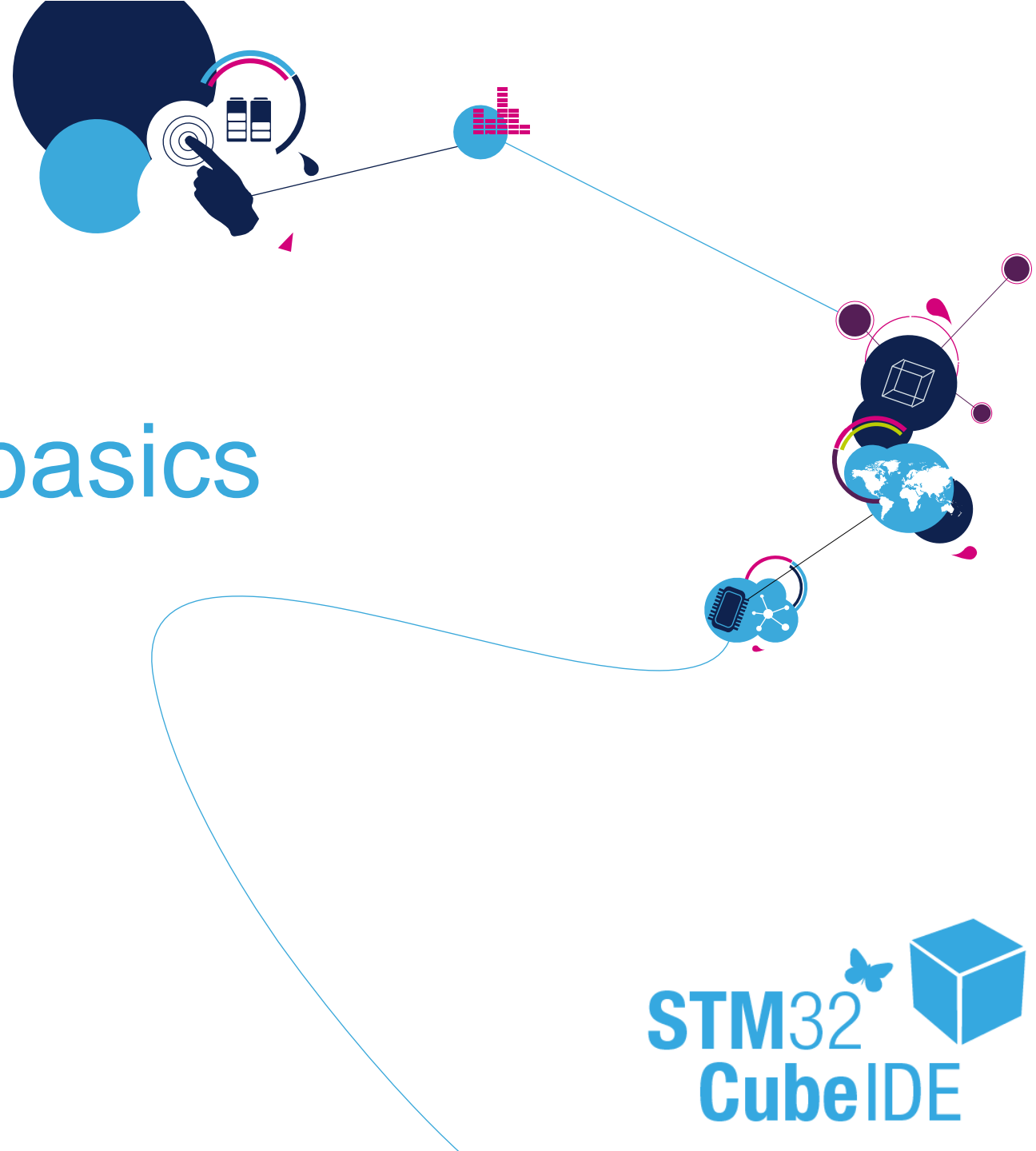
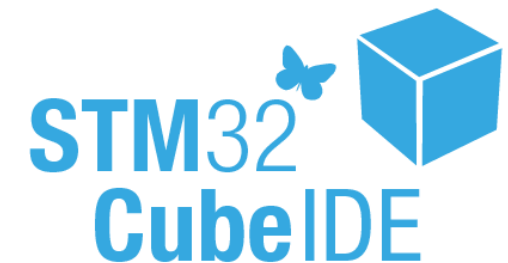


# STM32CubeIDE basics

CMSIS\_OS lab: Basic project on FreeRTOS



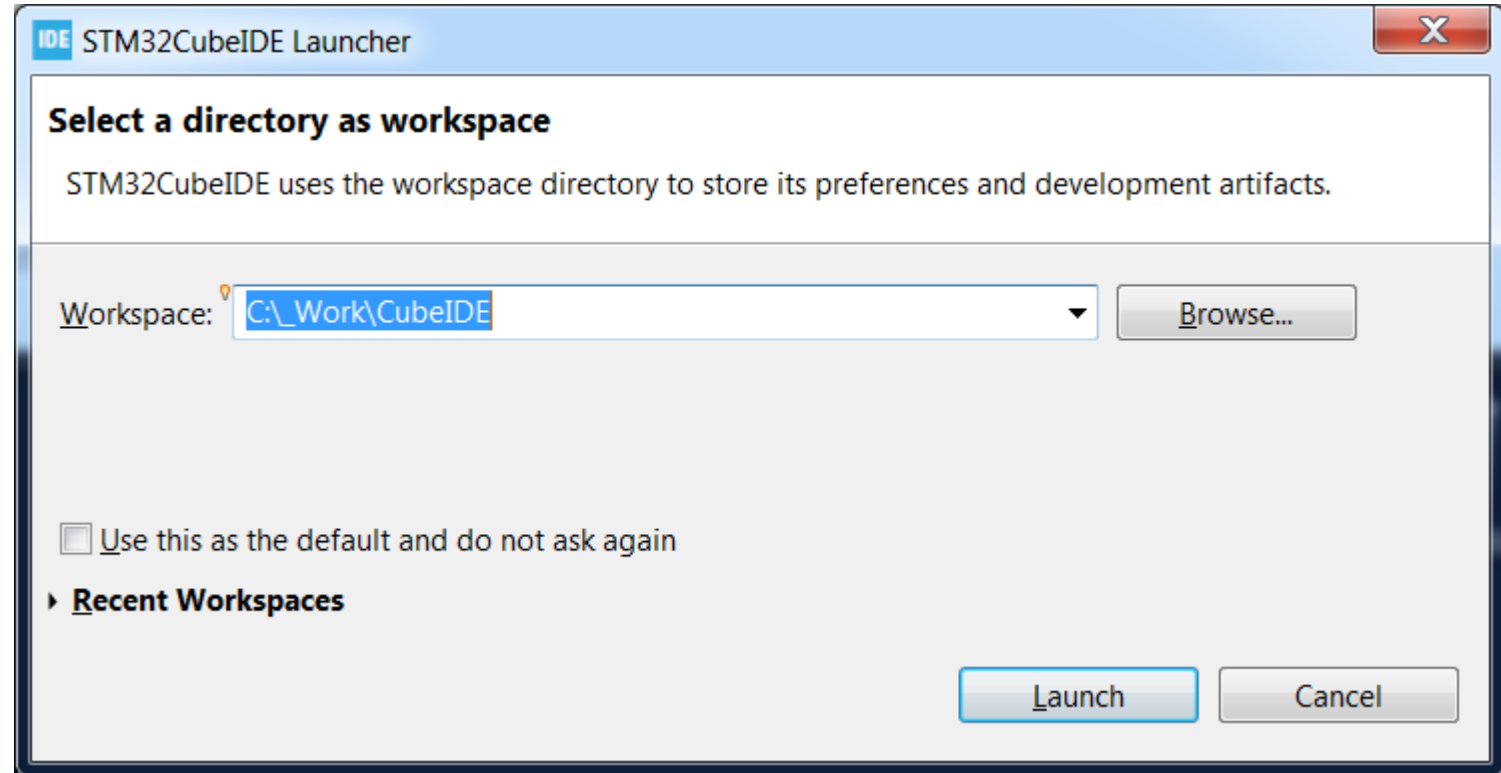
# Lab: Basic project on FreeRTOS

## Objectives:

- Configure PA5 as GPIO Output with LED\_GREEN label
- Configure PC13 as GPIO\_EXTI13 with BLUE\_BUTTON label
- Add FreeRTOS middleware to the project with CMSIS\_OS layer on top
- Add 2 tasks and one binary semaphore and use them to react on button press (interrupt) and LED control (ON/OFF)

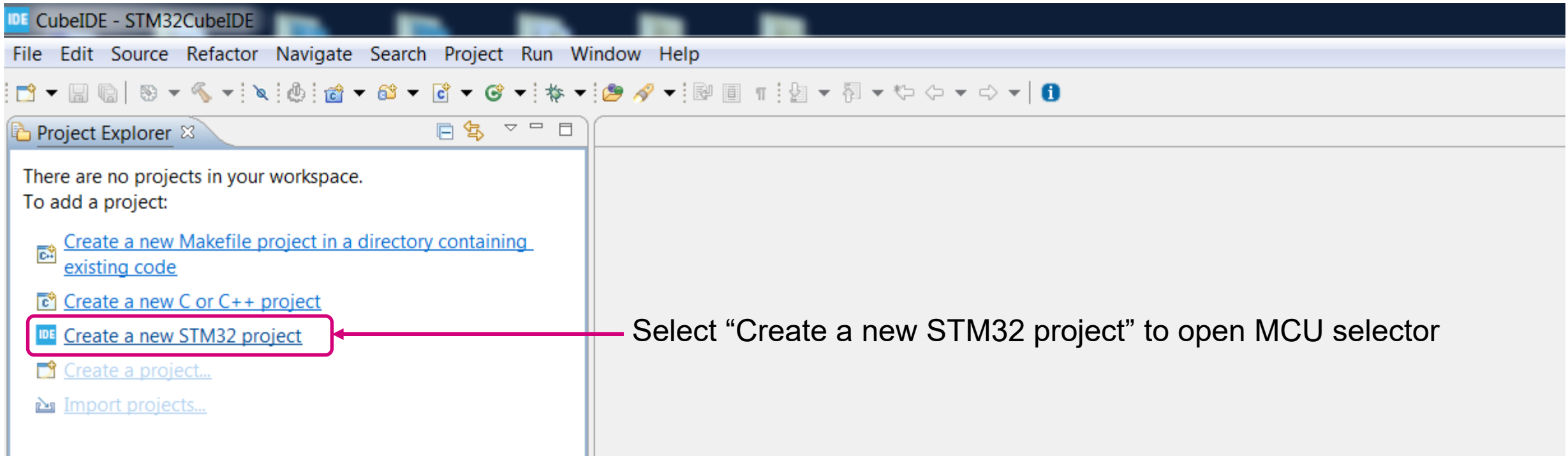
# Start a new workspace

- Run STM32CubeIDE
- Select a folder to store a workspace



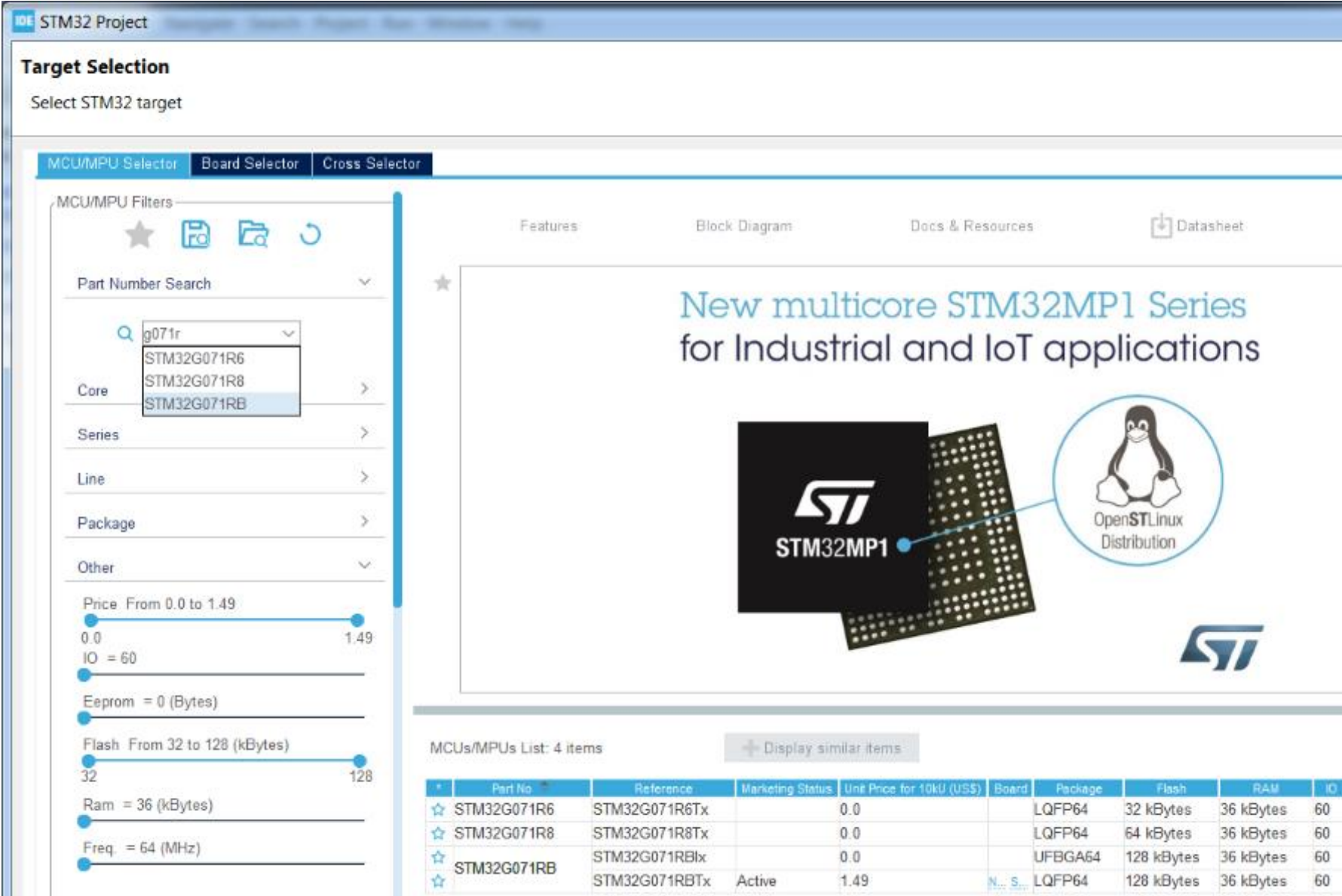
# Create a new project

- Click on “Create a new STM32 project”



# Select target MCU: STM32G071RBTx

- We will use STM32G071RBTx MCU



**Target Selection**  
Select STM32 target

MCU/MPU Selector | Board Selector | Cross Selector

MCU/MPU Filters

Part Number Search: g071r

Core: STM32G071R6, STM32G071R8, **STM32G071RB**

Series: >

Line: >

Package: >

Other: >

Price: From 0.0 to 1.49

IO: = 60

Eeprom: = 0 (Bytes)

Flash: From 32 to 128 (kBytes)

Ram: = 36 (kBytes)

Freq.: = 64 (MHz)

MCUs/MPUs List: 4 items

	Part No	Reference	Marketing Status	Unit Price for 10kU (US\$)	Board	Package	Flash	RAM	IO
☆	STM32G071R6	STM32G071R6Tx		0.0		LQFP64	32 kBytes	36 kBytes	60
☆	STM32G071R8	STM32G071R8Tx		0.0		LQFP64	64 kBytes	36 kBytes	60
☆	STM32G071RB	STM32G071RBTx	Active	1.49	N... S...	UFBGA64	128 kBytes	36 kBytes	60
☆	STM32G071RBTx	STM32G071RBTx				LQFP64	128 kBytes	36 kBytes	60

New multicore STM32MP1 Series for Industrial and IoT applications

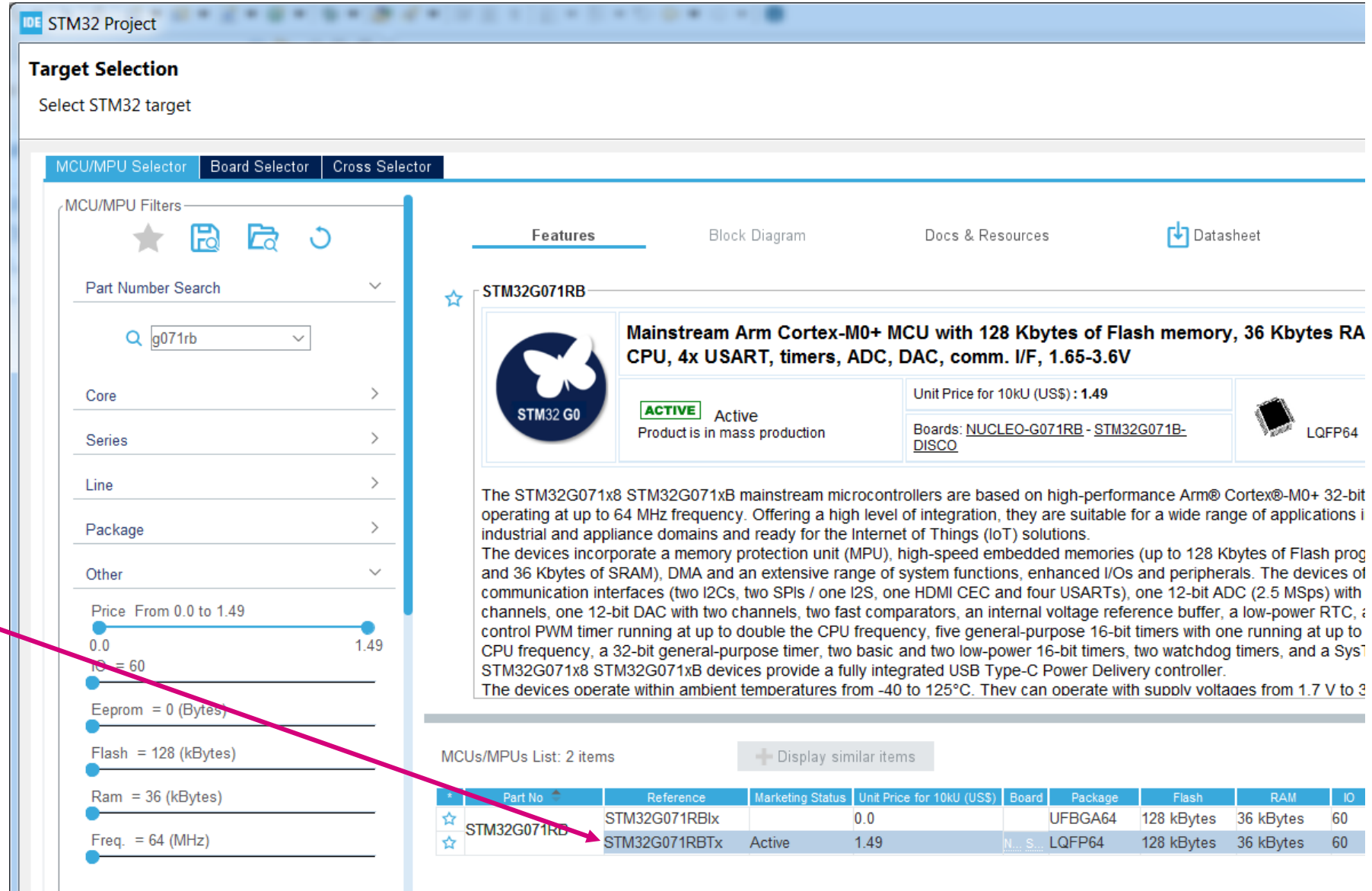
STM32MP1

OpenSTLinux Distribution

ST

# Select target MCU: STM32G071RBTx

- It is possible to view on main MCU features, download its documentation
- To start a new project we need to double click on the part number



**Target Selection**  
Select STM32 target

MCU/MCU Selector | Board Selector | Cross Selector

MCU/MCU Filters

Part Number Search

g071rb

Core

Series

Line

Package

Other

Price From 0.0 to 1.49

0.0 1.49

IC = 60

Eeprom = 0 (Bytes)

Flash = 128 (kBytes)

Ram = 36 (kBytes)

Freq. = 64 (MHz)

**Features** | Block Diagram | Docs & Resources | Datasheet

★ STM32G071RB

**Mainstream Arm Cortex-M0+ MCU with 128 Kbytes of Flash memory, 36 Kbytes RA CPU, 4x USART, timers, ADC, DAC, comm. I/F, 1.65-3.6V**

**ACTIVE** Active  
Product is in mass production

Unit Price for 10kU (US\$) : 1.49

Boards: [NUCLEO-G071RB](#) - [STM32G071B-DISCO](#)

LQFP64

The STM32G071x8 STM32G071xB mainstream microcontrollers are based on high-performance Arm® Cortex®-M0+ 32-bit operating at up to 64 MHz frequency. Offering a high level of integration, they are suitable for a wide range of applications in industrial and appliance domains and ready for the Internet of Things (IoT) solutions.

The devices incorporate a memory protection unit (MPU), high-speed embedded memories (up to 128 Kbytes of Flash prog and 36 Kbytes of SRAM), DMA and an extensive range of system functions, enhanced I/Os and peripherals. The devices of communication interfaces (two I2Cs, two SPIs / one I2S, one HDMI CEC and four USARTs), one 12-bit ADC (2.5 MSps) with channels, one 12-bit DAC with two channels, two fast comparators, an internal voltage reference buffer, a low-power RTC, a control PWM timer running at up to double the CPU frequency, five general-purpose 16-bit timers with one running at up to CPU frequency, a 32-bit general-purpose timer, two basic and two low-power 16-bit timers, two watchdog timers, and a SysT

STM32G071x8 STM32G071xB devices provide a fully integrated USB Type-C Power Delivery controller.

The devices operate within ambient temperatures from -40 to 125°C. They can operate with supply voltages from 1.7 V to 3

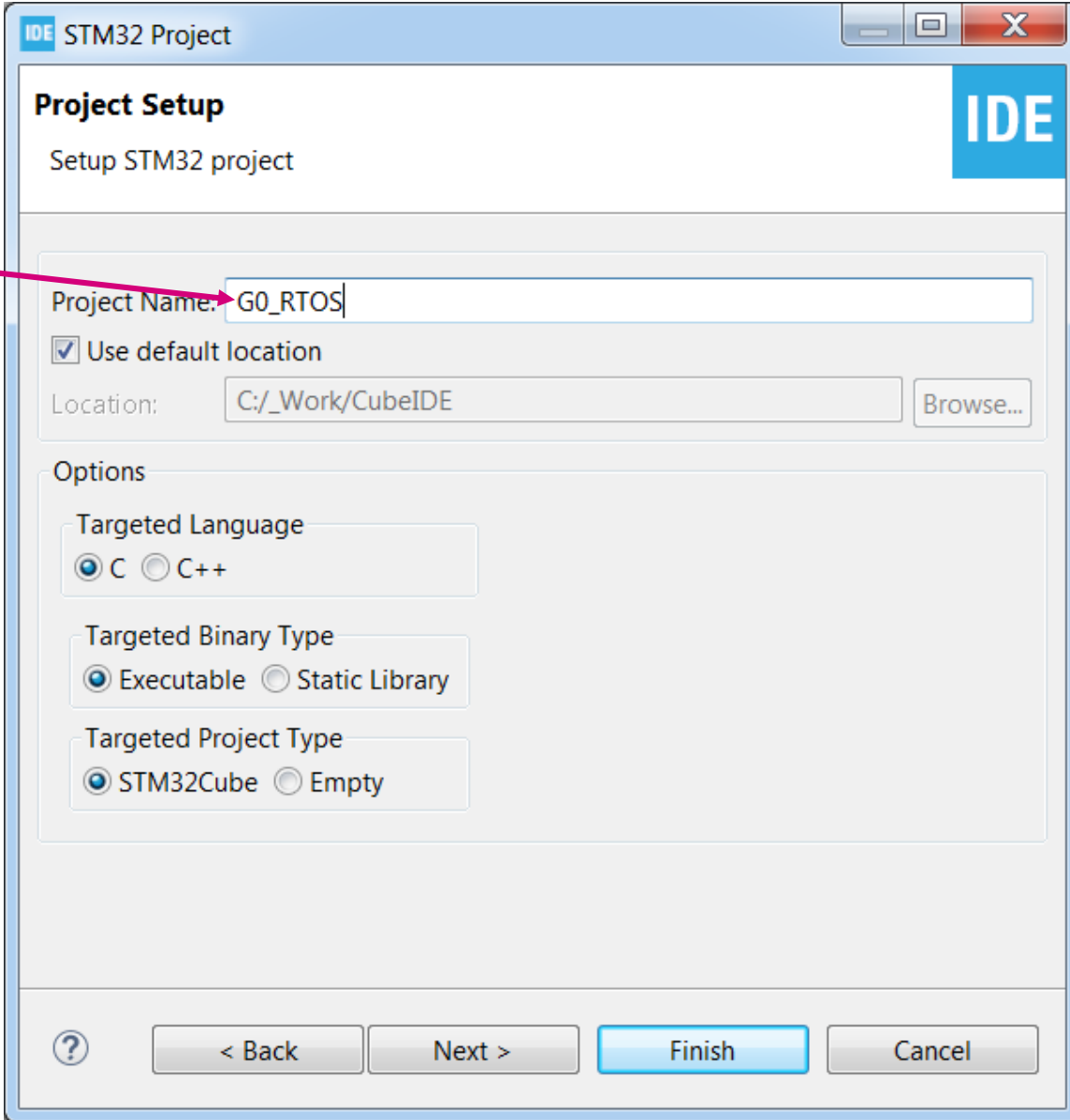
MCUs/MPUs List: 2 items

Display similar items

★	Part No	Reference	Marketing Status	Unit Price for 10kU (US\$)	Board	Package	Flash	RAM	IO
★	STM32G071RBx	STM32G071RBx		0.0		UFPGA64	128 kBytes	36 kBytes	60
★	STM32G071RBTx	STM32G071RBTx	Active	1.49	N... S...	LQFP64	128 kBytes	36 kBytes	60

# Enter project name

- Specify project name, optionally its location (if different from workspace one)
- Additionally we can specify target language (C or C++), binary type (executable or static library) and project type (generated by STM32CubeMX or an empty one)



**Project Setup**  
Setup STM32 project

Project Name: G0\_RTOS

☒ Use default location

Location: C:/\_Work/CubeIDE Browse...

**Options**

Targeted Language  
☒ C ☐ C++

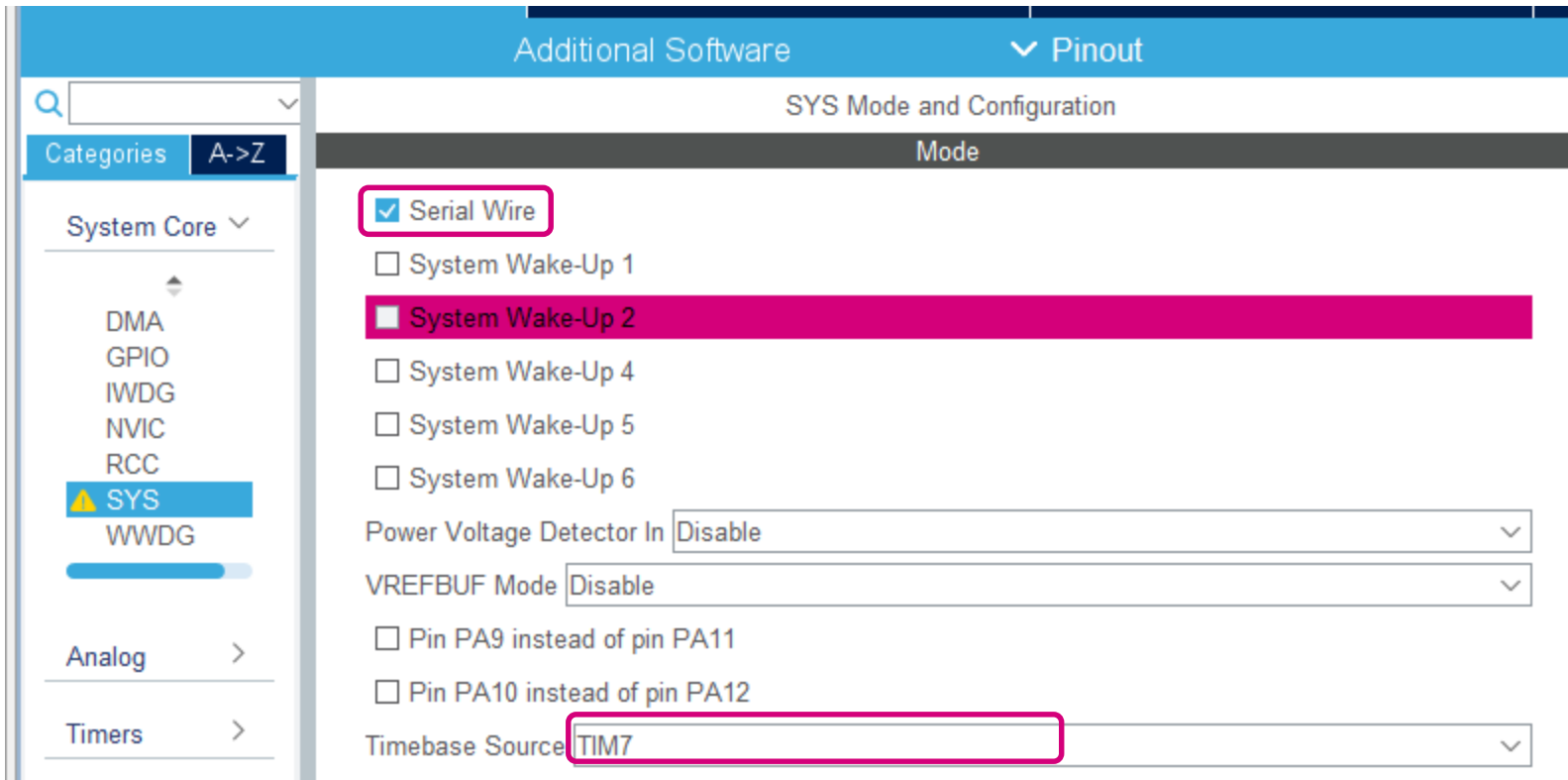
Targeted Binary Type  
☒ Executable ☐ Static Library

Targeted Project Type  
☒ STM32Cube ☐ Empty

? < Back Next > Finish Cancel

# Enabling Serial Wire debug interface

- Select “Serial Wire” from System Core -> SYS peripheral group
- As a result PA13 and PA14 will be assigned to SWD interface
- Change a timebase source (used by HAL library functions) from SysTick to other timer (i.e. TIM7) as SysTick will be used by operating system

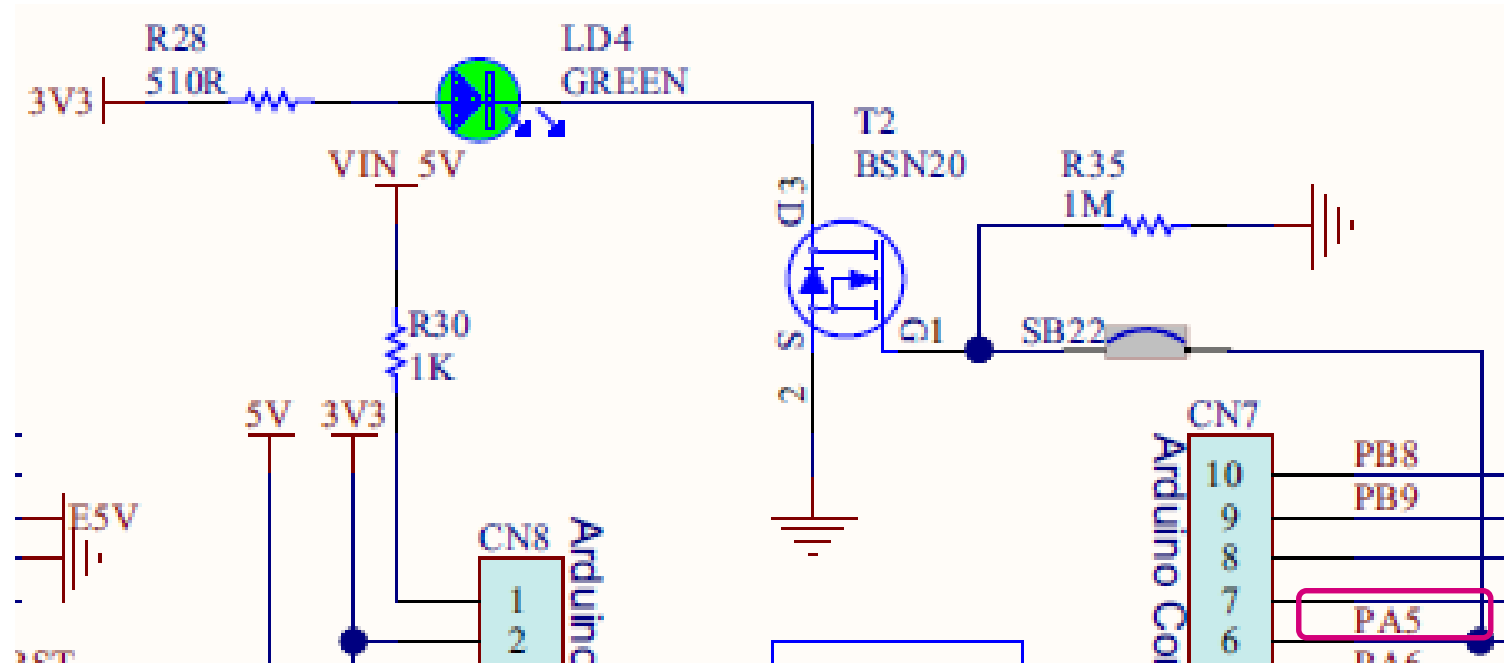




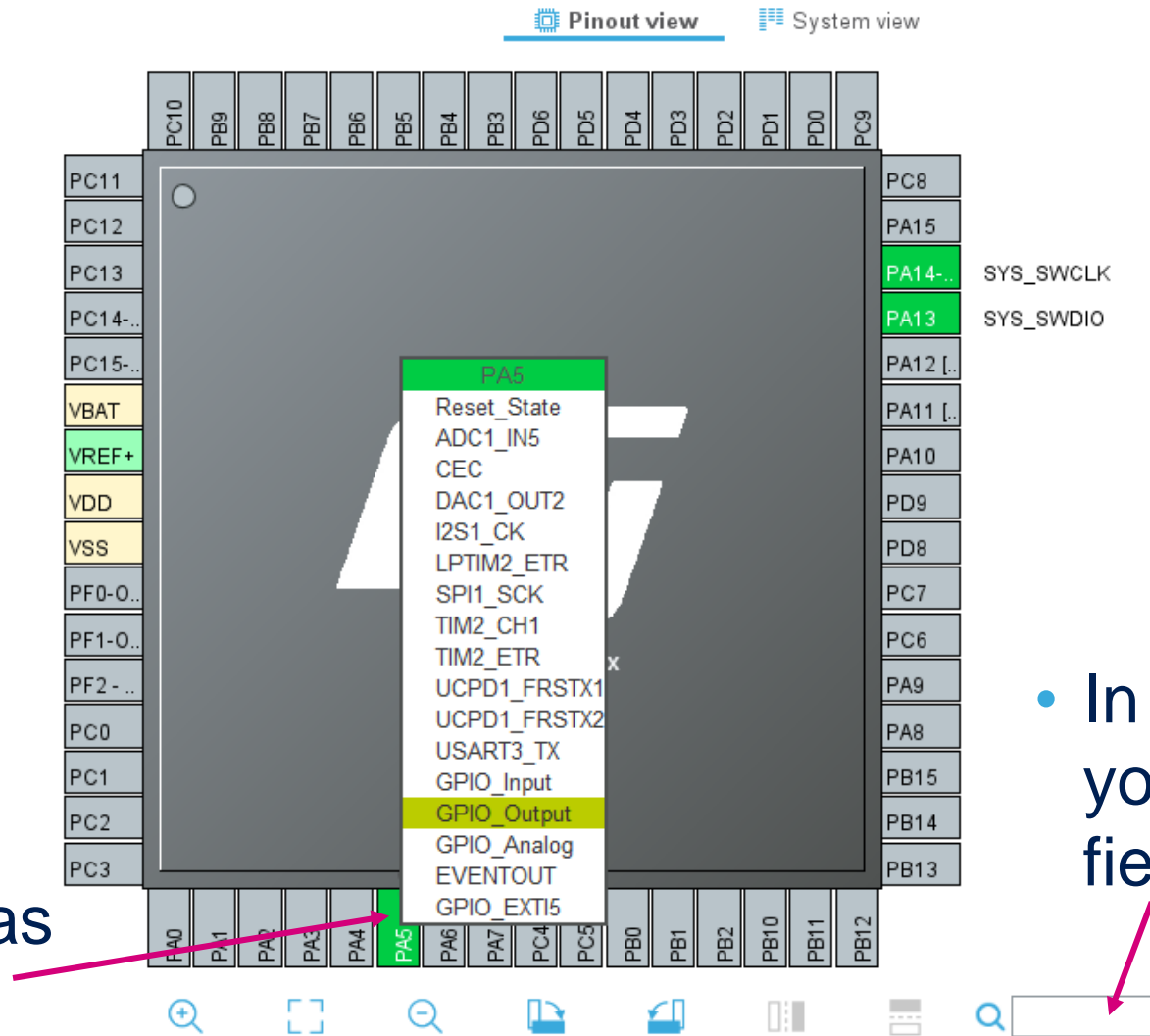
# Pin Configuration

## Green LED

- In this example we are going to use one of the LEDs present on the STM32G0 Nucleo board (connected to PA5 as seen in the schematic below)



# Configuring PA5 as Output



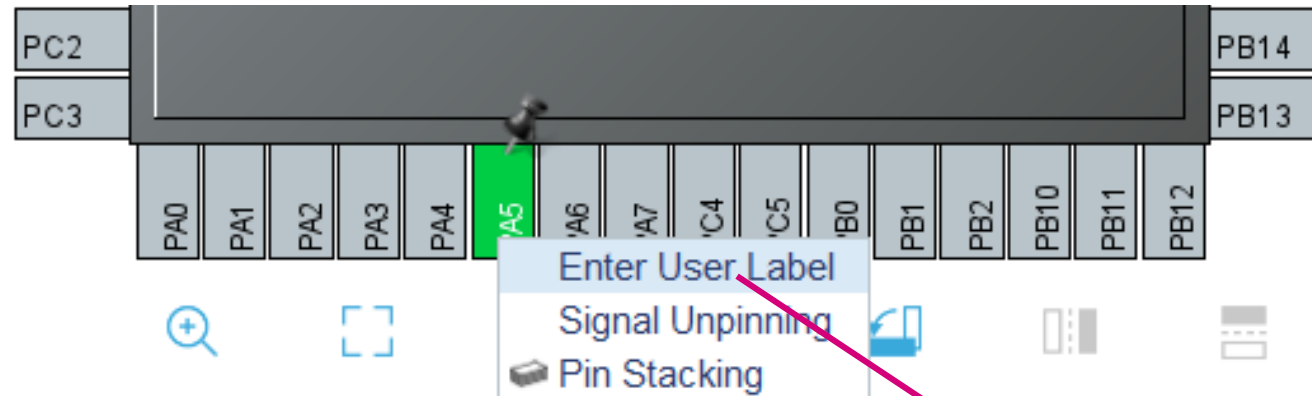
- In bigger packages you can use search field to locate the pin



Using configure PA5 as  
GPIO\_Output

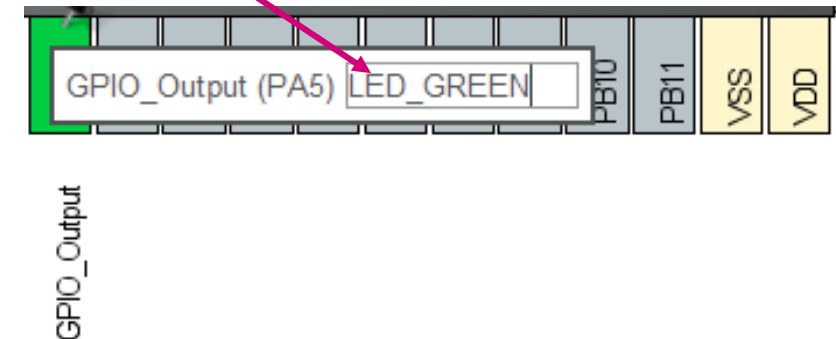
# Assign label to PA5

- Using select Enter User Label and insert LED\_GREEN label



## Hint:

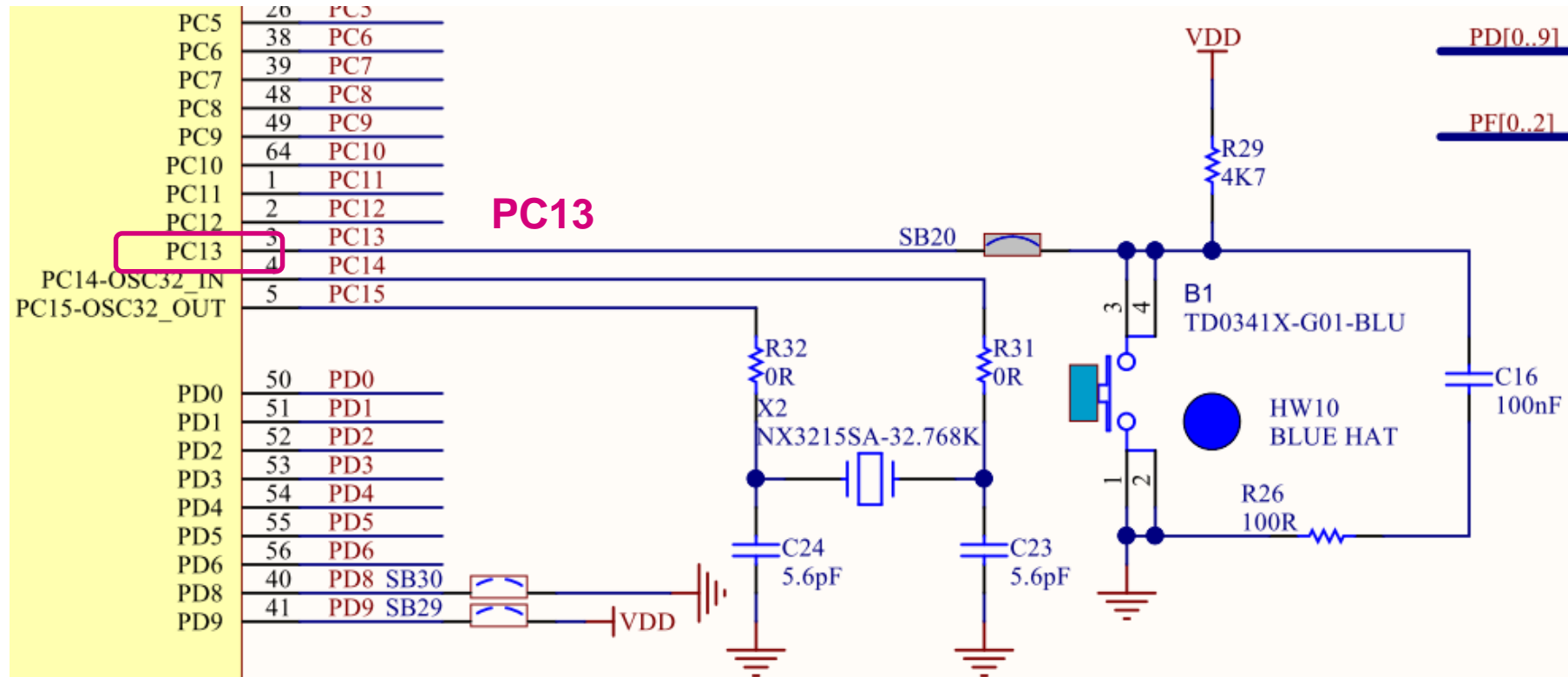
Labels are defined in **main.h** file within generated project (private defines section)



# Pins Configuration

external interrupt button

- In this example we are going to use one button - blue one (connected to PA5 as seen in the schematic below)



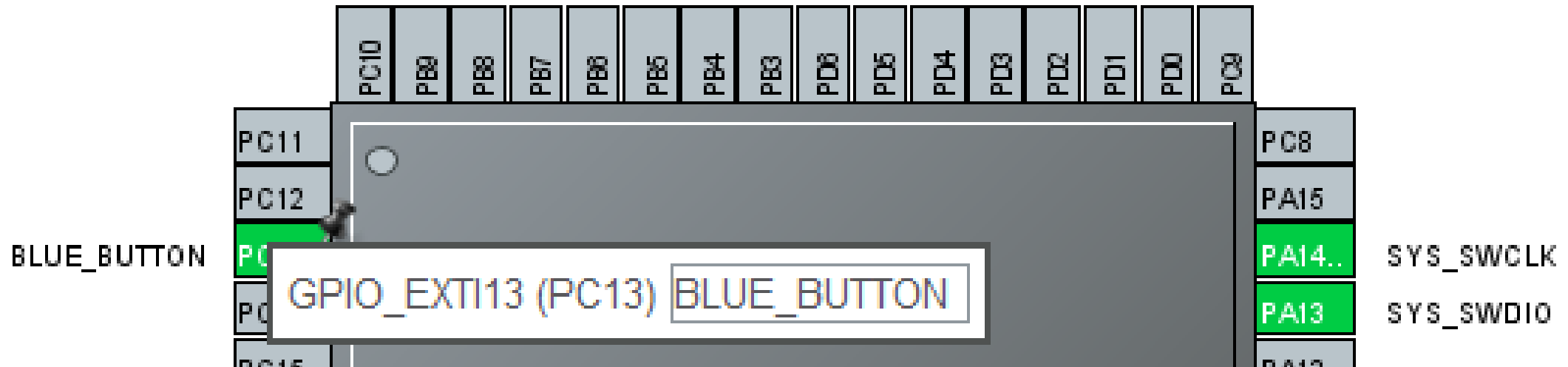
- 



# Assign label to PC13



- Using select Enter User Label and insert BLUE\_BUTTON label

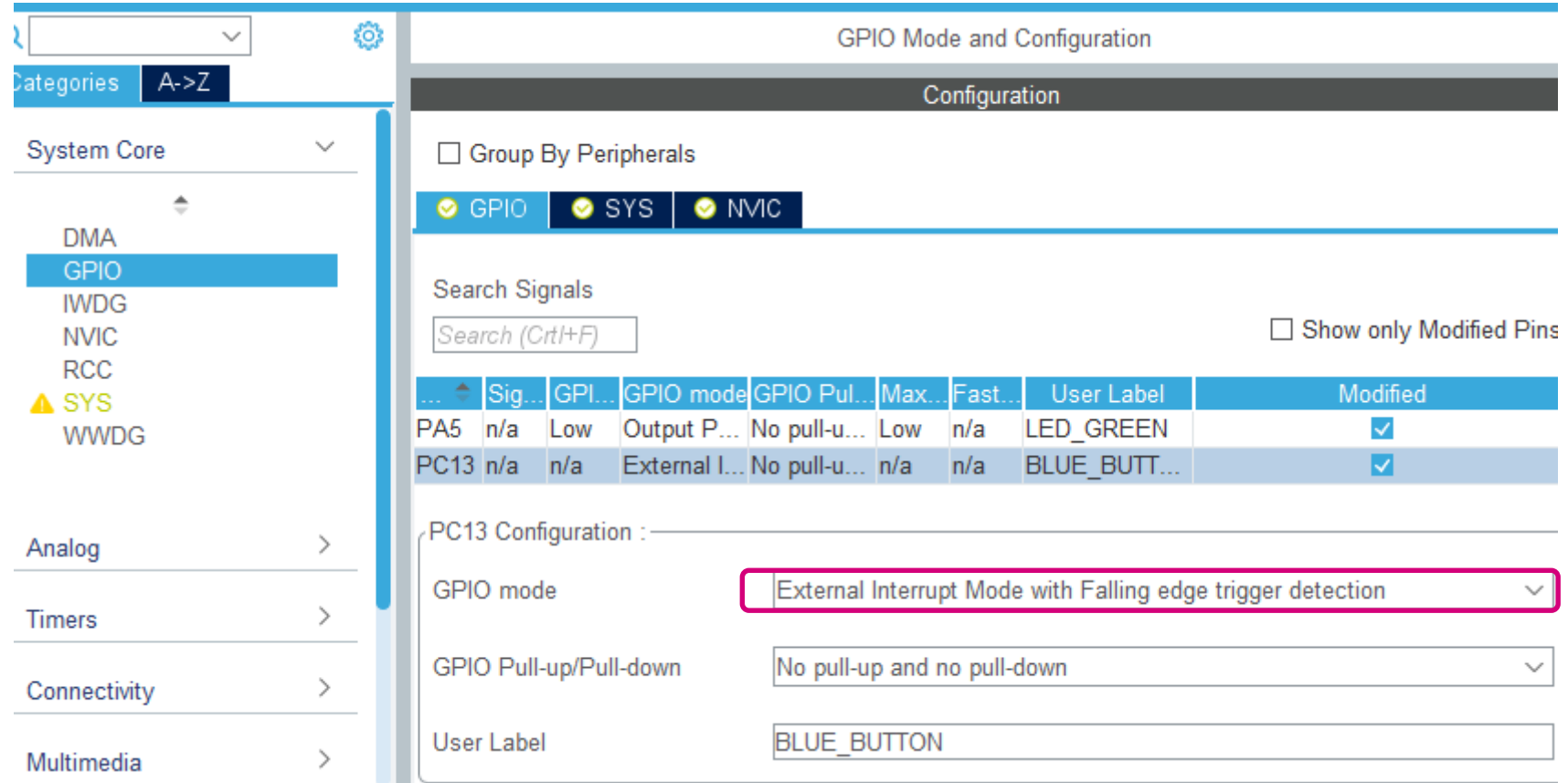


## Hint:

Labels are defined in **main.h** file within generated project (private defines section)

# Configure PC13

- Go to System Core -> GPIO, GPIO tab
- Select PC13 from the list
- Change GPIO mode to External Interrupt Mode with Falling edge trigger detection



The screenshot shows the STM32 CubeIDE interface for configuring GPIO. On the left, the 'System Core' menu is open, and 'GPIO' is selected. The main panel displays the 'GPIO Mode and Configuration' for PC13. The 'Configuration' tab is active, showing a table of GPIO pins and their configurations. The 'PC13' row is highlighted, showing it is configured as an 'External Interrupt Mode with Falling edge trigger detection'.

**GPIO Mode and Configuration**

**Configuration**

☐ Group By Peripherals

☒ GPIO ☒ SYS ☒ NVIC

Search Signals  ☐ Show only Modified Pins

...	Sig...	GPI...	GPIO mode	GPIO Pul...	Max...	Fast...	User Label	Modified
PA5	n/a	Low	Output P...	No pull-u...	Low	n/a	LED_GREEN	<input checked="" type="checkbox"/>
PC13	n/a	n/a	External I...	No pull-u...	n/a	n/a	BLUE_BUTT...	<input checked="" type="checkbox"/>

**PC13 Configuration :**

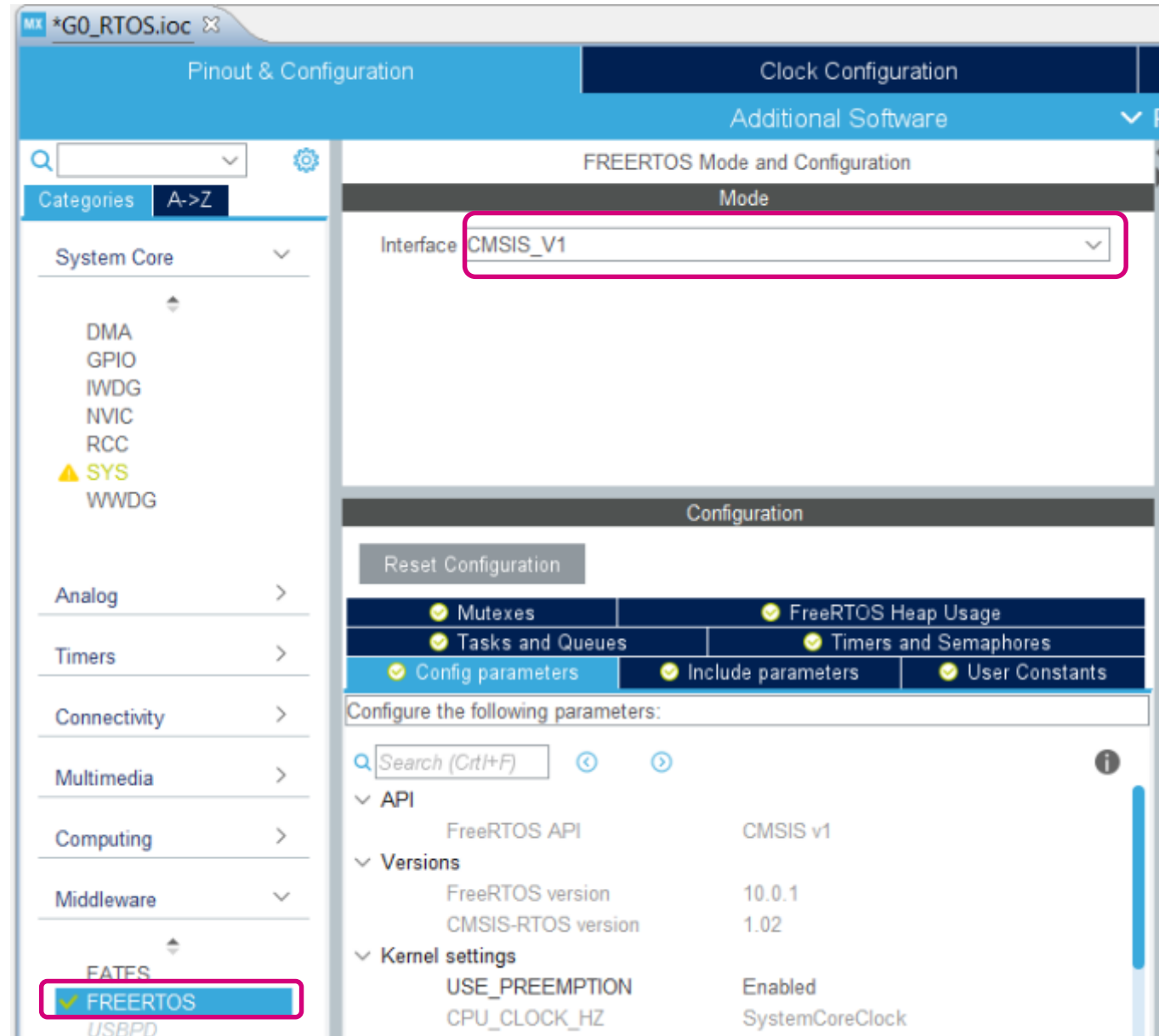
GPIO mode:

GPIO Pull-up/Pull-down:

User Label:

# Add FreeRTOS middleware

- Select Middleware -> FREERTOS from peripherals list
- Select its interface as CMSIS\_V1





# Configure FreeRTOS middleware

- Within FreeRTOS configuration modify  
LIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY from 3 to 2

FREERTOS Mode and Configuration

Mode

Interface

Configuration

<input checked="" type="checkbox"/> Timers and Semaphores	<input checked="" type="checkbox"/> Mutexes	<input checked="" type="checkbox"/> FreeRTOS Heap Usage
<input checked="" type="checkbox"/> User Constants	<input checked="" type="checkbox"/> Tasks and Queues	
<input checked="" type="checkbox"/> Config parameters	<input checked="" type="checkbox"/> Include parameters	

Configure the following parameters:

▼ Interrupt nesting behaviour configuration

LIBRARY\_LOWEST\_INTERRUPT\_ 3

LIBRARY\_MAX\_SYSCALL\_INTE. 2

# Add two tasks within FreeRTOS

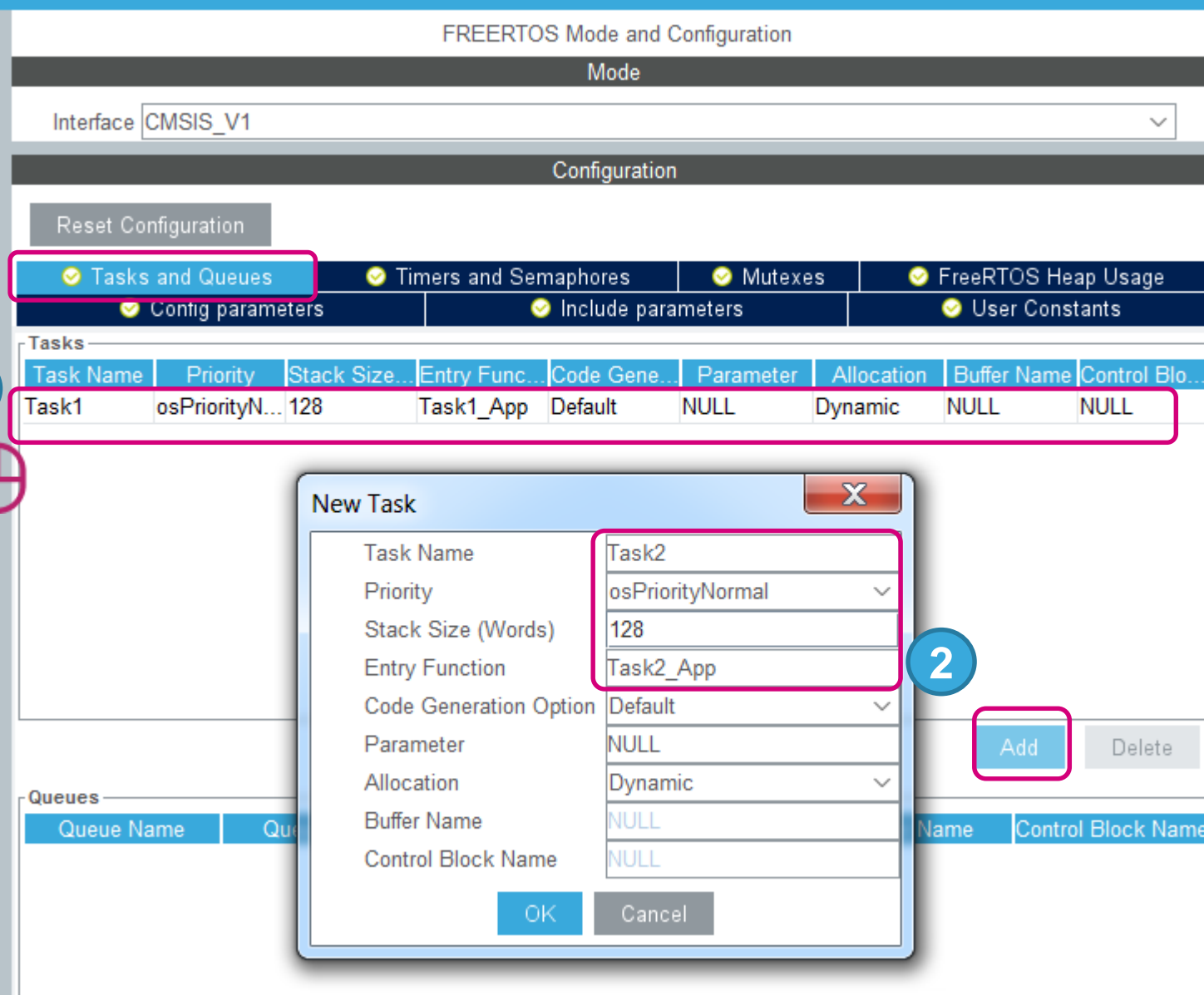
- Go to FreeRTOS Configuration, Tasks and Queues tab

## 1 Change defaultTask to:

- Task Name: **Task1**
- Priority: **osPriorityNormal**
- Stack Size: **128 Words**
- Entry Function: **Task1\_App**

## 2 Add new task:

- Task Name: **Task2**
- Priority: **osPriorityNormal**
- Stack Size: **128 Words**
- Entry Function: **Task2\_App**



The screenshot shows the 'FREERTOS Mode and Configuration' window in STM32 CubeIDE. The 'Interface' is set to 'CMSIS\_V1'. Under the 'Configuration' section, 'Tasks and Queues' is selected. The 'Tasks' table shows 'Task1' with priority 'osPriorityNormal', stack size '128', and entry function 'Task1\_App'. A 'New Task' dialog is open, showing 'Task2' with priority 'osPriorityNormal', stack size '128', and entry function 'Task2\_App'. The 'Add' button is highlighted.

**1** Change defaultTask to:

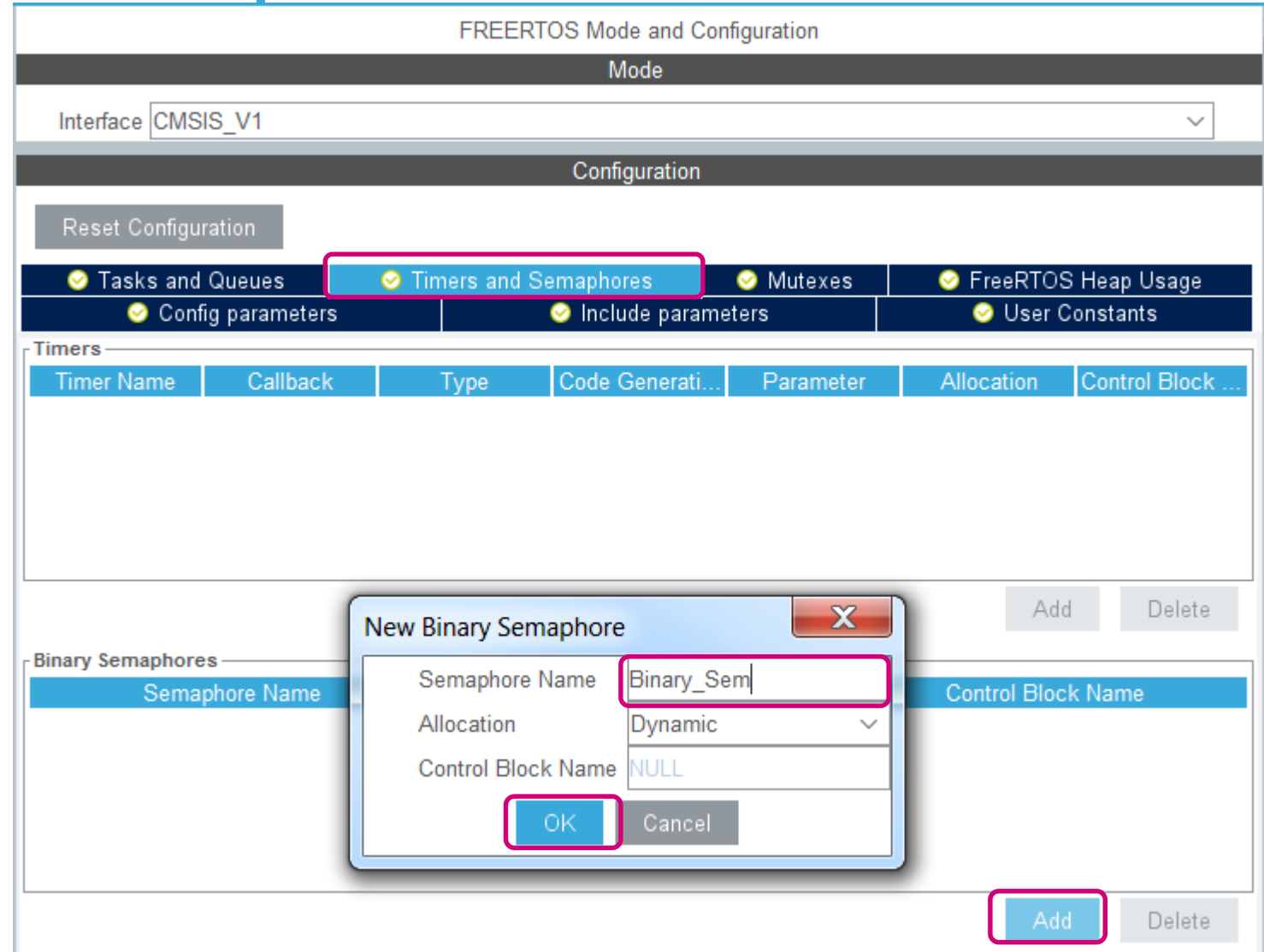
- Task Name: **Task1**
- Priority: **osPriorityNormal**
- Stack Size: **128 Words**
- Entry Function: **Task1\_App**

**2** Add new task:

- Task Name: **Task2**
- Priority: **osPriorityNormal**
- Stack Size: **128 Words**
- Entry Function: **Task2\_App**

# Add binary semaphore within FreeRTOS

- Go to FreeRTOS Configuration, Timers and Semaphores tab
- Add new Binary Semaphore:
  - Semaphore Name: **Binary\_Sem**



# Verify how much memory OS has used

- Go to FreeRTOS Configuration, FreeRTOS Heap USAGE tab
- Check how many bytes is used by tasks, semaphores.

FREERTOS Mode and Configuration

Mode

Interface

Configuration

Reset Configuration

☒ Tasks and Queues   
 ☒ Timers and Semaphores   
 ☒ Mutexes   
 ☒ FreeRTOS Heap Usage

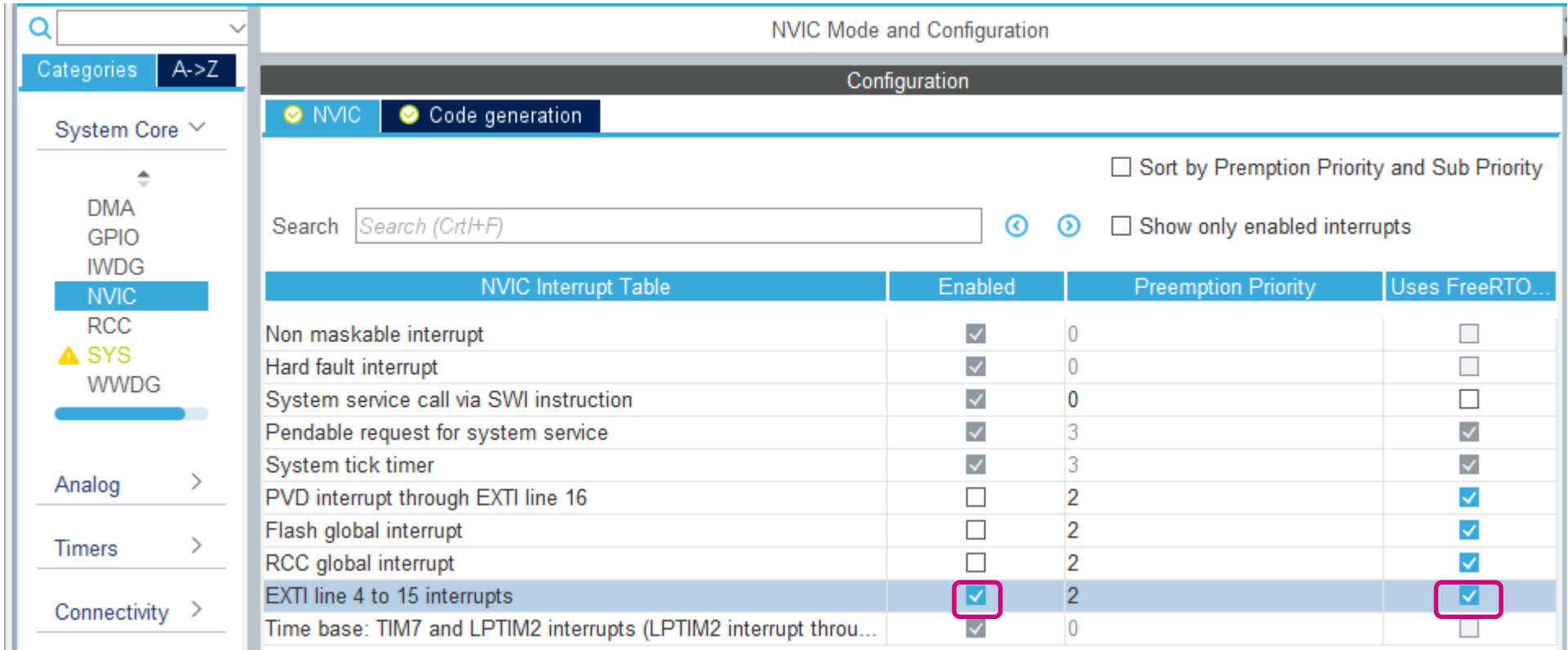
☒ Config parameters   
 ☒ Include parameters   
 ☒ User Constants

Summary

HEAP STILL AVAILABLE	1112 Bytes
TOTAL HEAP USED	1960 Bytes
Total amount for tasks	1872 Bytes
Total amount for queues	0 Bytes
Total amount for timers	0 Bytes
Total amount for mutexes and semaphores	88 Bytes
FreeRTOS tasks	
Idle task (FreeRTOS internal)	624 Bytes
Task1	624 Bytes
Task2	624 Bytes
FreeRTOS mutexes ans semaphores	
Binary_Sem	88 Bytes

# Configure NVIC

- Go to System Core -> NVIC
- Check whether EXTI line 4 to 15 interrupts is marked as “Uses FreeRTOS functions and is enabled



NVIC Mode and Configuration

Configuration

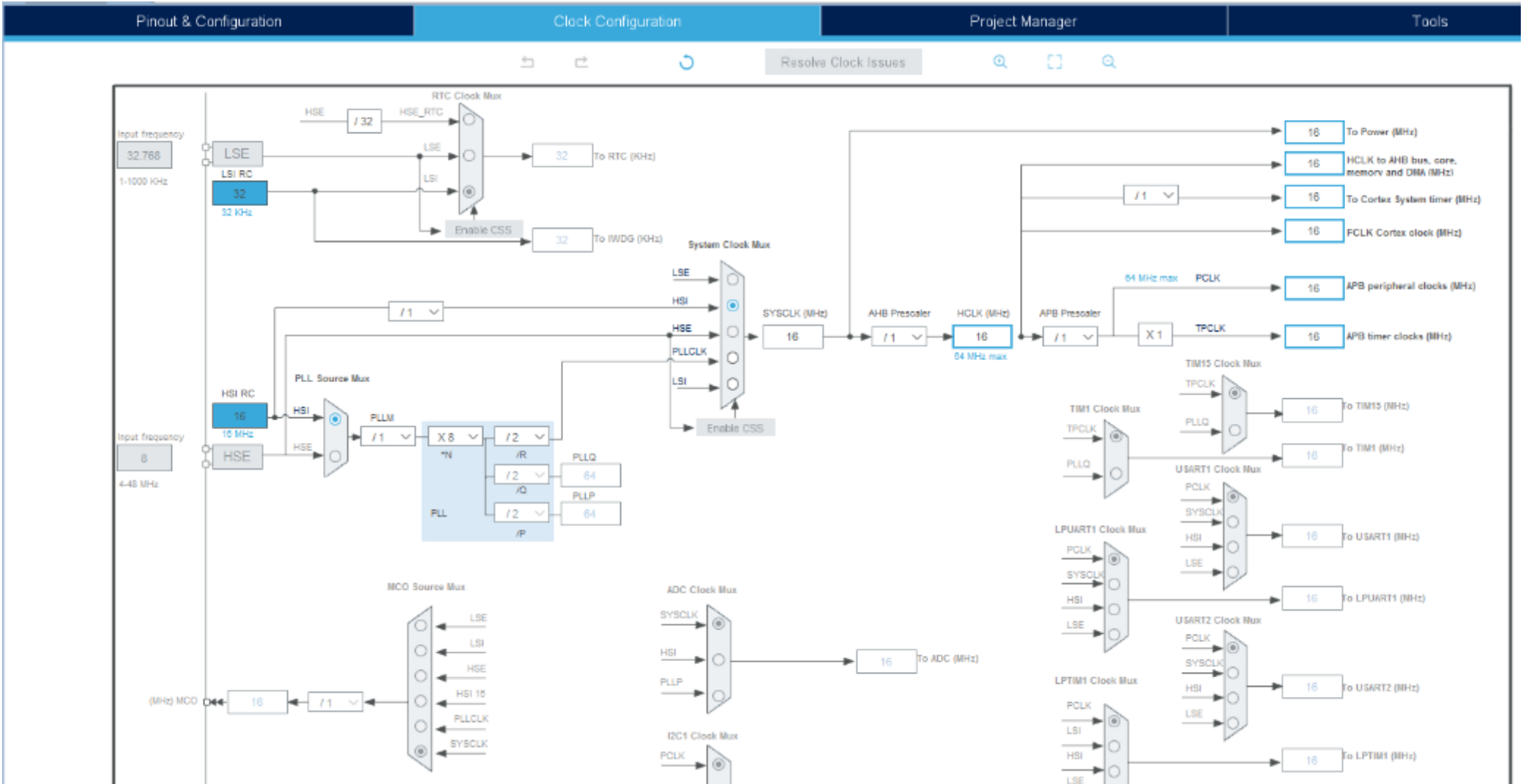
☒ NVIC ☒ Code generation

☐ Sort by Preemption Priority and Sub Priority

Search    ☐ Show only enabled interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Uses FreeRTOS...
Non maskable interrupt	<input checked="" type="checkbox"/>	0	<input type="checkbox"/>
Hard fault interrupt	<input checked="" type="checkbox"/>	0	<input type="checkbox"/>
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	<input type="checkbox"/>
Pendable request for system service	<input checked="" type="checkbox"/>	3	<input checked="" type="checkbox"/>
System tick timer	<input checked="" type="checkbox"/>	3	<input checked="" type="checkbox"/>
PVD interrupt through EXTI line 16	<input type="checkbox"/>	2	<input checked="" type="checkbox"/>
Flash global interrupt	<input type="checkbox"/>	2	<input checked="" type="checkbox"/>
RCC global interrupt	<input type="checkbox"/>	2	<input checked="" type="checkbox"/>
EXTI line 4 to 15 interrupts	<input checked="" type="checkbox"/>	2	<input checked="" type="checkbox"/>
Time base: TIM7 and LPTIM2 interrupts (LPTIM2 interrupt throu...	<input checked="" type="checkbox"/>	0	<input type="checkbox"/>

# Default clock configuration – no change

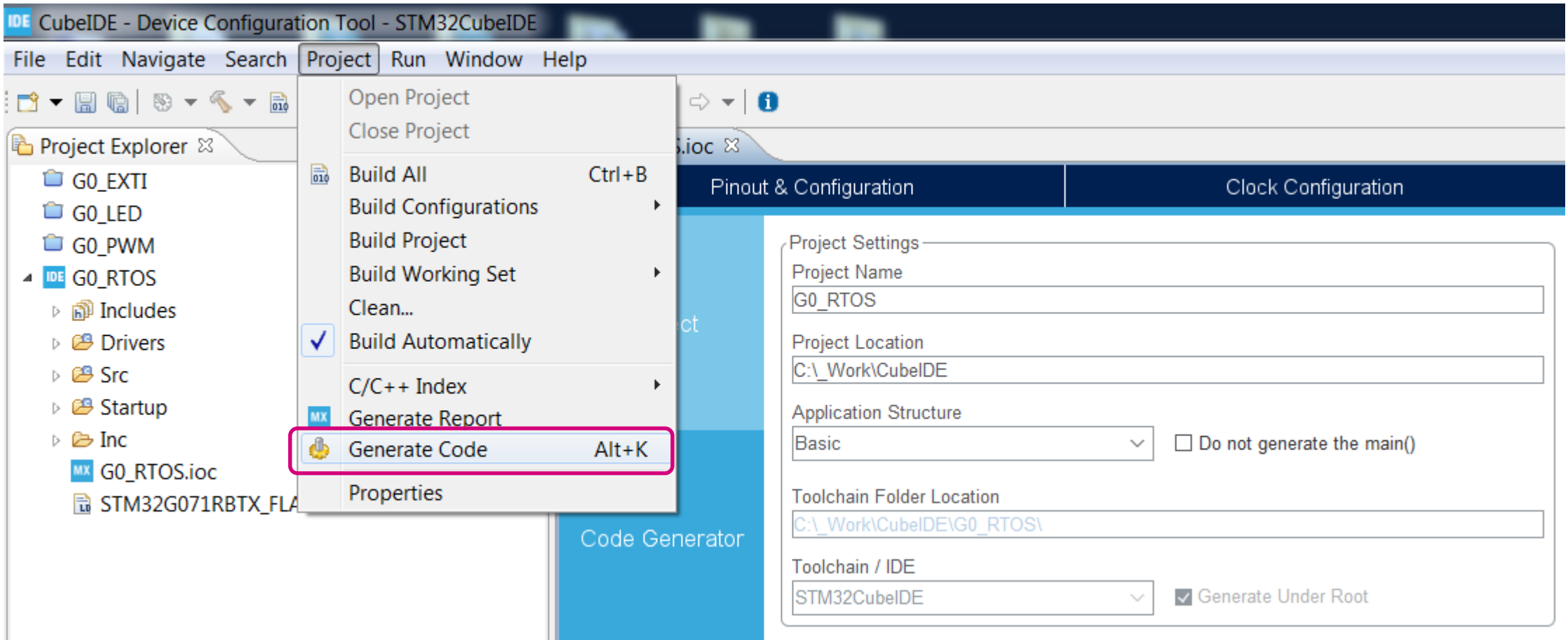


# Basic project settings – no change

Pinout & Configuration	Clock Configuration	Project Manager
Project	<p><b>Project Settings</b></p> <p>Project Name G0_EXTI</p> <p>Project Location C:\_Work\CubeIDE</p> <p>Application Structure Basic <input type="checkbox"/> Do not generate the main()</p> <p>Toolchain Folder Location C:\_Work\CubeIDE\G0_EXTI\</p> <p>Toolchain / IDE STM32CubeIDE <input checked="" type="checkbox"/> Generate Under Root</p>	
Code Generator		
Advanced Settings	<p><b>Linker Settings</b></p> <p>Minimum Heap Size <input type="text" value="0x200"/></p> <p>Minimum Stack Size <input type="text" value="0x400"/></p>	
	<p><b>Mcu and Firmware Package</b></p> <p>Mcu Reference STM32G071RBTx</p> <p>Firmware Package Name and Version STM32Cube FW_G0 V1.2.0</p> <p><input type="text" value="C:/_Work/_CubeMX/STM32Cube_FW_G0_V1.2.0"/> <input type="button" value="Browse"/></p>	

# Code generation

- It is necessary to add to an empty template our project we have just prepared



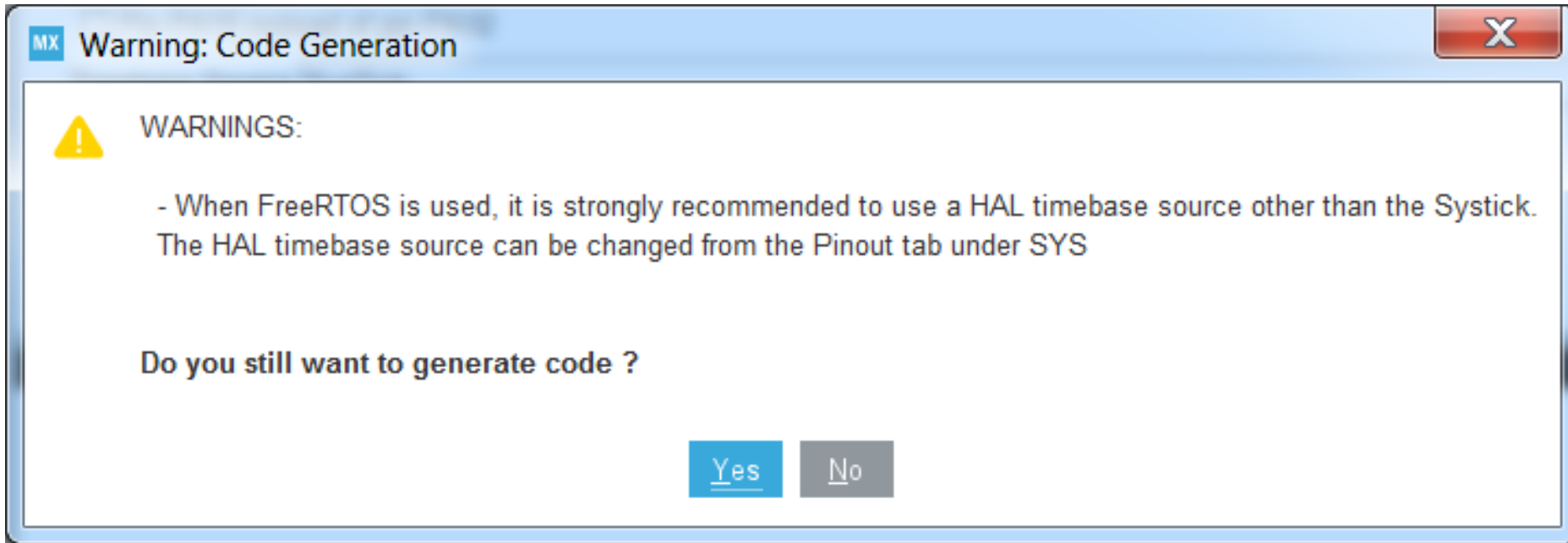
## Hint:

It is enough to save the configuration project (.ioc file) to launch project generation



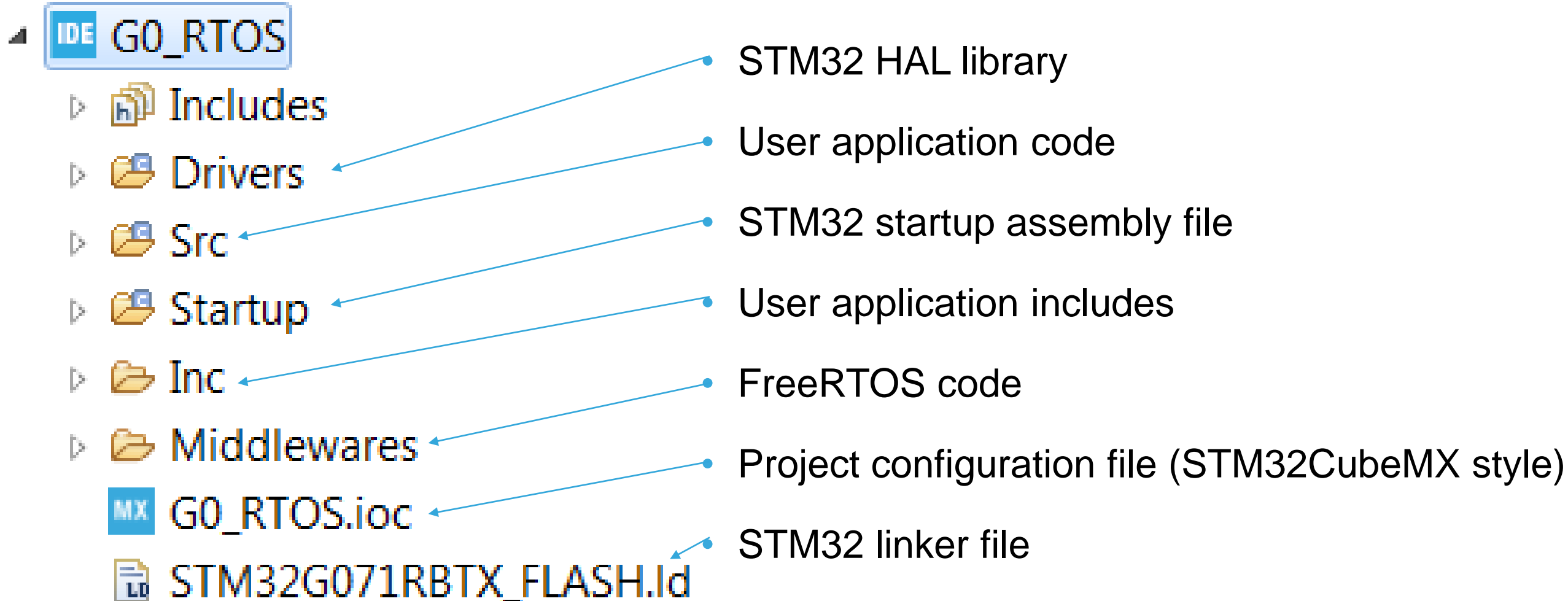
# In case we forget about timebase timer

- If we will not change timebase timer from SysTick (reserved by FreeRTOS) to any other timer, there would be a warning generated before code generation



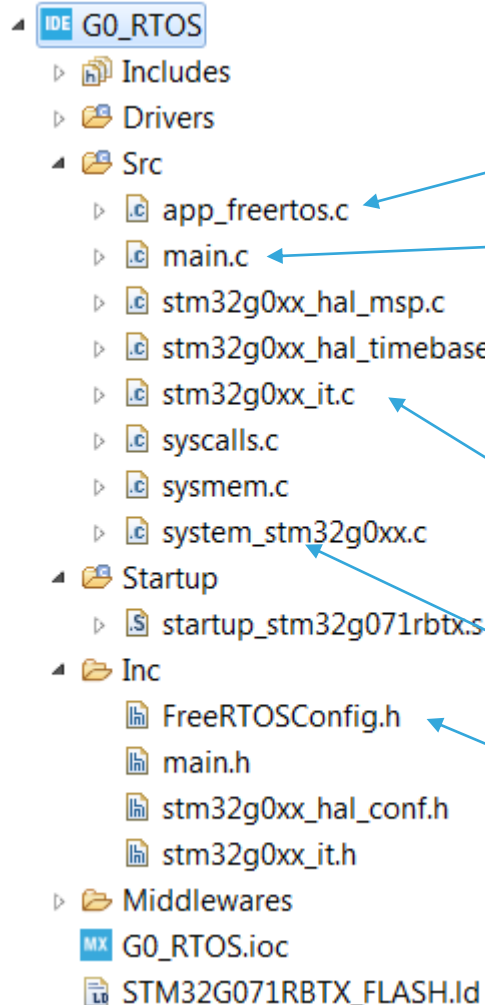
# FreeRTOS based application files structure

- Once we generate the project, we should see the following files structure



# User application files structure

- Once we generate the project, we should see the following files structure



FreeRTOS application code (like hooks). It replaces freertos.c file from previous implementations

Main user application source file

Main user application HW configuration file (HAL origin)

Timebase routines using selected timer (TIM7 in our case) used by HAL library for all timing functions

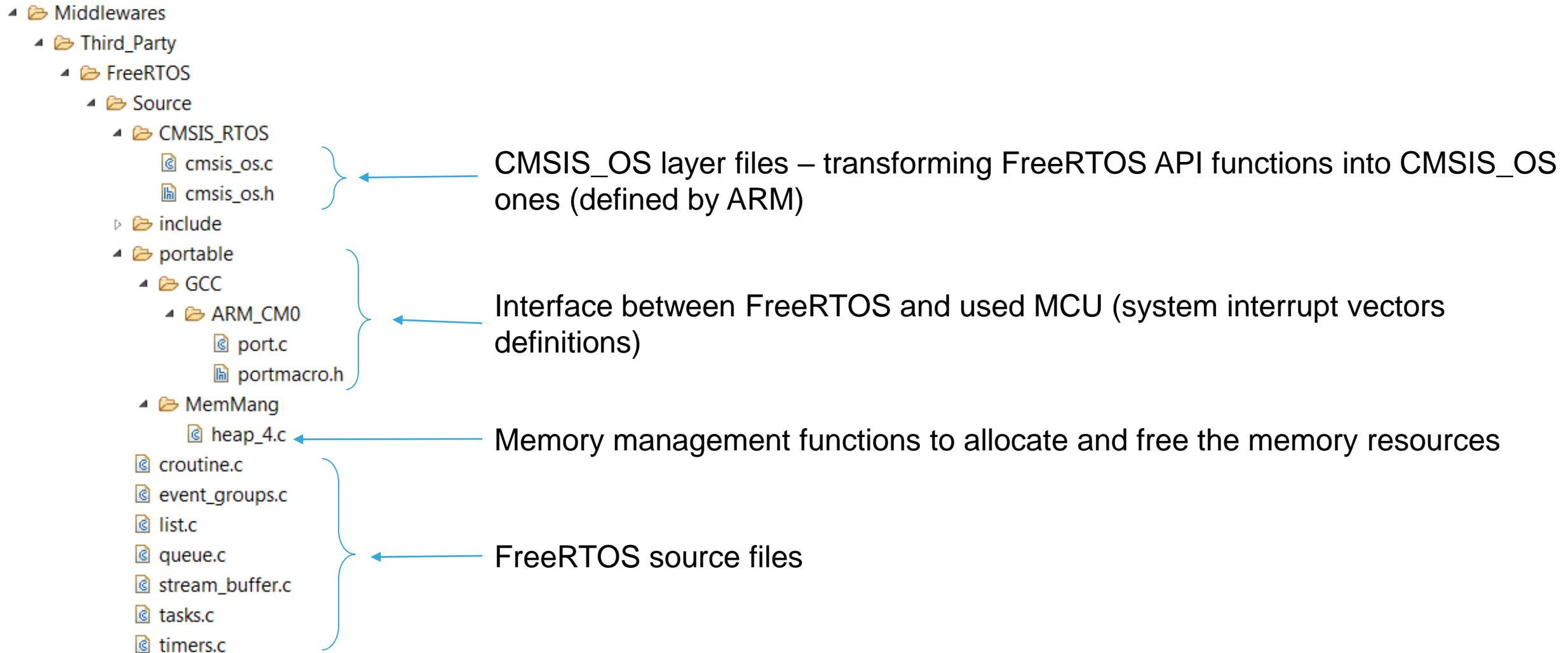
Interrupt routines source file

Initial MCU configuration file (SystemInit() location)

Main FreeRTOS configuration file

# FreeRTOS files structure

- Once we generate the project, we should see the following files structure



- Turn ON GREEN LED within Task1 application code (Task1\_App() function), then go to Blocked state for 1 seconds

```
void Task1_App(void const * argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
        osDelay(1000);
    }
    /* USER CODE END 5 */
}
```

- Turn OFF GREEN LED within Task2 application code (Task2\_App() function), then go to Blocked state for 1 second

```
void Task2_App(void const * argument)
{
    /* USER CODE BEGIN Task2_App */
    /* Infinite loop */
    for(;;)
    {
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
        osDelay(1000);
    }
    /* USER CODE END Task2_App */
}
```

# Application run

- Let's compile the code and run the debug session
- As a result we could see either green LED always ON or OFF as both tasks are active for a very short time one just after the other

- Wait for Binary\_Sem semaphore (being in blocked state)

```
void Task1_App(void const * argument)
{

    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
        osSemaphoreWait(Binary_SemHandle, osWaitForever);
    }
    /* USER CODE END 5 */
}
```



- Release Binary\_Sem on each BLUE\_BUTTON press

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Falling_Callback(uint16_t GPIO_Pin)
{
    if(BLUE_BUTTON_Pin == GPIO_Pin)
        osSemaphoreRelease(Binary_SemHandle);
}
/* USER CODE END 4 */
```

# Application run


- Let's compile the code and run the debug session
- As a result on each blue button press we are unblocking Task1 (GREEN LED turned on), then it depends on which stage is Task2 (turning LED off) for how long our LED will be in ON state.

# Thank you



 [/STM32](https://www.facebook.com/STM32)

 [@ST\\_World](https://twitter.com/@ST_World)

 [community.st.com](https://community.st.com)