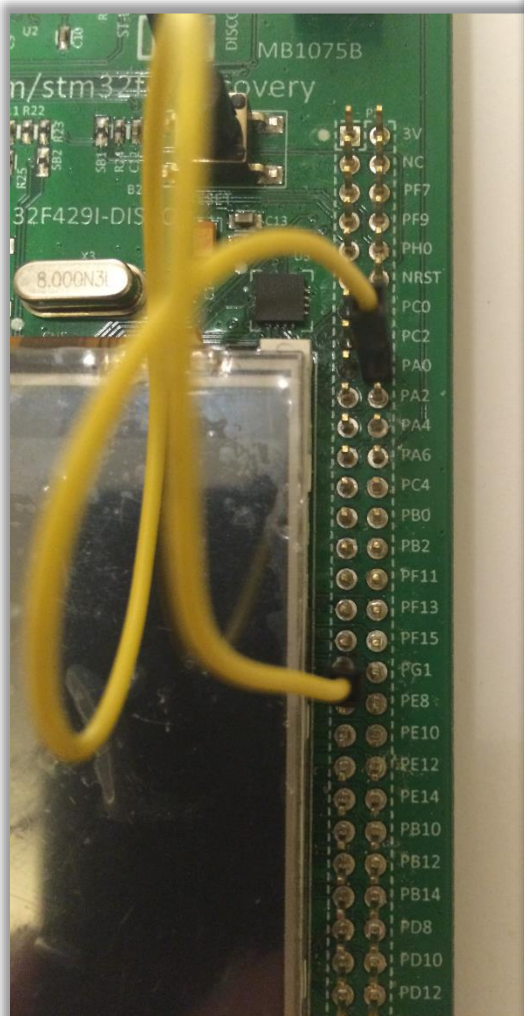
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX TIM selection
  - Select TIM clock source ETR2
  - Enable GPIO for LED PG14

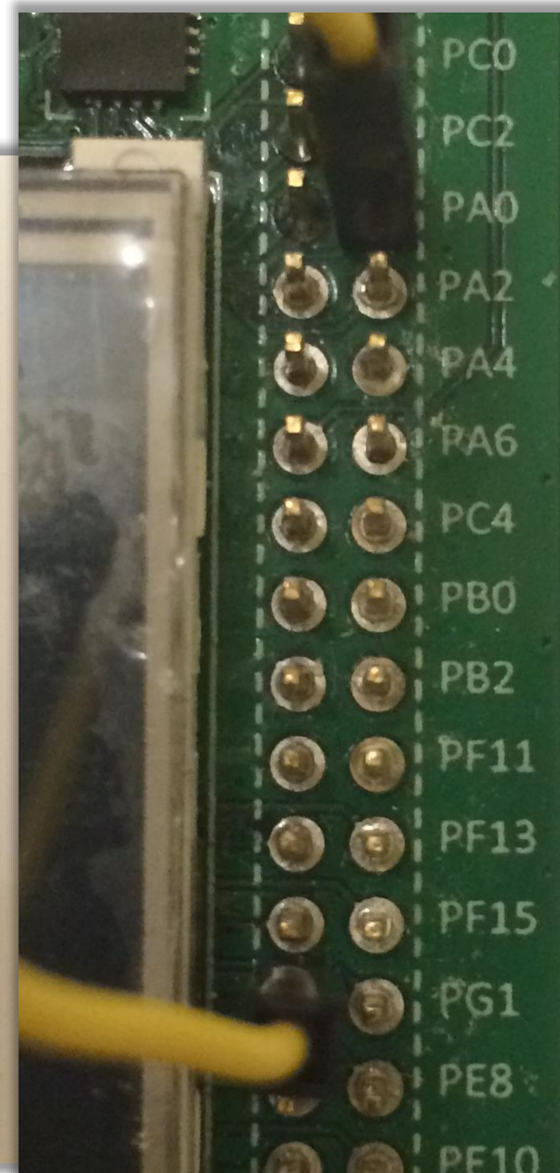




3.2.4 Use TIM as pulse counter

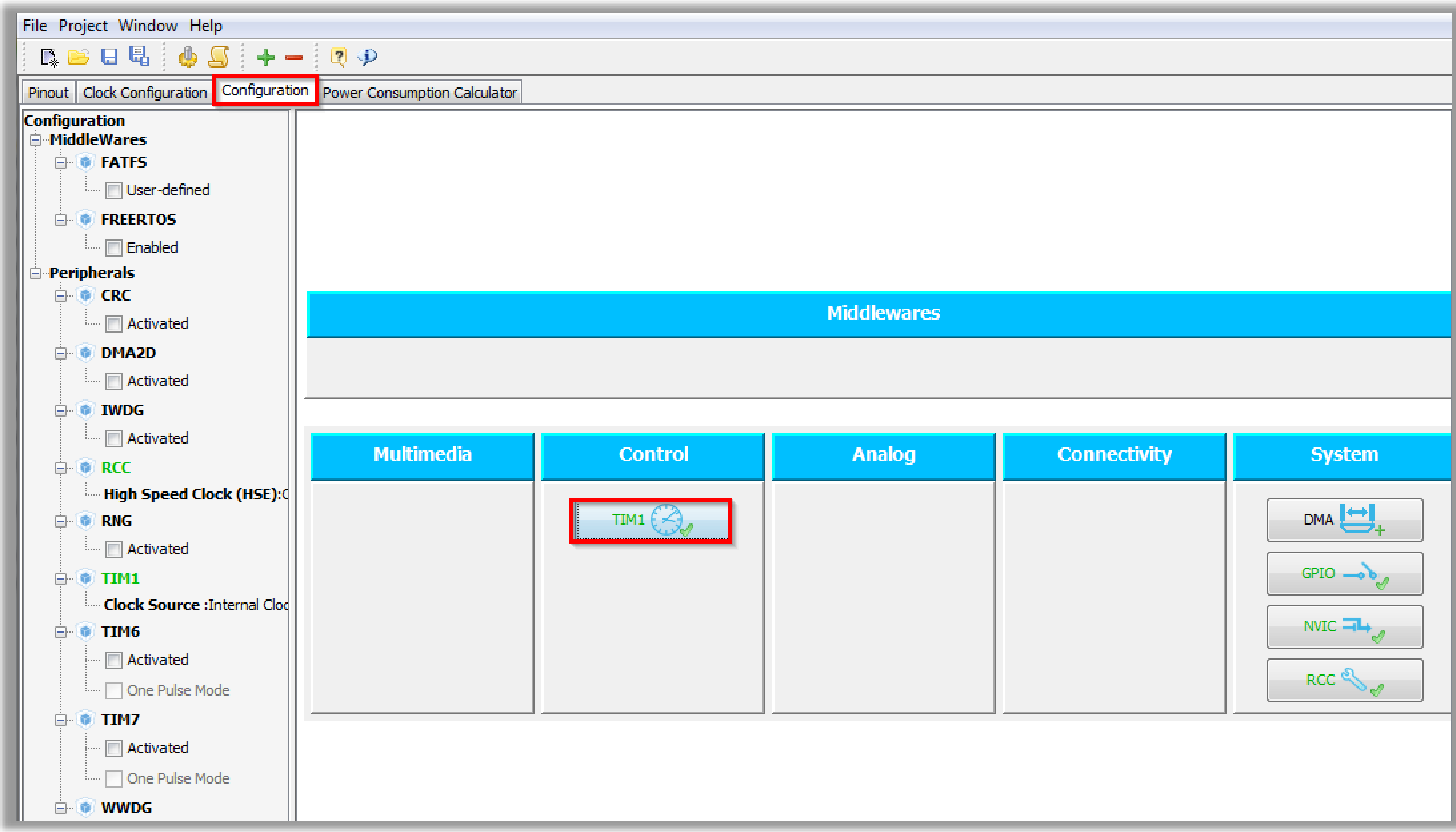
- Hard ware setting
  - Connect Button PA0 and ETR pin PE7 with wire together





3.2.4 Use TIM as pulse counter

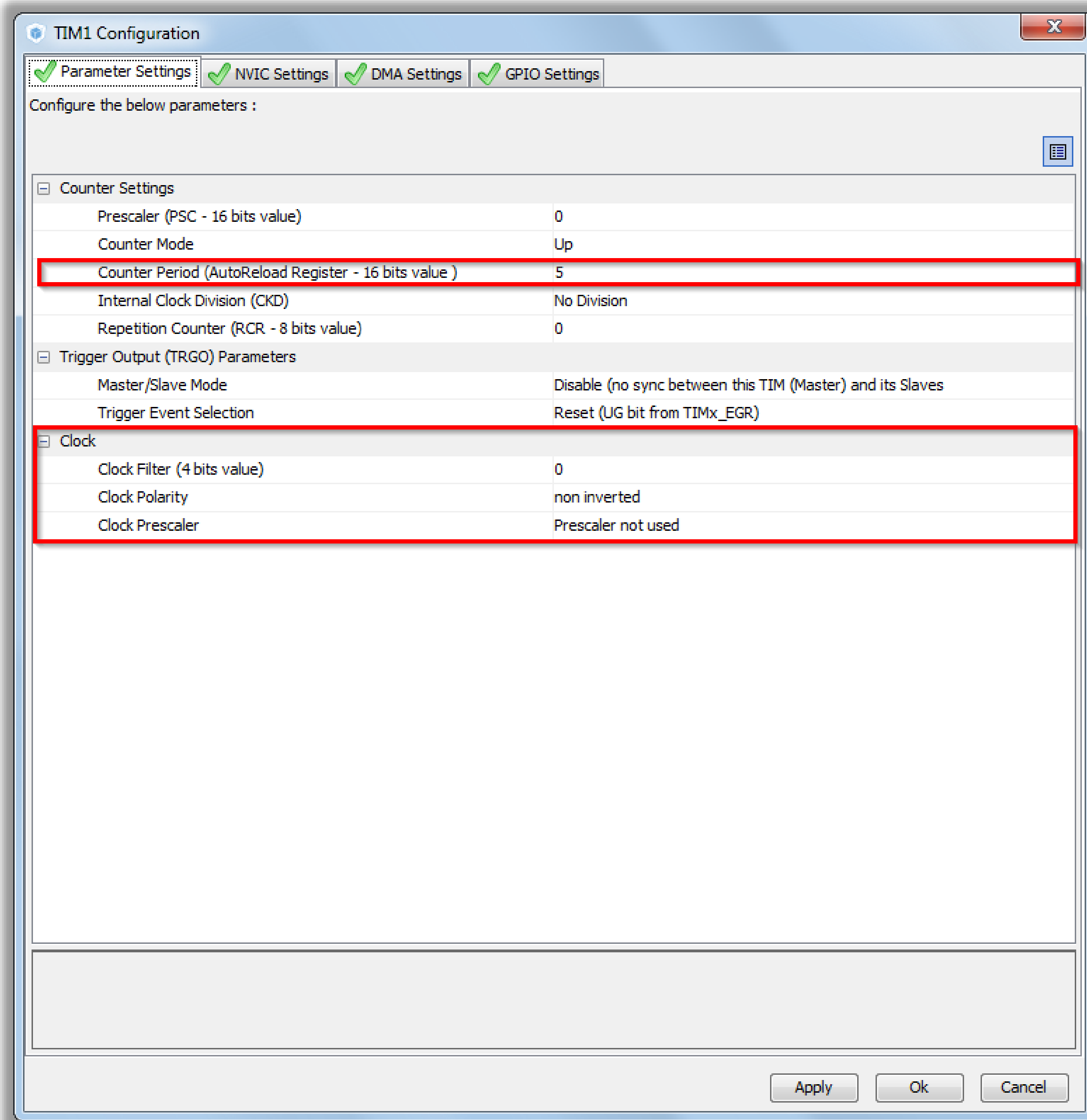
- CubeMX TIM configuration
  - Tab>Configuration>Control>TIM1
  - Check the settings



life.augmented

3.2.4 Use TIM as pulse counter

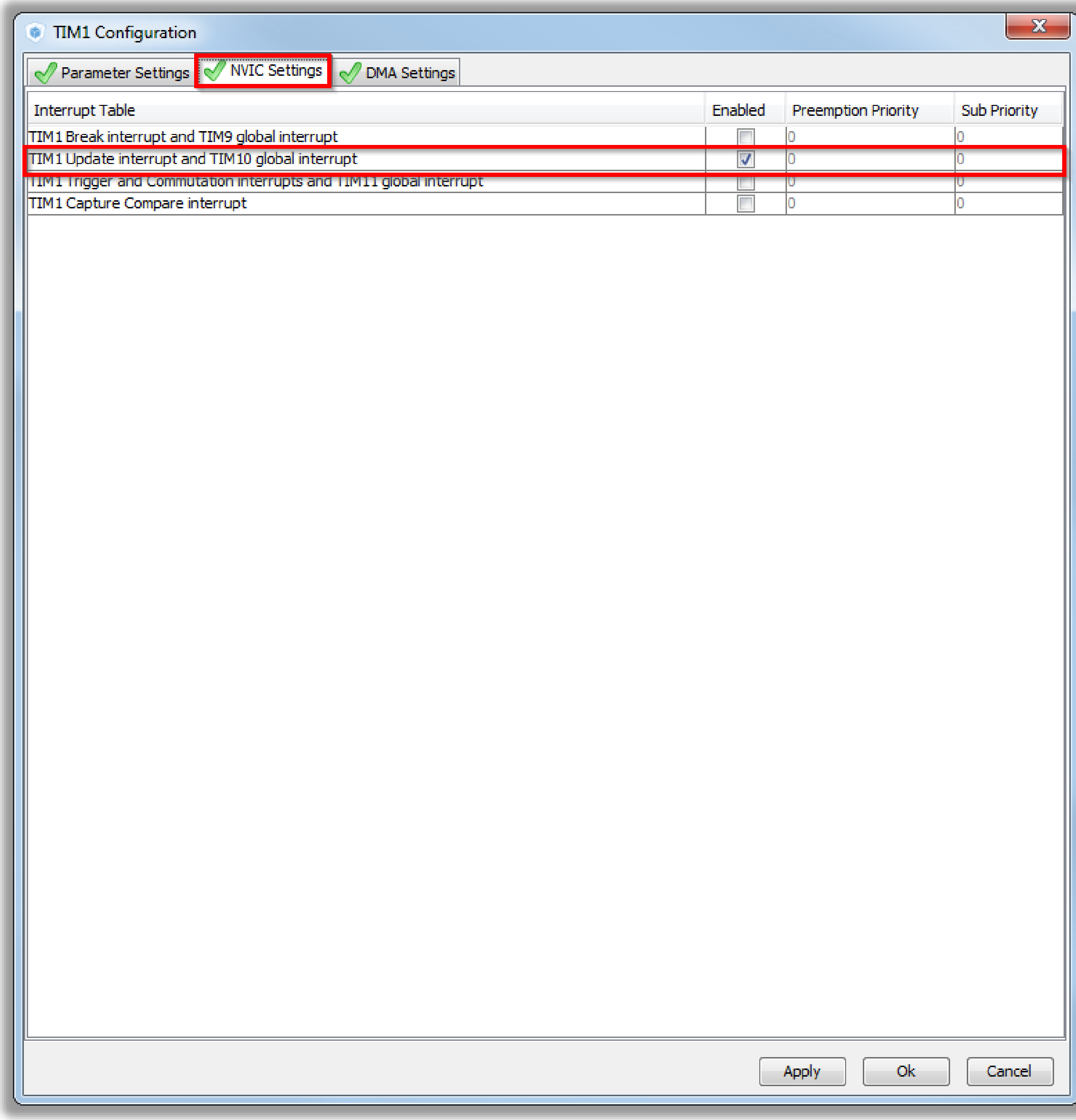
- CubeMX TIM configuration
  - Tab>Parameter Settings
  - Counter set to 5, 5 button press
  - Clock set the ETR pin filter and edge reaction



life.augmented

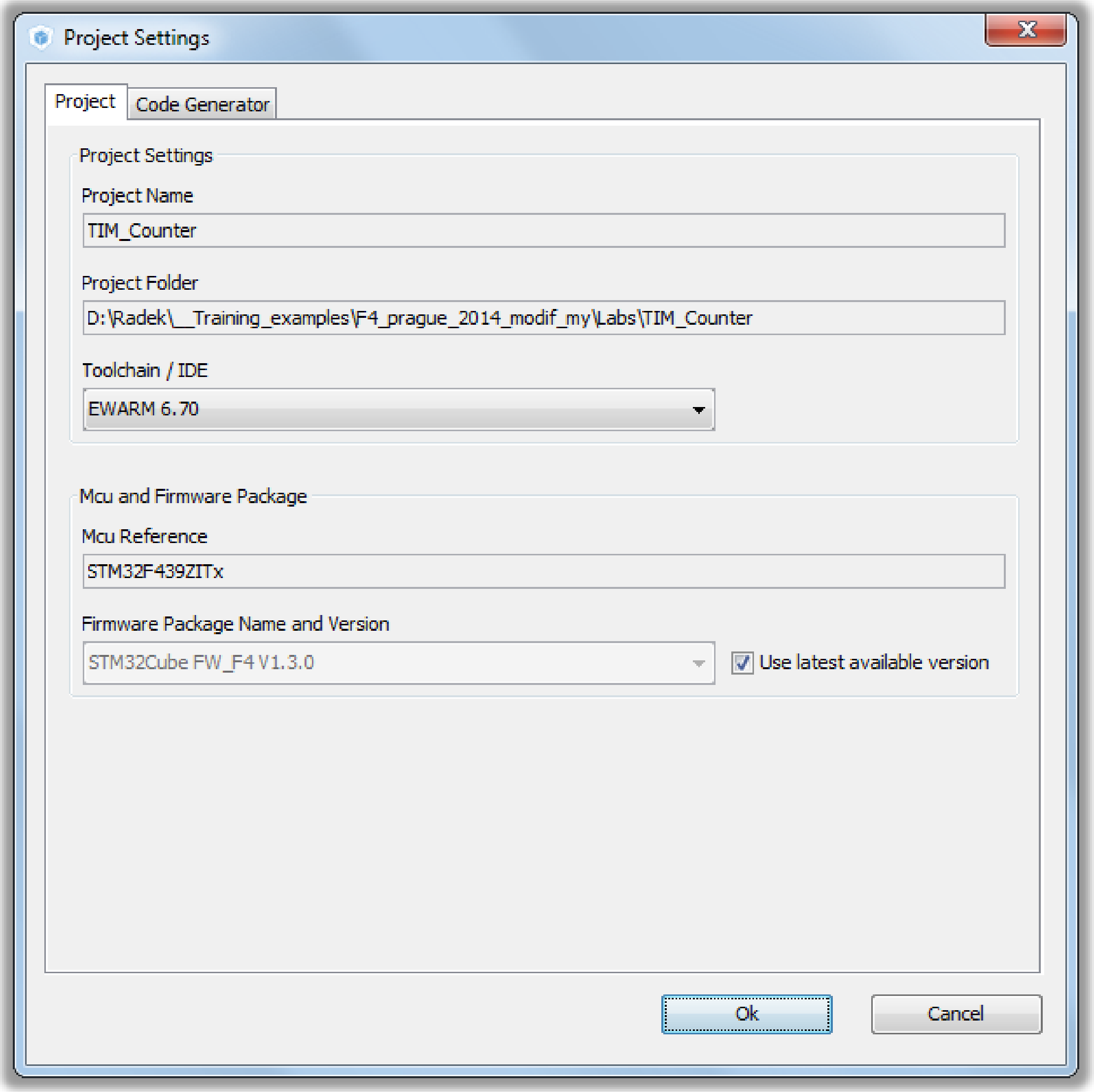
3.2.4 Use TIM as pulse counter

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

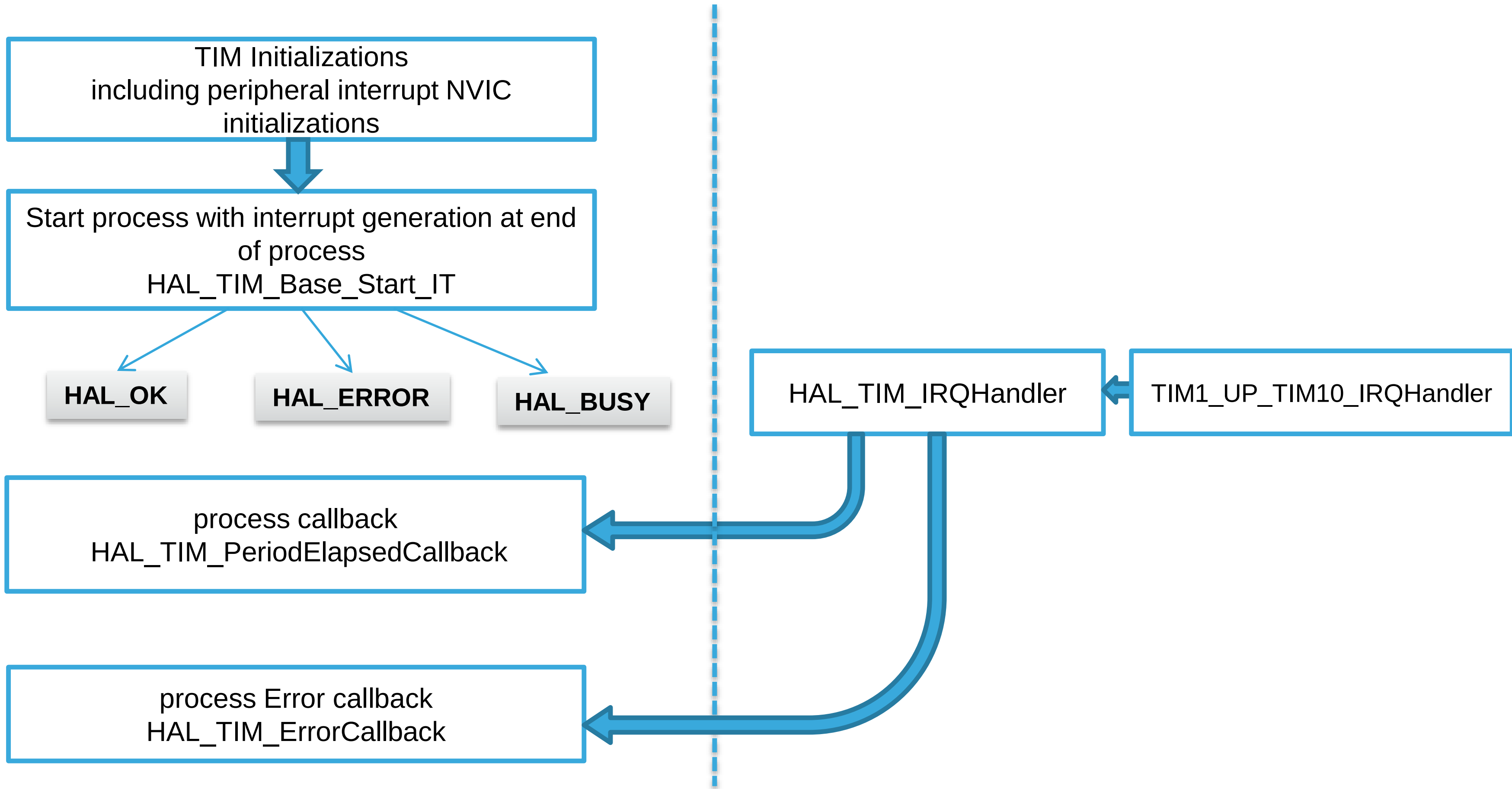


3.2.4 Use TIM as pulse counter

- 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



3.2.4 Use TIM as pulse counter

- 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)

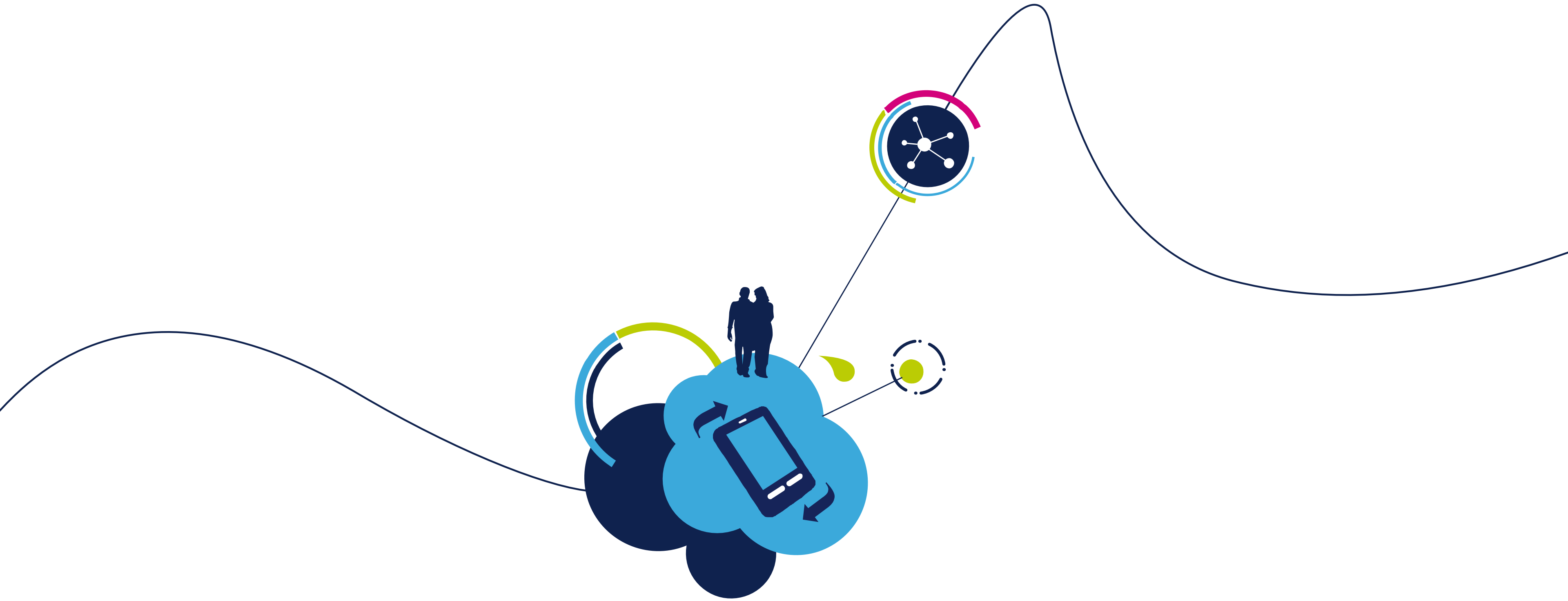
3.2.4 Use TIM as pulse counter

- 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 */
```



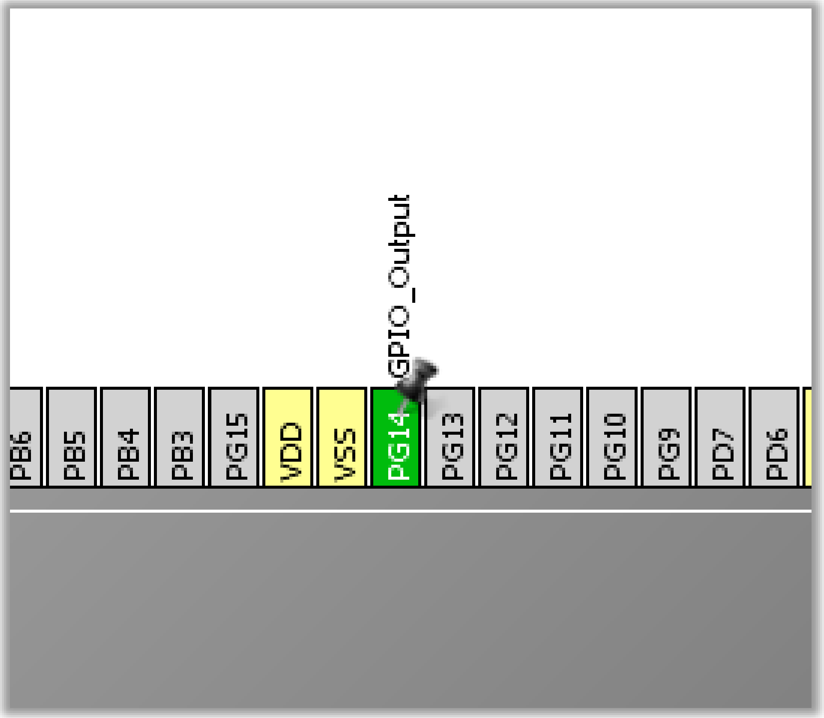
3.3.1 WWDG lab

3.3.1 Use WWDG

- 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

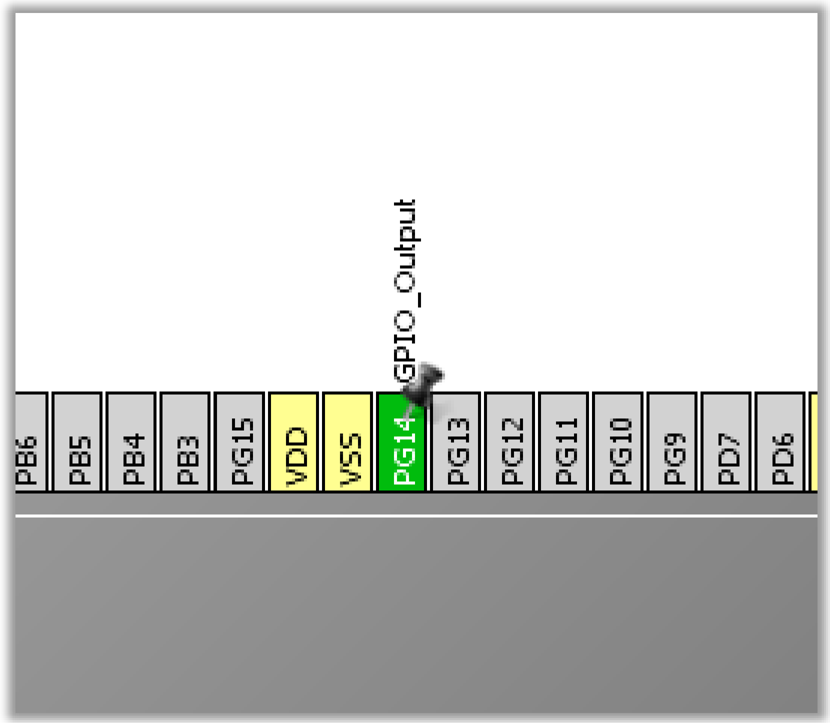
3.3.1 Use WWDG

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



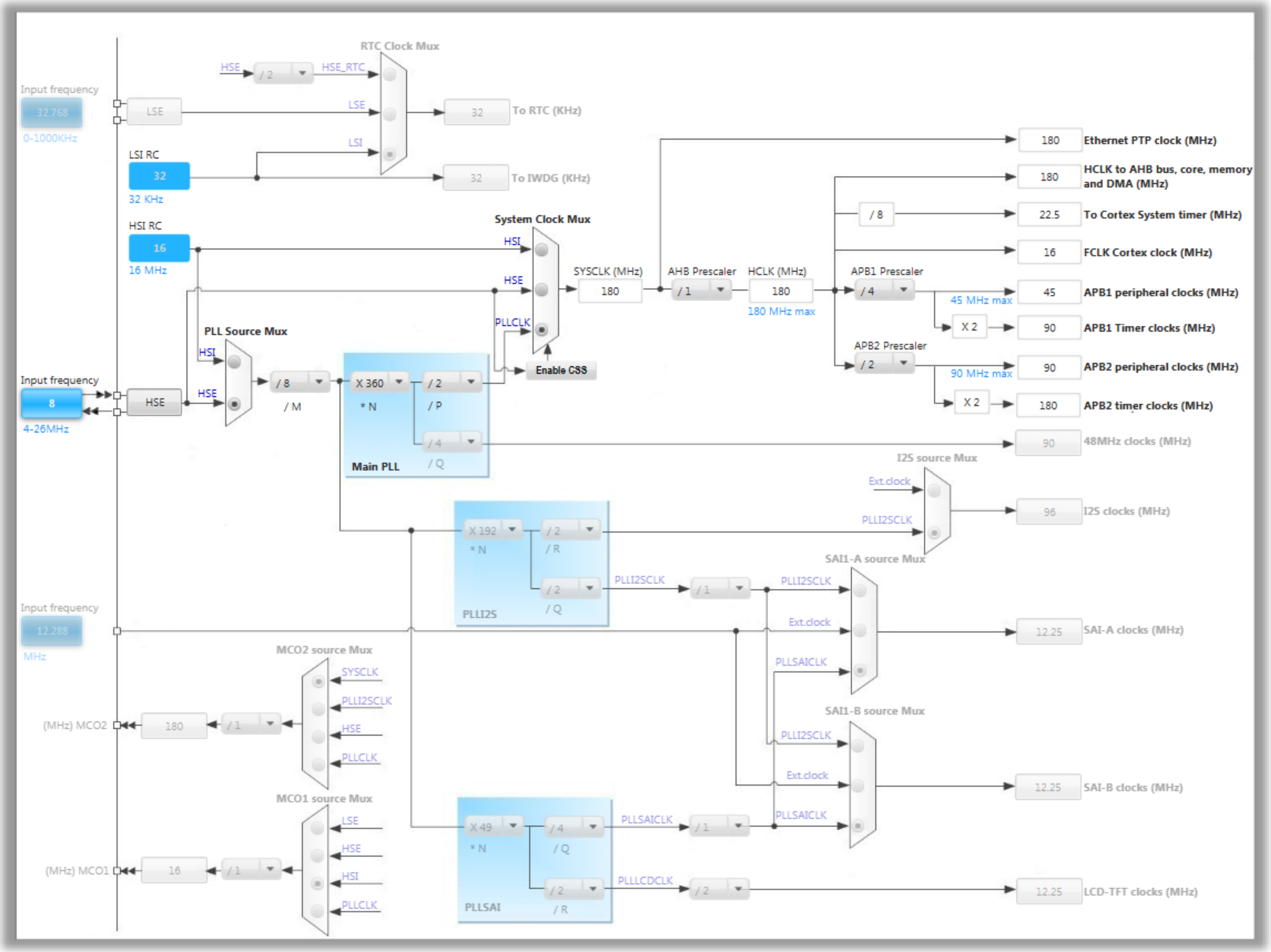
3.3.1 Use WWDG

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



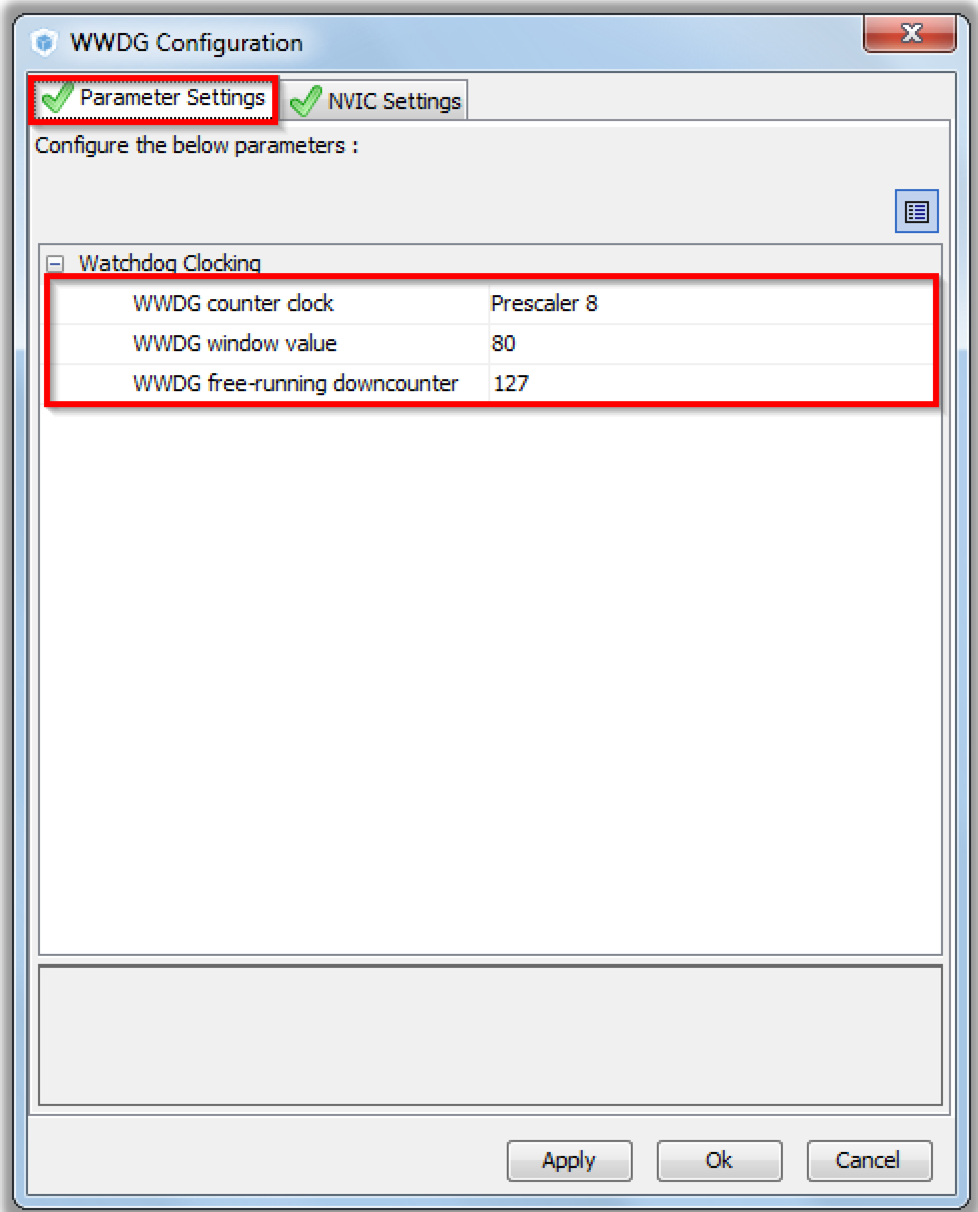
3.3.1 Use WWDG

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



3.3.1 Use WWDG

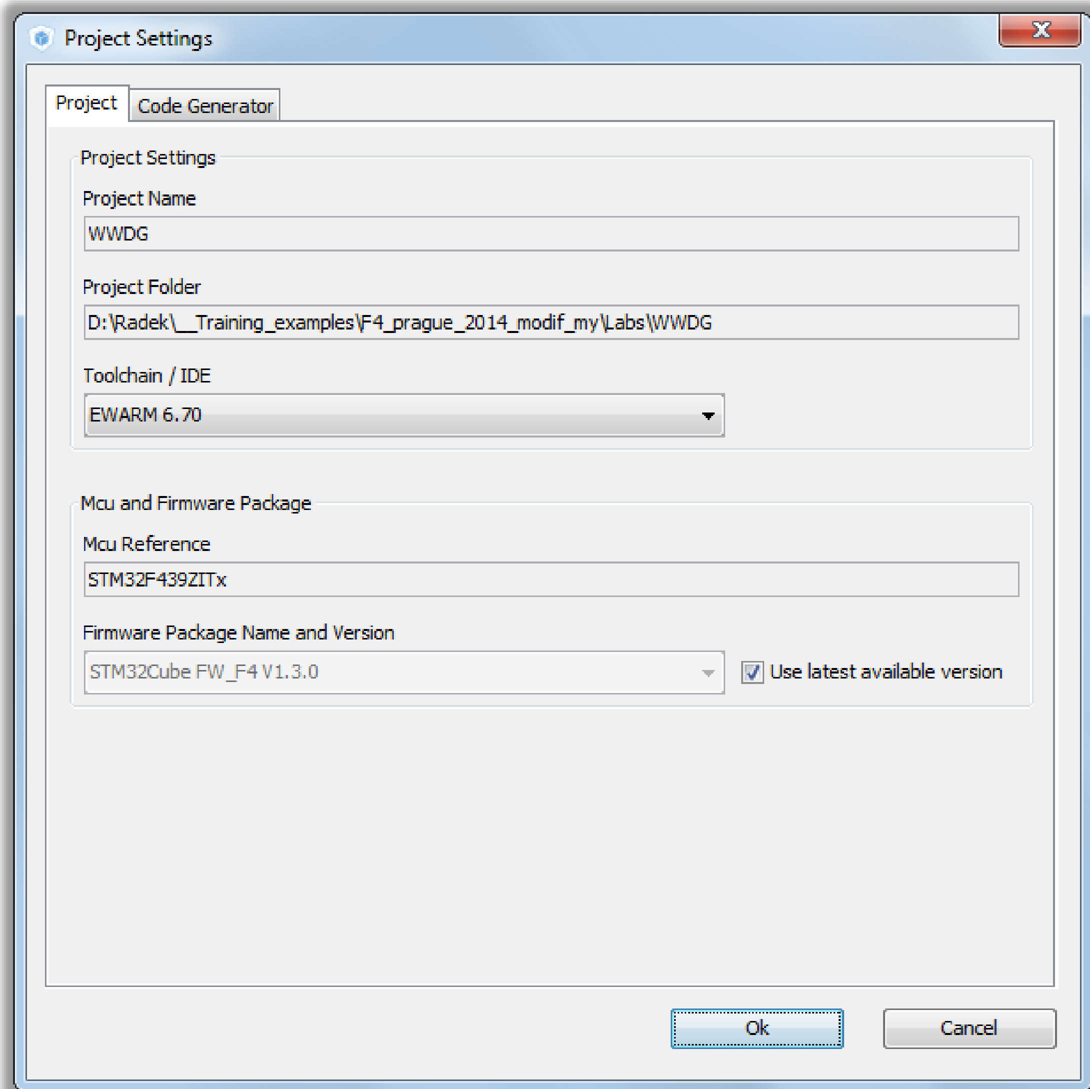
- CubeMX ADC configuration
  - TAB>Configuration>System> >WWDG>Parameter Settings
  - Set prescaller to 8
  - WWDG window to 80
  - And free running counter to 127
  - Button OK



life.augmented

3.3.1 Use WWDG

- 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



life.augmented

3.3.1 Use WWDG

- How calculate the window APB1$_{freq}$=45MHz, prescaller 8

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

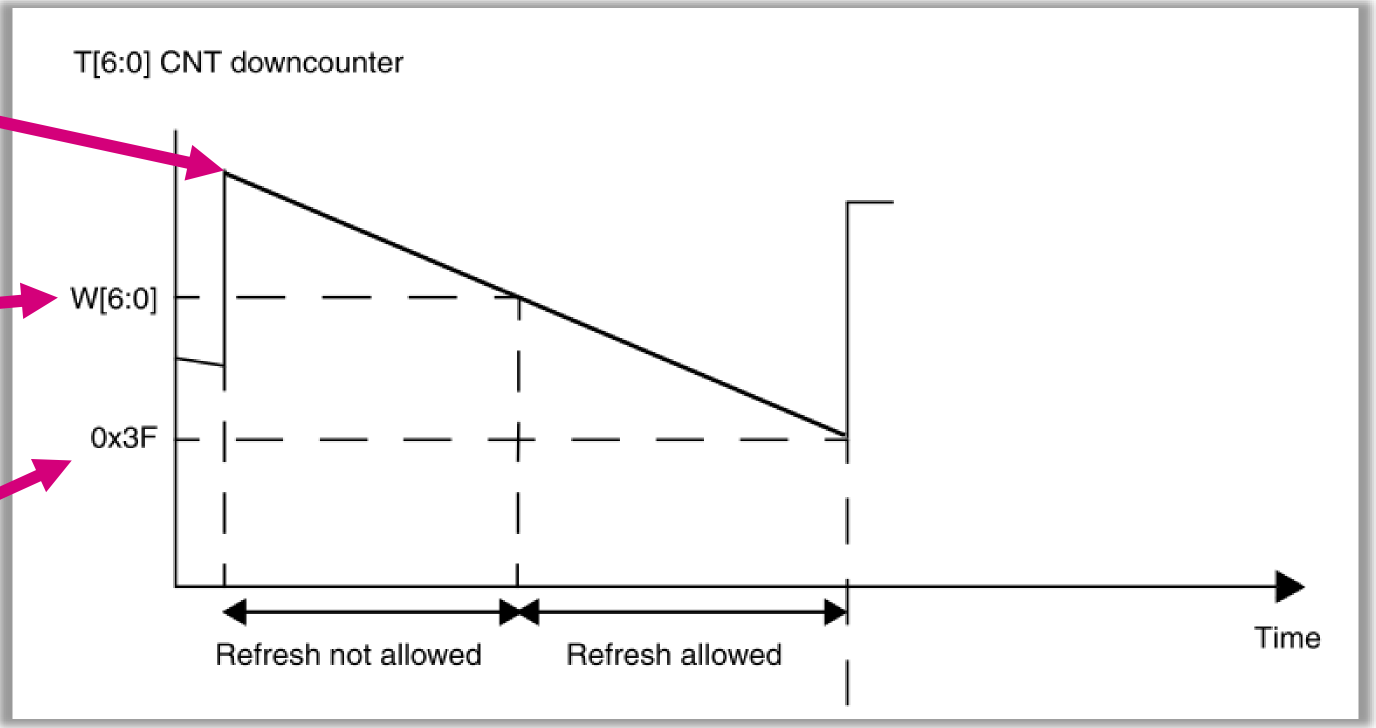
$$= \left(\frac{1}{45*10^6}\right)*4096*8*(127-80) = 34.2ms$$

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

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 */
```

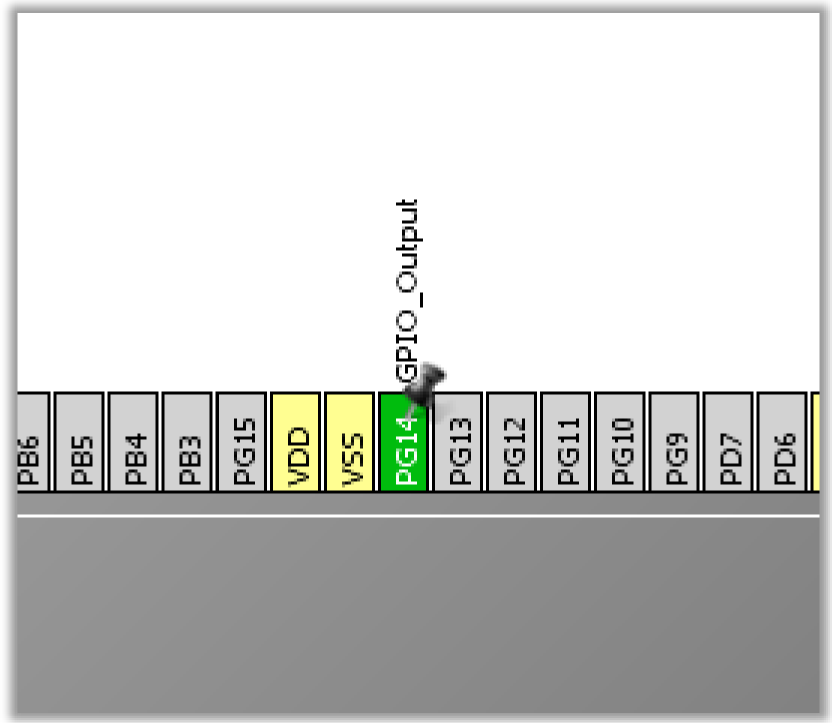
3.3.2 IWDG lab

3.3.2 Use IWDG

- 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

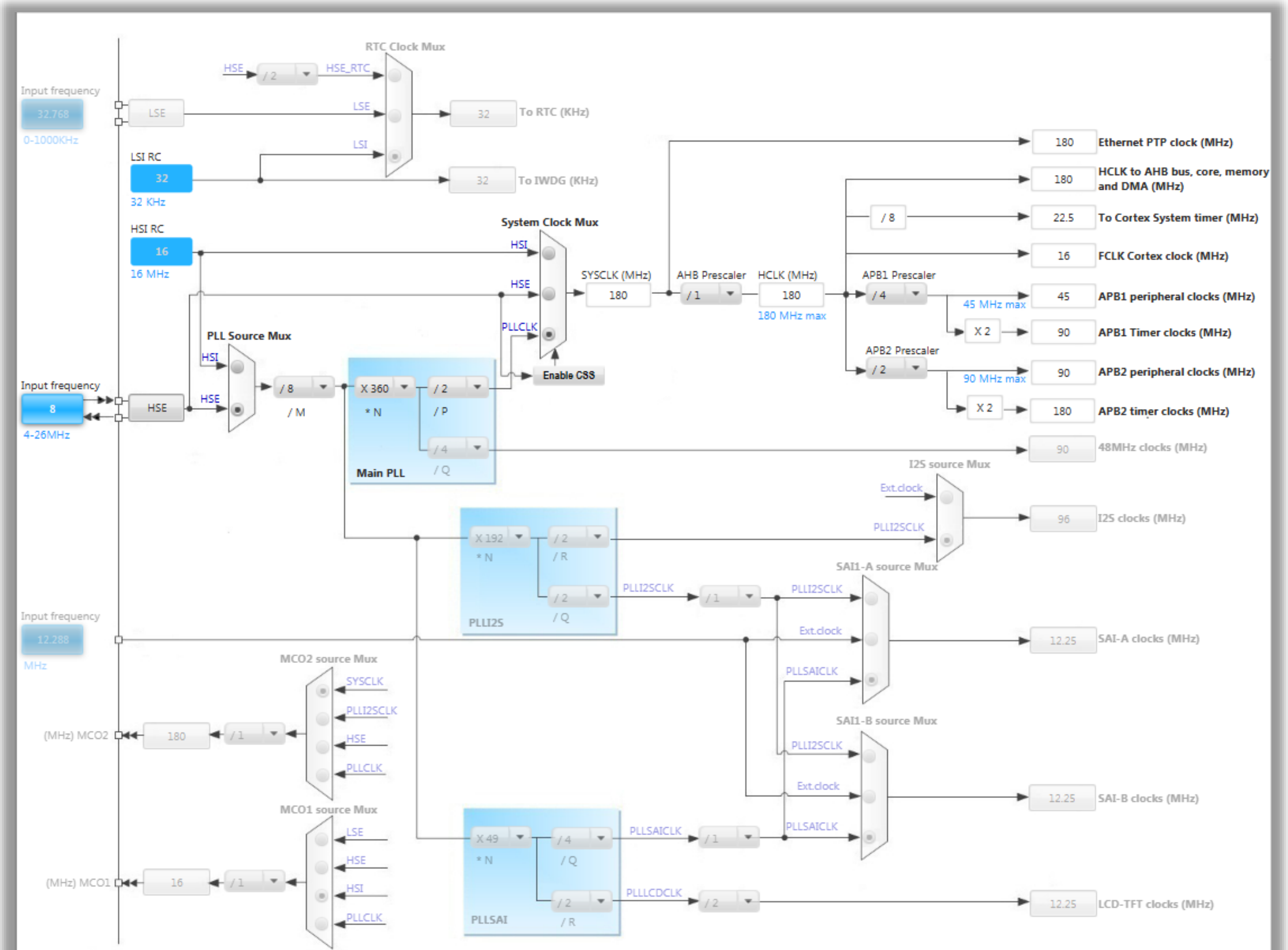
3.3.2 Use IWDG

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



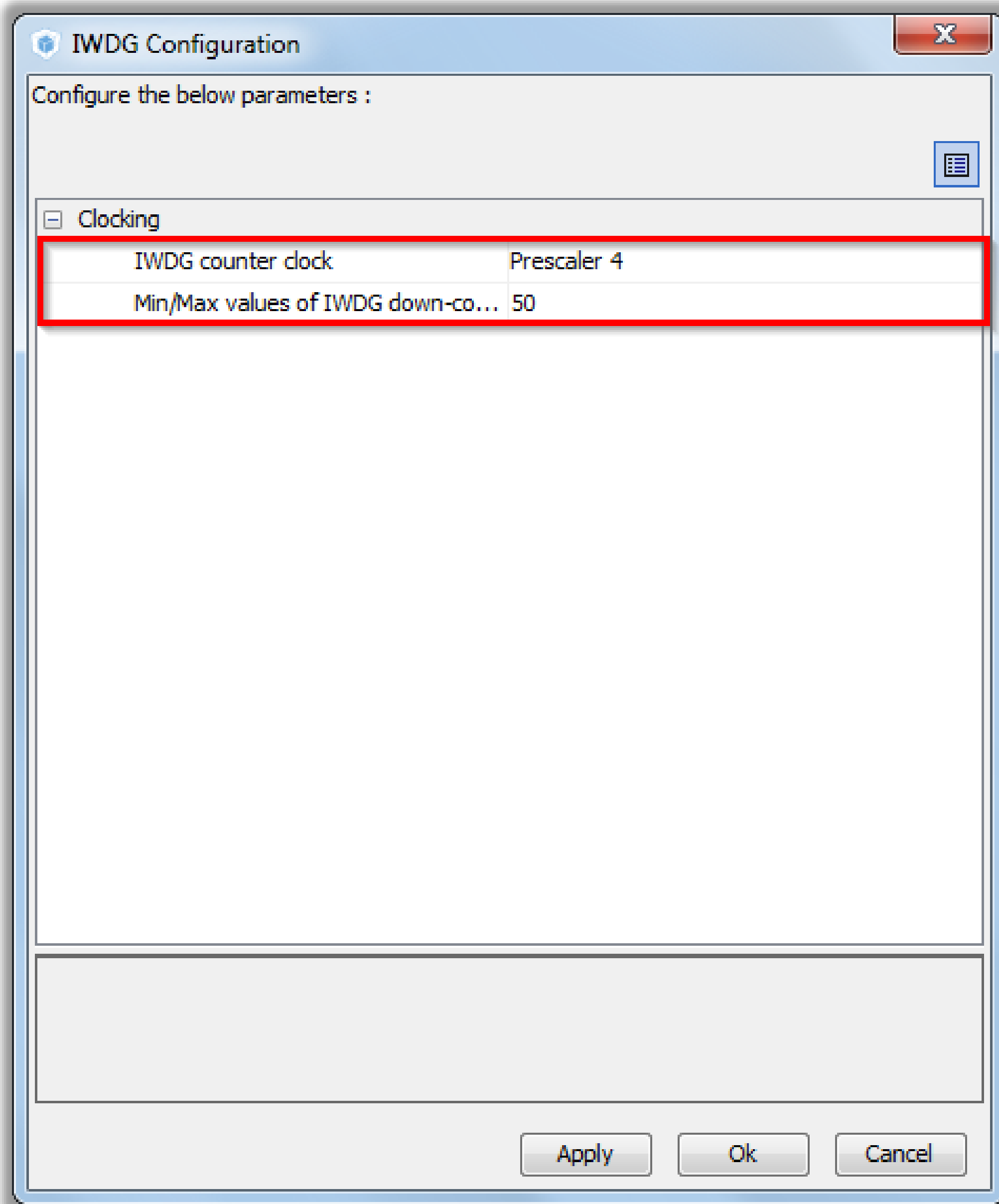
3.3.2 Use IWDG

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



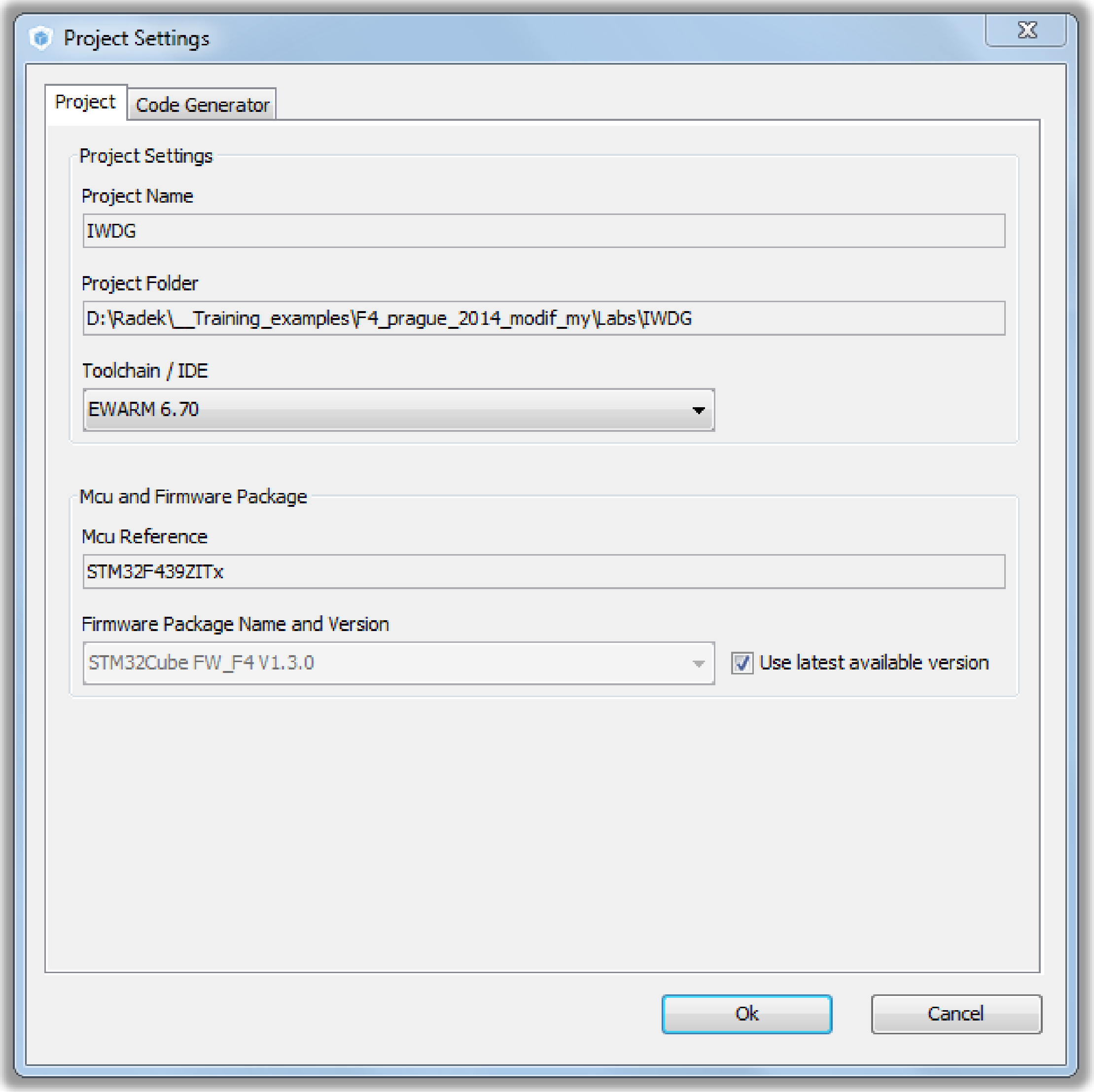
3.3.2 Use IWDG

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



3.3.2 Use IWDG

- 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



life.augmented

3.3.2 Use IWDG

- IWDG refresh interval

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

- 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 */
```

3.3.2 Use IWDG

- Hardware IWDG
  - Remove IWDG start from project

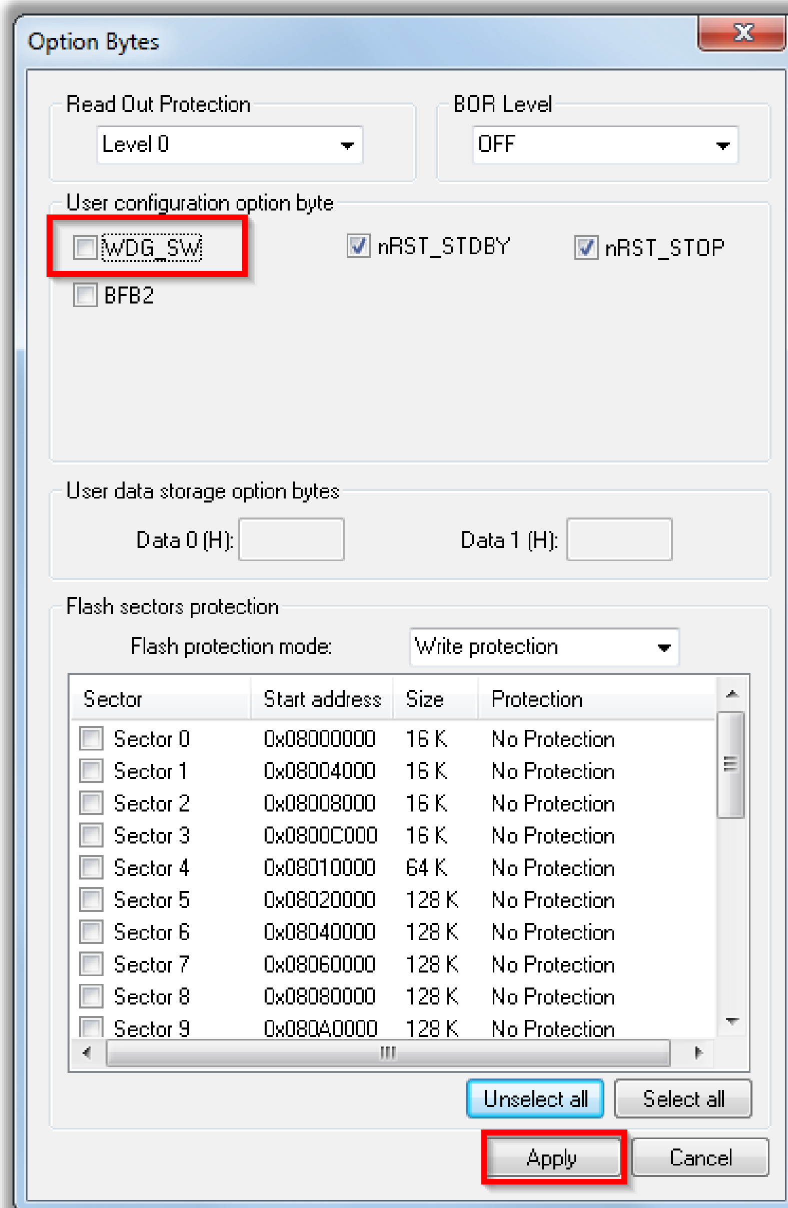
```
/* USER CODE BEGIN 2 */
/* USER CODE END 2 */
```

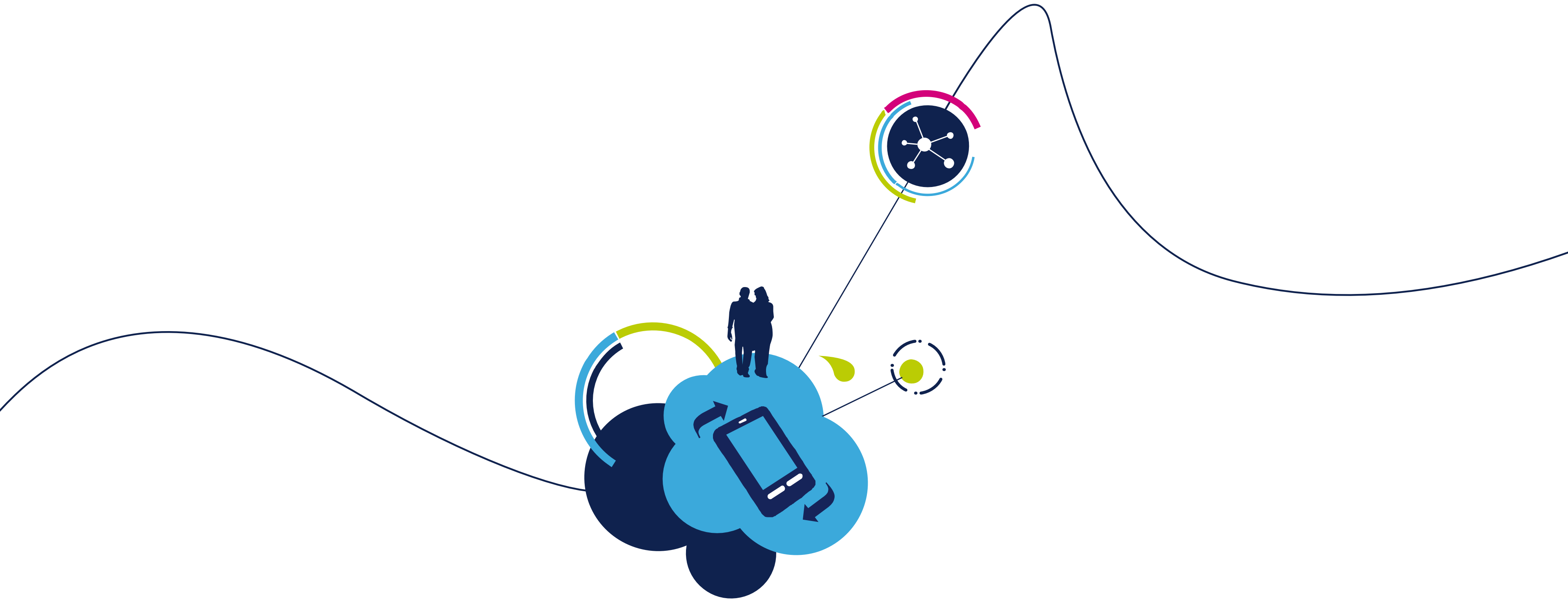
  - Use ST-Link utility and enable IWDG Hardware start

3.3.2 Use IWDG

- 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





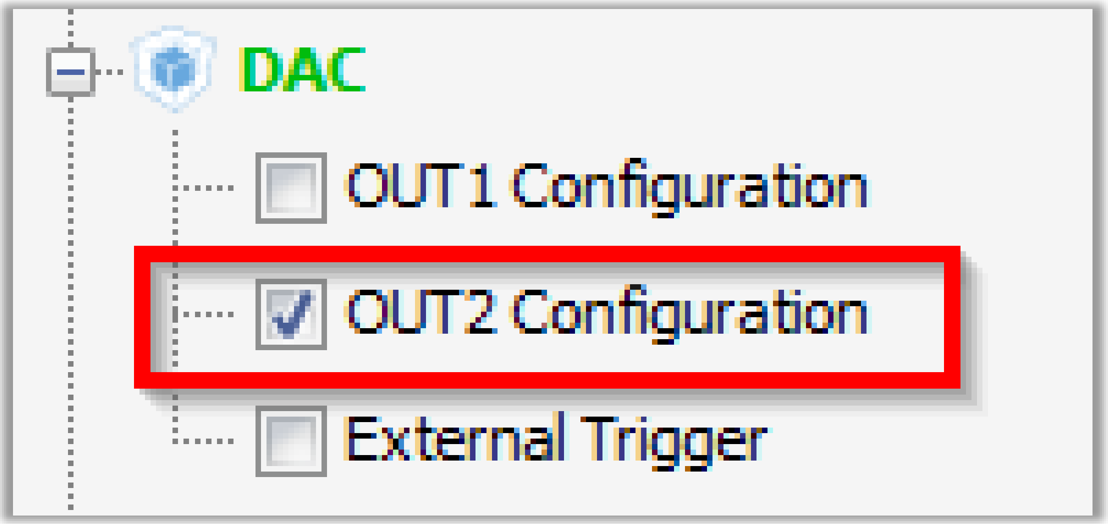
4.1.1 DAC wave generator lab

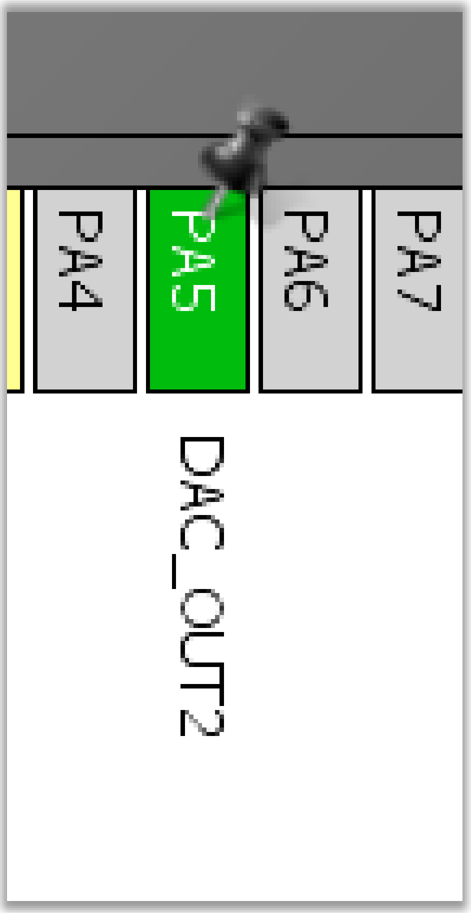
4.1.1 Use DAC as wave generator

- 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

4.1.1 Use DAC as wave generator

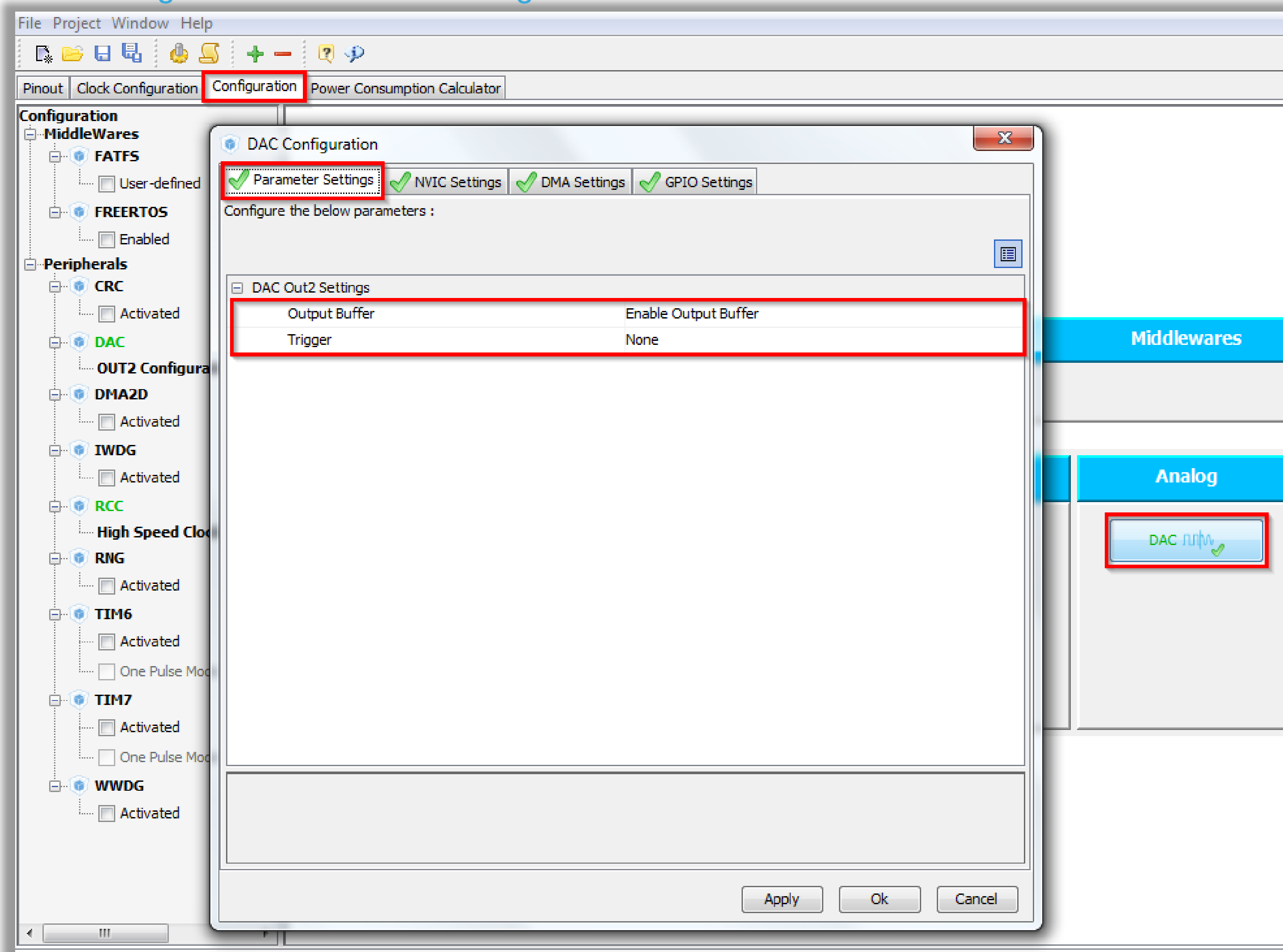
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- CubeMX DAC selection
  - Select DAC OUT2





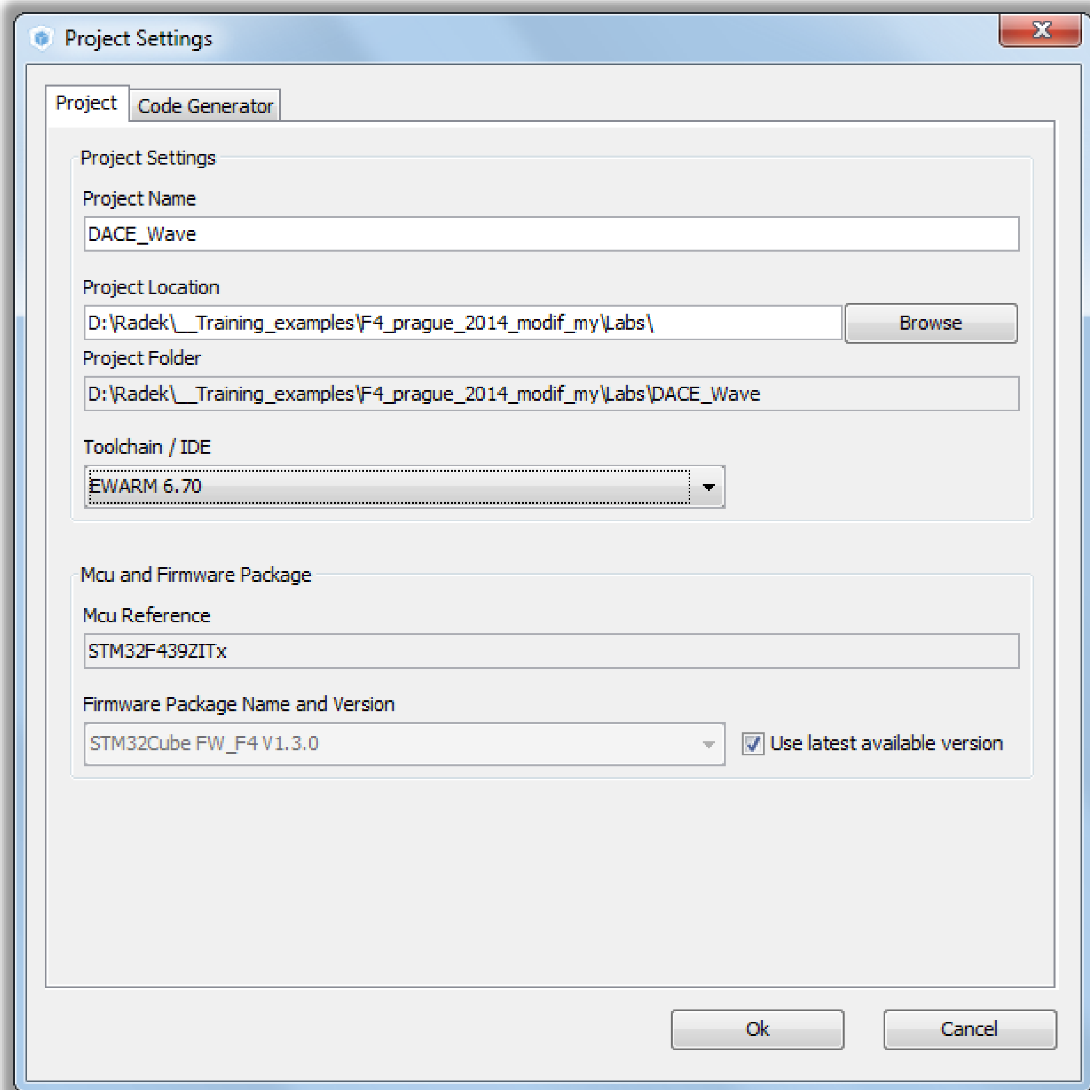
4.1.1 Use DAC as wave generator

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



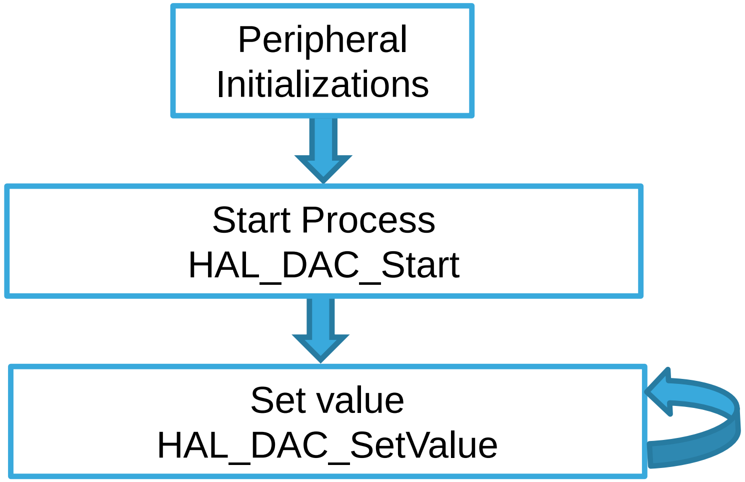
4.1.1 Use DAC as wave generator

- 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



4.1.1 Use DAC as wave generator

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



4.1.1 Use DAC as wave generator

- 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)

4.1.1 Use DAC as wave generator

- 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 */
```

4.2.1 ADC Poll lab

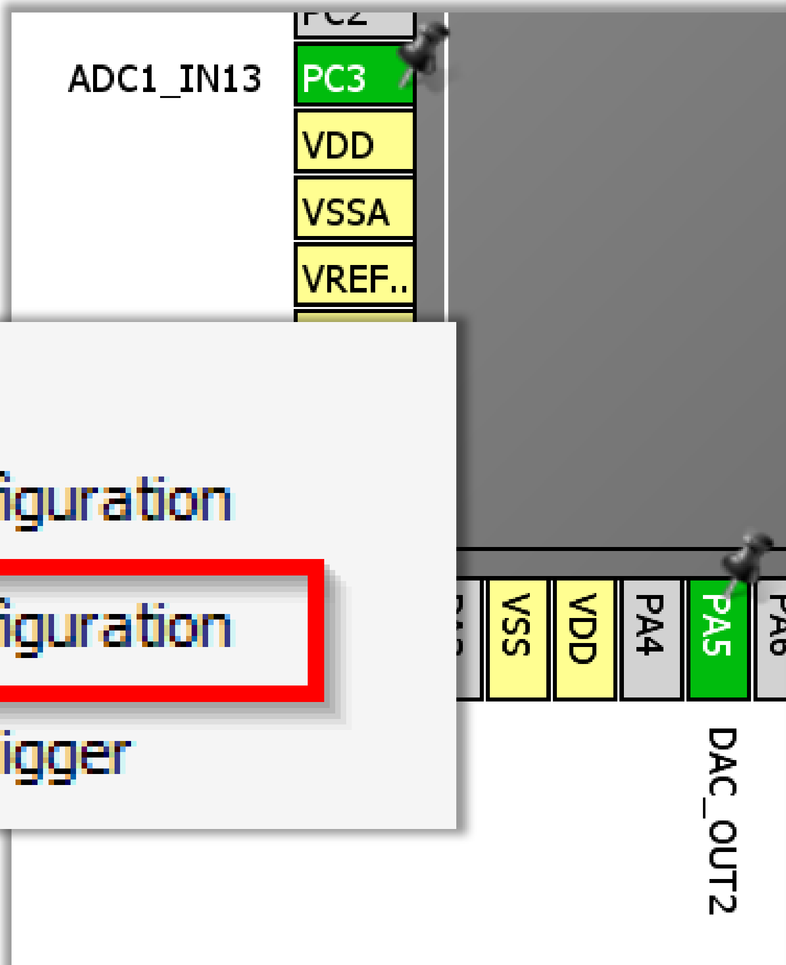
4.2.1 Use ADC in polling mode

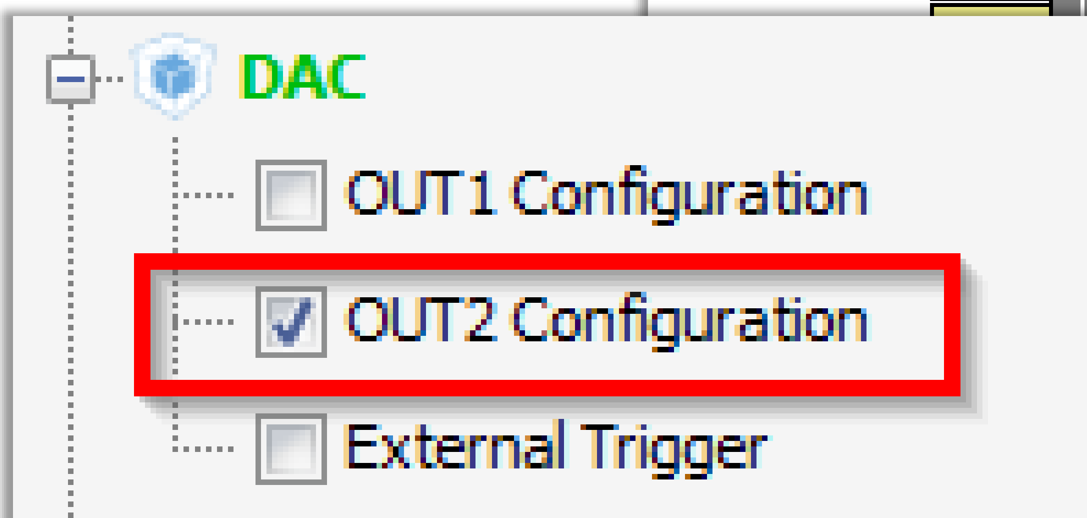
- Objective
  - Use the DAC part from previous lab
  - Learn how to setup ADC in CubeMX
  - How to Generate Code in CubeMX and use HAL functions
- Goal
  - Configure ADC in poll in CubeMX and Generate Code
  - Learn how to start ADC and measure the DAC
  - Verify the measured wave in STMStudio (http://www.st.com/web/en/catalog/tools/PF251373 require JAVA)

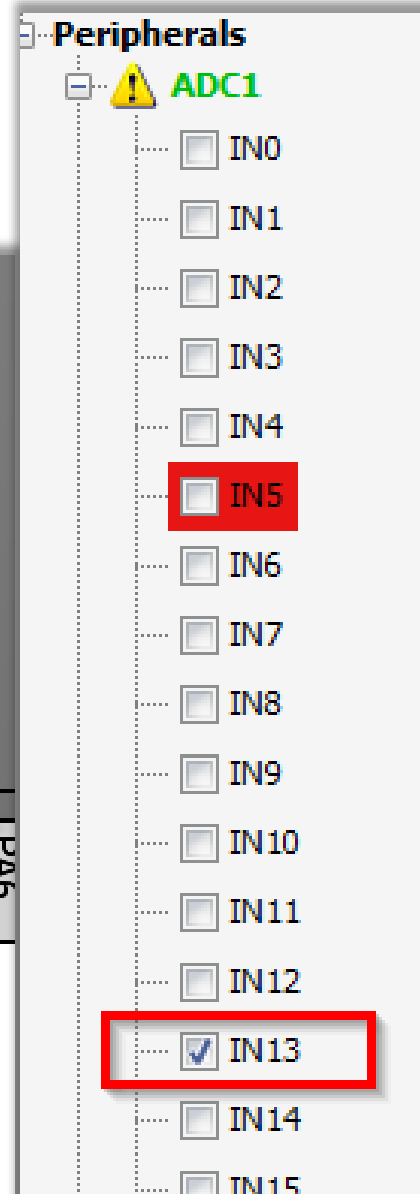
life.augmented

4.2.1 Use ADC in polling mode

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



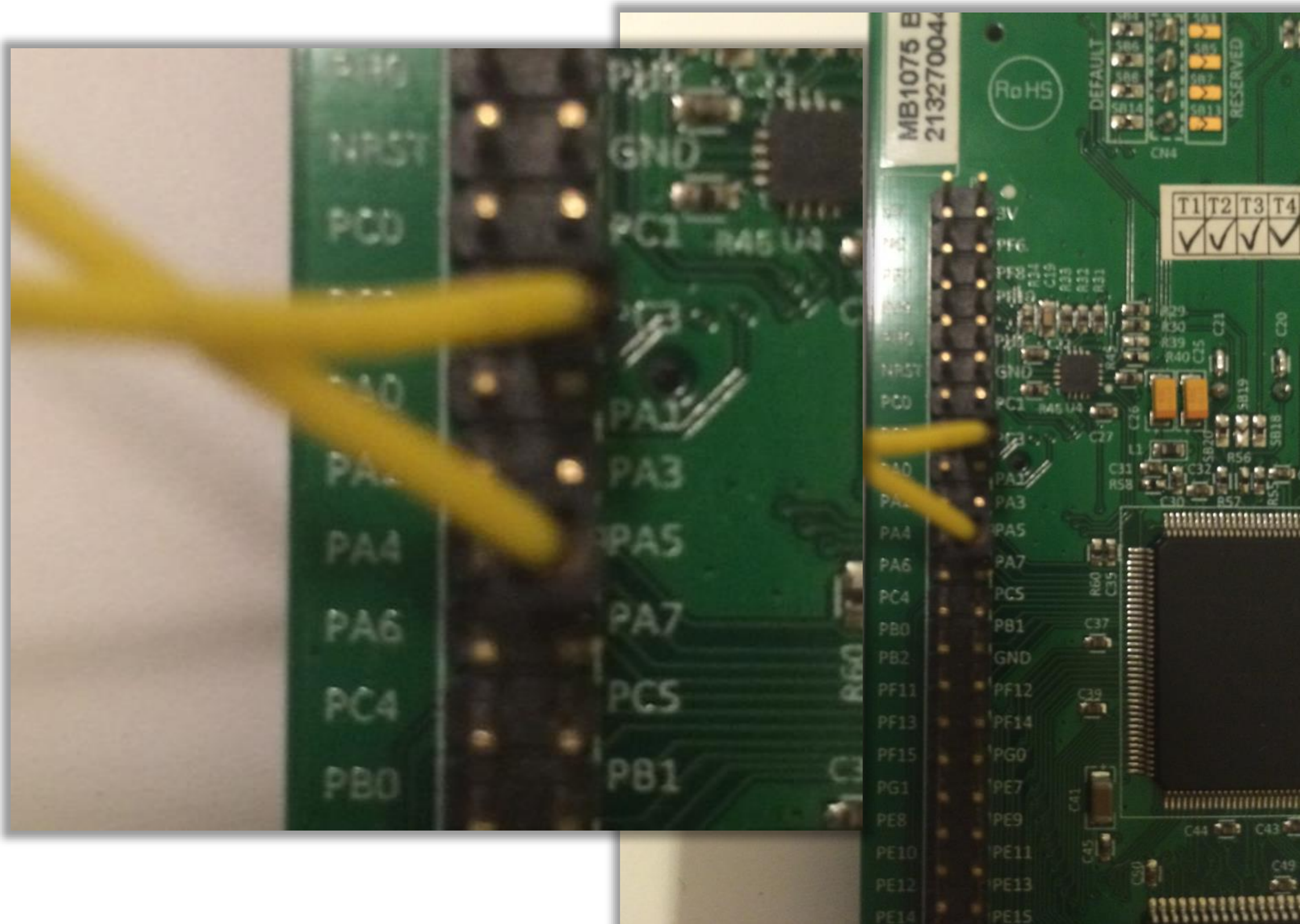




life.augmented

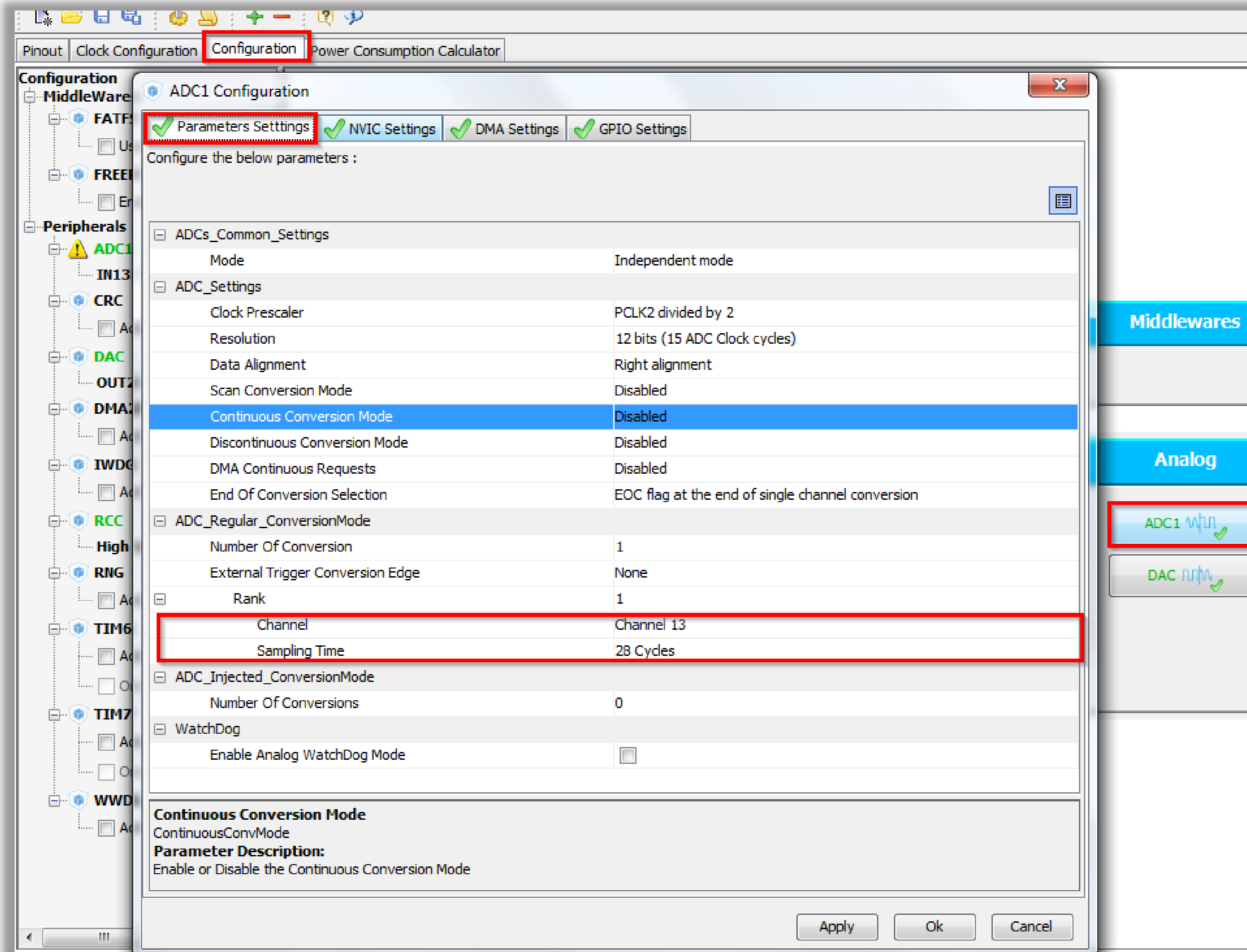
4.2.1 Use ADC in polling mode

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



4.2.1 Use ADC in polling mode

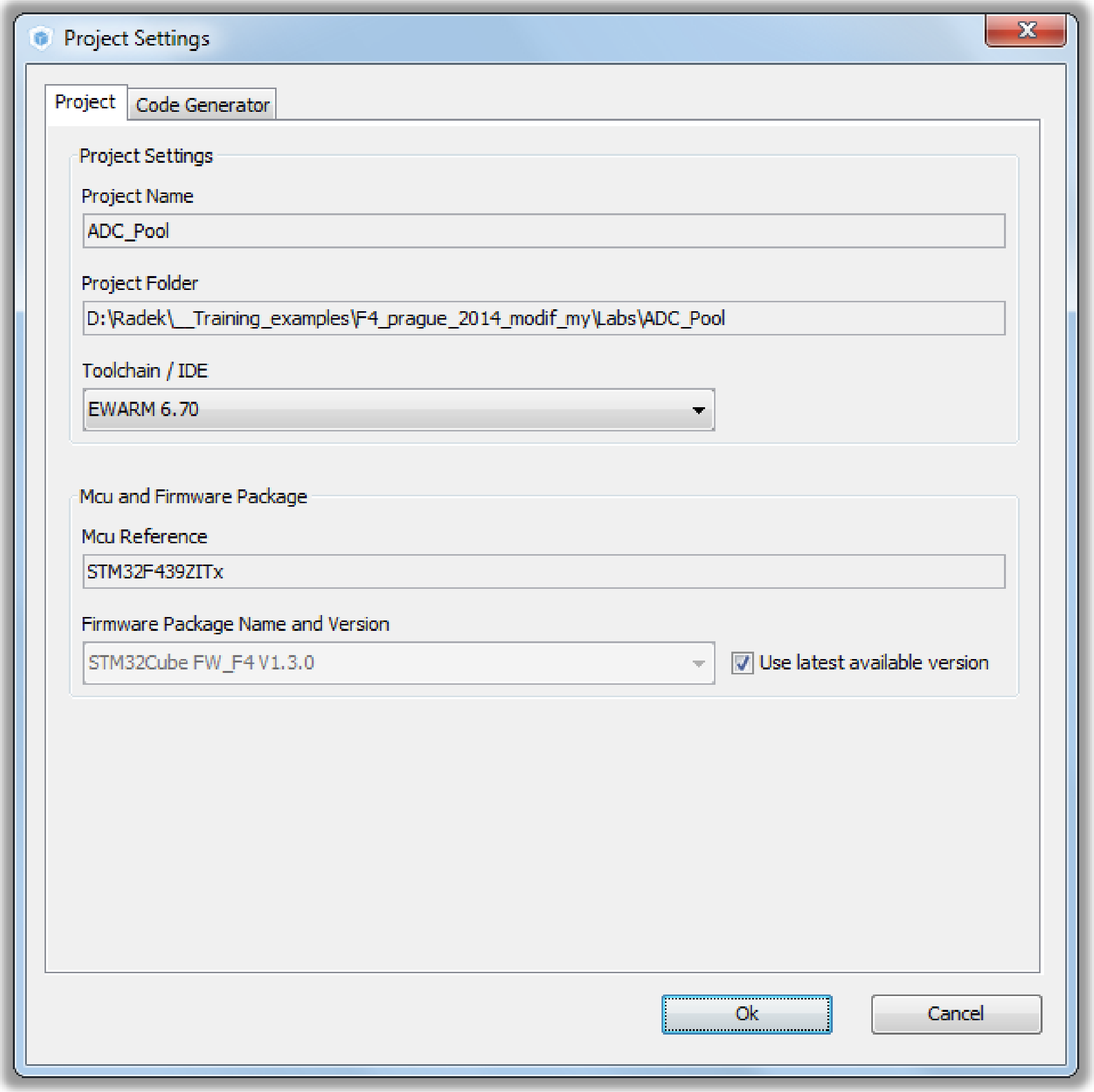
- CubeMX ADC configuration
  - TAB>Configuration>Analog>ADC1>Parametr Settings
  - Set ADC1
  - Set sampling time for CH13
  - Button OK
- DAC from previous example



life.augmented

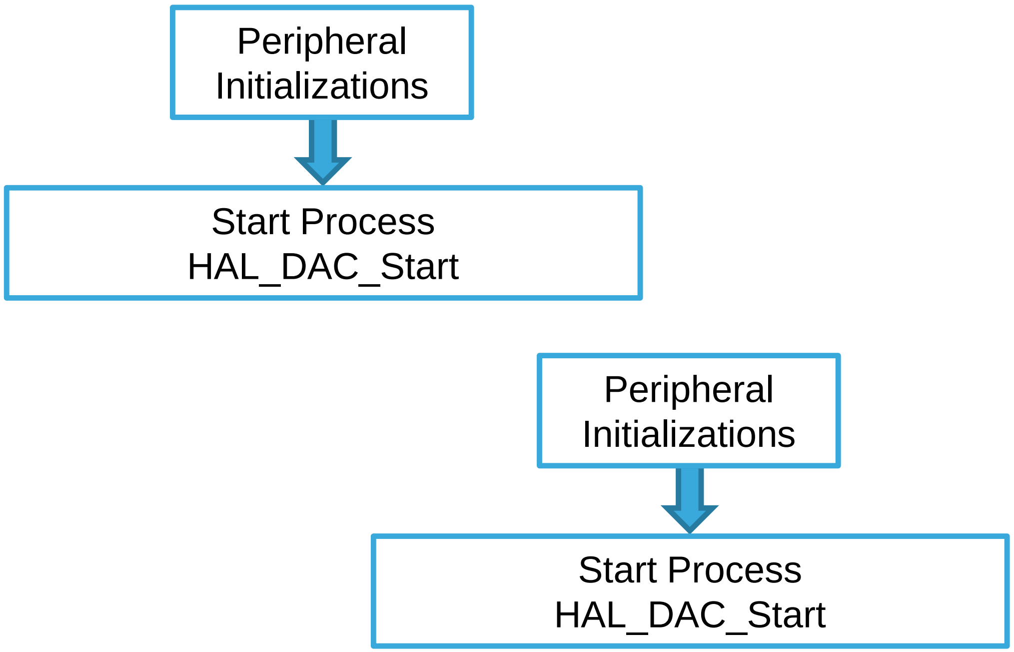
4.2.1 Use ADC in polling mode

- 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



4.2.1 Use ADC in polling mode

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



4.2.1 Use ADC in polling mode

- 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)

4.2.1 Use ADC in polling mode

- 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 */
```

4.2.1 Use ADC in polling mode

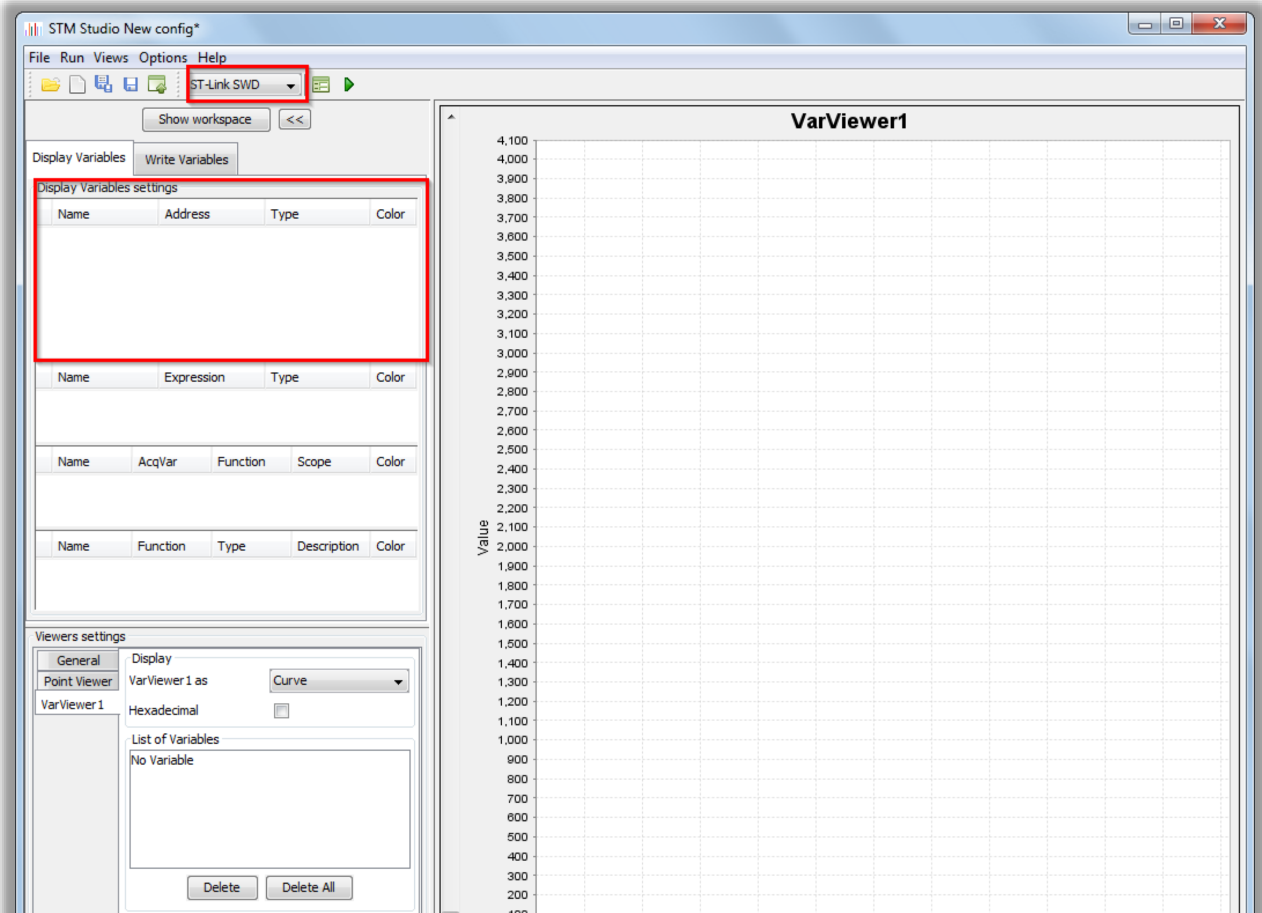
- Solution
  - Main loop with DAC set and ADC set

```
/* 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 */
```

life.augmented

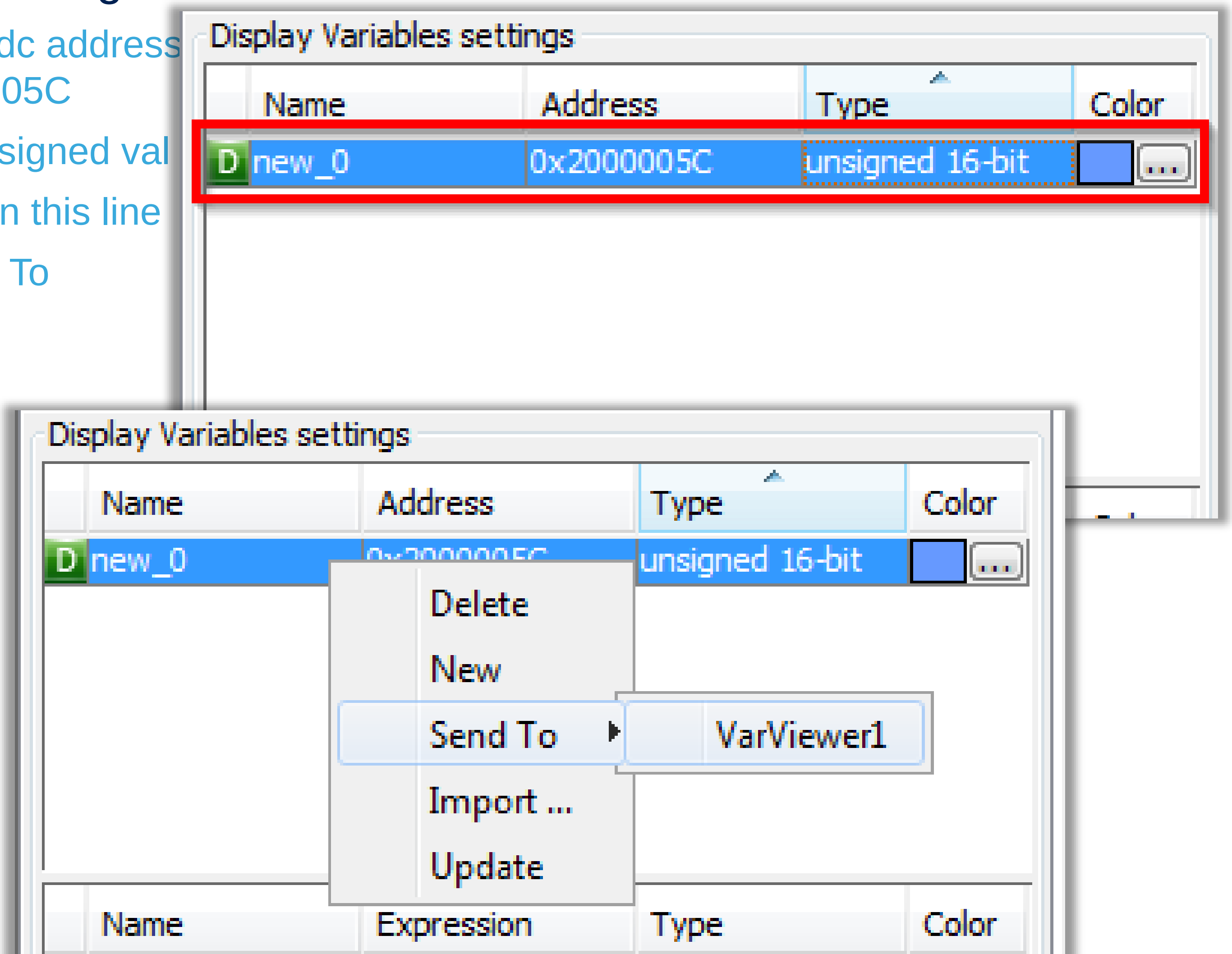
4.2.1 Use ADC in polling mode

- 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 compilator and optimizations)
- Start the STMStudio
  - Set the ST Link SWD
  - Right click into Display variable settings
  - Select NEW



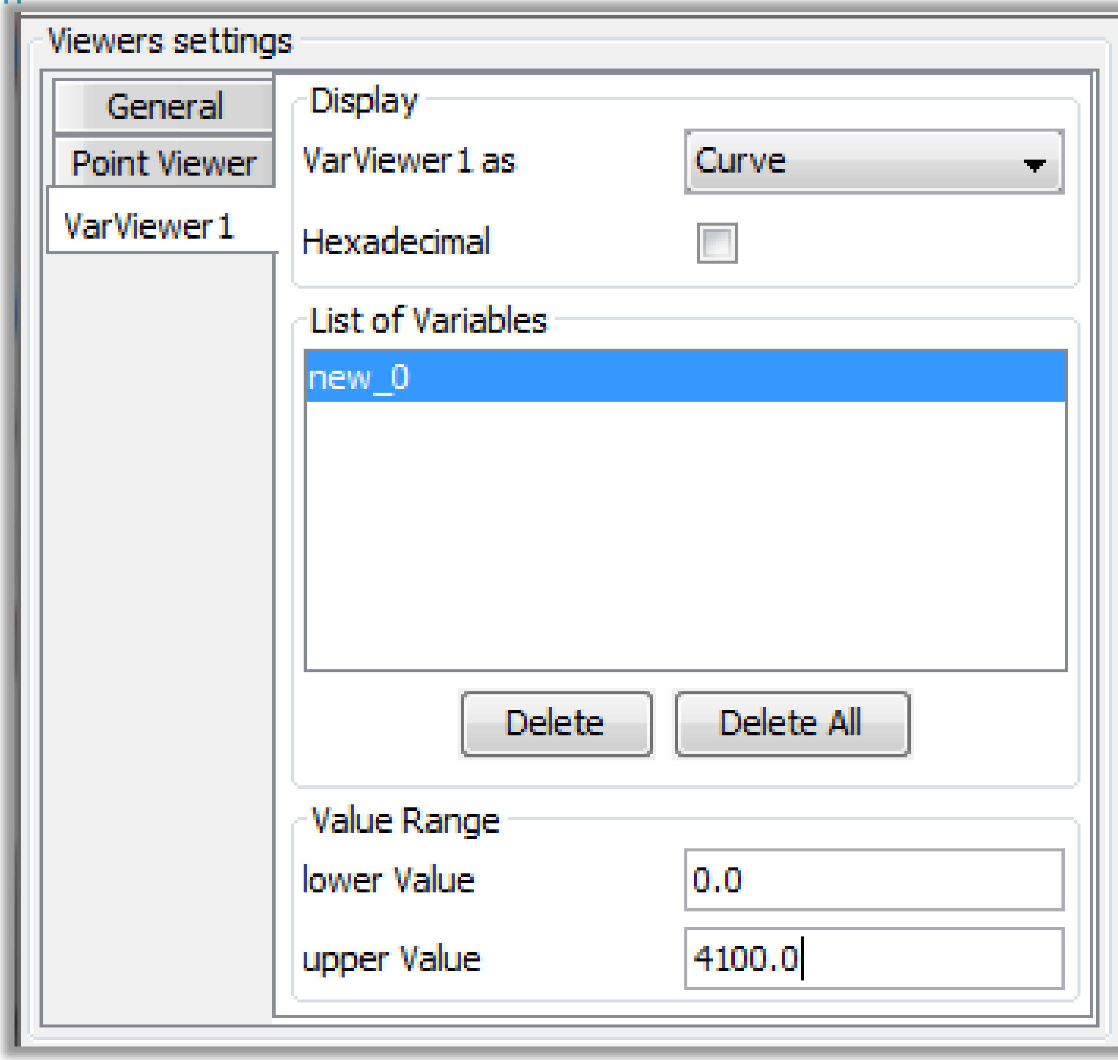
4.2.1 Use ADC in polling mode

- STM studio settings
  - Set value_adc address my 0x2000005C
  - Set 16bit unsigned val
  - Right click on this line
  - Select Send To VarViewer1



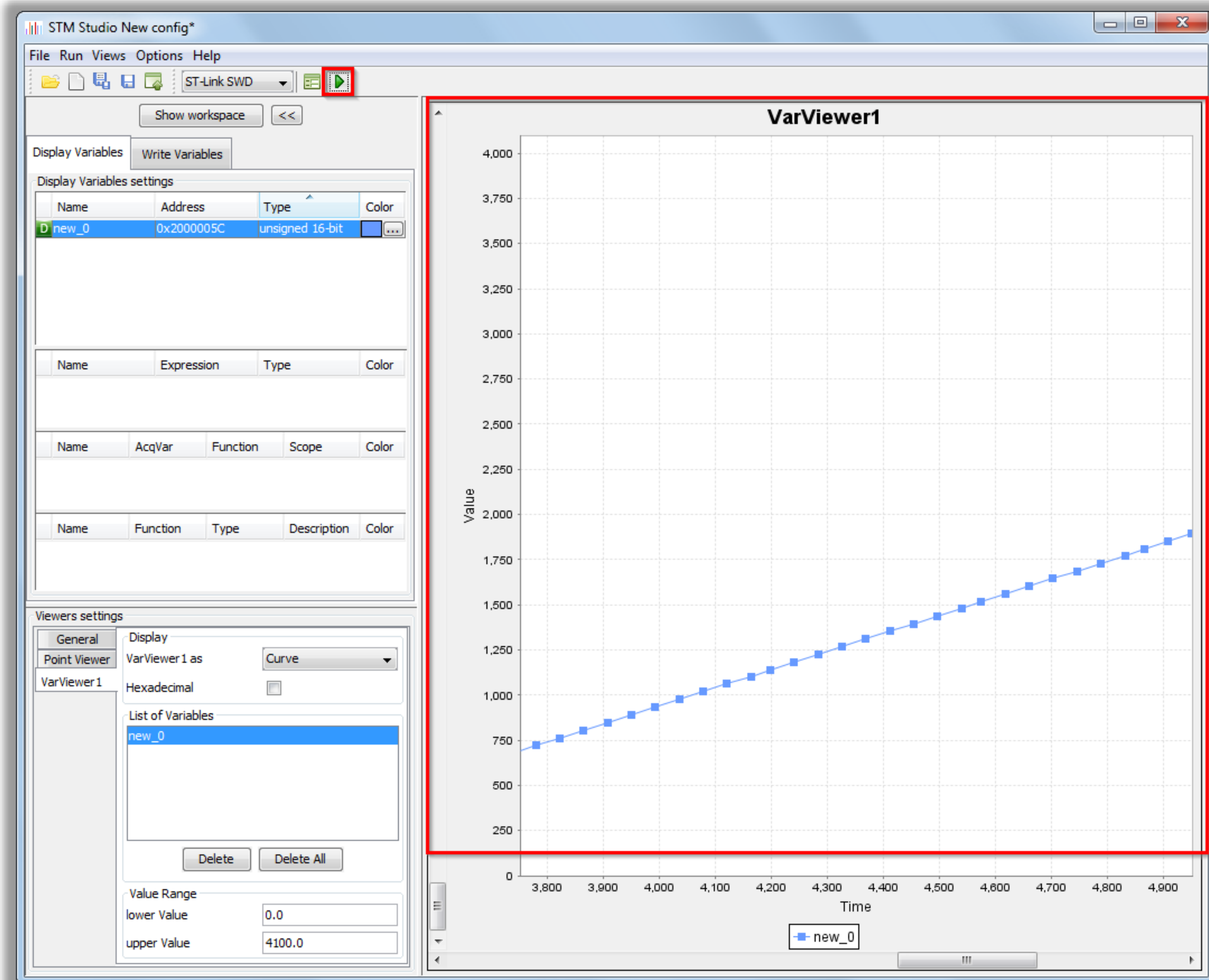
4.2.1 Use ADC in polling mode

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

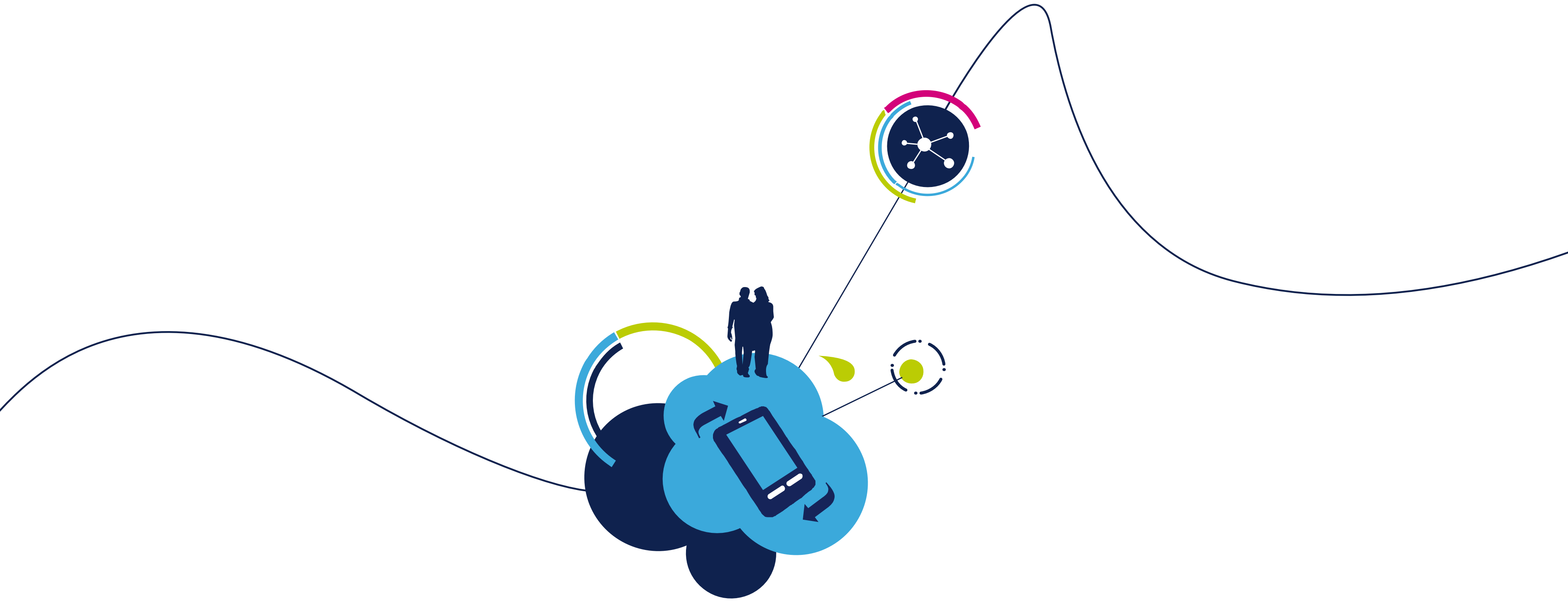


4.2.1 Use ADC in polling mode

- STM studio settings
  - Now press green play button
  - And you will see content of value_adc



life.augmented



4.2.2 ADC Interrupt lab

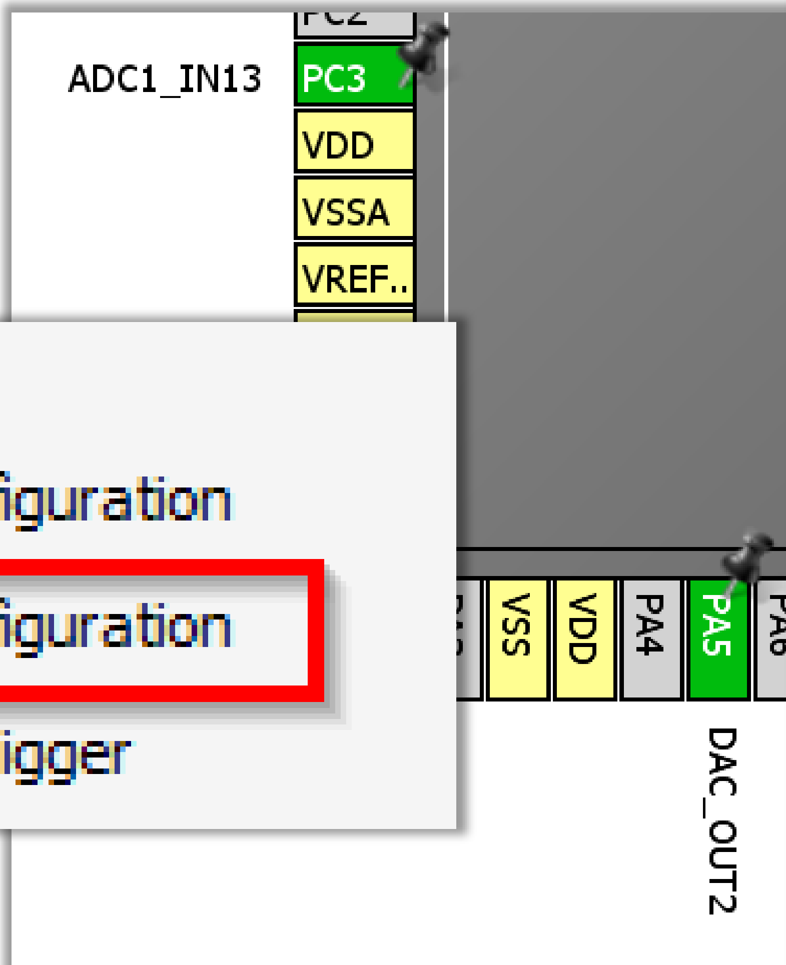
4.2.2 Use ADC with interrupt

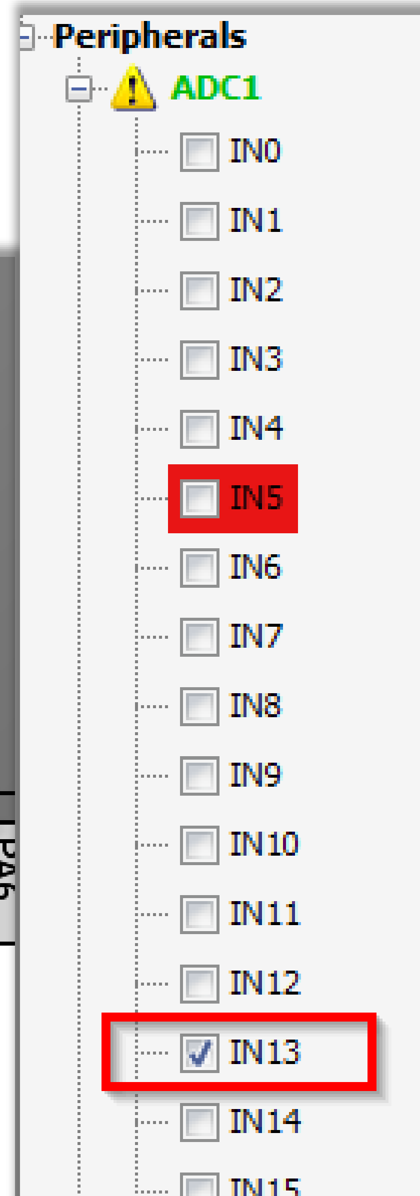
- Objective
  - Use the DAC part from previous lab
  - Learn how to setup ADC with interrupt in CubeMX
  - How to Generate Code in CubeMX and use HAL functions
- Goal
  - Configure ADC in interrupt in CubeMX and Generate Code
  - Learn how to start ADC and measure the DAC
  - Verify the measured wave in STMStudio (http://www.st.com/web/en/catalog/tools/PF251373 require JAVA)

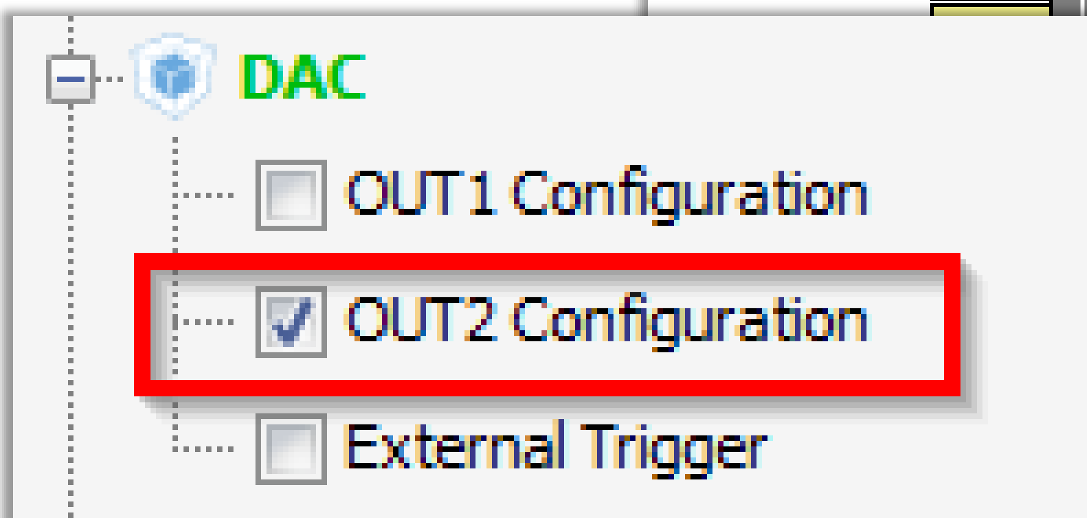
life.augmented

4.2.2 Use ADC with interrupt

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

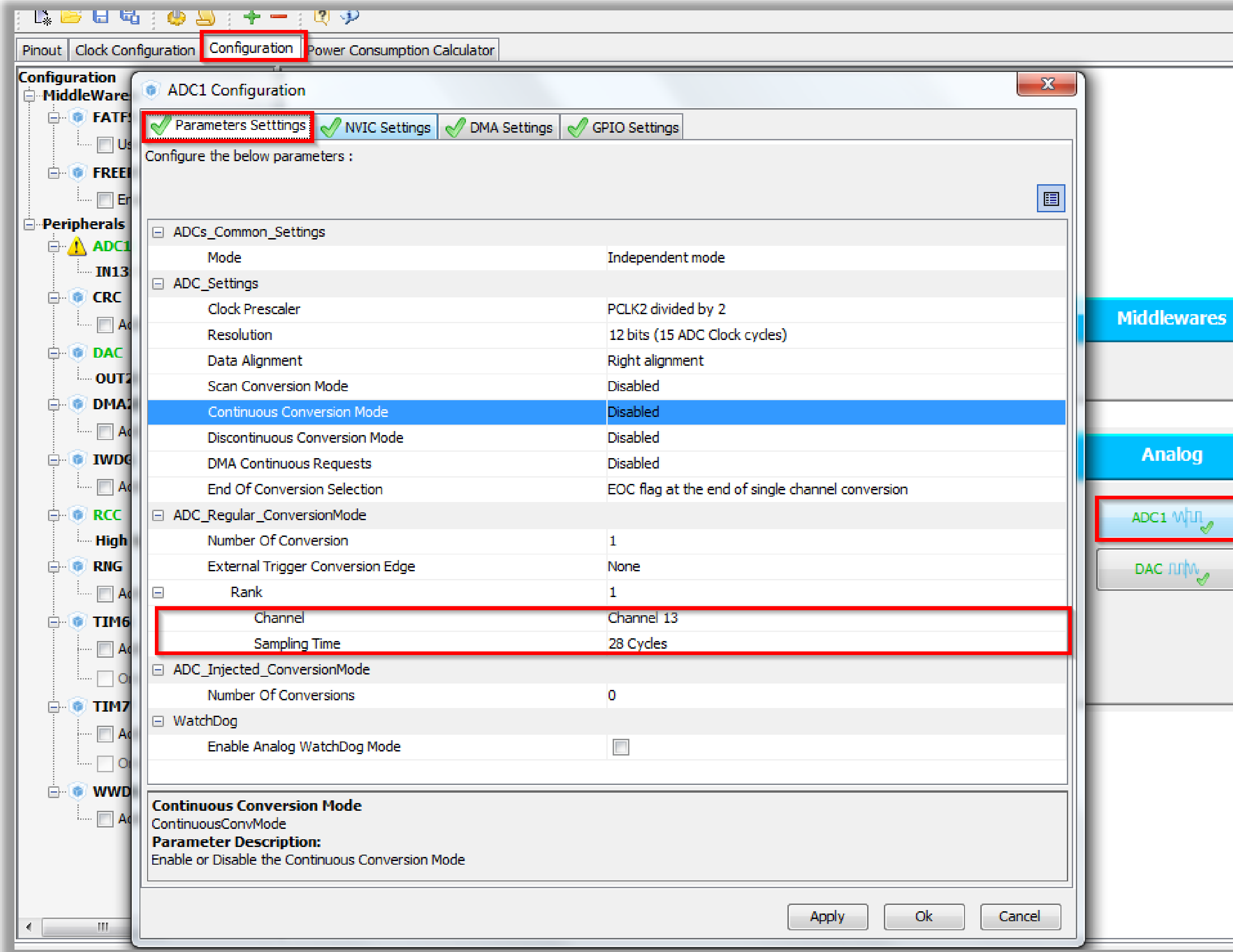






4.2.2 Use ADC with interrupt

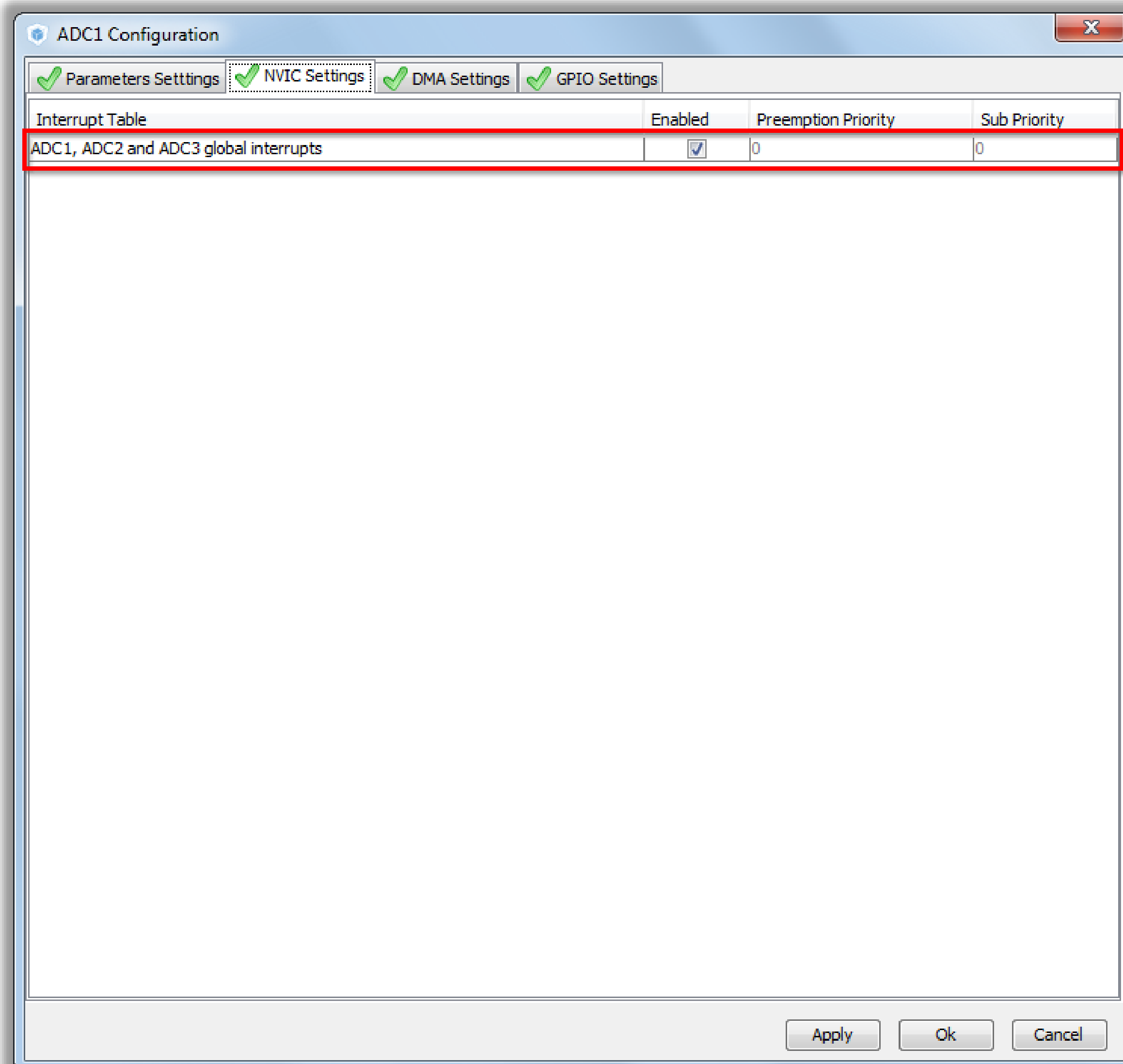
- CubeMX ADC configuration
  - TAB>Configuration>Analog>ADC1>Parametr Settings
  - Set ADC1
  - Set sampling time for CH13
  - Button OK
- DAC from previous example



life.augmented

4.2.2 Use ADC with interrupt

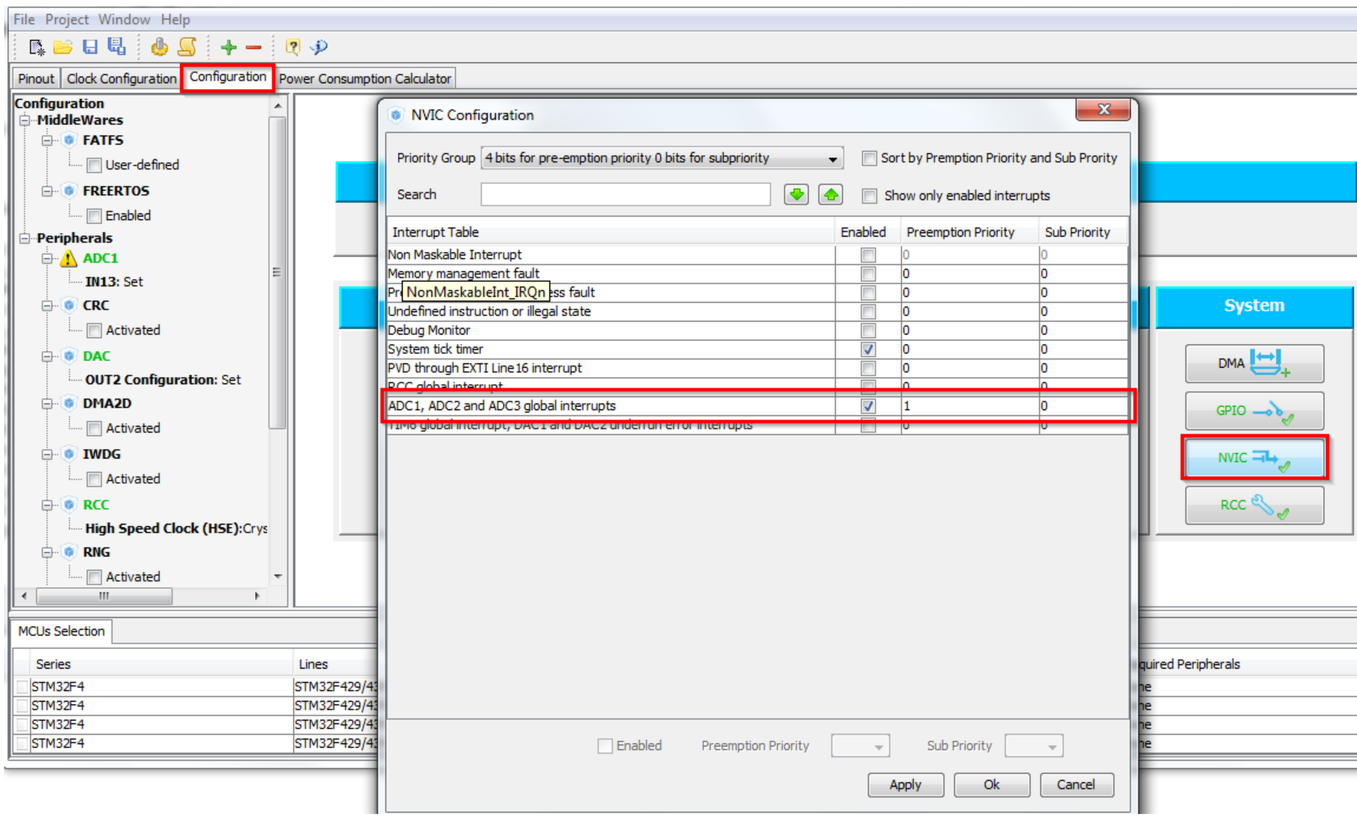
- CubeMX ADC configuration
  - TAB>NVIC settings
  - Enable ADC1 interrupt
  - Button OK



life.augmented

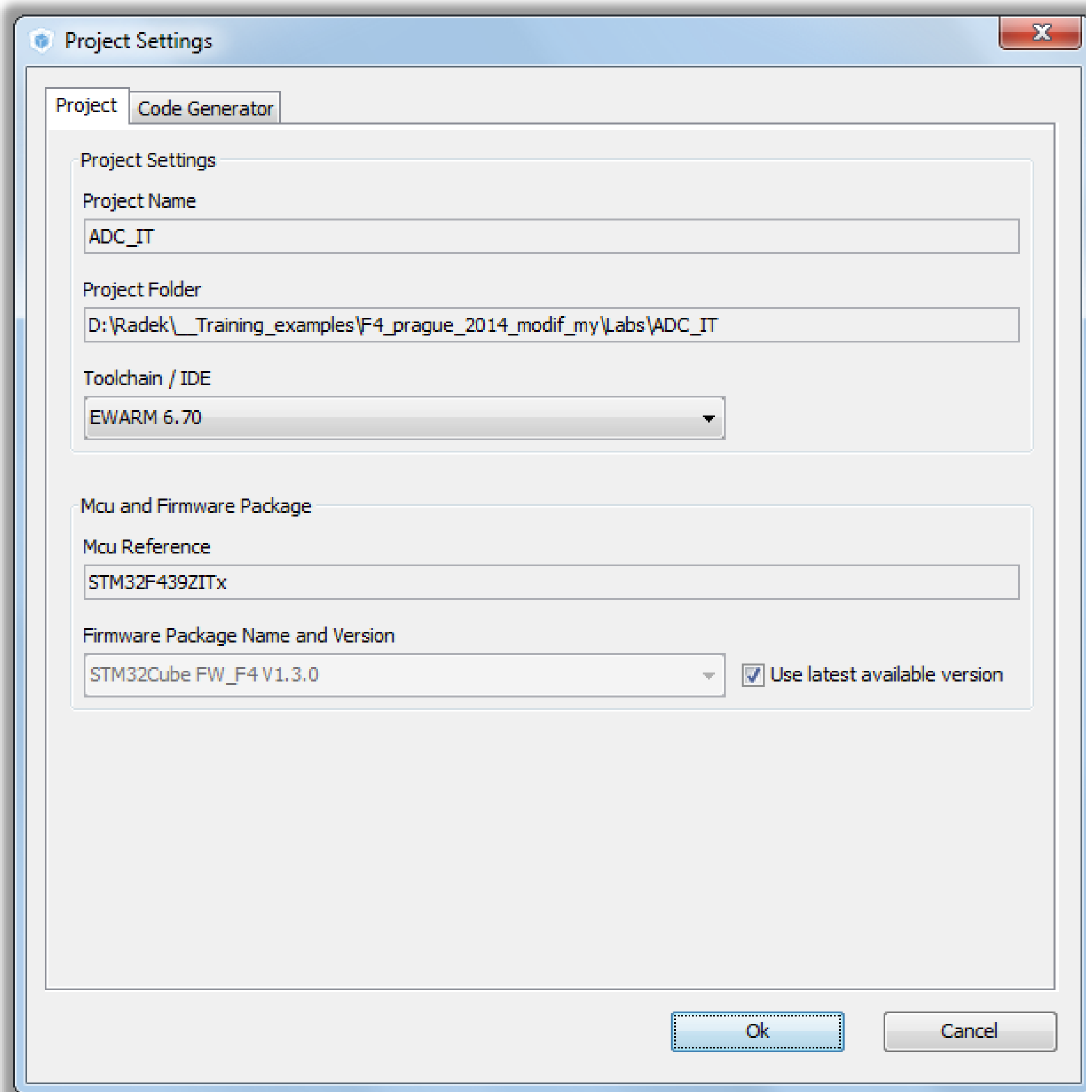
4.2.2 Use ADC with interrupt

- 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



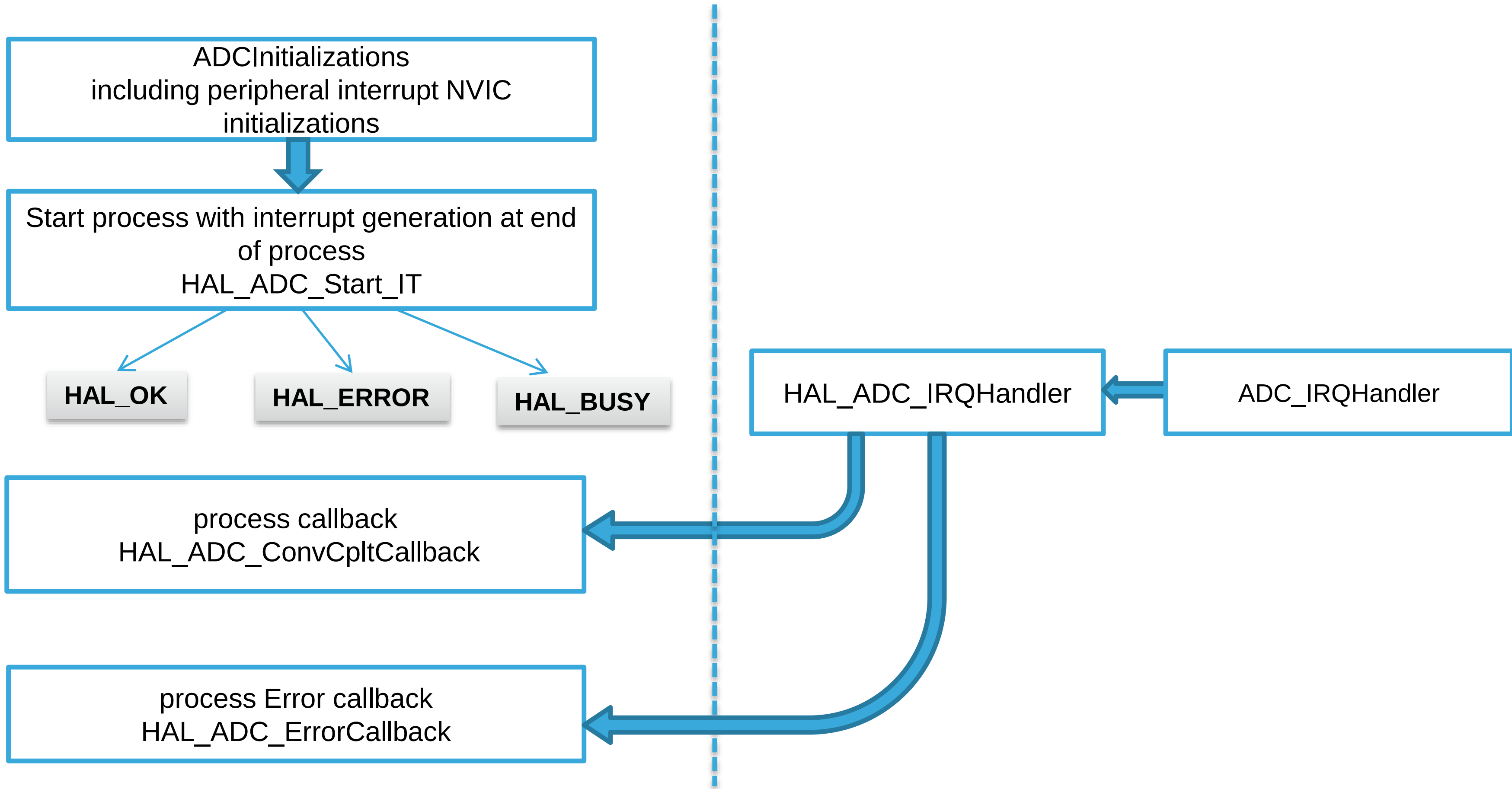
4.2.2 Use ADC with interrupt

- 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



4.2.2 Use ADC with interrupt

HAL Library ADC with IT flow



4.2.2 Use ADC with interrupt

- 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)

4.2.2 Use ADC with interrupt

- 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 */
```

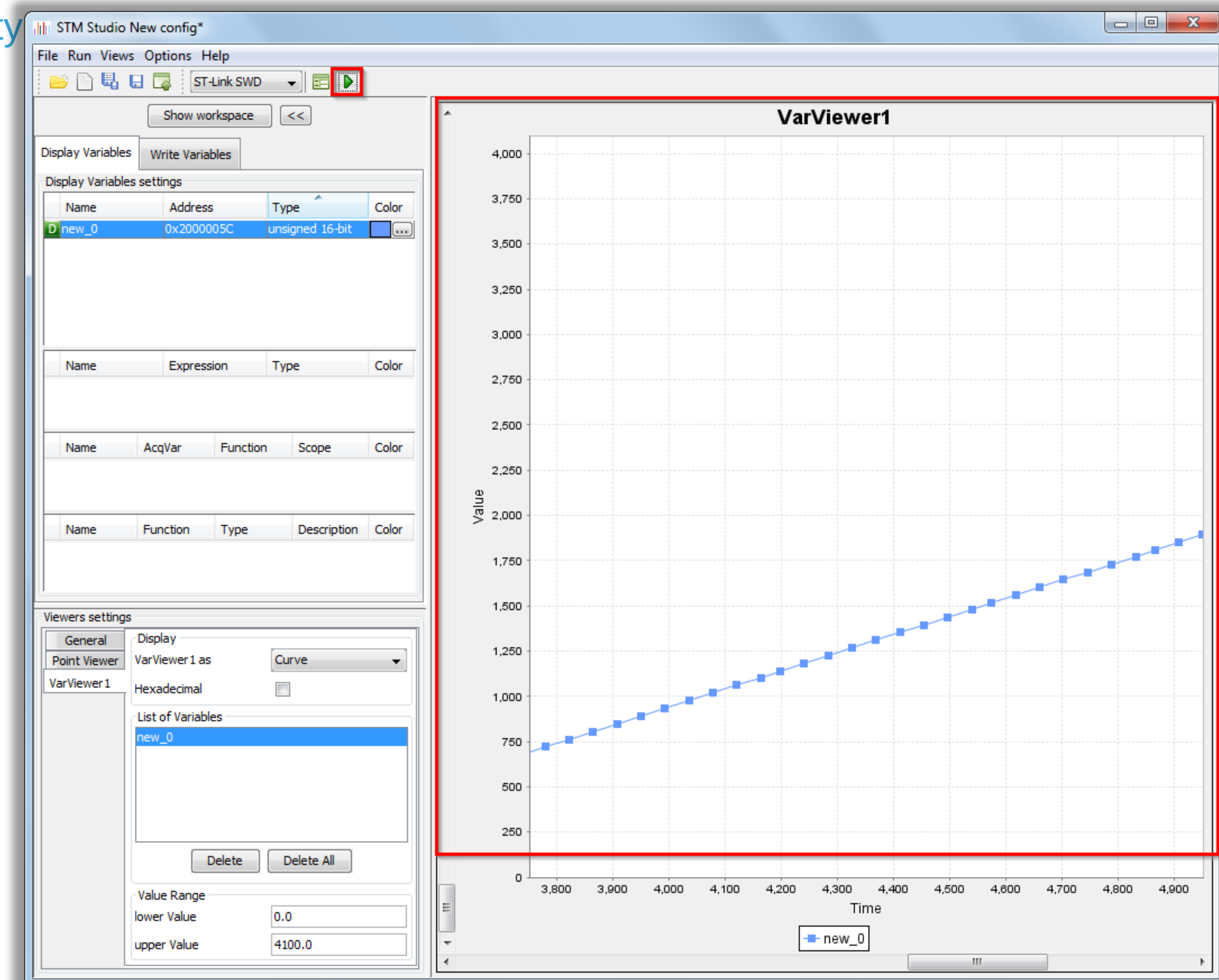
4.2.2 Use ADC with interrupt

- 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 */
```

4.2.2 Use ADC with interrupt

- STM studio settings
  - Check functionality again with STMstudio



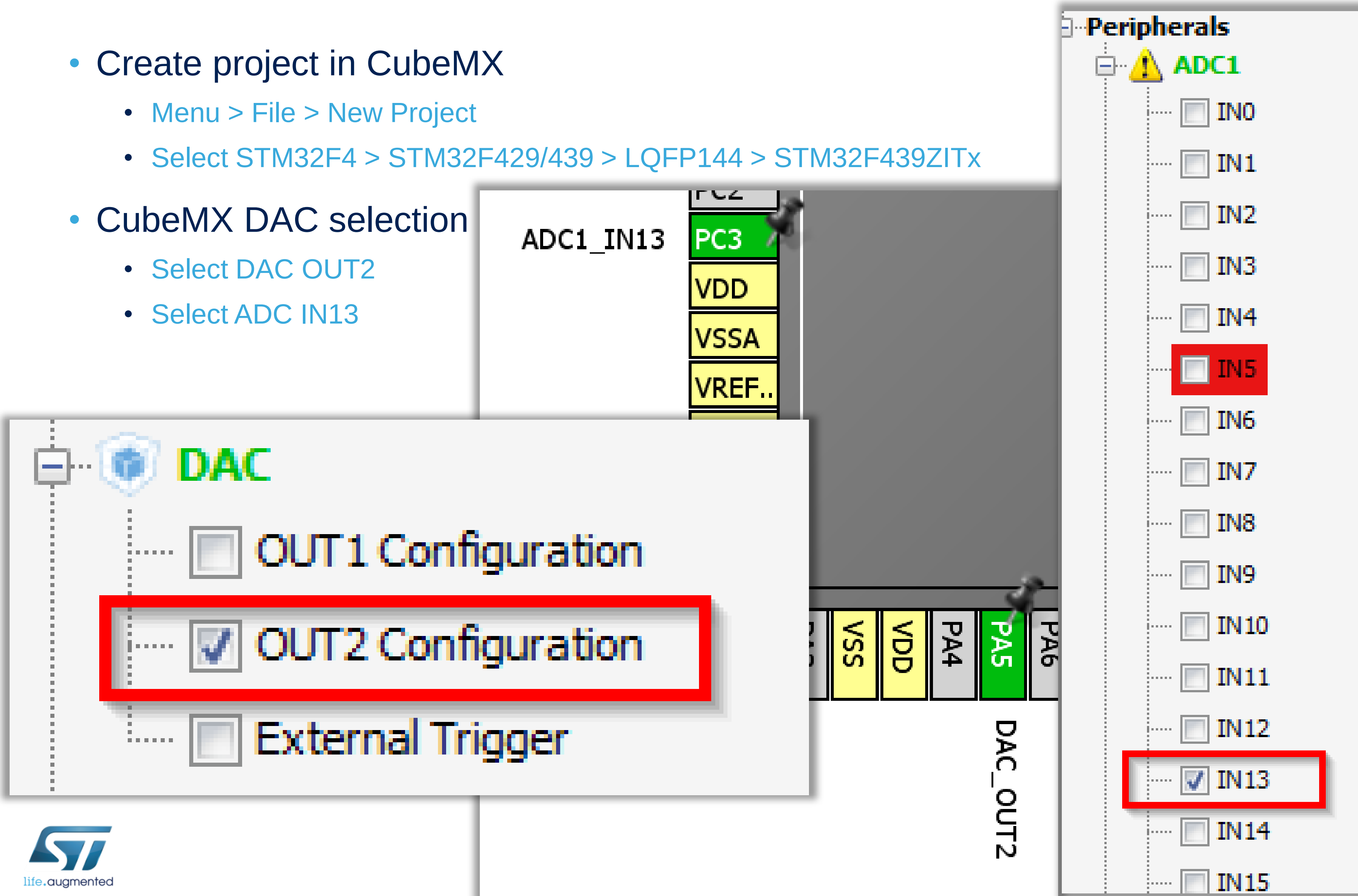
4.2.3 ADC with DMA lab

4.2.3 Use ADC with DMA

- 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)

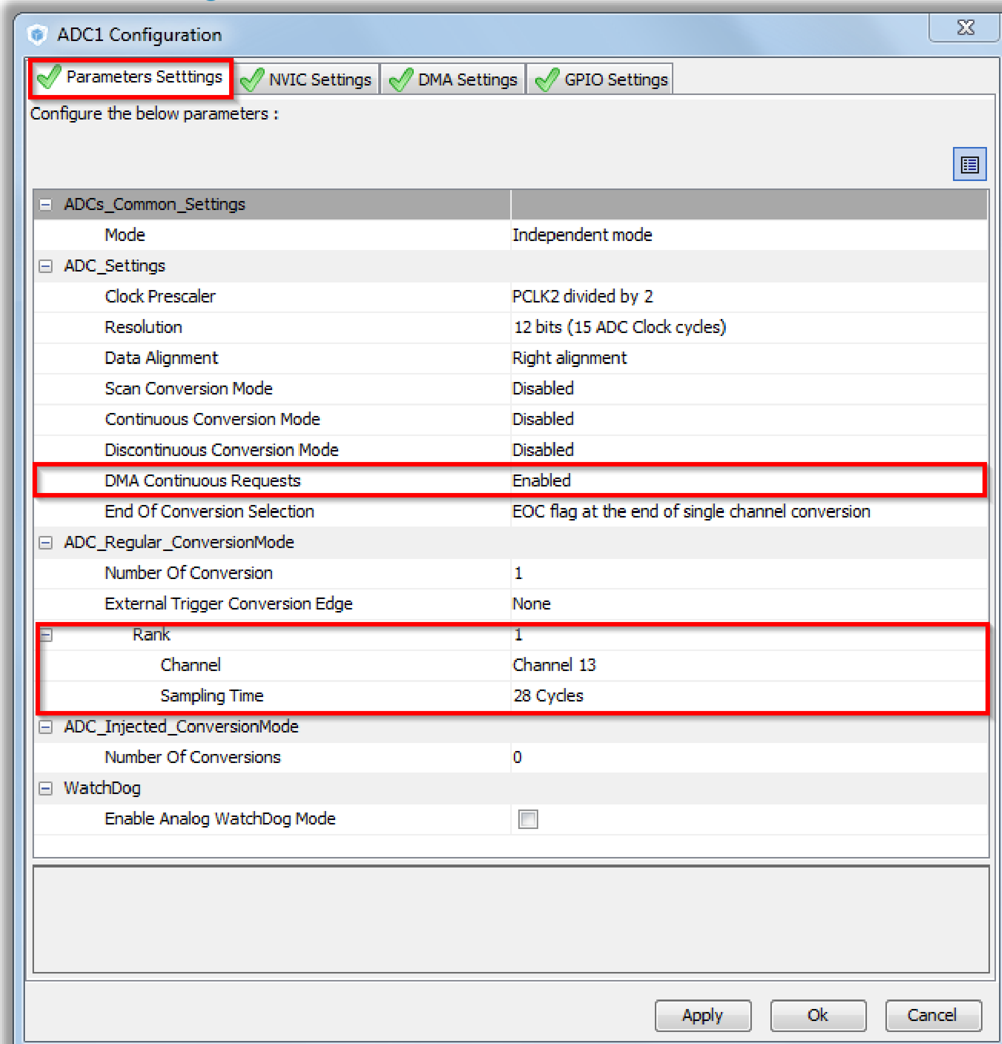
4.2.3 Use ADC with DMA

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



4.2.3 Use ADC with DMA

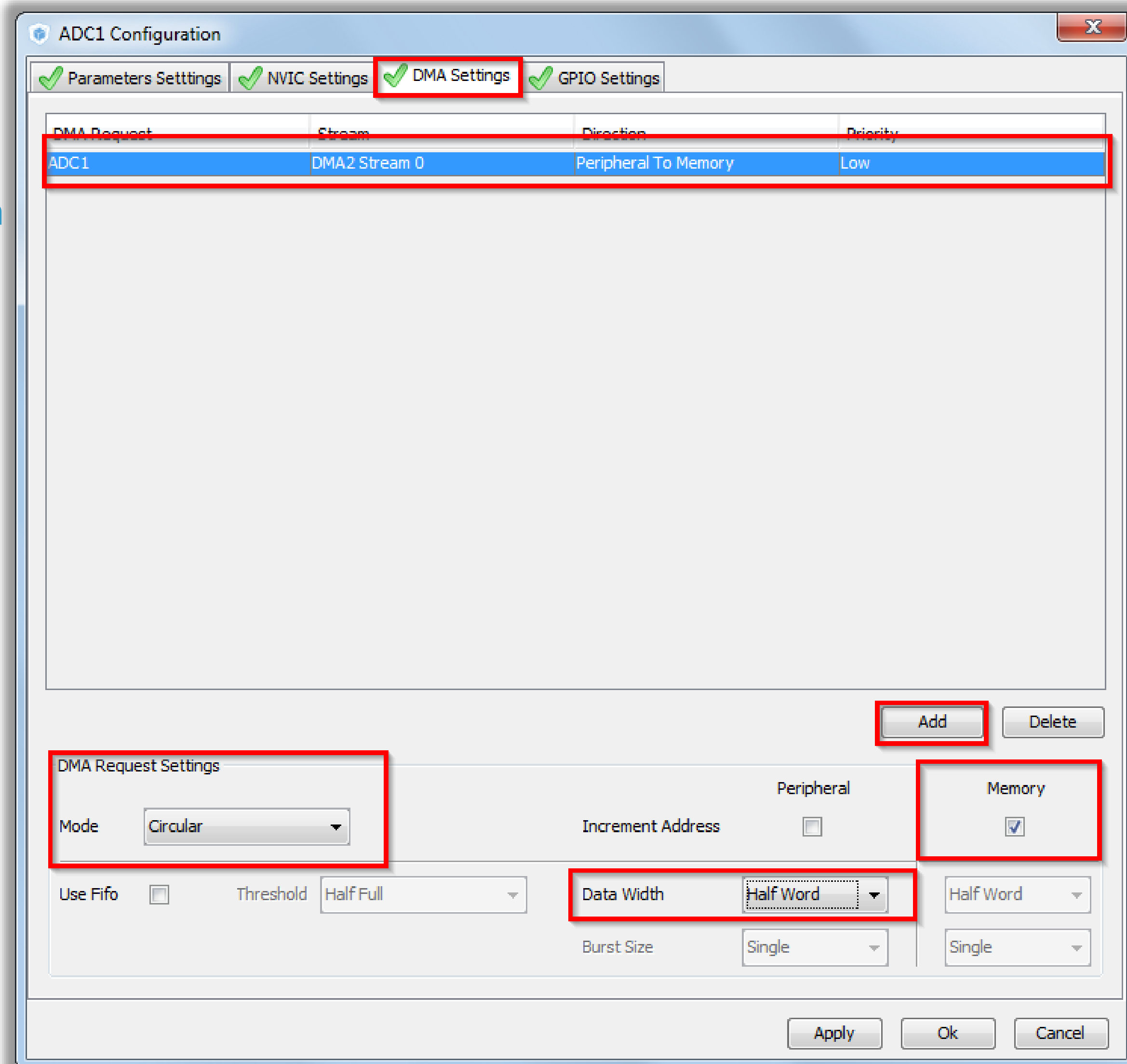
- CubeMX ADC configuration
  - TAB>Configuration>Analog>ADC1>Parameter Settings
  - Set ADC1
  - Set sampling time for CH13
  - DMA Continuous requests
  - Button OK
- DAC from previous example



life.augmented

4.2.3 Use ADC with DMA

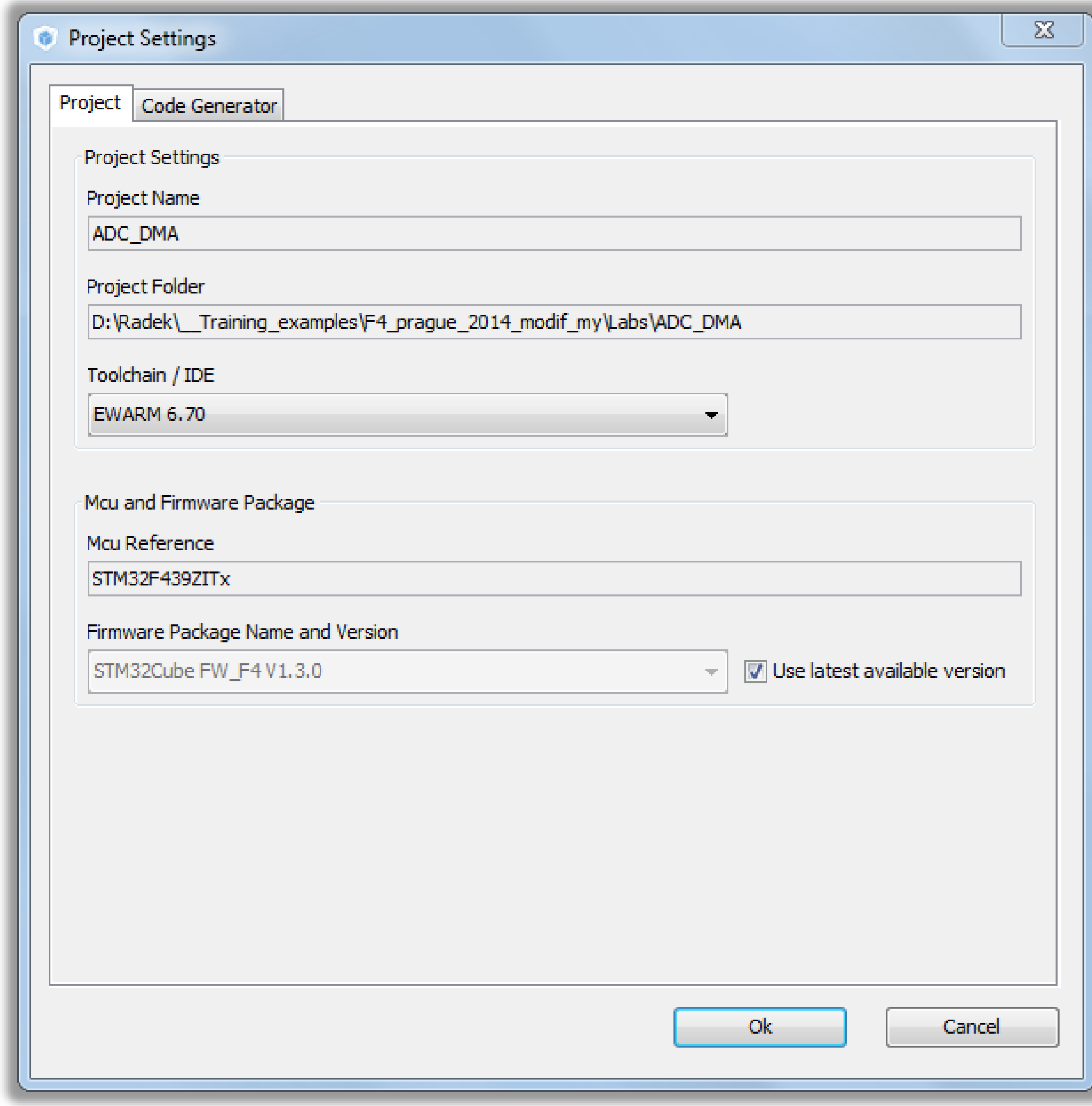
- CubeMX ADC configuration
  - TAB>DMA Settings
  - Button ADD
  - DMA request ADC1
  - Peripheral to memory direction
  - Circular mode
  - Memory increment
  - Half word data width
  - Button OK



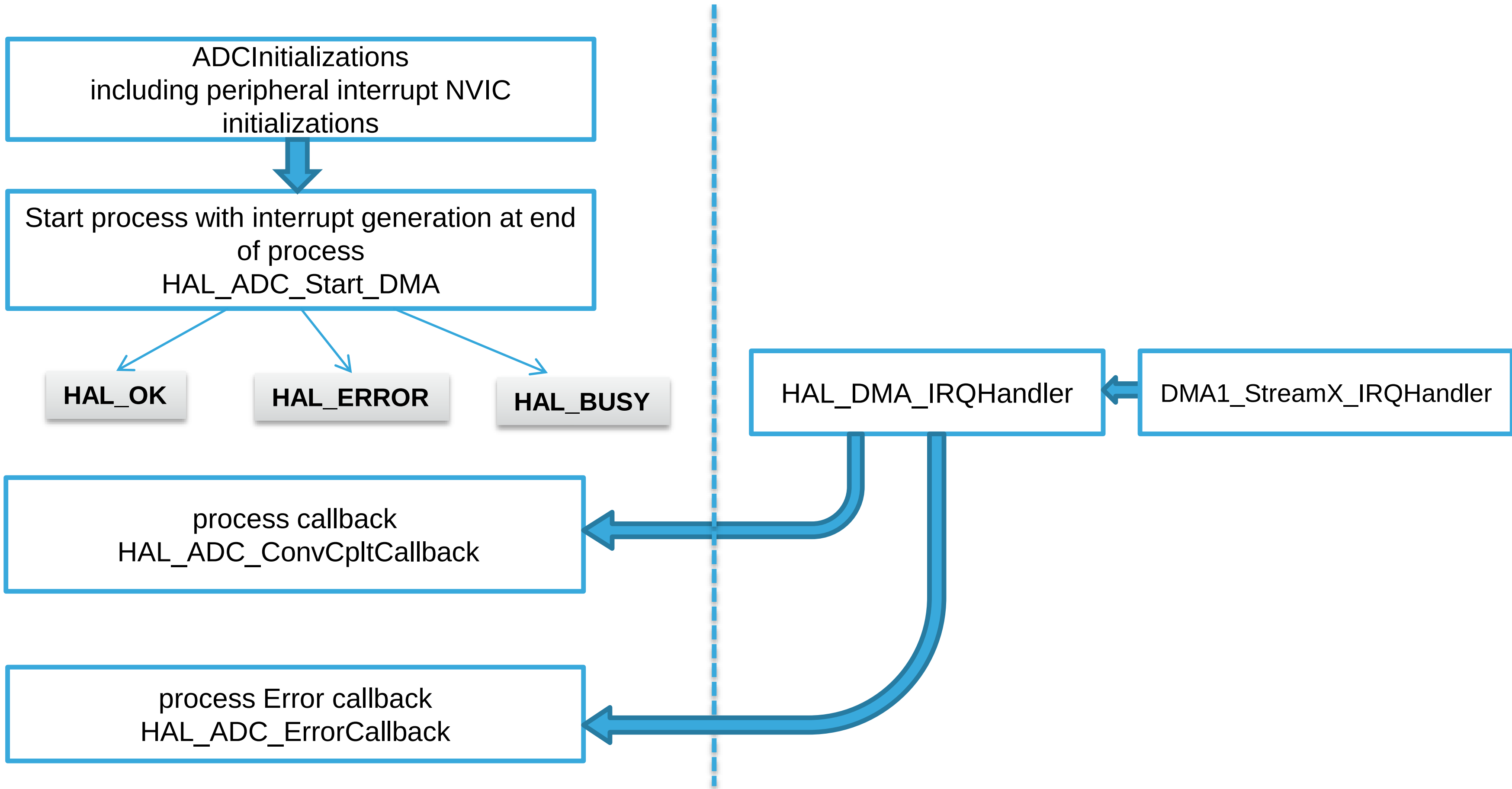
life.augmented

4.2.3 Use ADC with DMA

- 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



life.augmented

4.2.3 Use ADC with DMA

- 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)

4.2.3 Use ADC with DMA

- 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

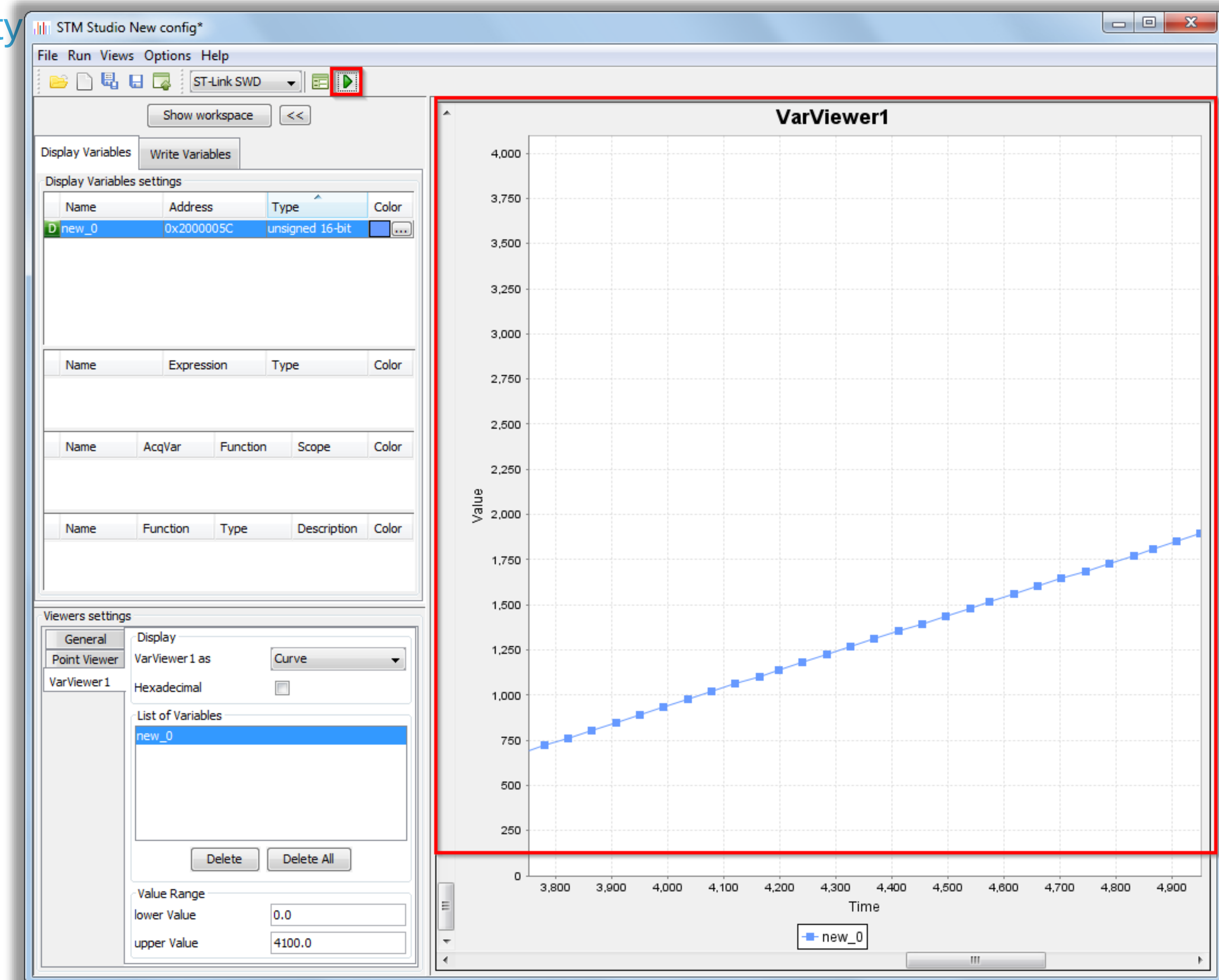
```
 /* USER CODE BEGIN 2 */
  HAL_DAC_Start(&hdac,DAC_CHANNEL_2);
  HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, value_dac);
  HAL_ADC_Start_DMA(&hadc1,(uint32_t*)&value_adc,1);
  /* USER CODE END 2 */
```

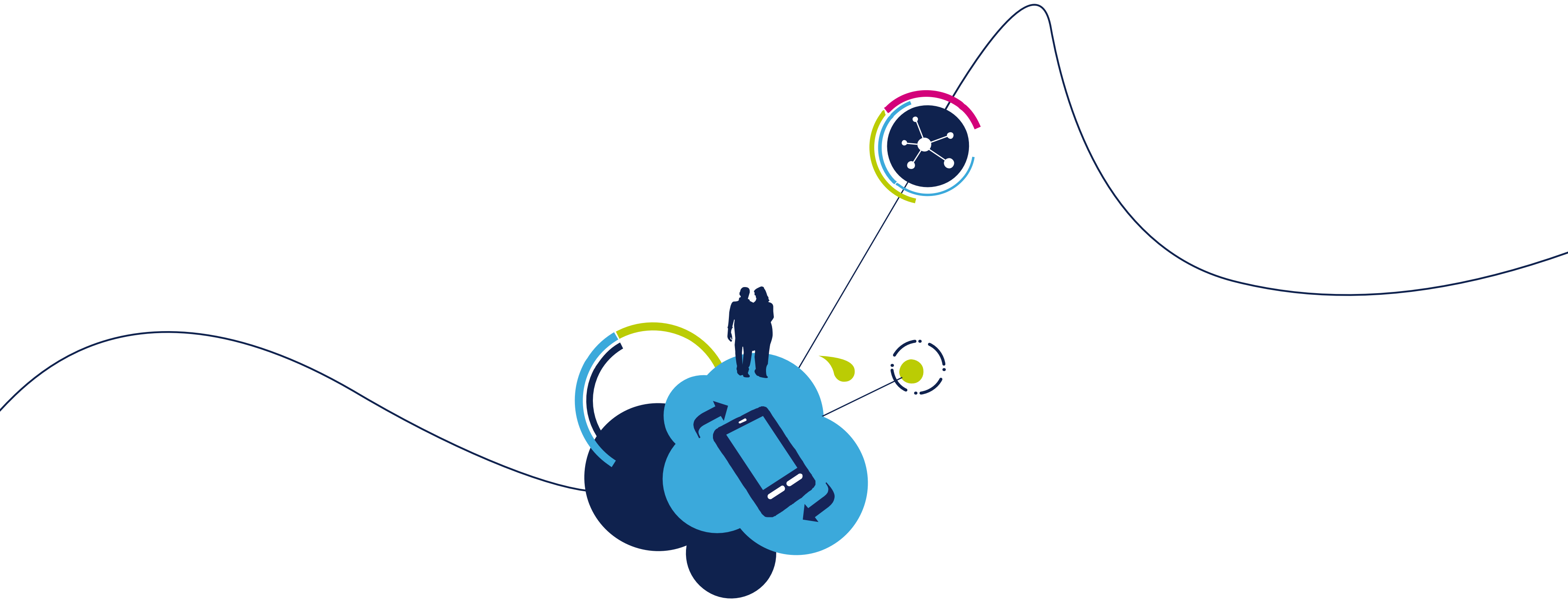
- Solution
  - ADC main routine

```
/* 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 */
```

4.2.3 Use ADC with DMA

- STM studio settings
  - Check functionality again with STMstudio





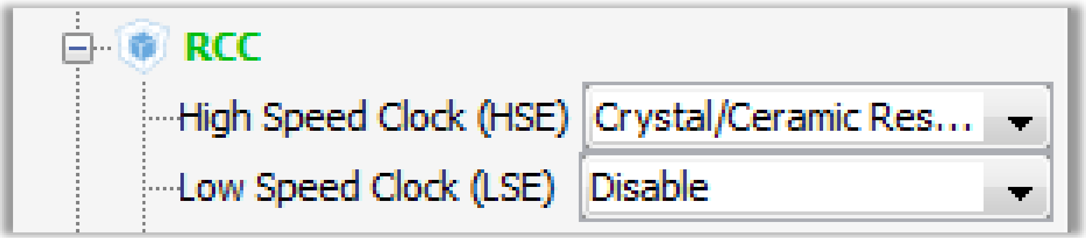
5.1 BSP SDRAM lab

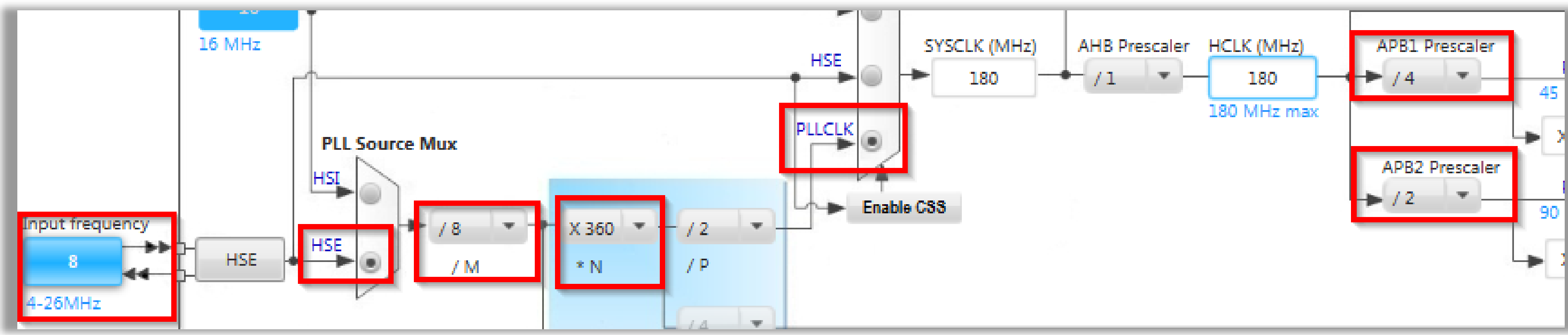
5.1 Use BSP for SDRAM initialization

- 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

5.1 Use BSP for SDRAM initialization

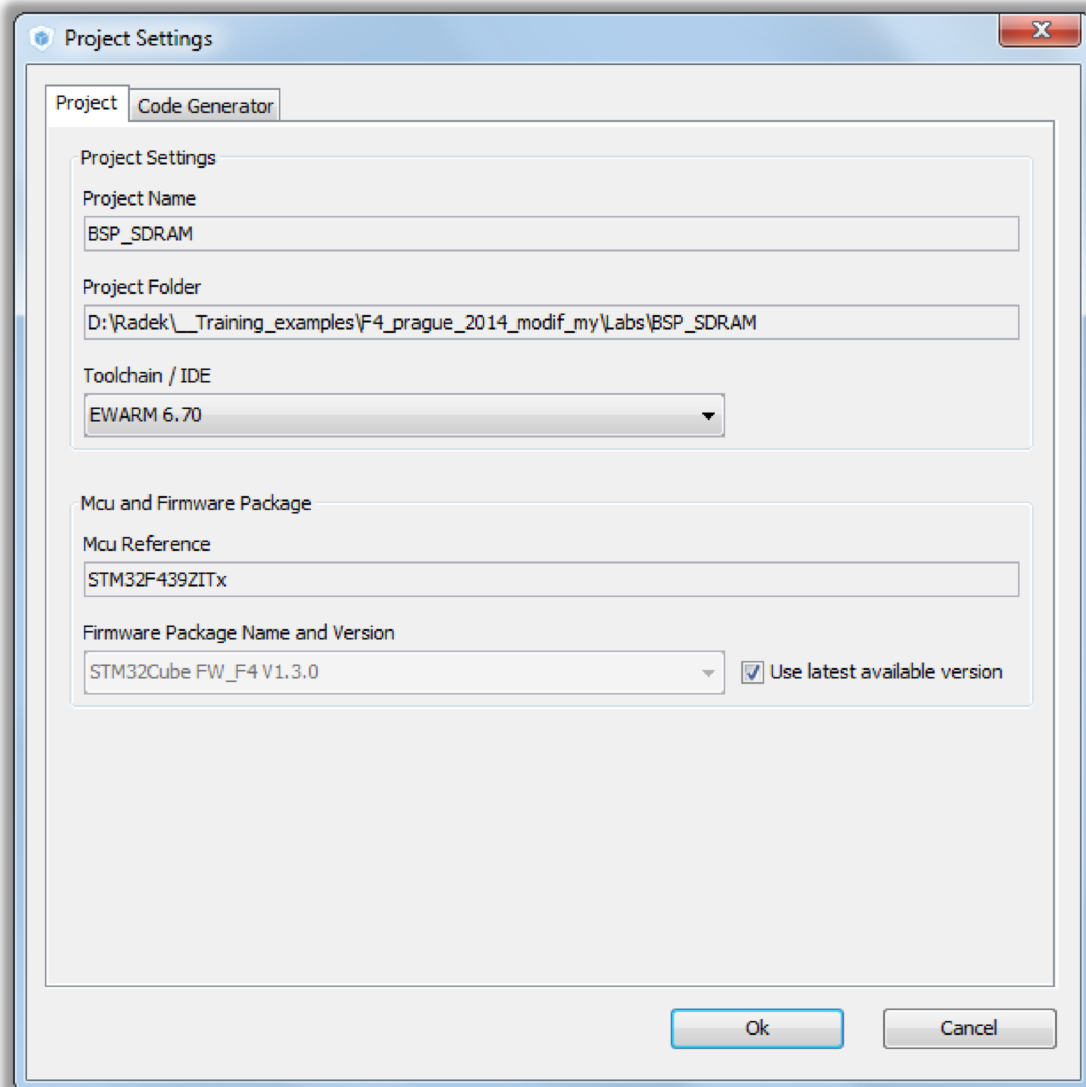
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- We need only blank project with clock initialization
  - We only set the RCC and configure the core to maximum speed





5.1 Use BSP for SDRAM initialization

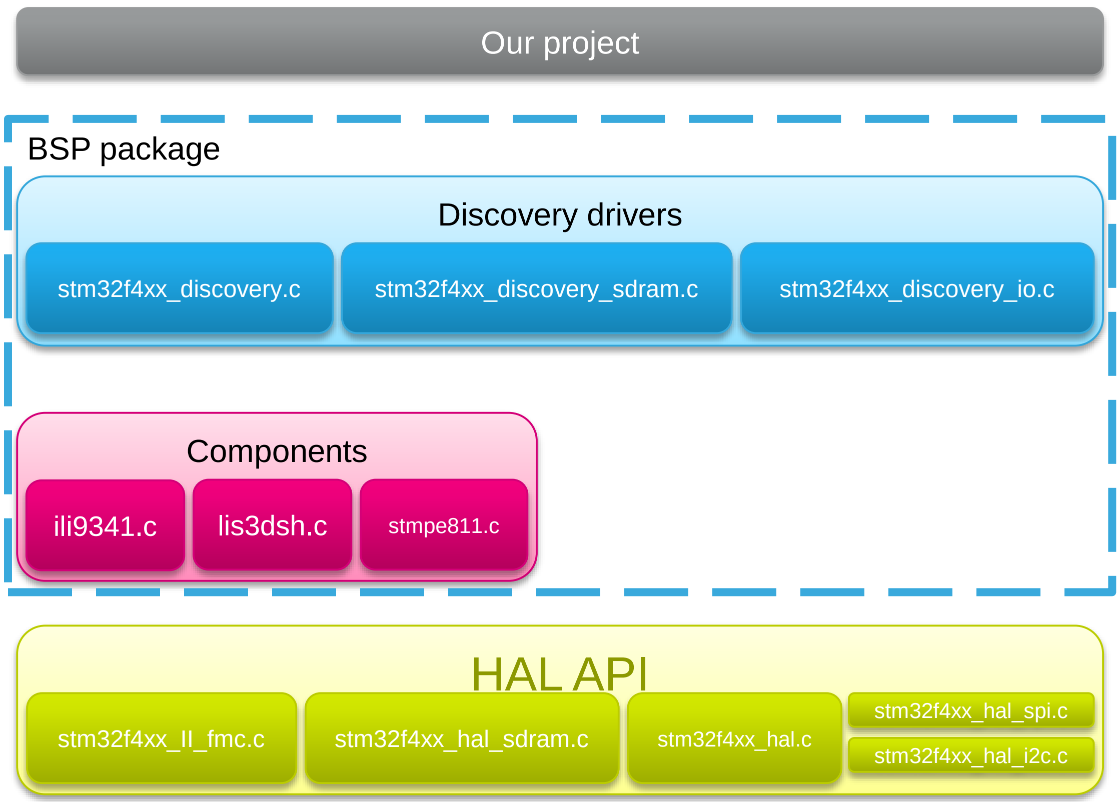
- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code

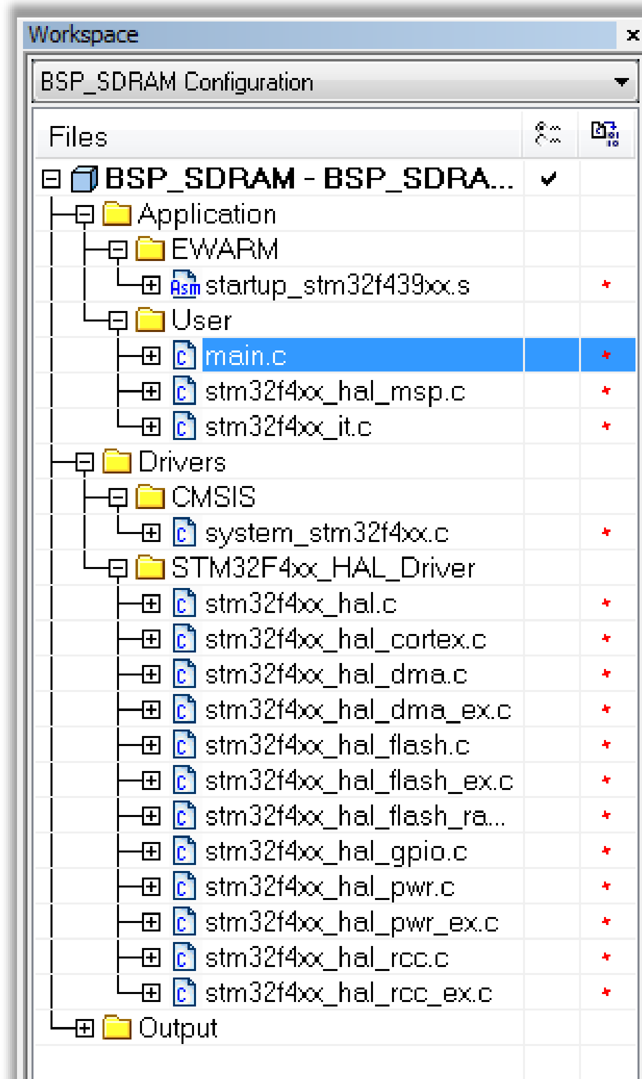


life.augmented

5.1 Use BSP for SDRAM initialization

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

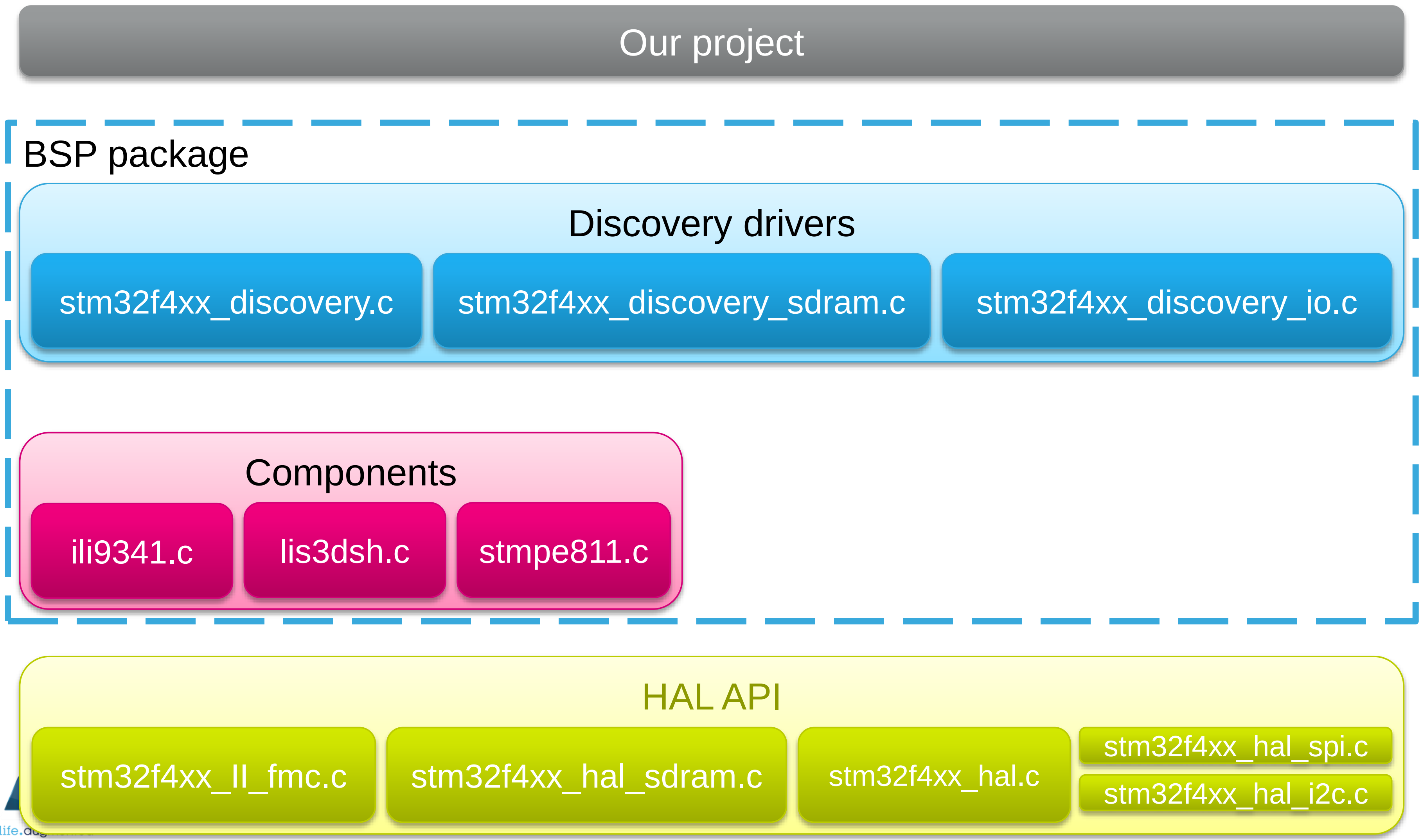




life.augmented

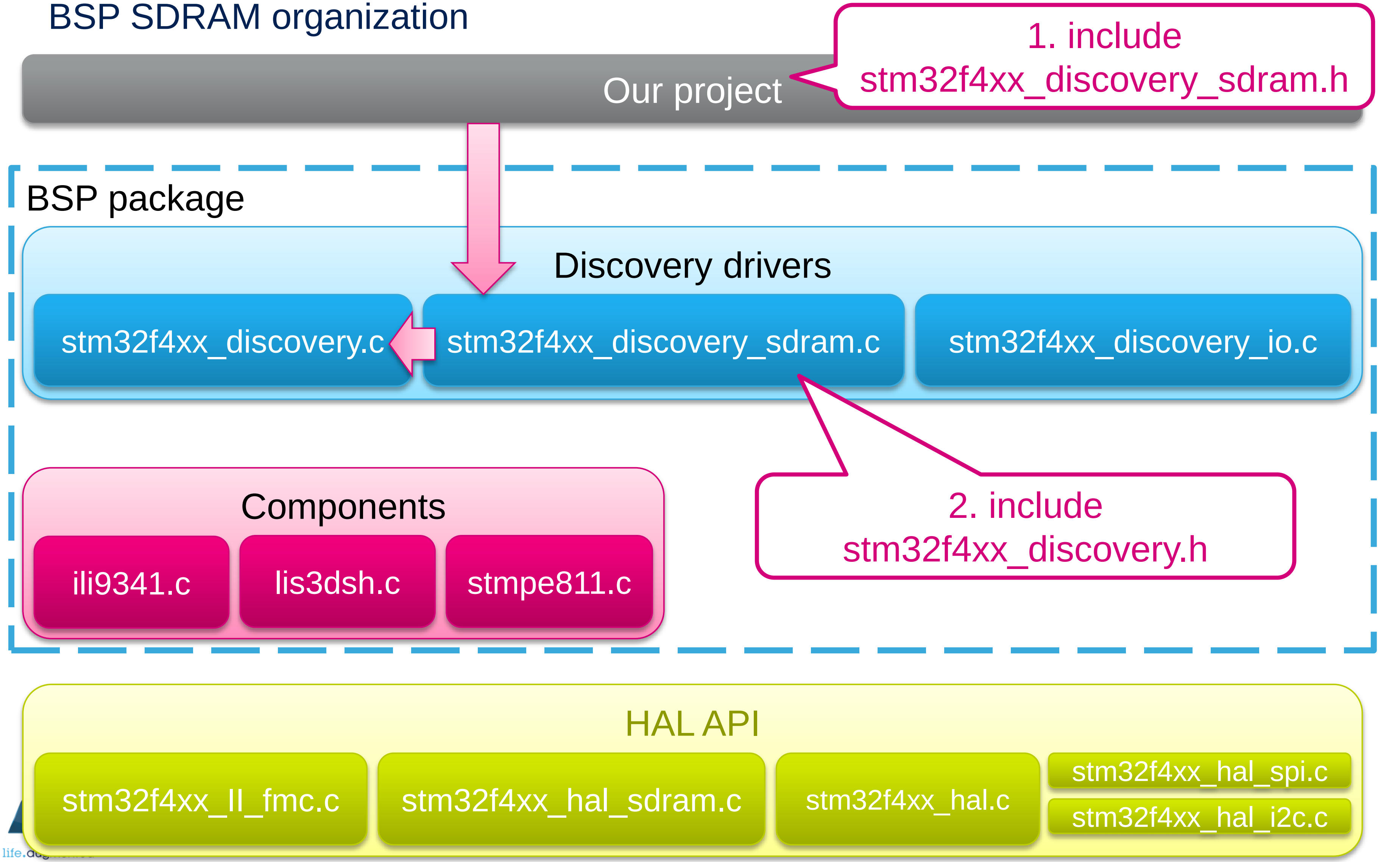
5.1 Use BSP for SDRAM initialization

BSP SDRAM organization



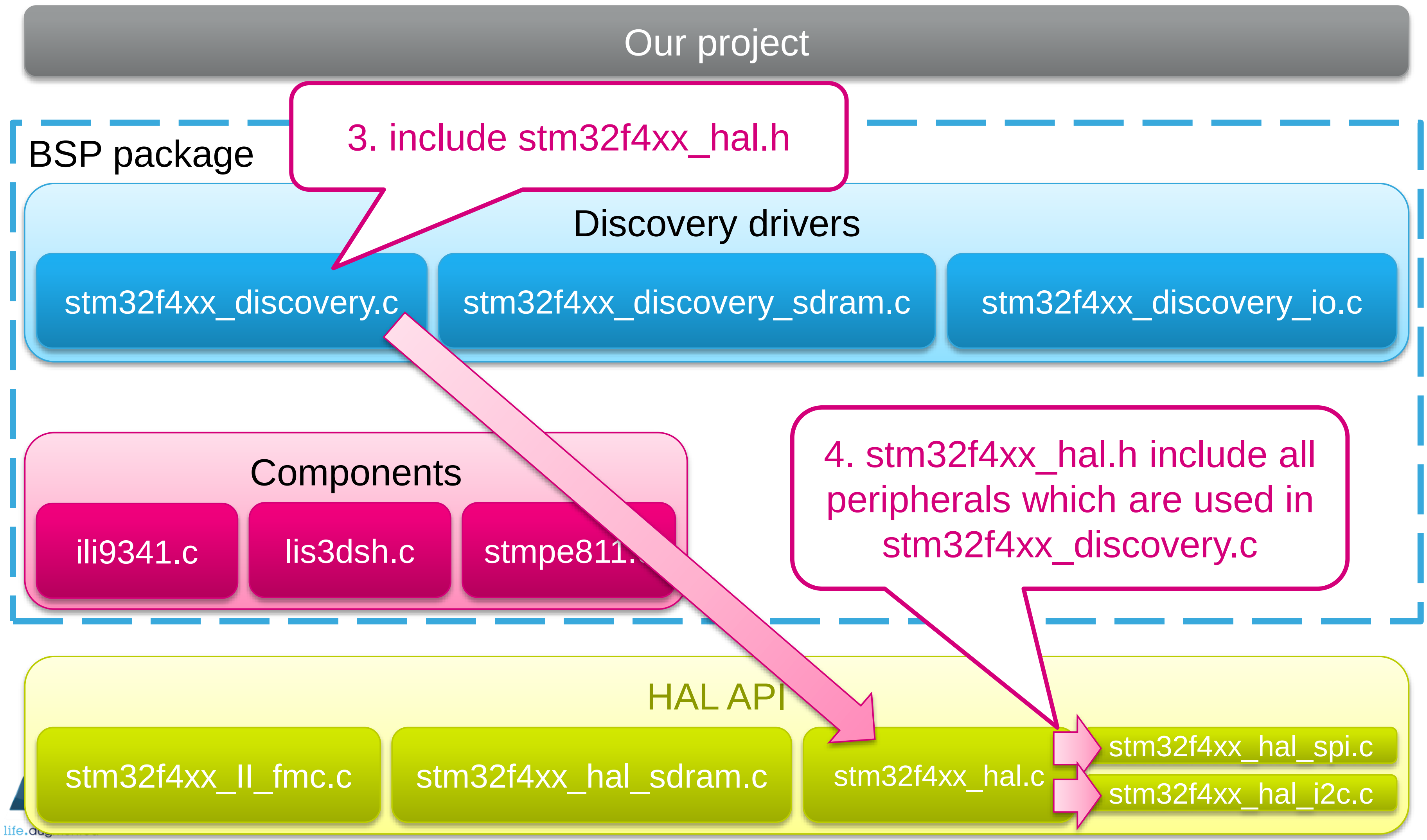
5.1 Use BSP for SDRAM initialization

BSP SDRAM organization



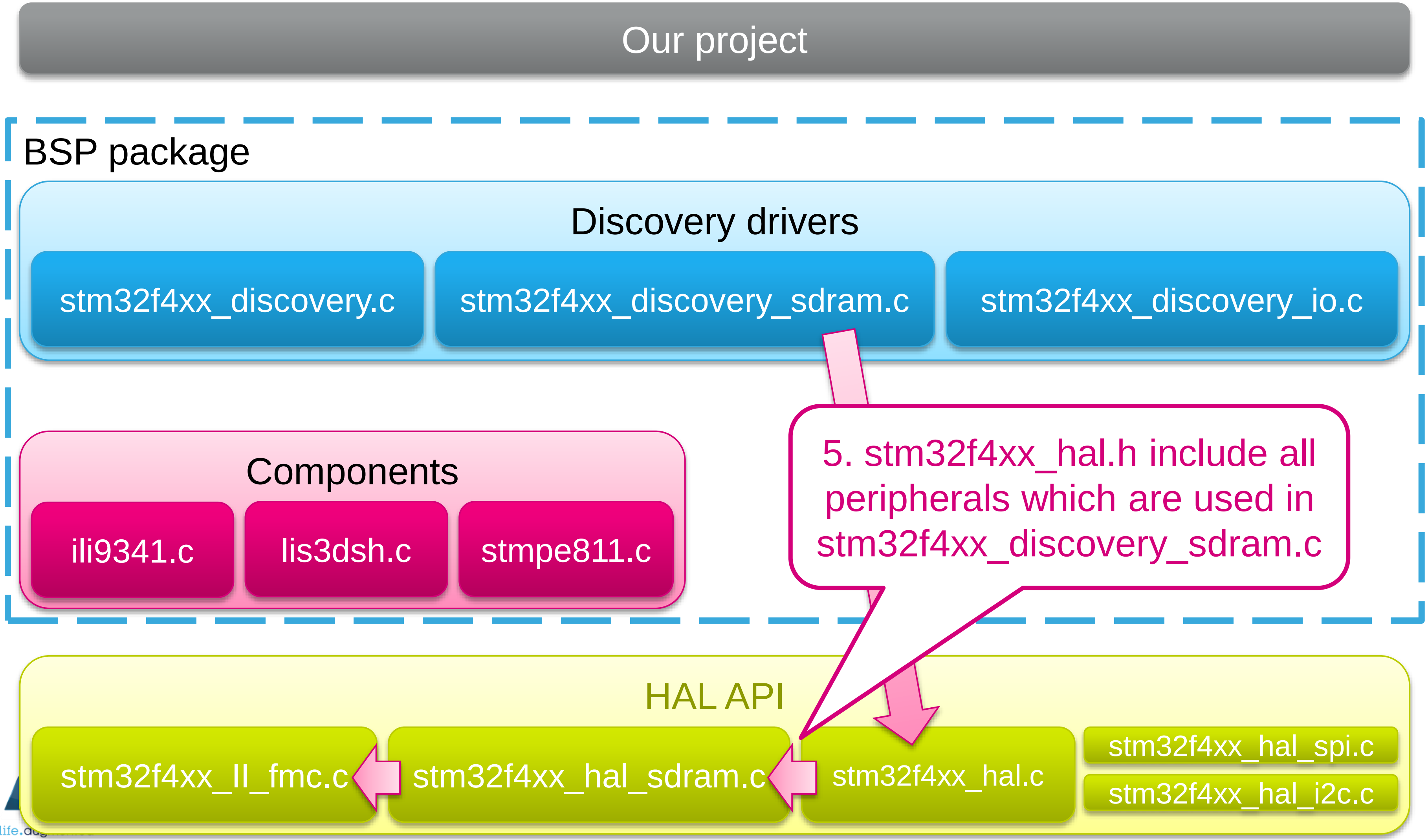
5.1 Use BSP for SDRAM initialization

BSP SDRAM organization



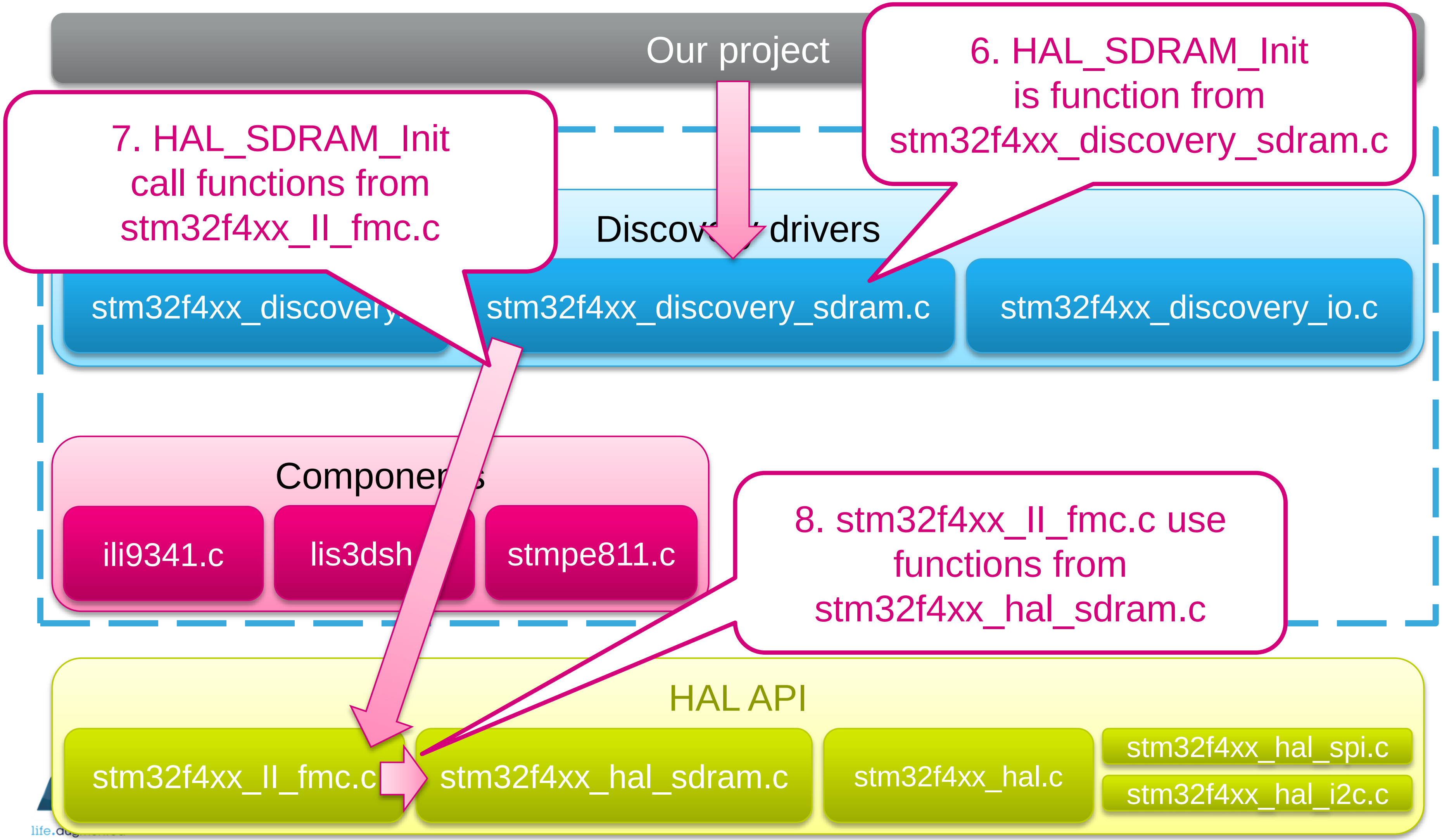
5.1 Use BSP for SDRAM initialization

BSP SDRAM organization

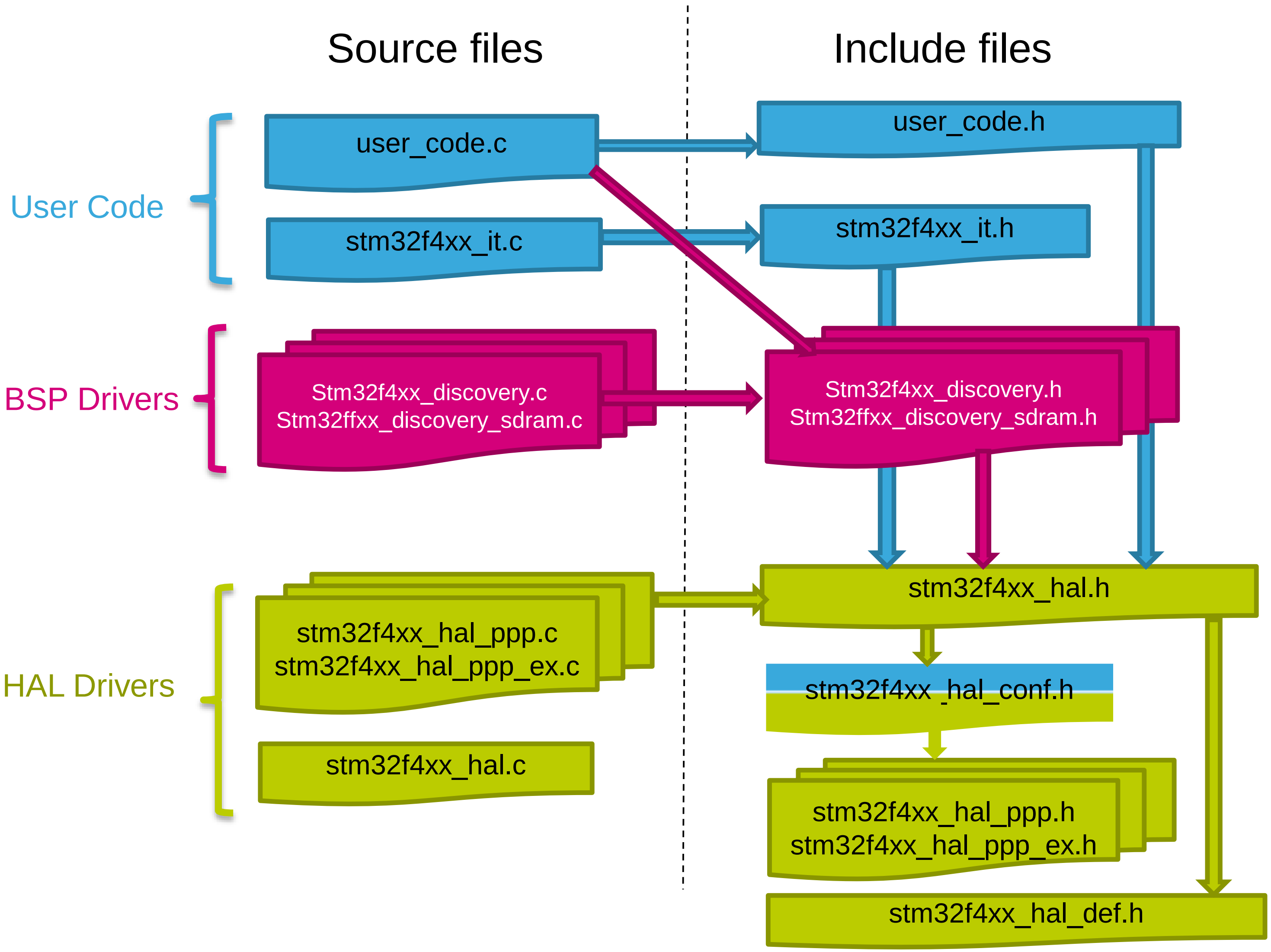


5.1 Use BSP for SDRAM initialization

BSP SDRAM organization

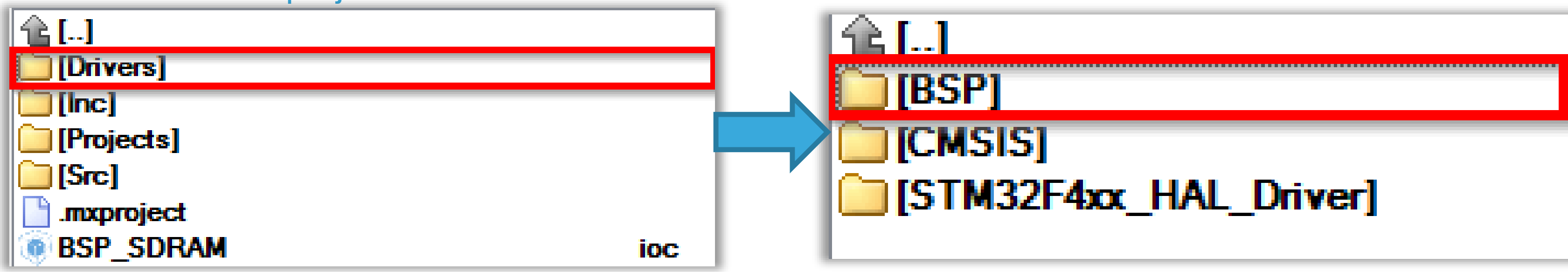


5.1 Use BSP for SDRAM initialization

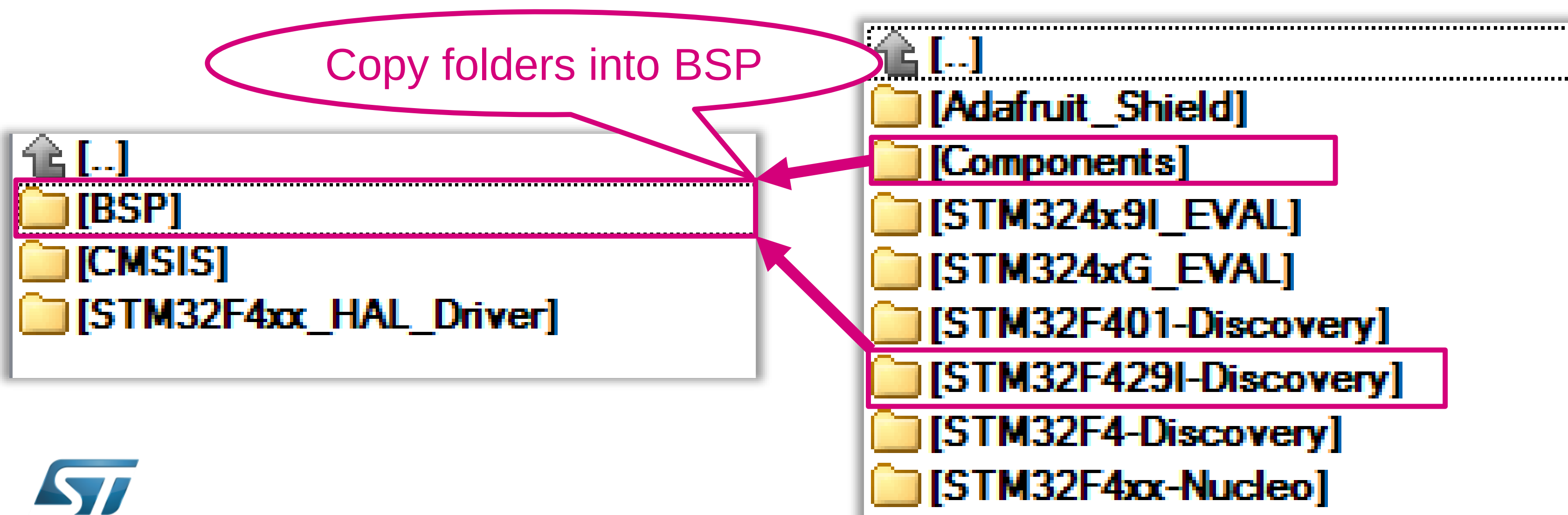


5.1 Use BSP for SDRAM initialization

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



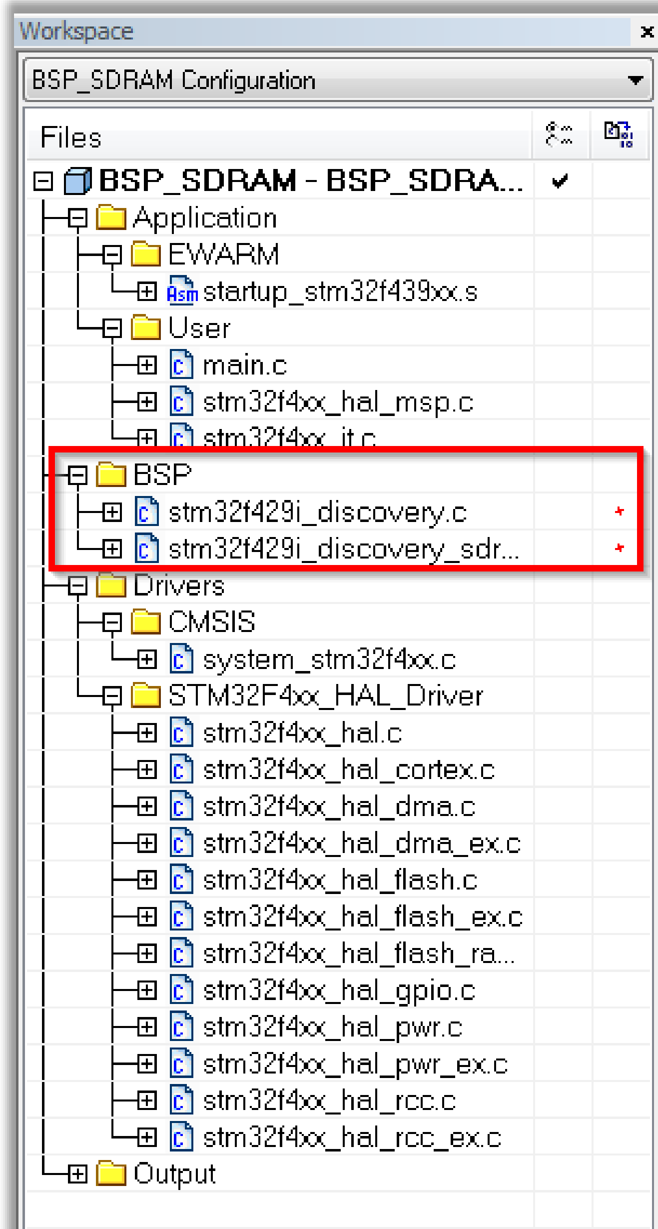
  - Now go into CubeMX repository …\STM32Cube_FW_F4_V1.3.0\Drivers\BSP\
  - And copy **Components** and **STM32F429I-Discovery** into **BSP** folder



life.augmented

5.1 Use BSP for SDRAM initialization

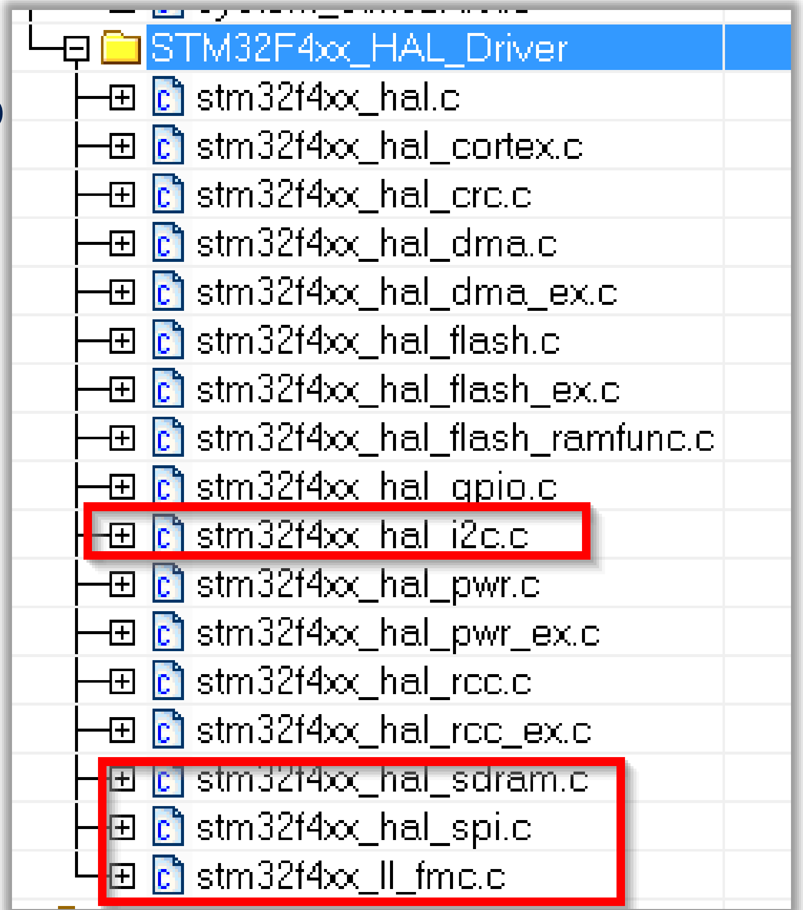
- Now we need to add this files also in project
  - Create BSP folder in project
  - Right click on project in Workplace>ADD>Group
  - Name it BSP
  - Now right click on BSP>ADD>Files
  - From Drivers\BSP\STM32F429I-Discovery\
    add stm32f429i_discovery.c
    and stm32f429i_discovery_sdram.c



life.augmented

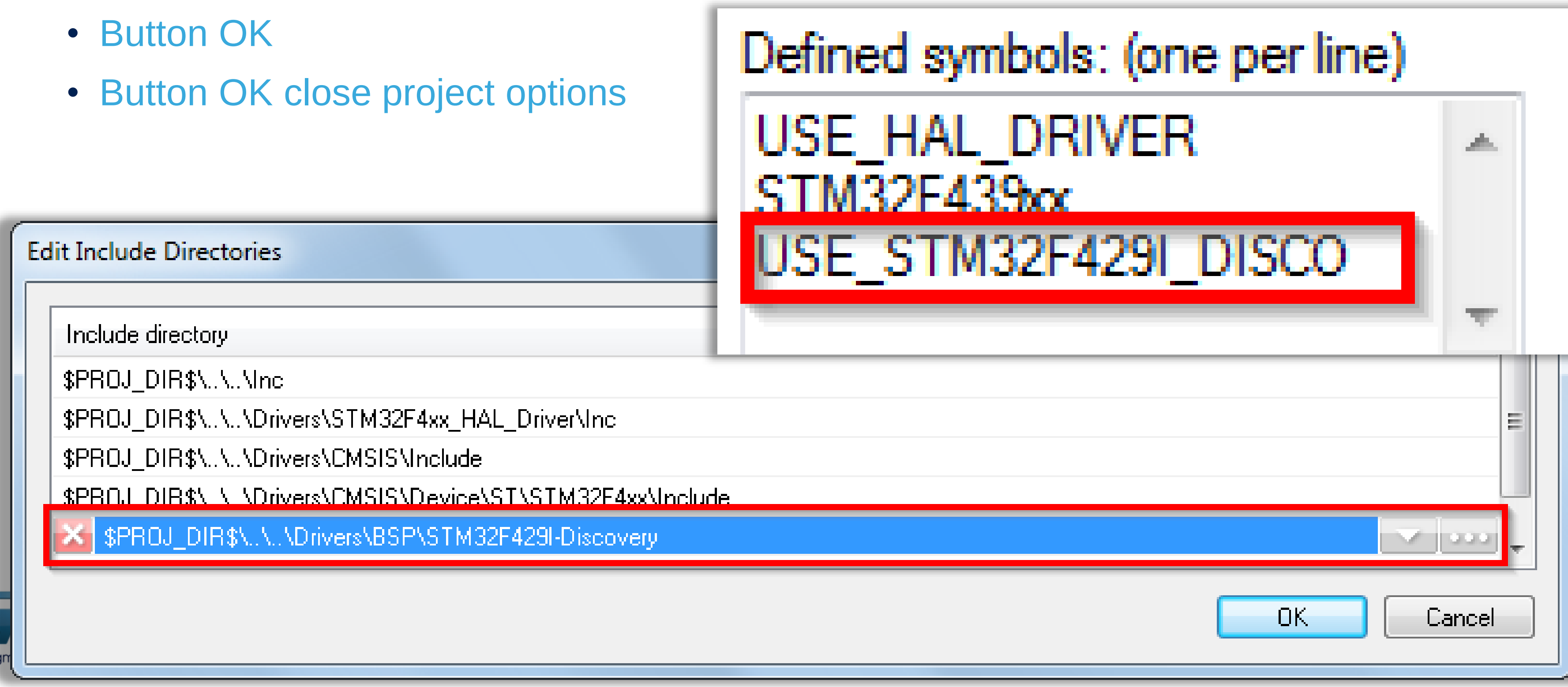
5.1 Use BSP for SDRAM initialization

- 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**



5.1 Use BSP for SDRAM initialization

- 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



5.1 Use BSP for SDRAM initialization

- 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
```

life.augmented

5.1 Use BSP for SDRAM initialization

- 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 */

5.1 Use BSP for SDRAM initialization

- 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 */
```

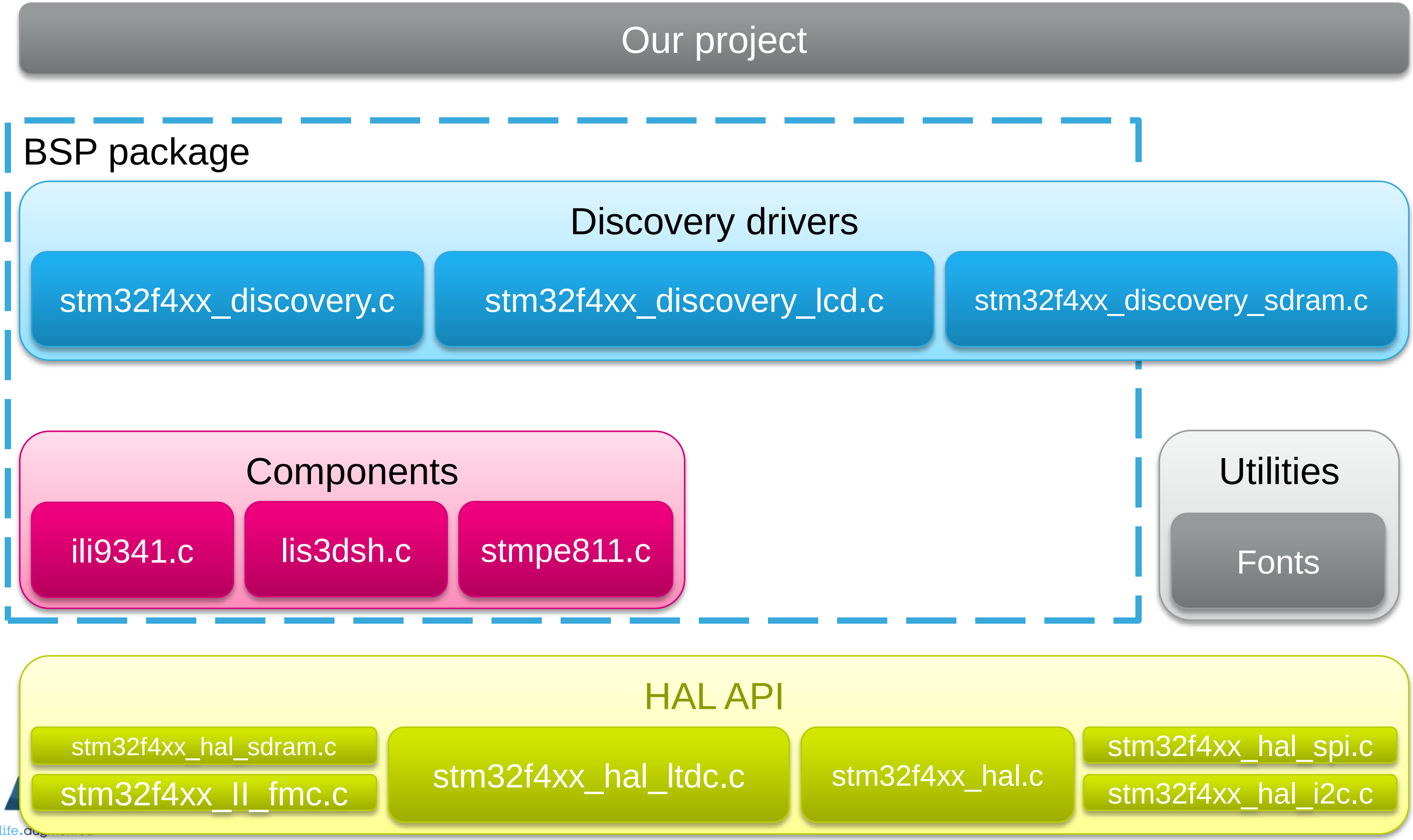
5.2 BSP LCD lab

5.2 Use BSP for LCD init and writing

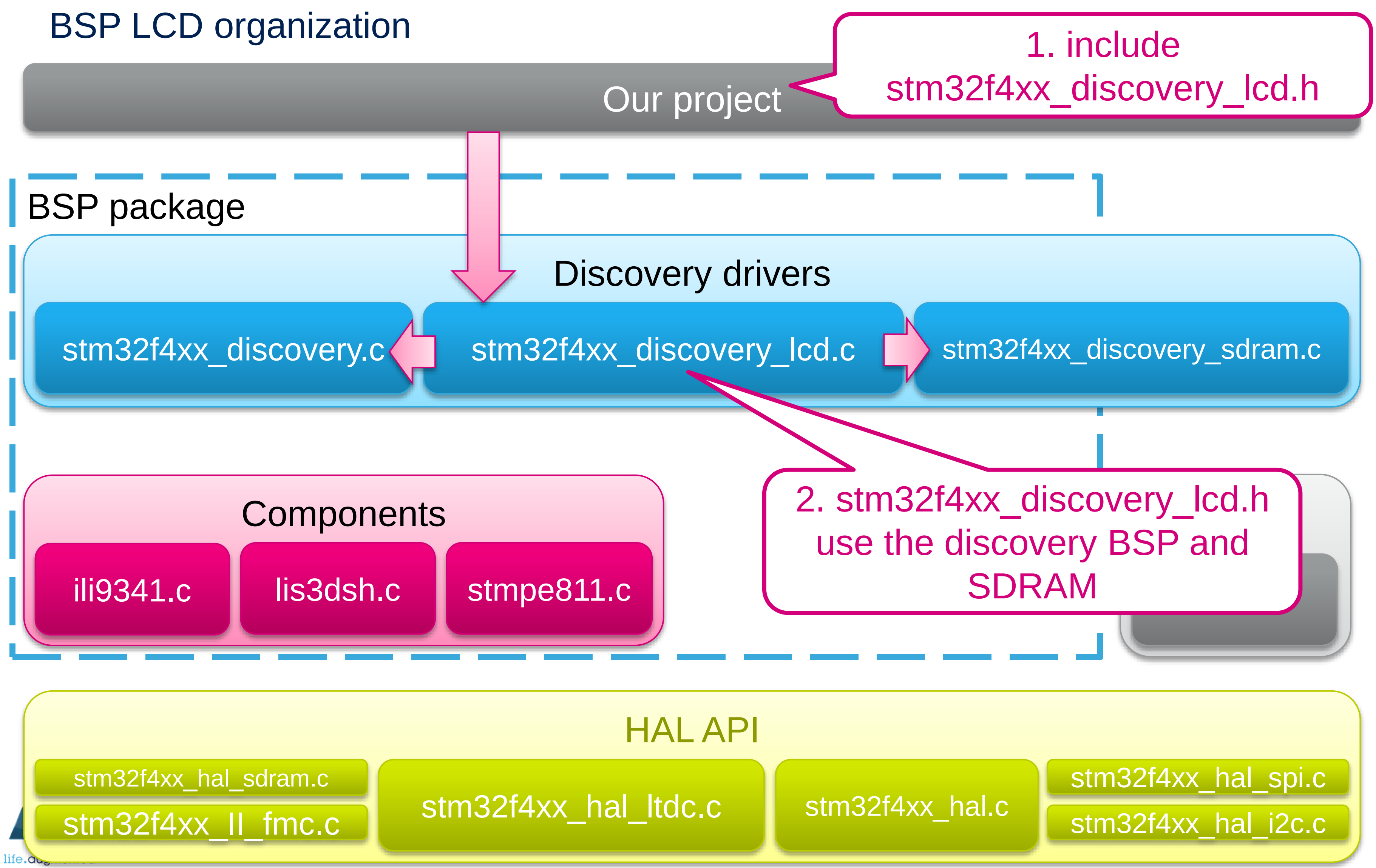
- Objective
  - Learn how import BSP LCD into project
  - Because the LCD use the SDRAM we use project from lab 25
  - Which part need to by configured in GUI
  - Try to write text on LCD
- Goal
  - Successfully import BSP LCD into your project
  - Learn which part you need to import
  - How to setup the project

5.2 Use BSP for LCD init and writing

BSP LCD organization

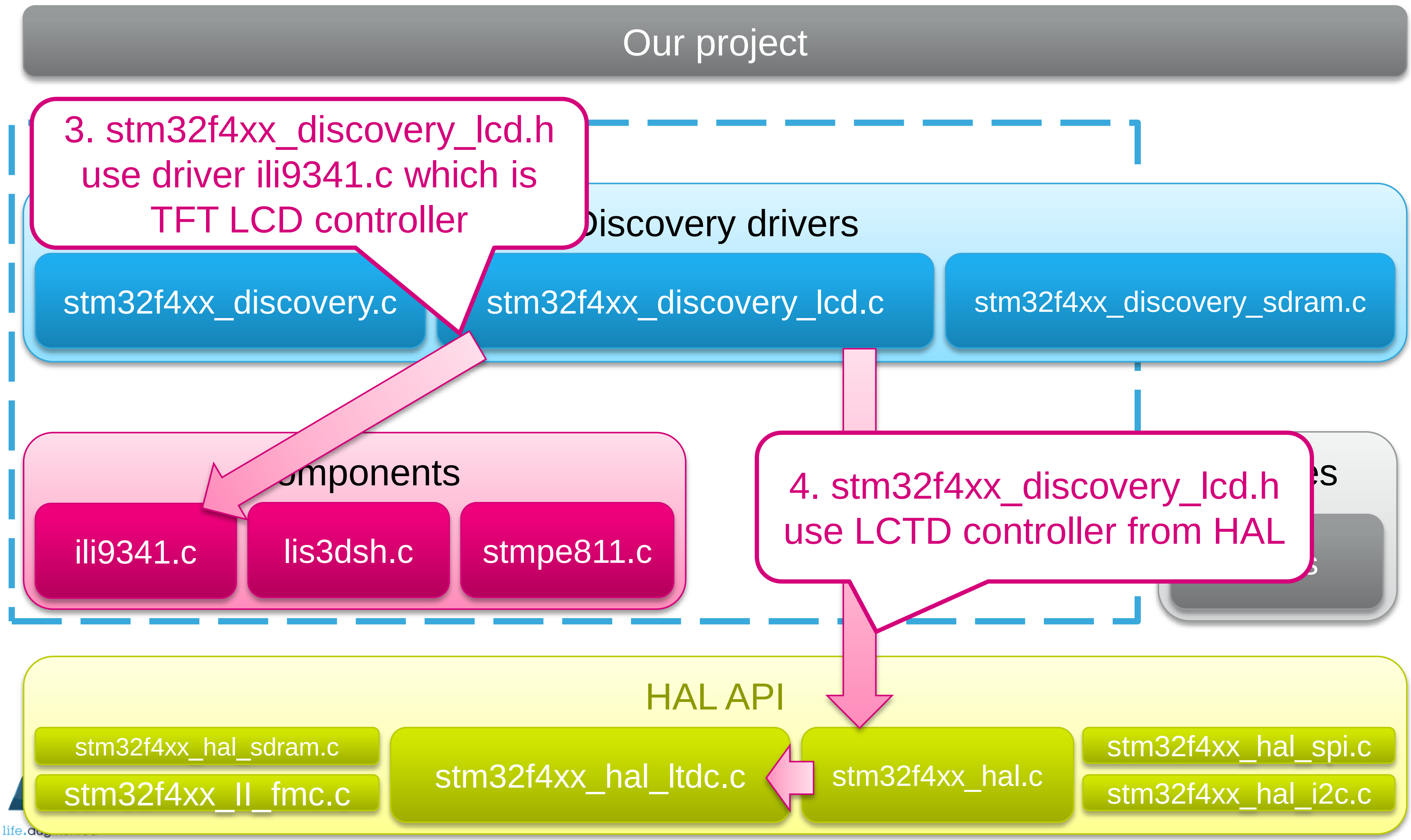


5.2 Use BSP for LCD init and writing



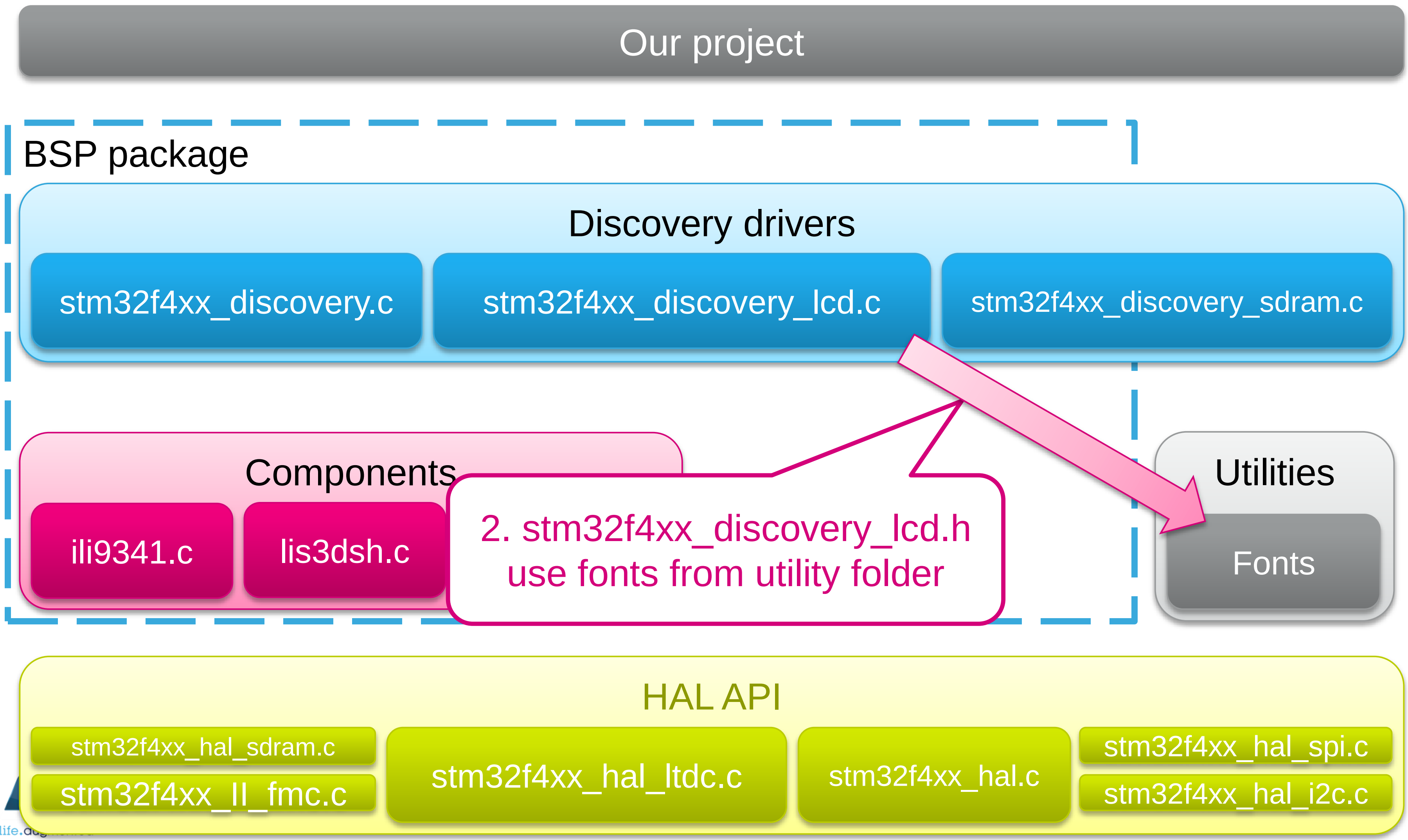
5.2 Use BSP for LCD init and writing

BSP LCD organization



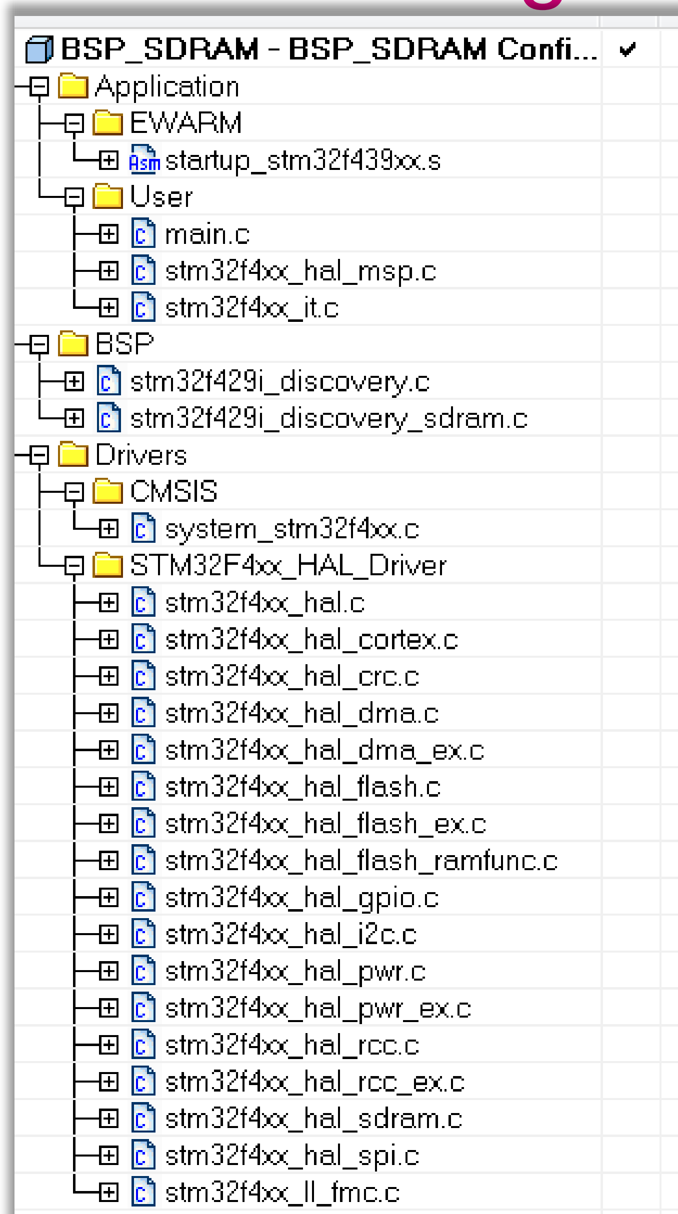
5.2 Use BSP for LCD init and writing

BSP LCD organization



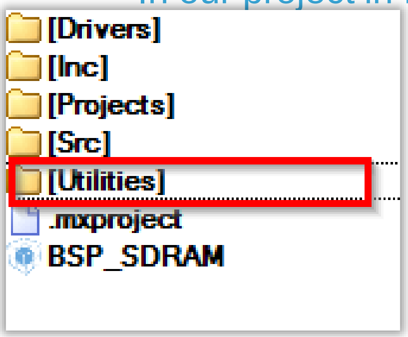
5.2 Use BSP for LCD init and writing

- 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

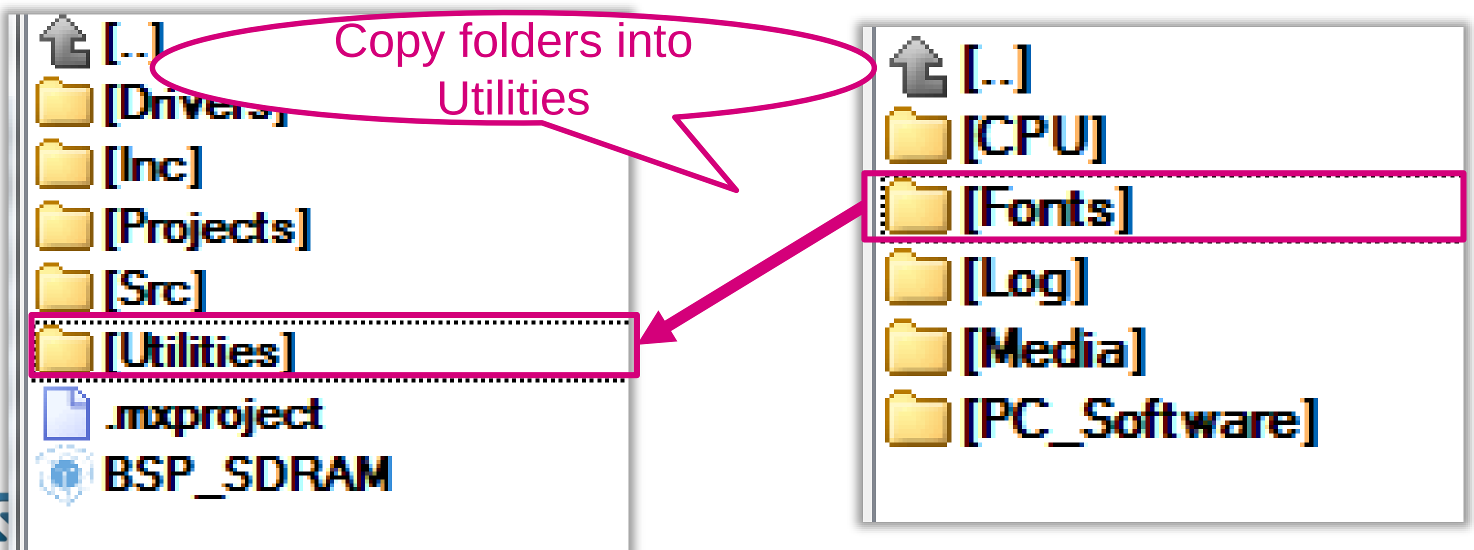


5.2 Use BSP for LCD init and writing

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

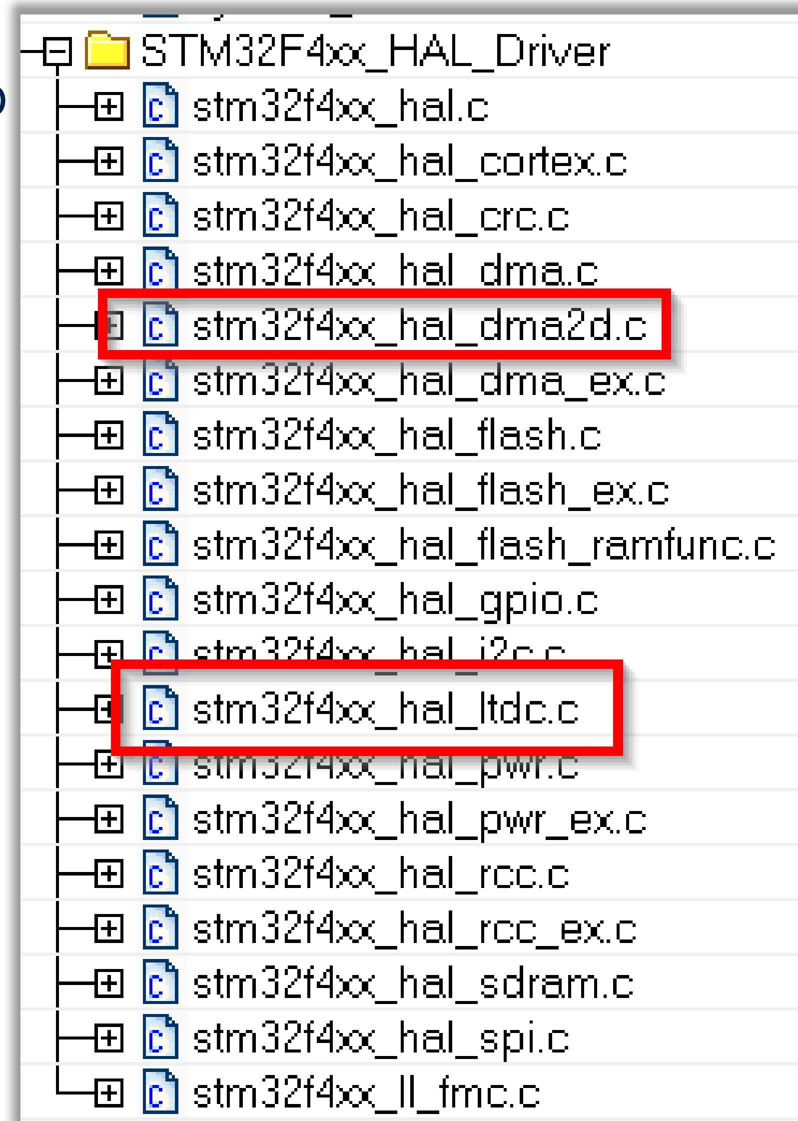


  - Now go into CubeMX repository …\STM32Cube_FW_F4_V1.3.0\Utilities\
  - And copy **Fonts** into **Utilities** folder



5.2 Use BSP for LCD init and writing

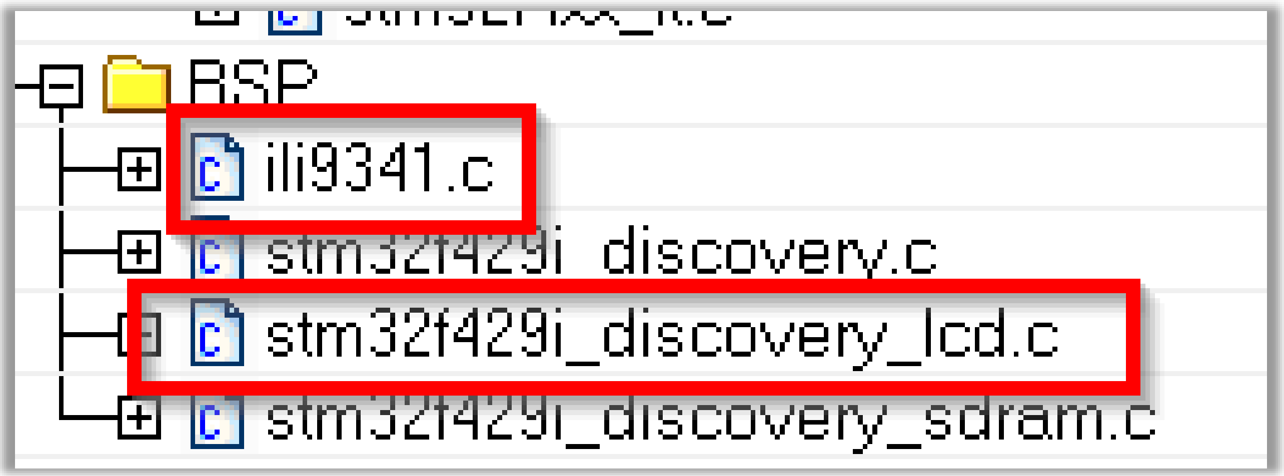
- We add the driver for LCD from HAL
- Right click on STM32F4xx_HAL_Drive>ADD from \Drivers\STM32F4xx_HAL_Driver\Src
    - **stm32f4xx_hal_ltdc.c**
    - **Stm32f4xx_hal_dma2d.c**



life.augmented

5.2 Use BSP for LCD init and writing

- 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**



5.2 Use BSP for LCD init and writing

- 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
```

life.augmented

5.2 Use BSP for LCD init and writing

- Into main.c now we modify include from stm32f429i_discovery_sdram.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 */
```

5.2 Use BSP for LCD init and writing

- 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 */
```

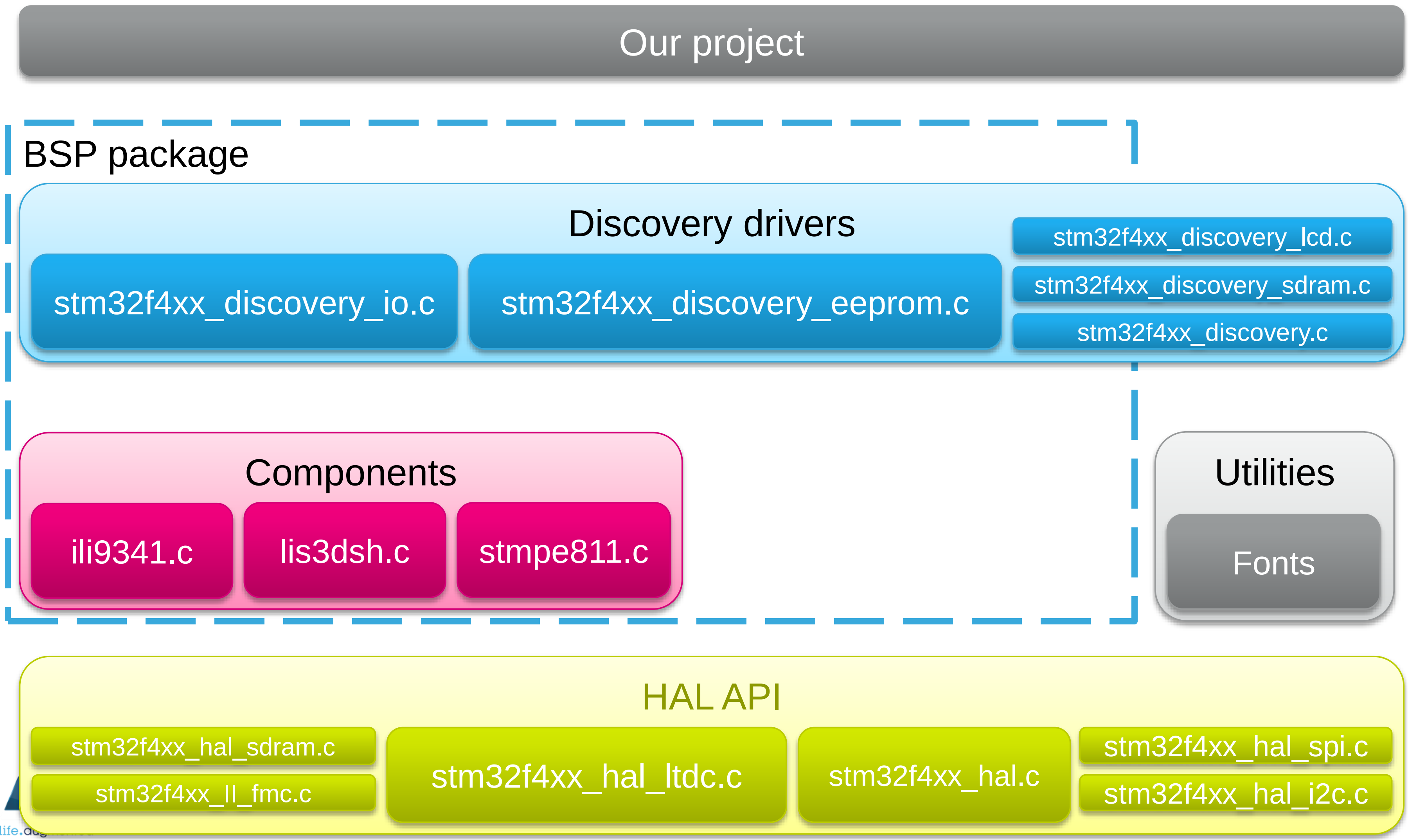
5.3 BSP EEPROM lab

5.3 Use BSP to access EEPROM

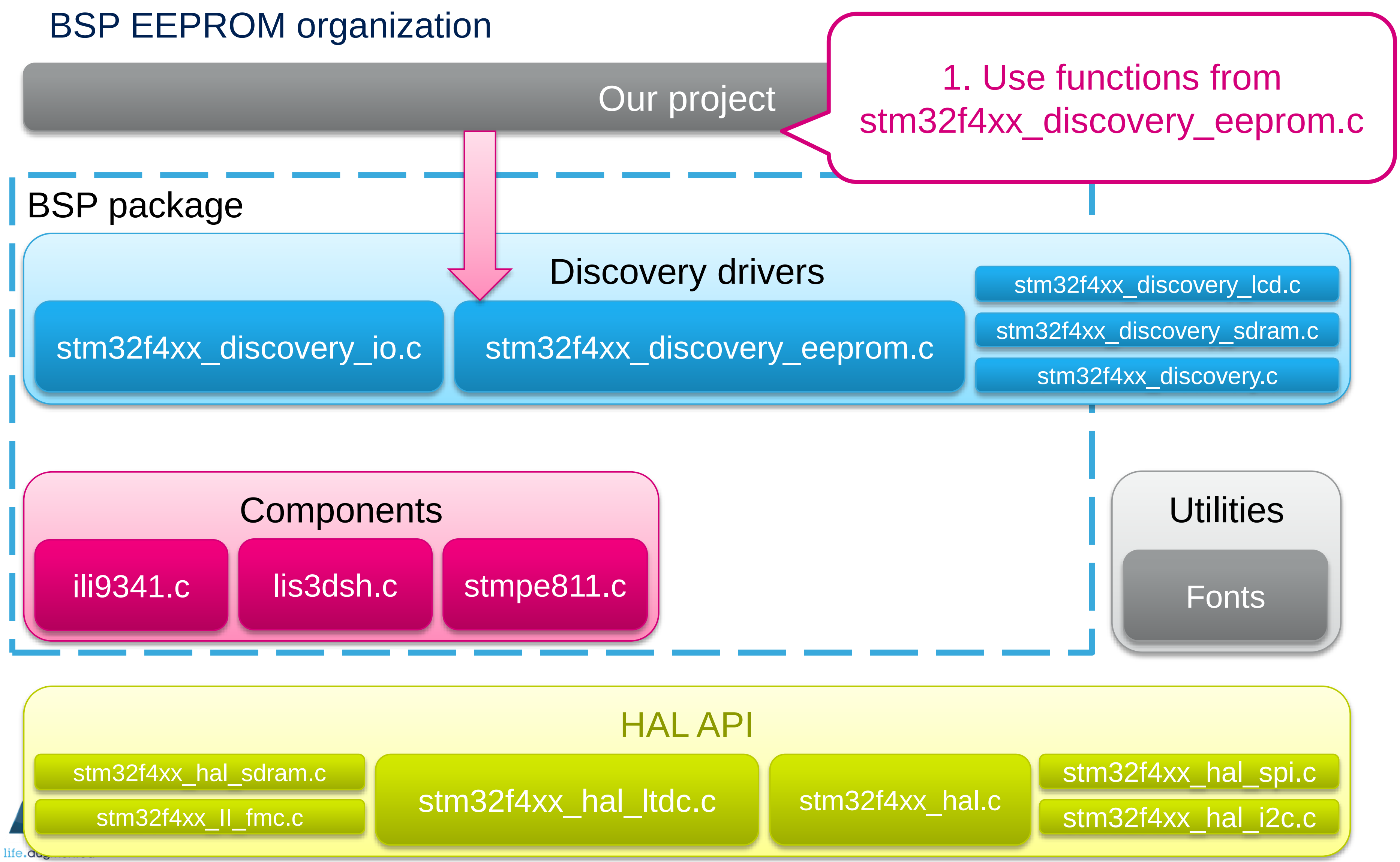
- Objective
  - Learn how import BSP EEPROM into project
  - We use the project from lab 26
  - Which part need to by configured in GUI
  - Try to write text into EEPROM and read it
  - Read text from EEPROM and display it on LCD
- Goal
  - Successfully import BSP EEPROM drivers into your project
  - Learn which part you need to import
  - How to setup the project

5.3 Use BSP to access EEPROM

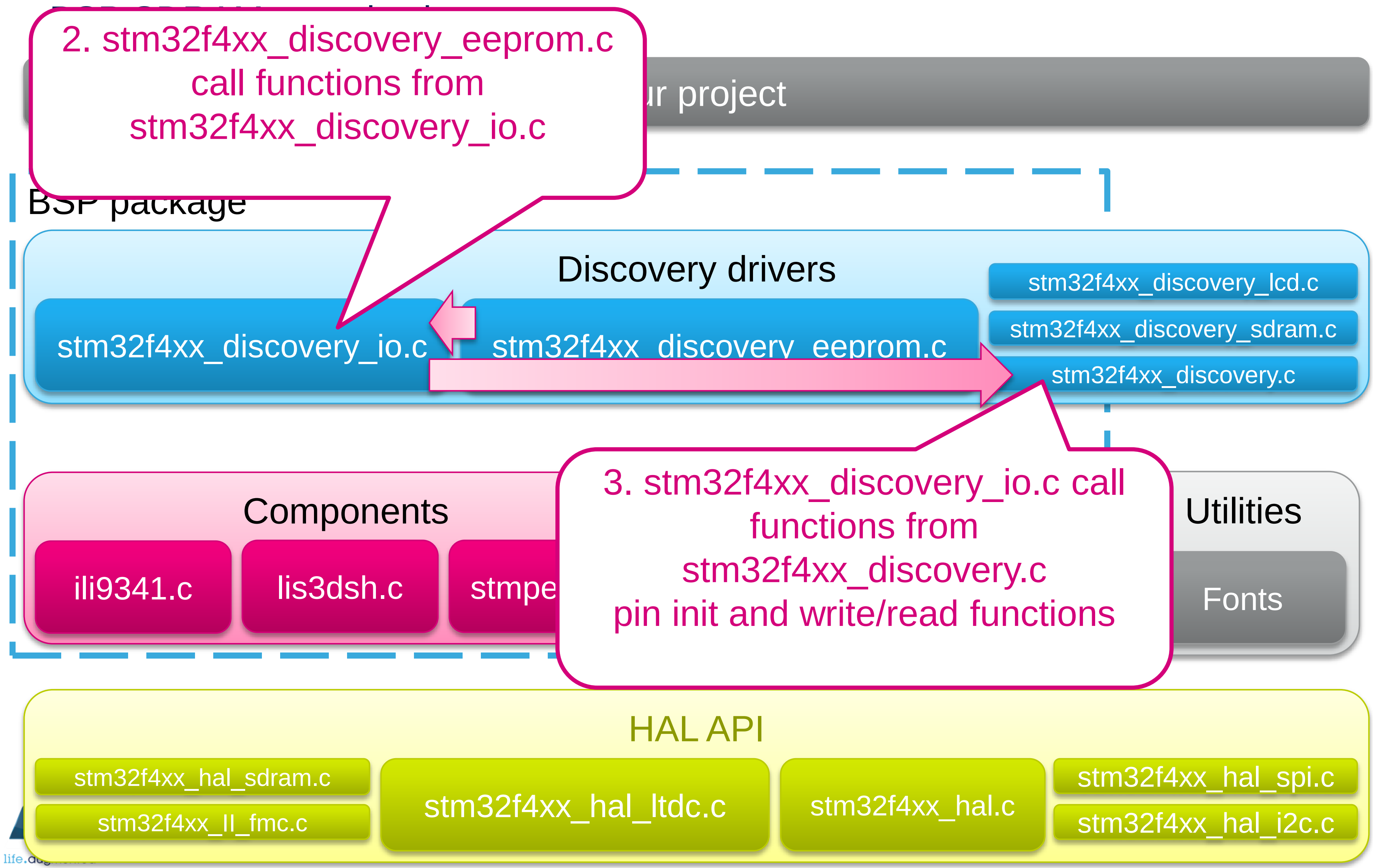
BSP EEPROM organization



5.3 Use BSP to access EEPROM

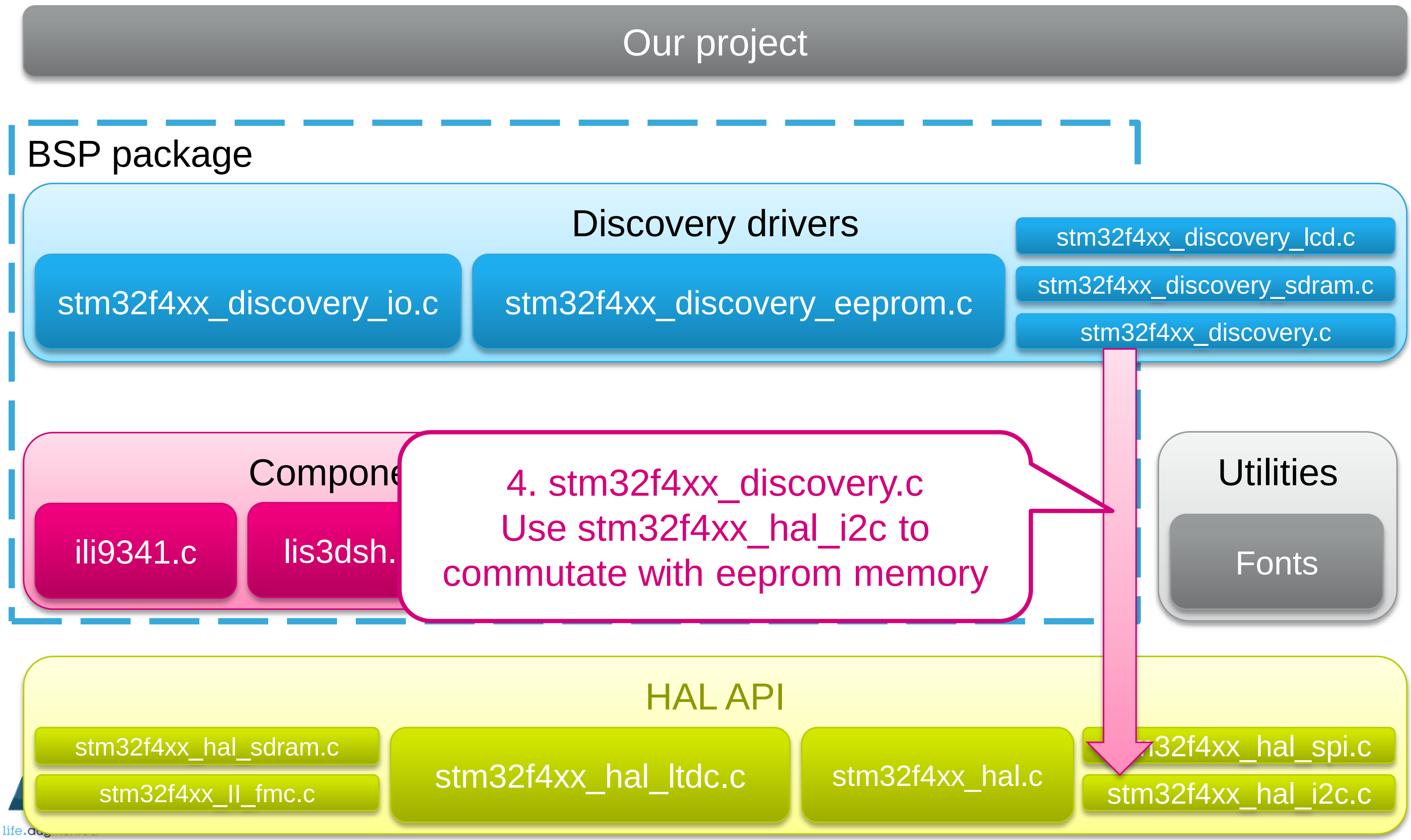


5.3 Use BSP to access EEPROM



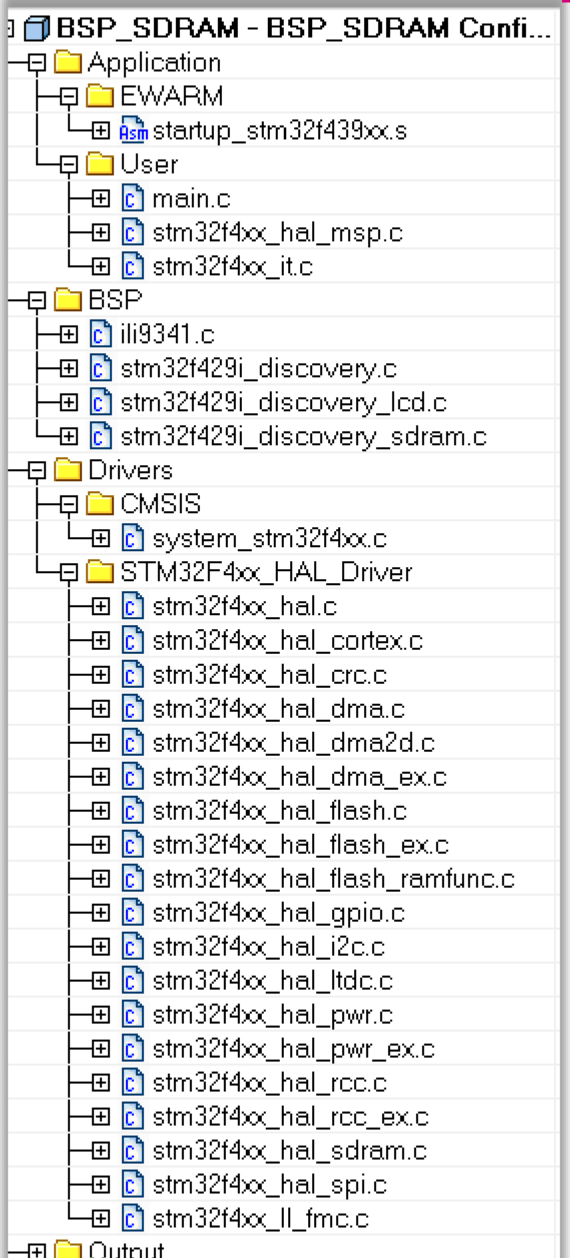
5.3 Use BSP to access EEPROM

BSP EEPROM organization



5.3 Use BSP to access EEPROM

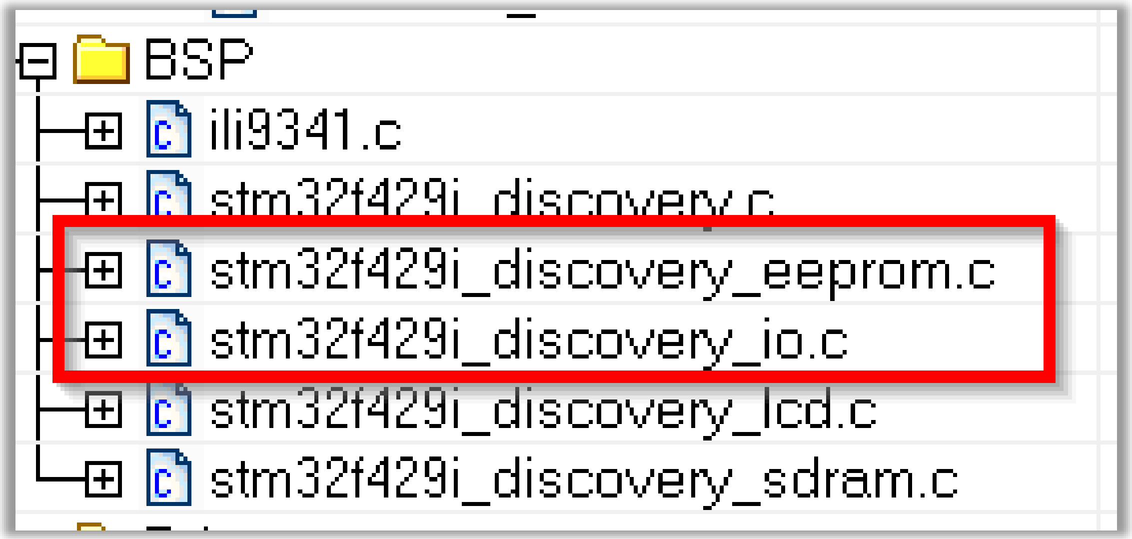
- We use the project from BSP LCD lab 26 because we want to display the memory content on LCD



life.augmented

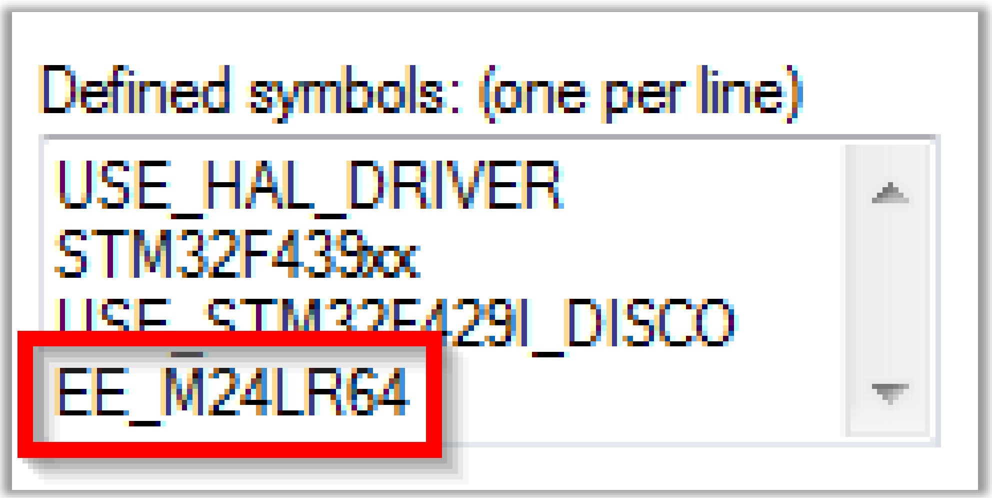
5.3 Use BSP to access EEPROM

- 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**



5.3 Use BSP to access EEPROM

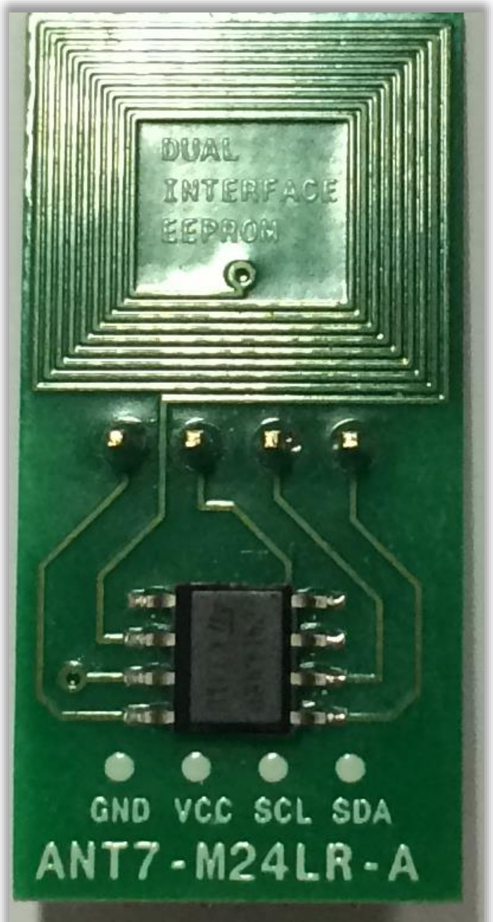
- Add the define of EEPROM into project options
  - Right click on project>Options>Category C/C++Compiler>Preprocesor
  - Into Defined symbols add EE_M24LR64
  - This allow use EEPROM functions
  - Button OK close project options

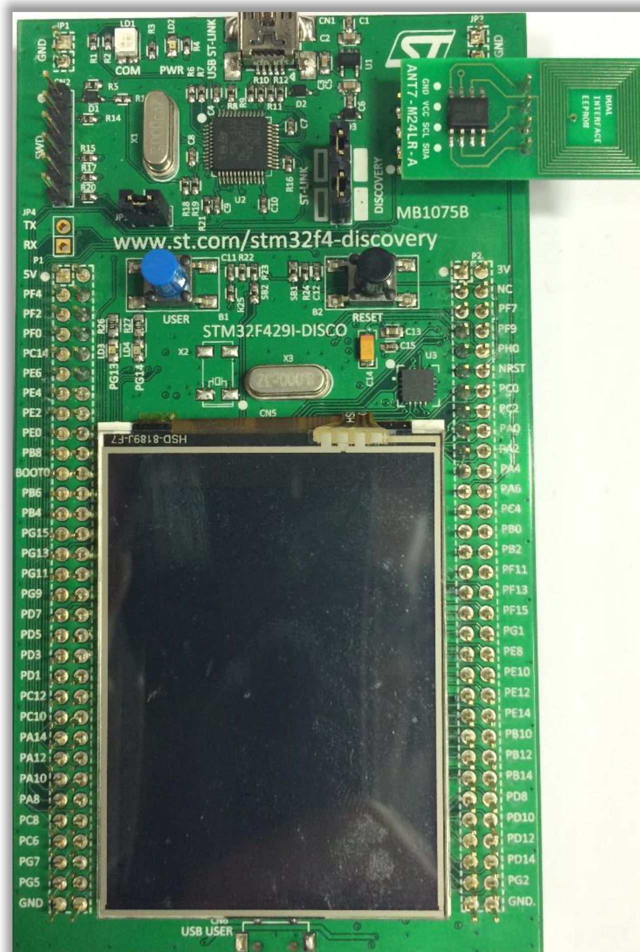


life.augmented

5.3 Use BSP to access EEPROM

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





5.3 Use BSP to access EEPROM

- 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 */
```

5.3 Use BSP to access EEPROM

- 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

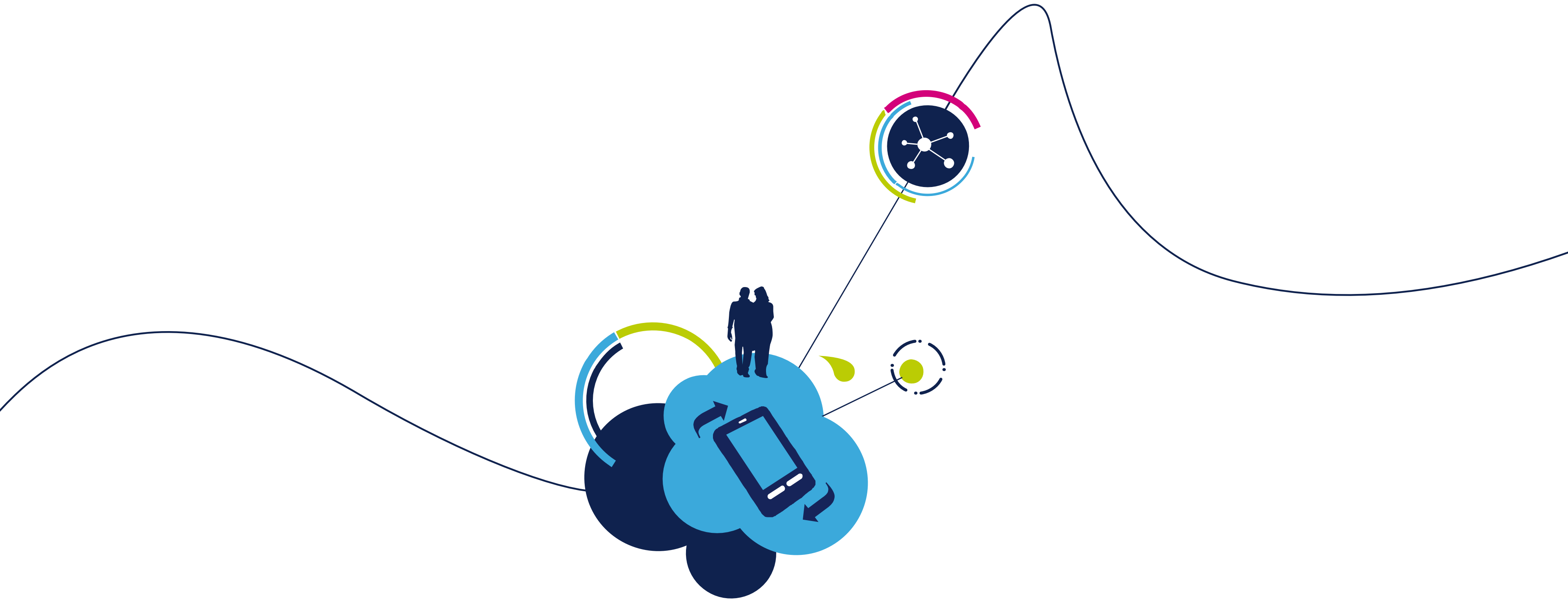
```
/* USER CODE BEGIN 1 */
void DMA1_Stream4_IRQHandler()
{
  HAL_DMA_IRQHandler(I2cHandle.hdmatx);
}

void DMA1_Stream2_IRQHandler()
{
  HAL_DMA_IRQHandler(I2cHandle.hdmarx);
}
/* USER CODE END 1 */
```

5.3 Use BSP to access EEPROM

- Into main.c add


```c
/* 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 */
```

life.augmented



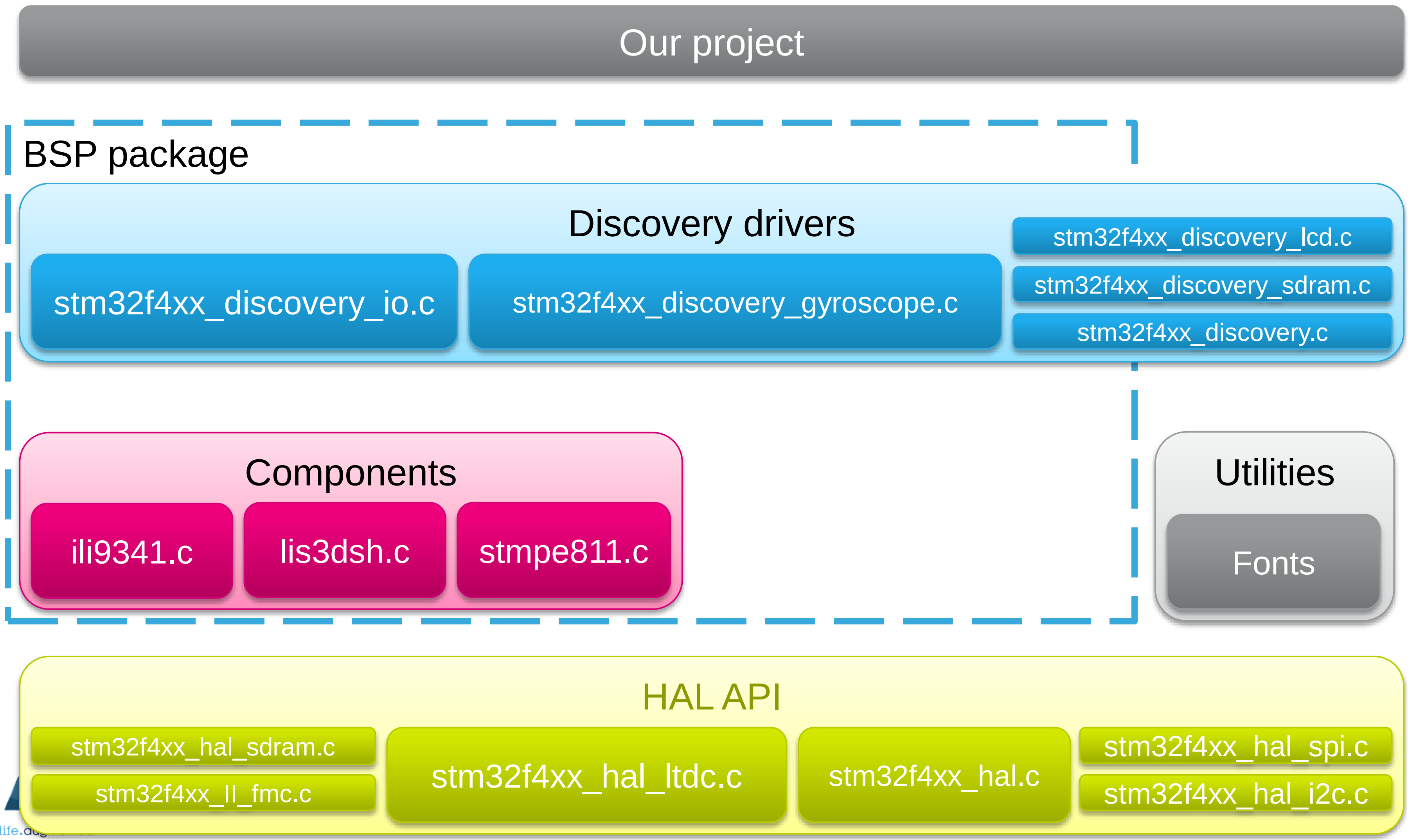
5.4 BSP GYRO lab

5.4 Use BSP to access GYROSCOPE

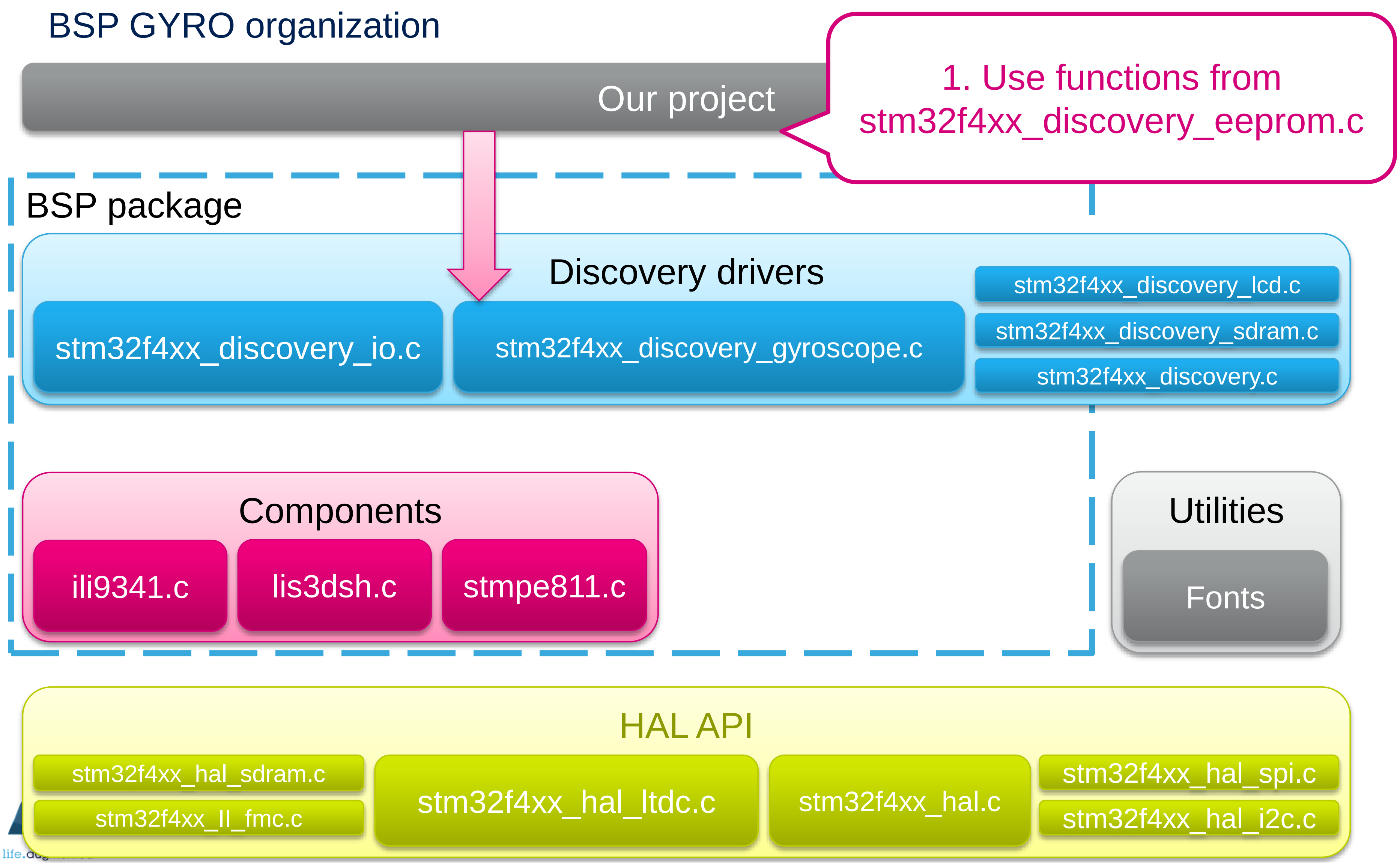
- Objective
  - Learn how import BSP GYROSCOPE into project
  - We use the project from lab 26
  - Which part need to by configured in GUI
  - Read data from GYROSCOPE and display it on LCD
- Goal
  - Successfully import BSP GYROSCOPE drivers into your project
  - Learn which part you need to import
  - How to setup the project

5.4 Use BSP to access GYROSCOPE

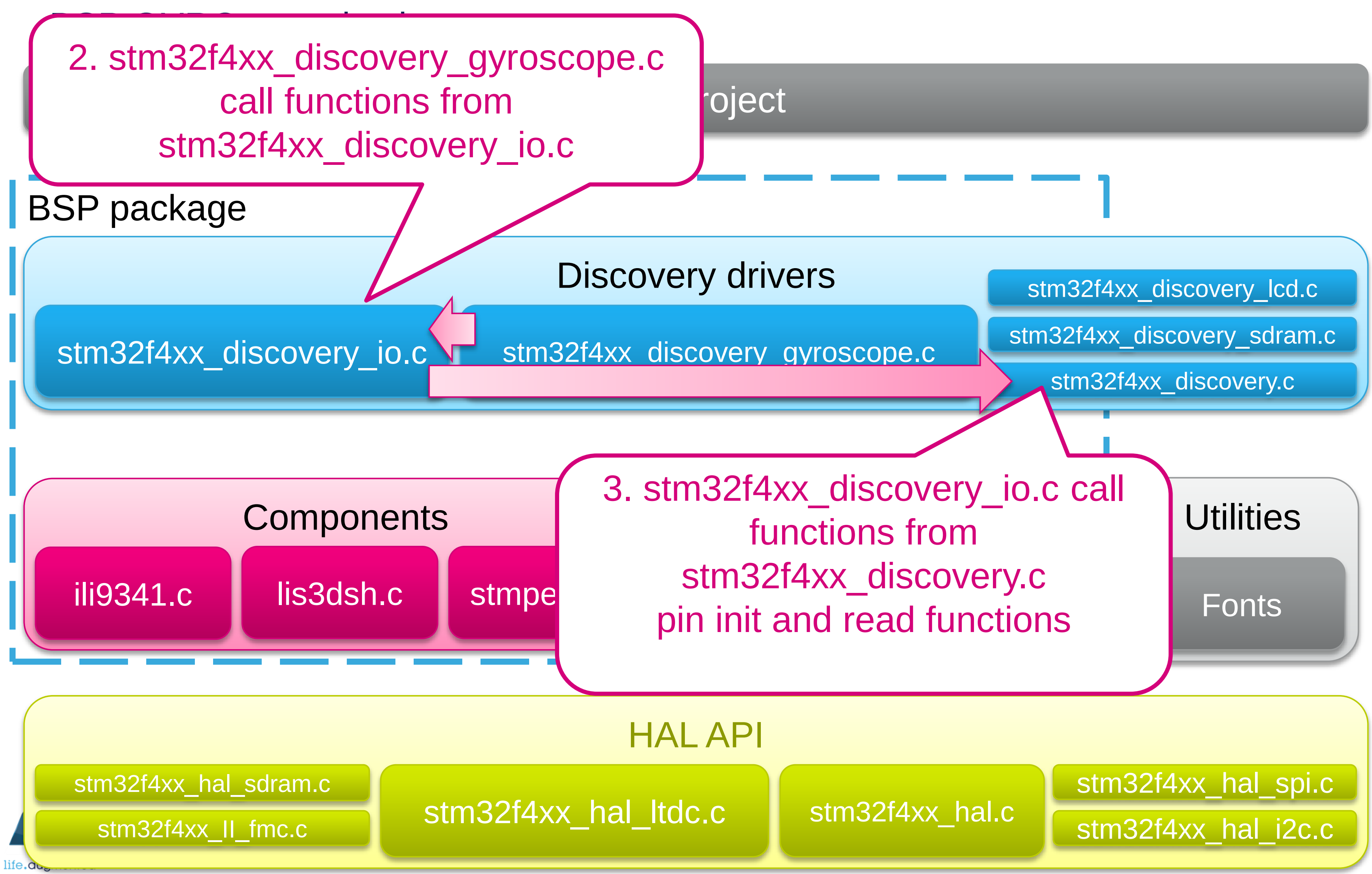
BSP GYRO organization



5.4 Use BSP to access GYROSCOPE

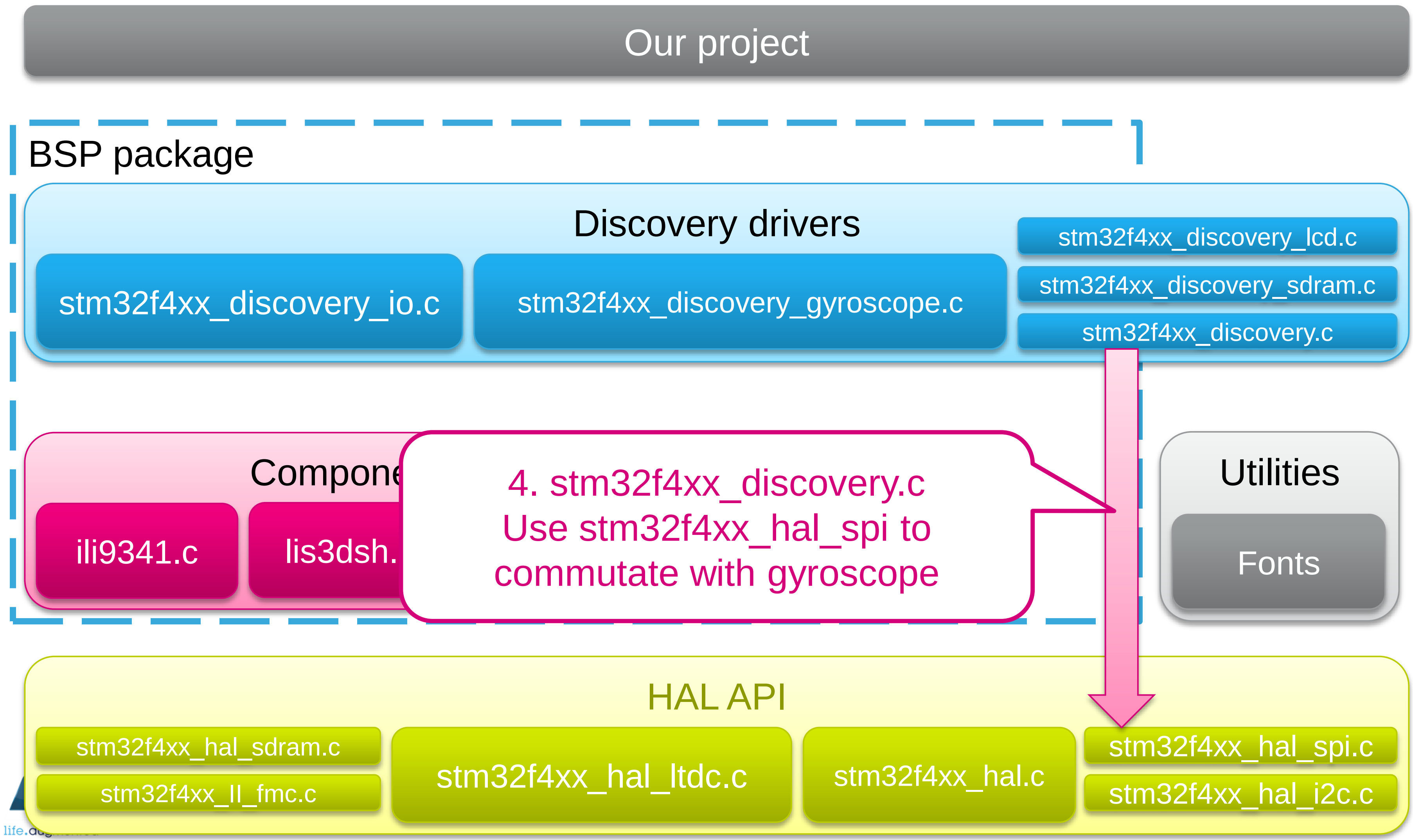


5.4 Use BSP to access GYROSCOPE



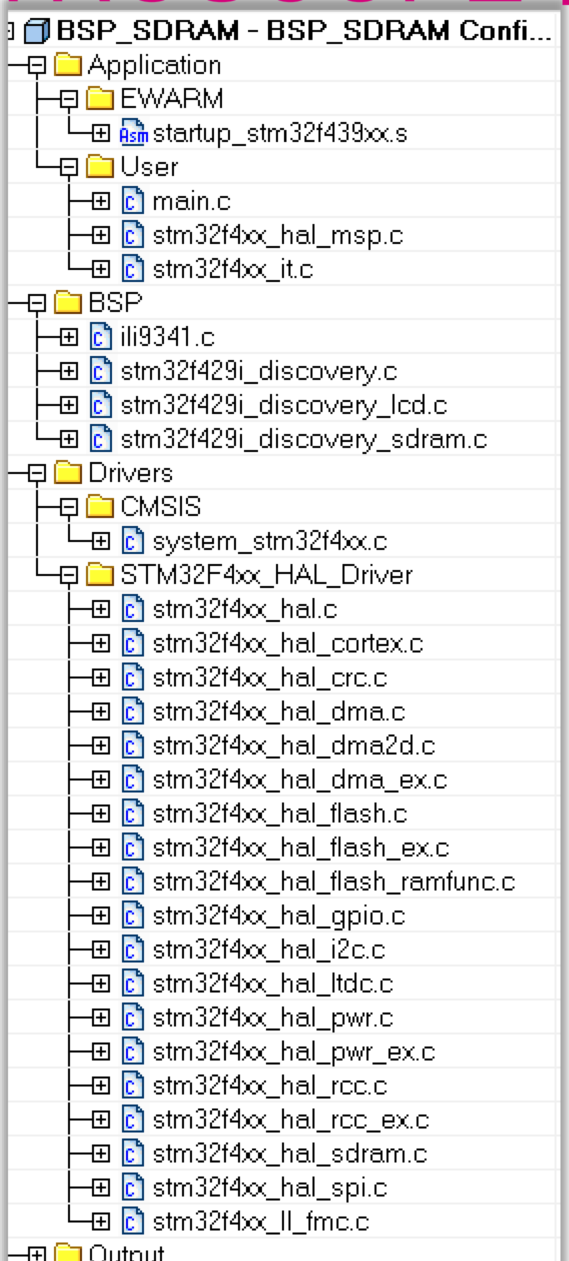
5.4 Use BSP to access GYROSCOPE

BSP GYRO organization



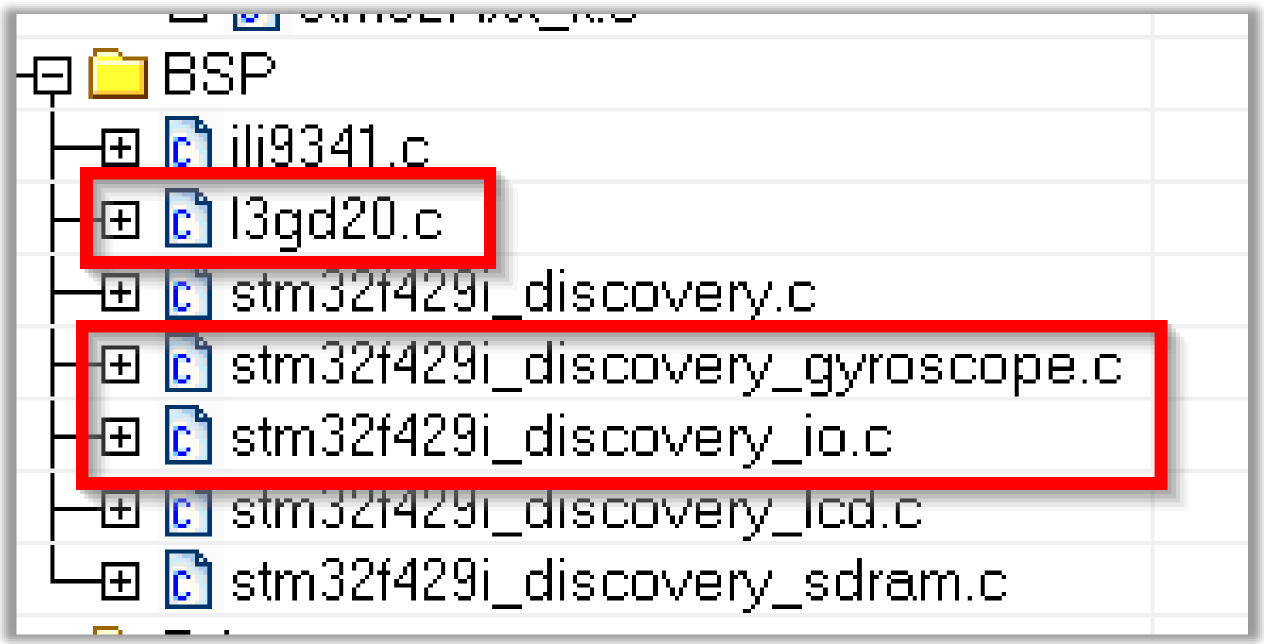
5.4 Use BSP to access GYROSCOPE

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



5.4 Use BSP to access GYROSCOPE

- 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\
  - **l3gd20.c**



life.augmented

5.4 Use BSP to access GYROSCOPE

- 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 */
```

5.4 Use BSP to access GYROSCOPE

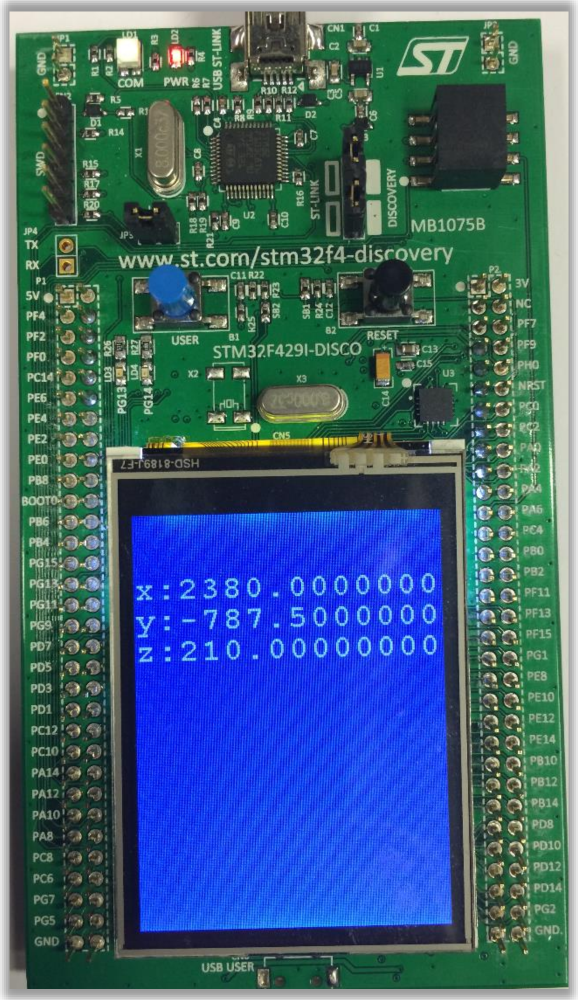
- Into main.c add

```
/* 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 */
```

5.4 Use BSP to access GYROSCOPE

- 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 */
```



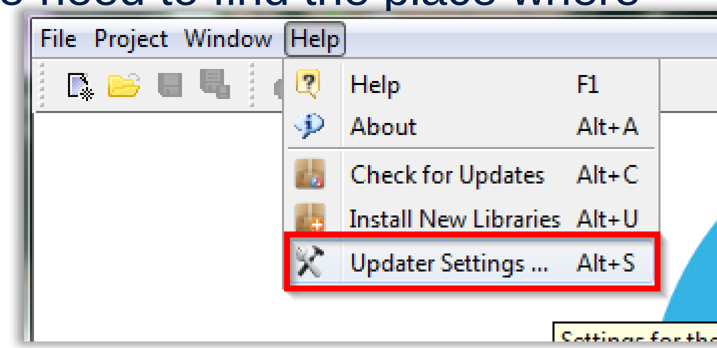
Appendix A CubeMX install

CubeMX install

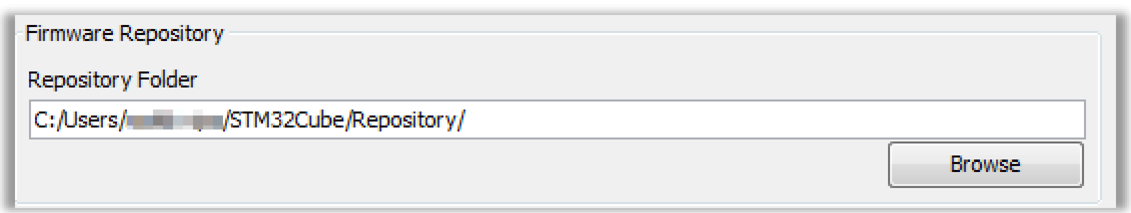
- CubeMX tool
  - http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242?s_searchtype=partnumber
- The CubeMX tool need the java
  - Please check if you have actual java on your pc, for sure 32bit and 64bit version
- Optionally you can download the Cube packages for STM32 device if you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

CubeMX install

- 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



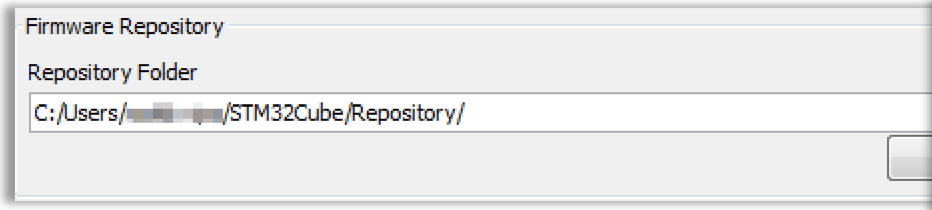
- MENU>Help>Updater Settings…
- You will see where is the repository folder
  - Default is C:/User/Acc_name/STM32Cube/Repository/
- You need to download STM32 packages into this folder
- Or CubeMX automatically download them into this folder

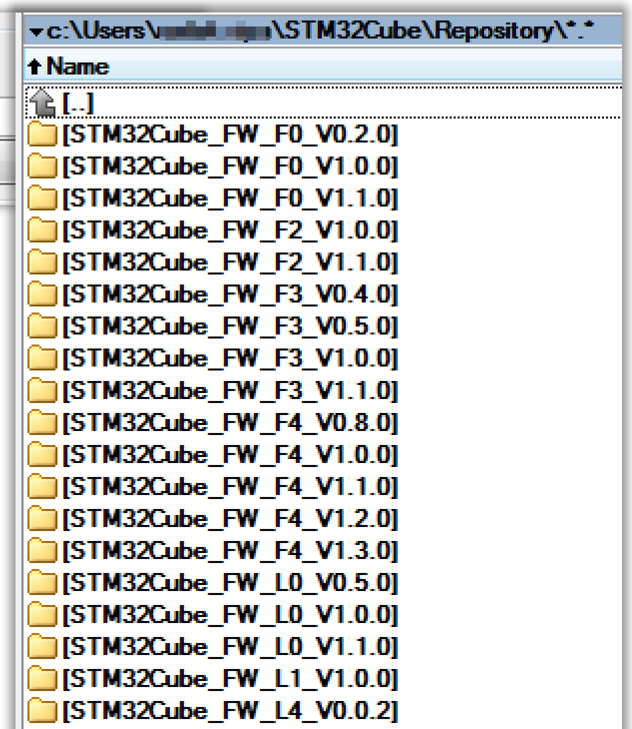


A

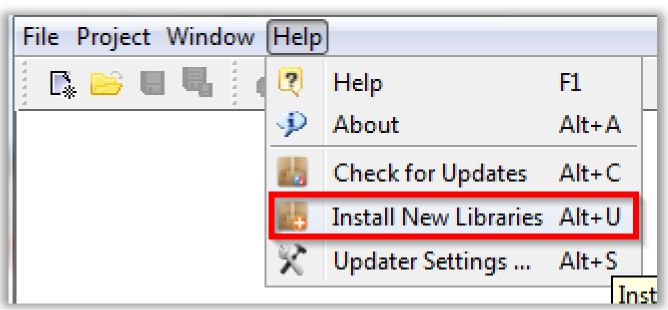
CubeMX install

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



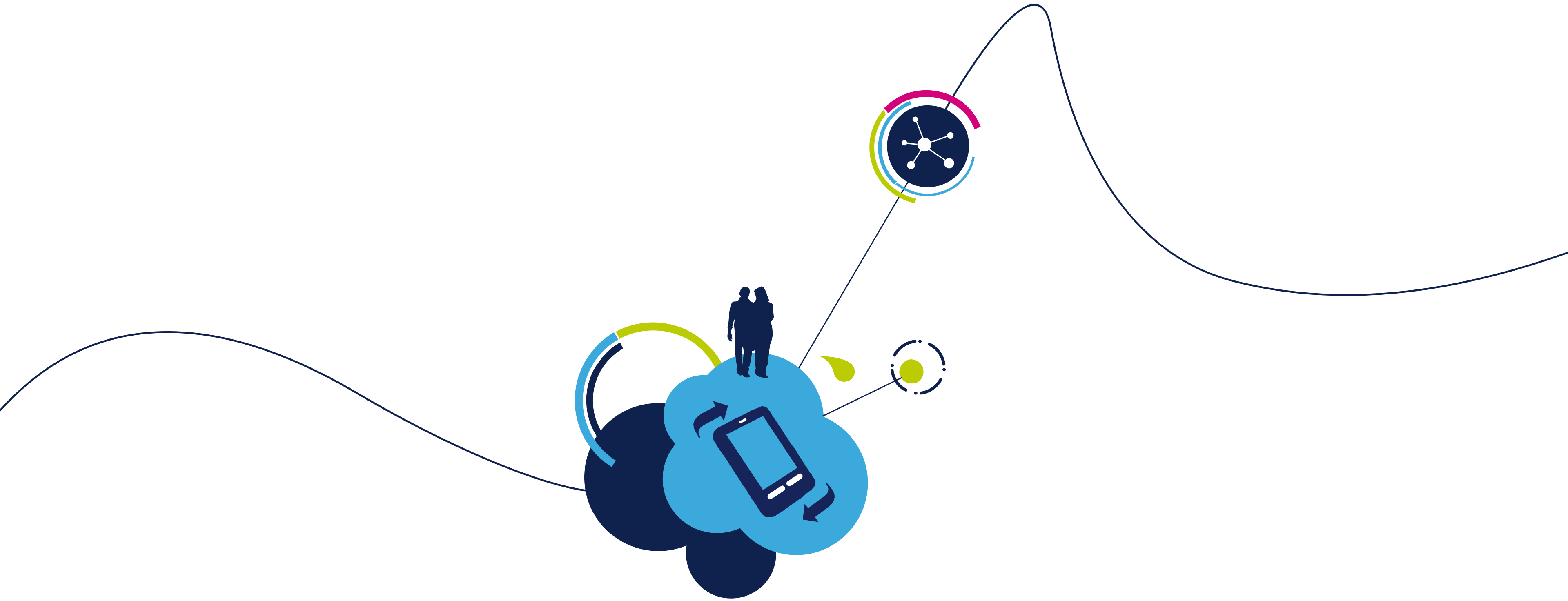


- In case you want to download this files automatically use in CubeMX
  - MENU>Help>Install New Libraries
  - Select libraries which you want
  - Force download with button Install Now



CubeMX install

- 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

B

CubeMX documentation

- CubeMX user manual UM1718
  - 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
- CubeMX technical note TN0072
  - http://www.st.com/st-web-ui/static/active/en/resource/technical/document/technical_note/CD00214439.pdf

STM32F4 documentation

- 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
- STM32F429 Programming manual
  - http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00046982.pdf

B STM32F429i-Discovery documentation

- STM32F429i-Discovery page
  - 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

www.st.com/stm32

life.augmented