STM32CubeIDE basics

CMSIS_OS lab: Basic project on FreeRTOS

life.augmented

STM32 CubeIDE

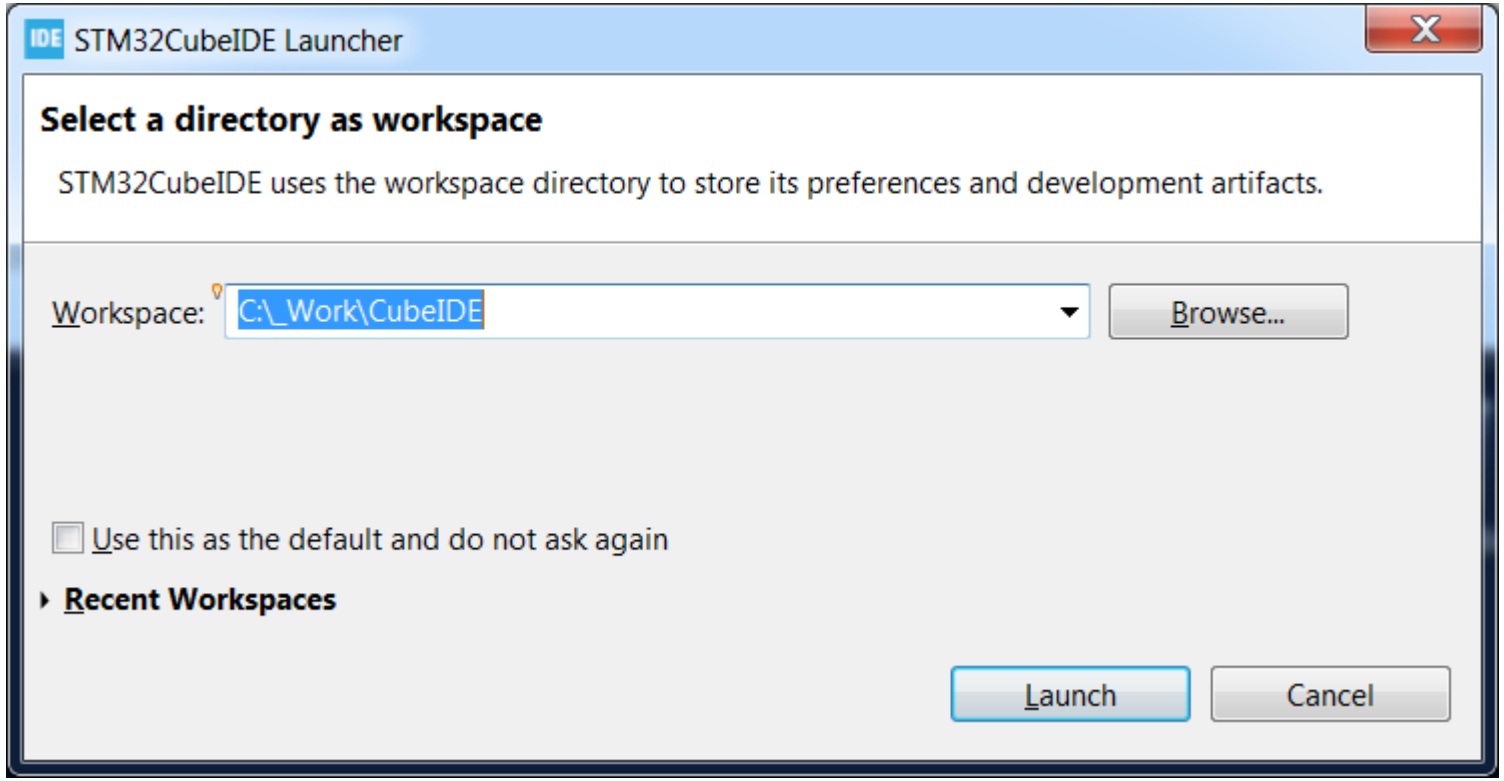
Lab: Basic project on FreeRTOS

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)

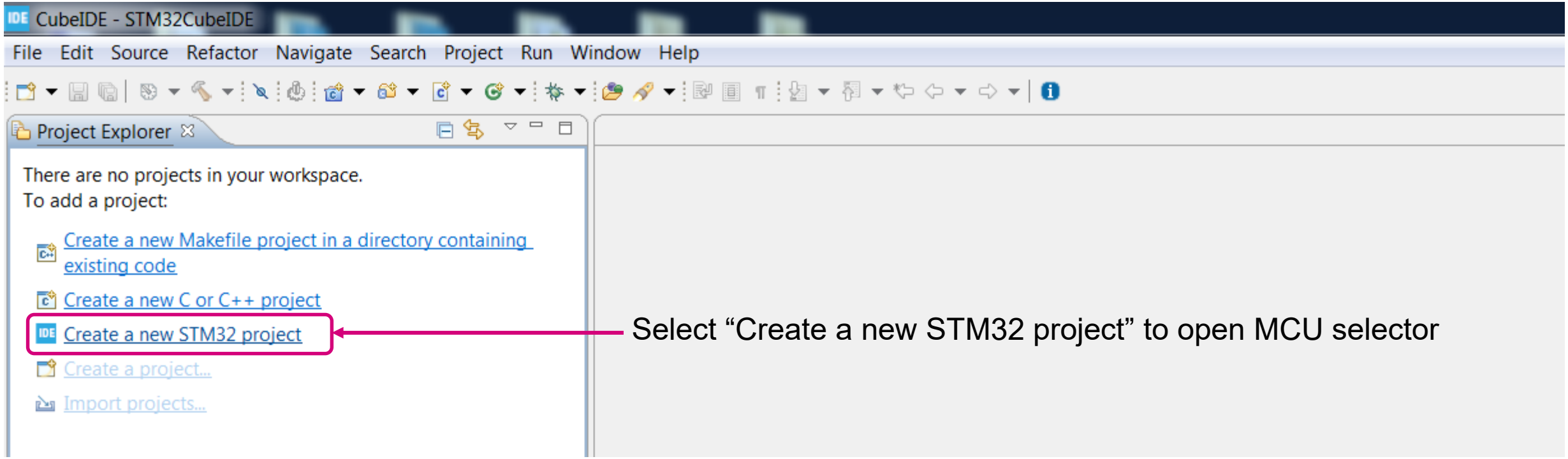
Start a new workspace

- Run STM32CubeIDE
- Select a folder to store a workspace



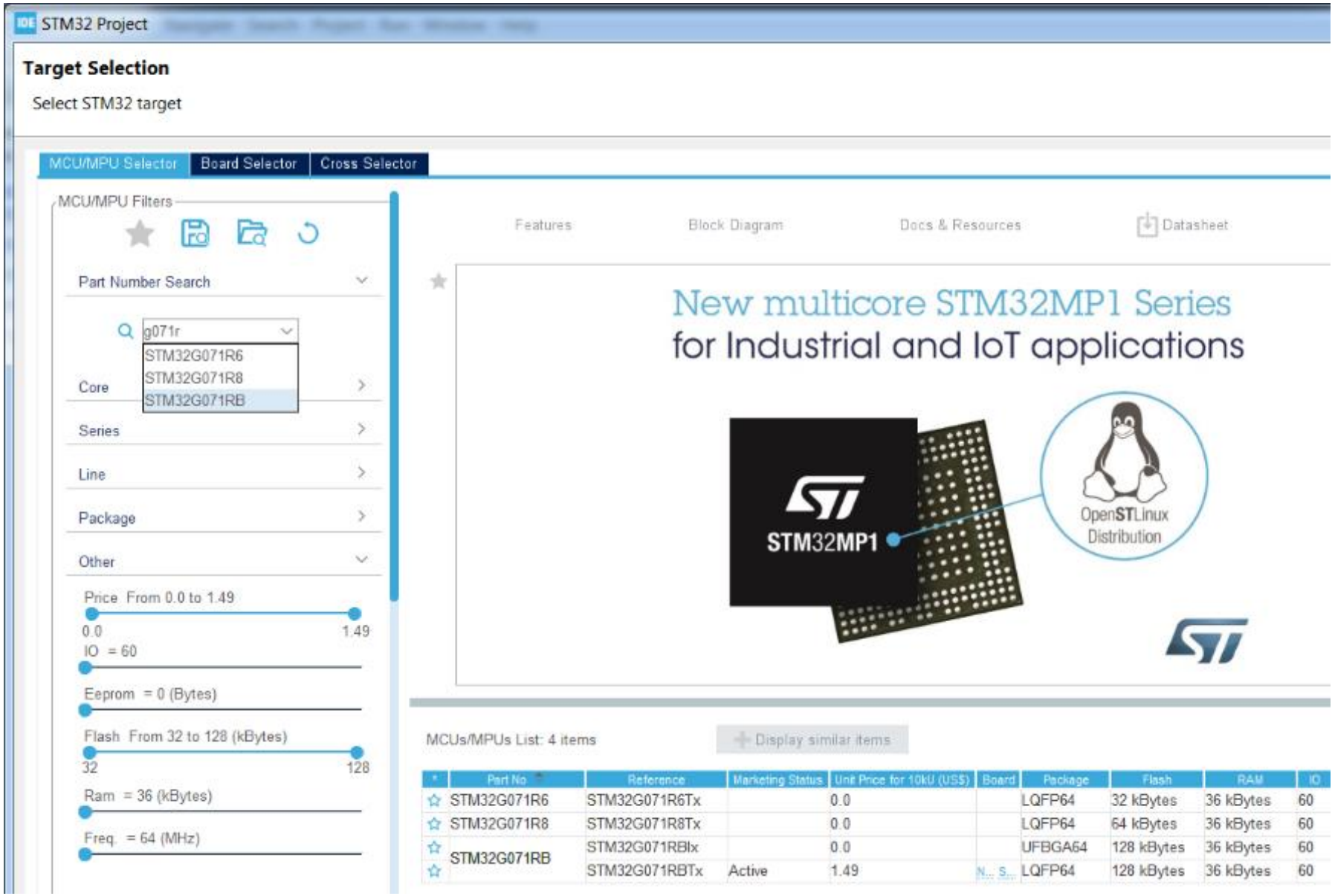
Create a new project

- Click on “Create a new STM32 project”



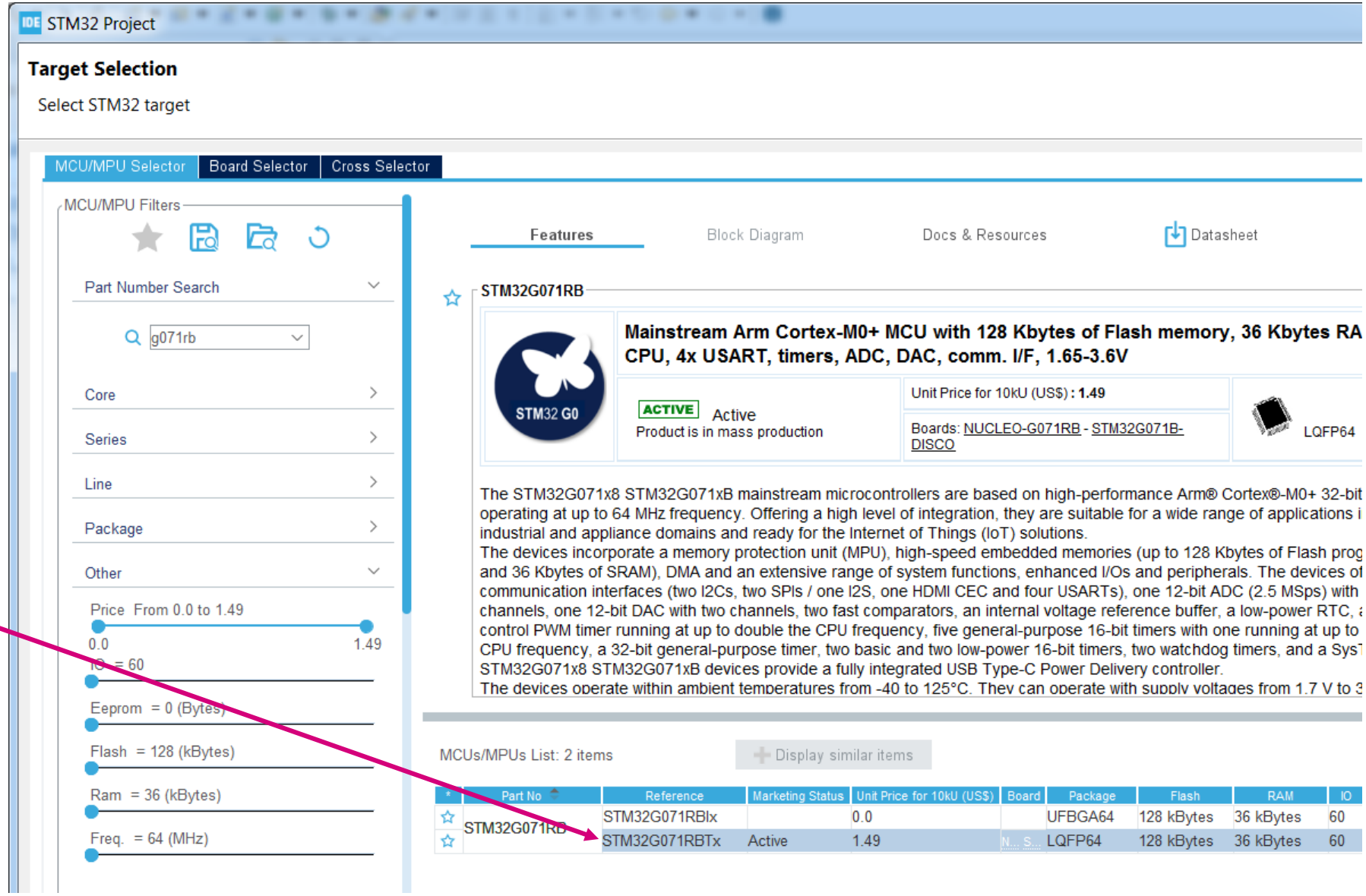
Select target MCU: STM32G071RBTx

- We will use STM32G071RBTx MCU



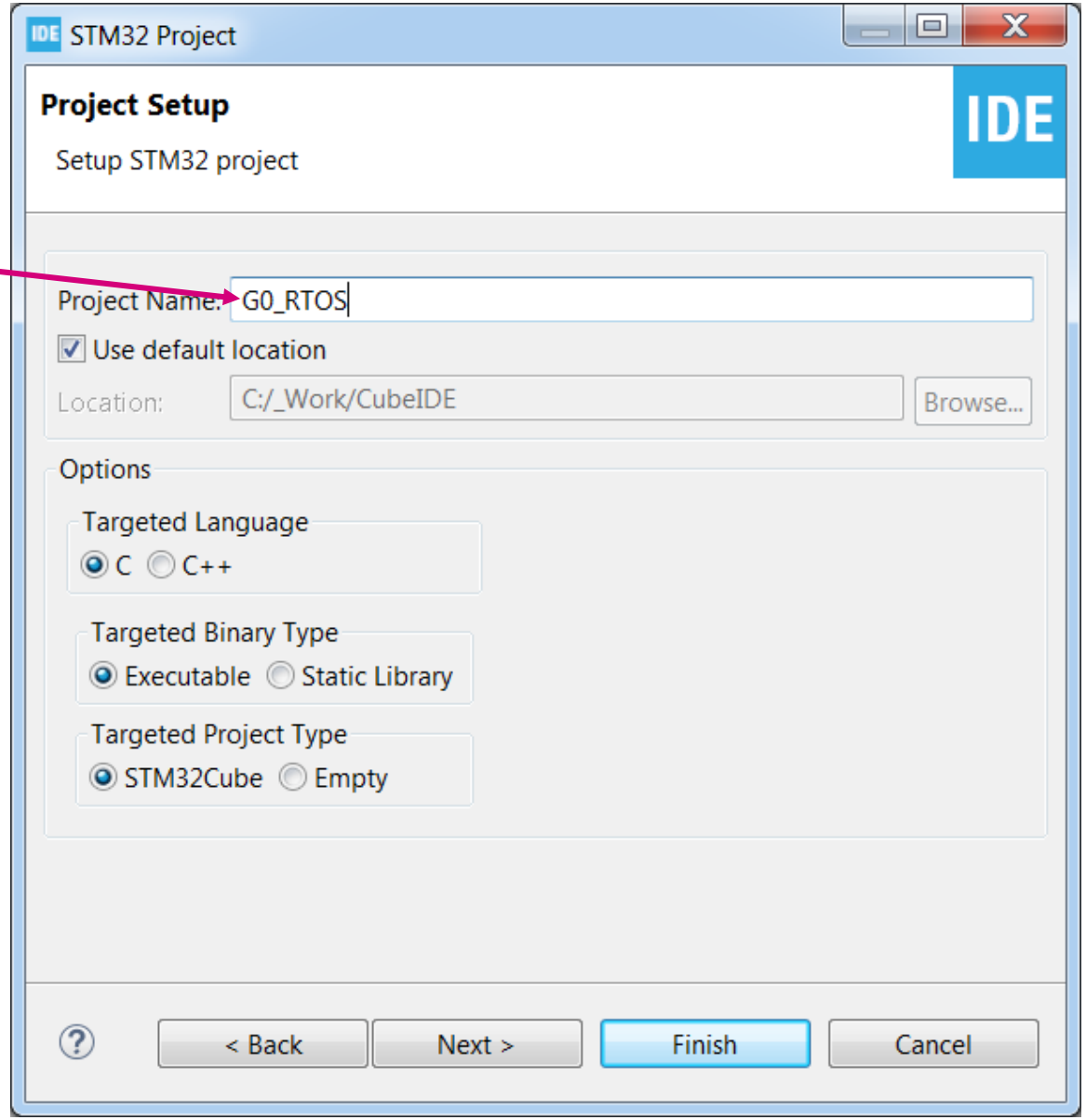
Select target MCU: STM32G071RBTx

- 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



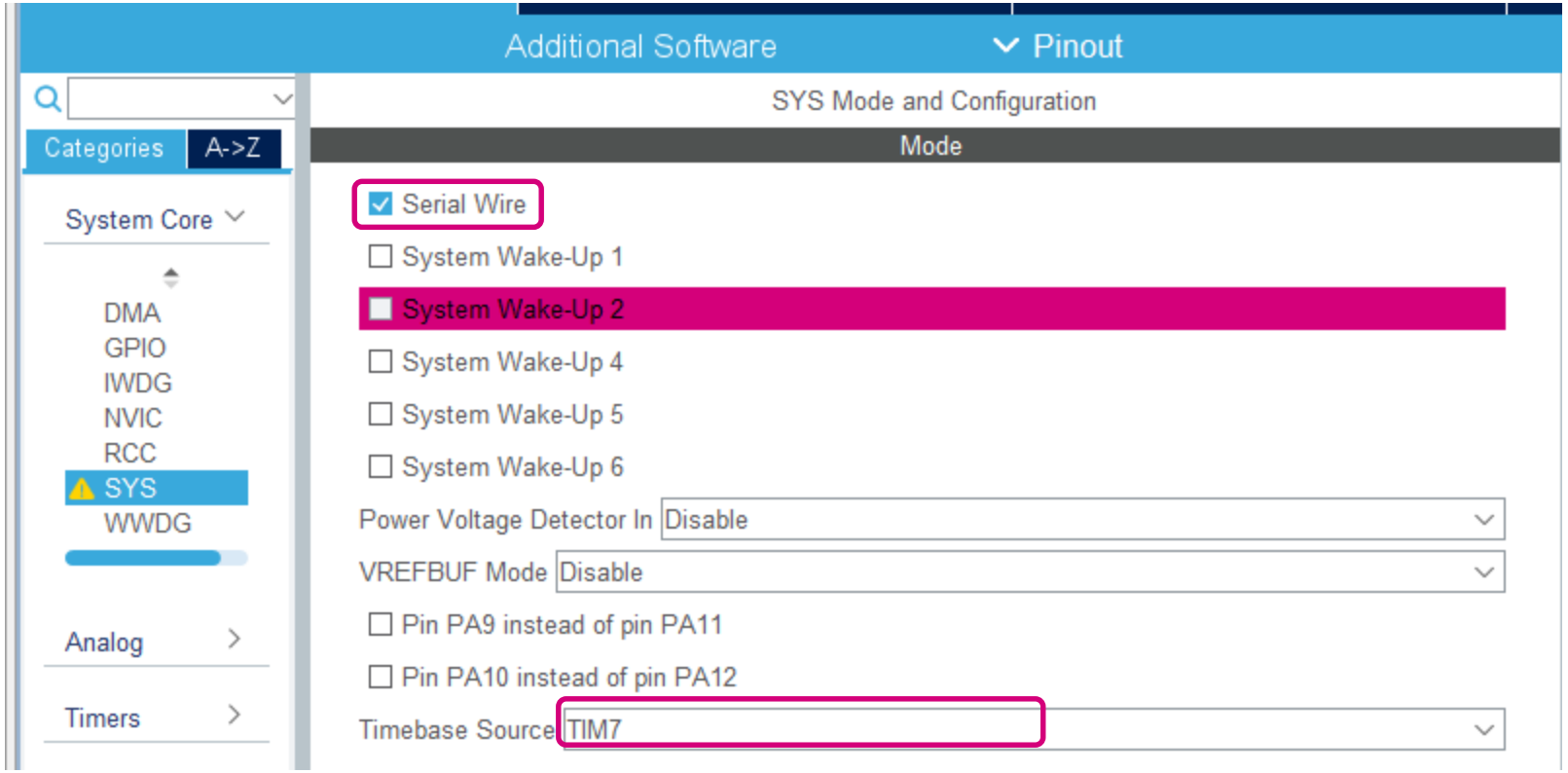
Enter project name

- 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)



Enabling Serial Wire debug interface

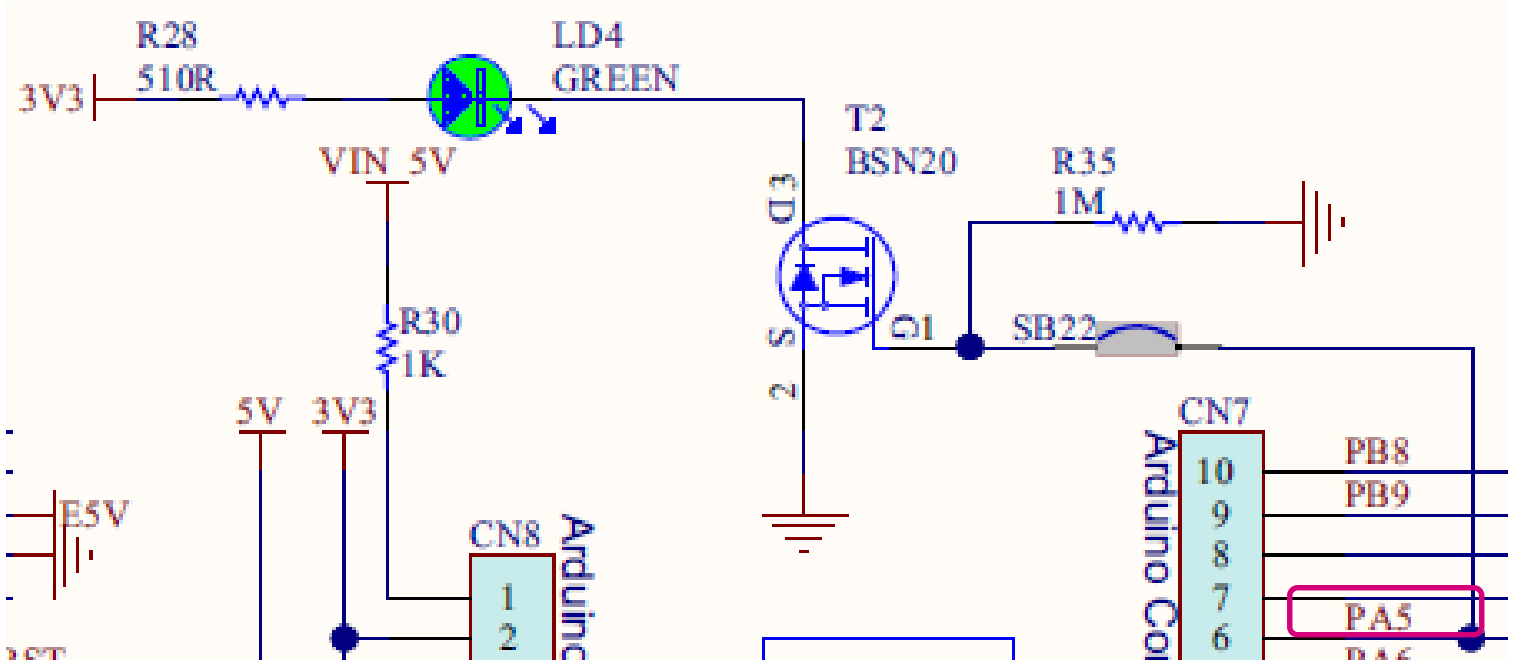
- Select “Serial Wire” from System Core -> SYS peripheral group
- As a result PA13 and PA14 will be assigned to SWD interface
- Change a timebase source (used by HAL library functions) from SysTick to other timer (i.e. TIM7) as SysTick will be used by operating system



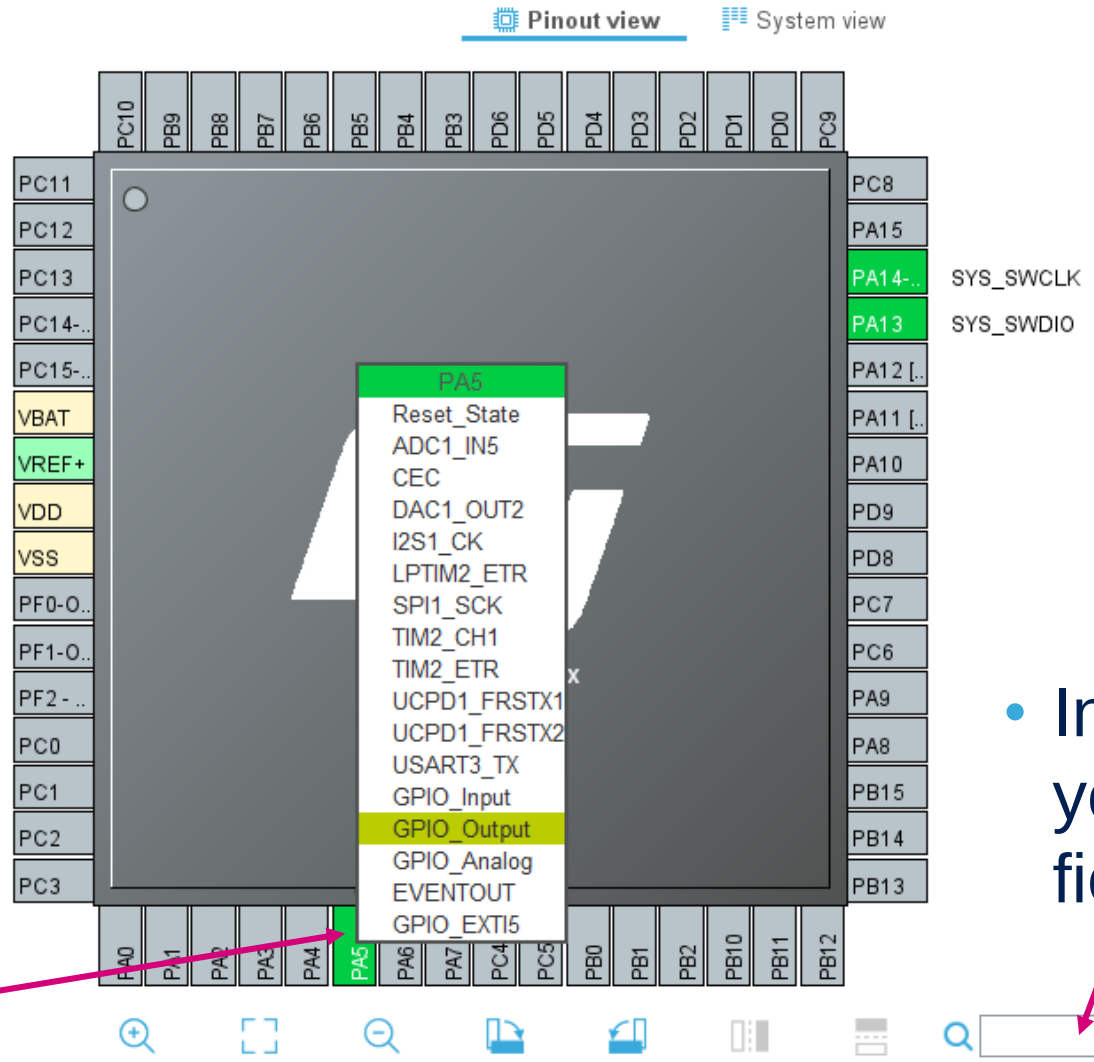
Pin Configuration

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

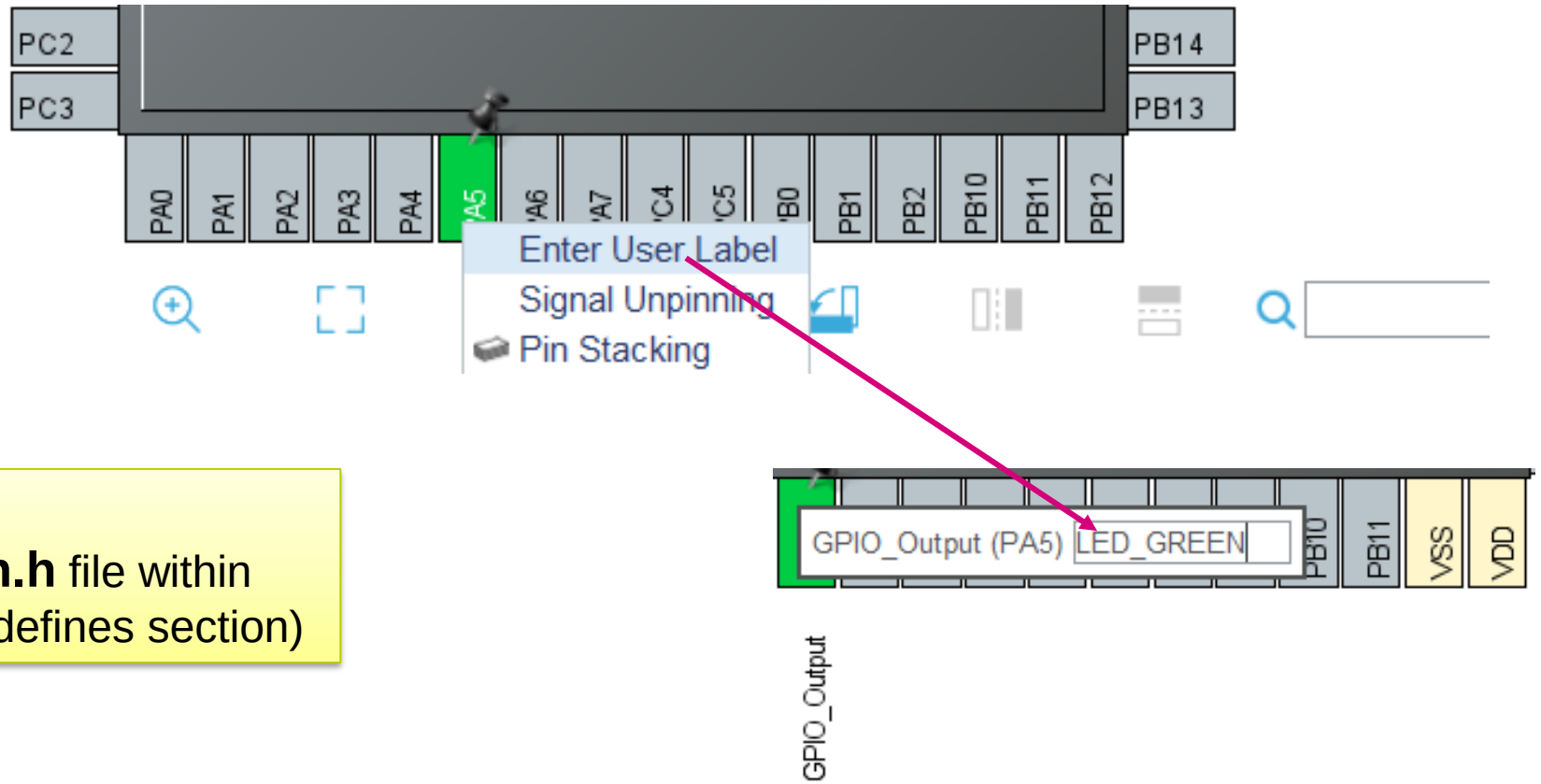


- Using configure PA5 as GPIO_Output

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

Assign label to PA5

- Using select Enter User Label and insert LED_GREEN label

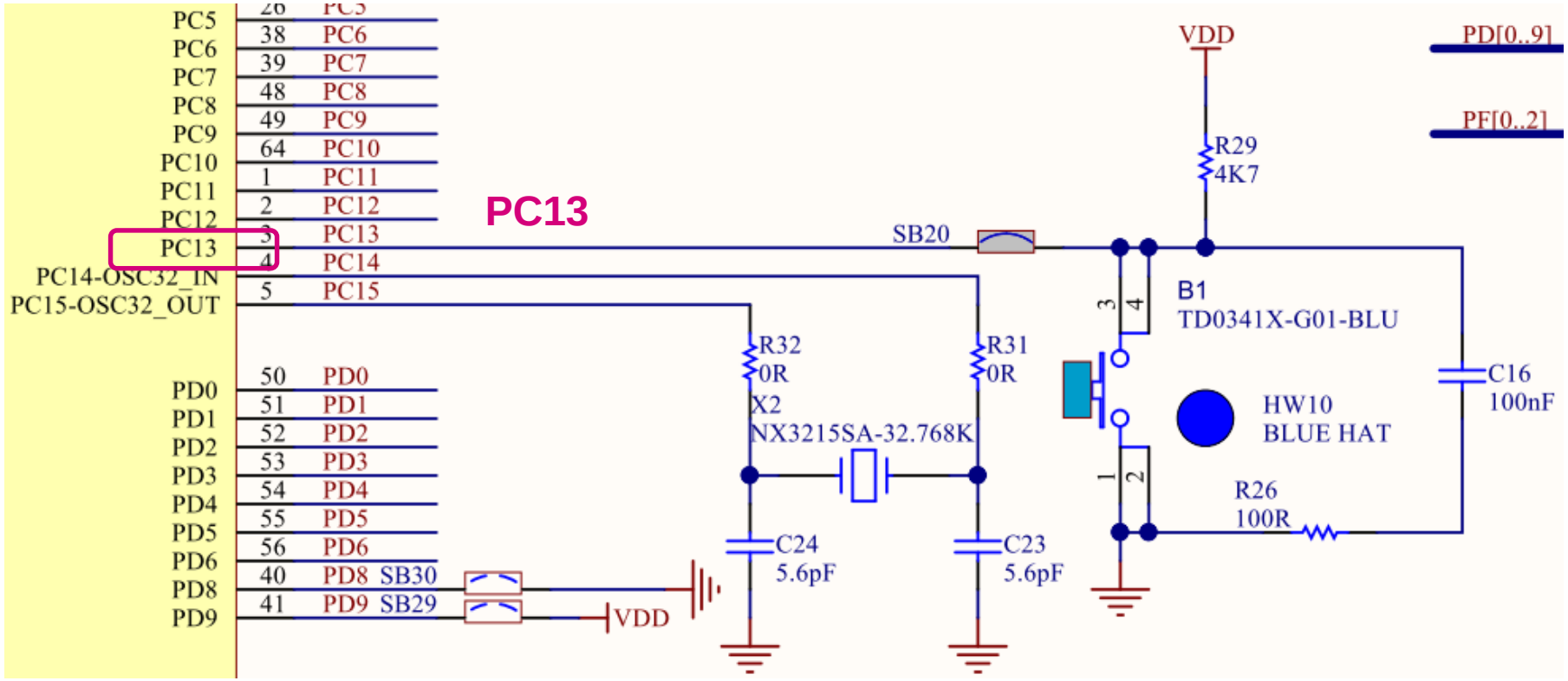


Hint:
Labels are defined in **main.h** file within generated project (private defines section)

Pins Configuration

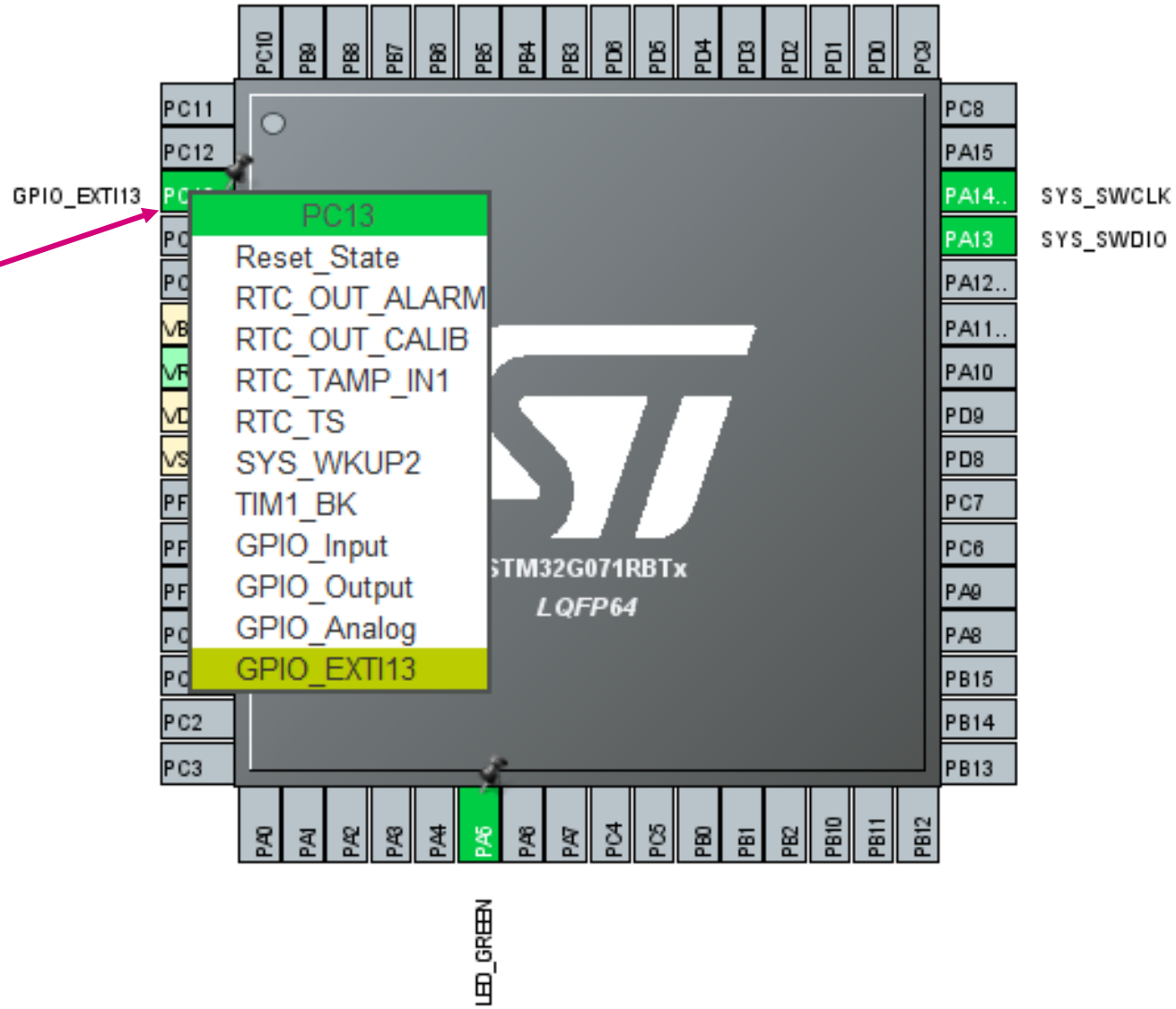
external interrupt button

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



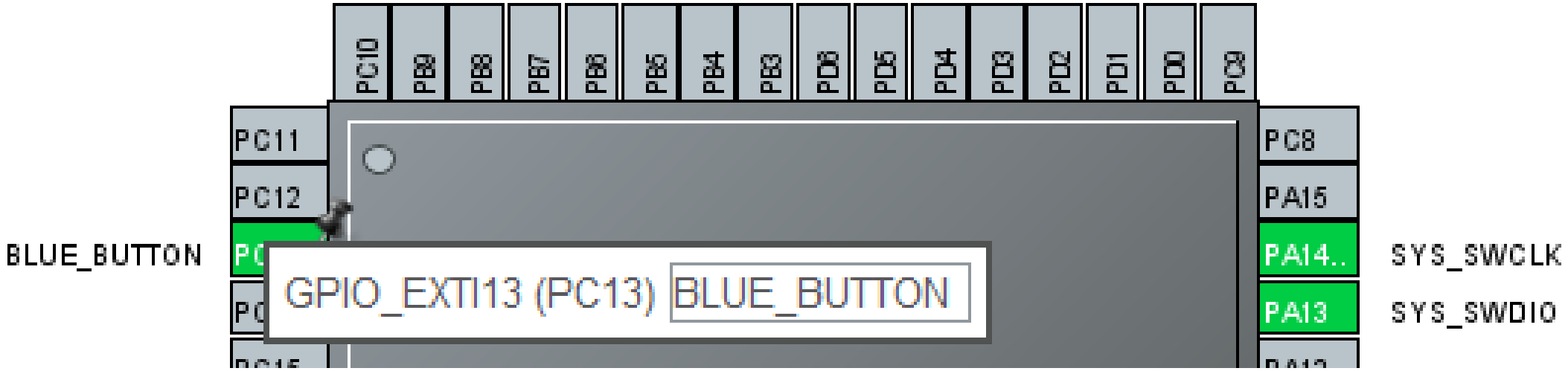
Configuring PC13 as GPIO_EXTI13

- Using configure PC13 as GPIO_EXTI13



Assign label to PC13

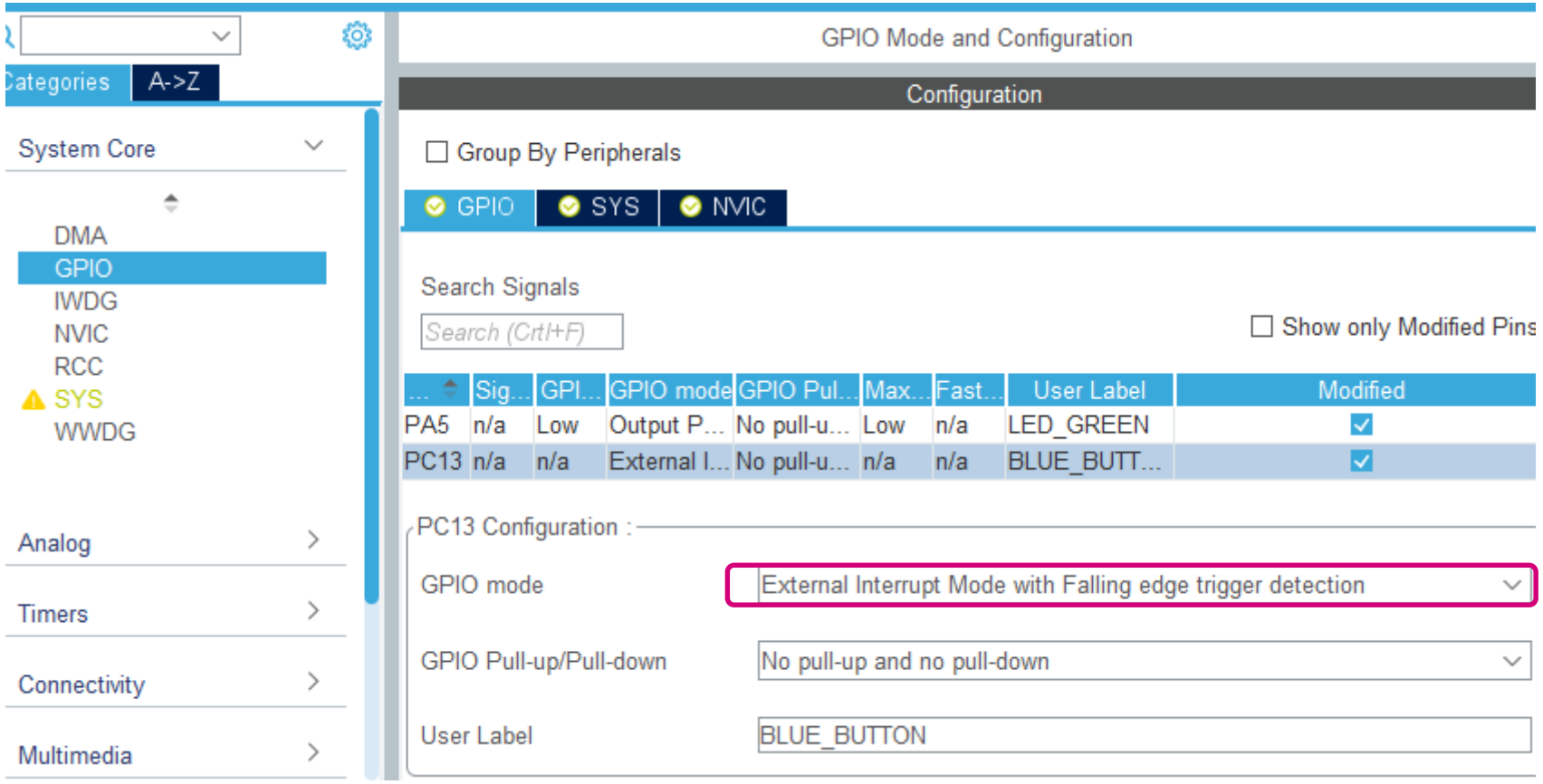
- Using select Enter User Label and insert BLUE_BUTTON label



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

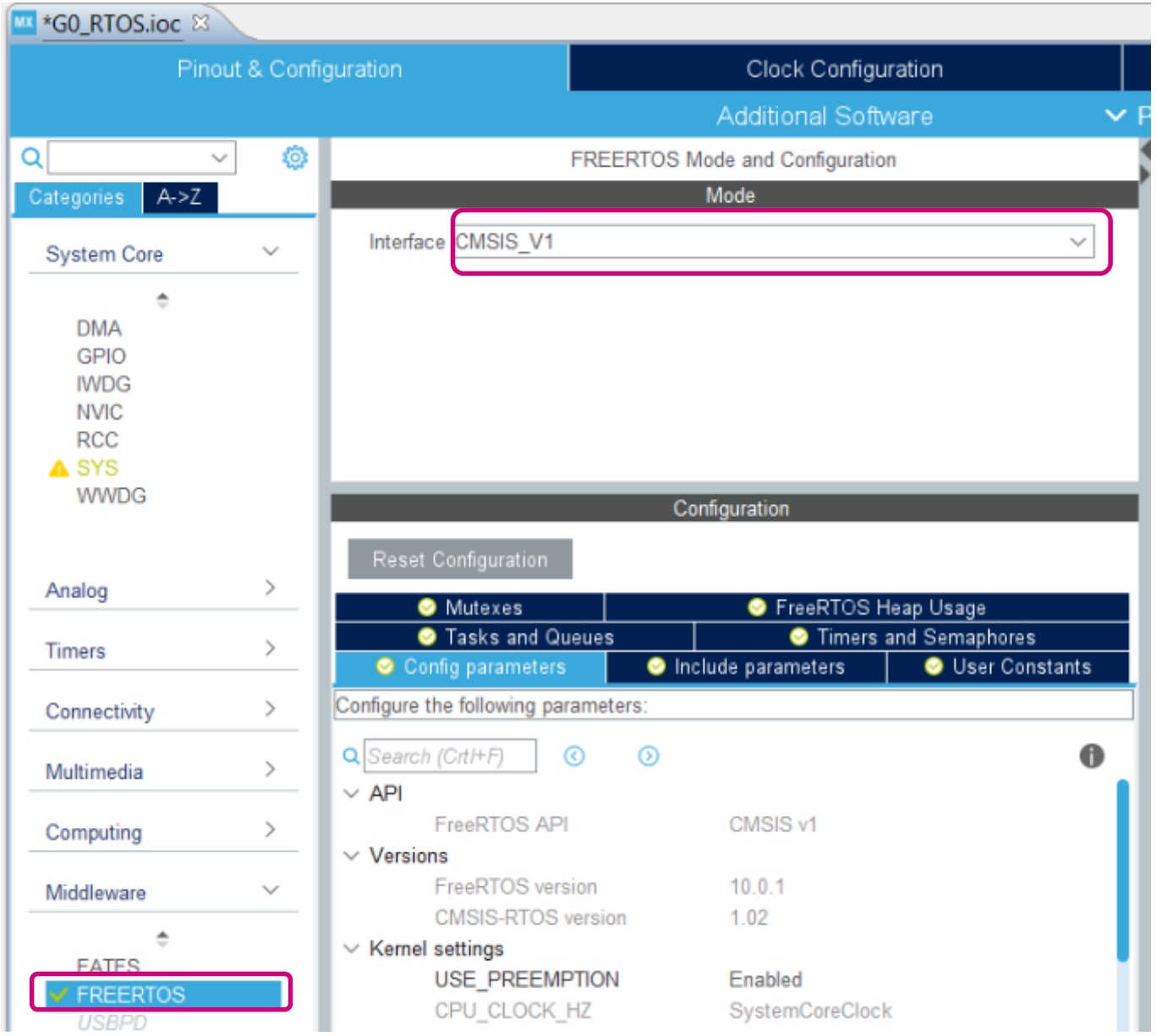
Configure PC13

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



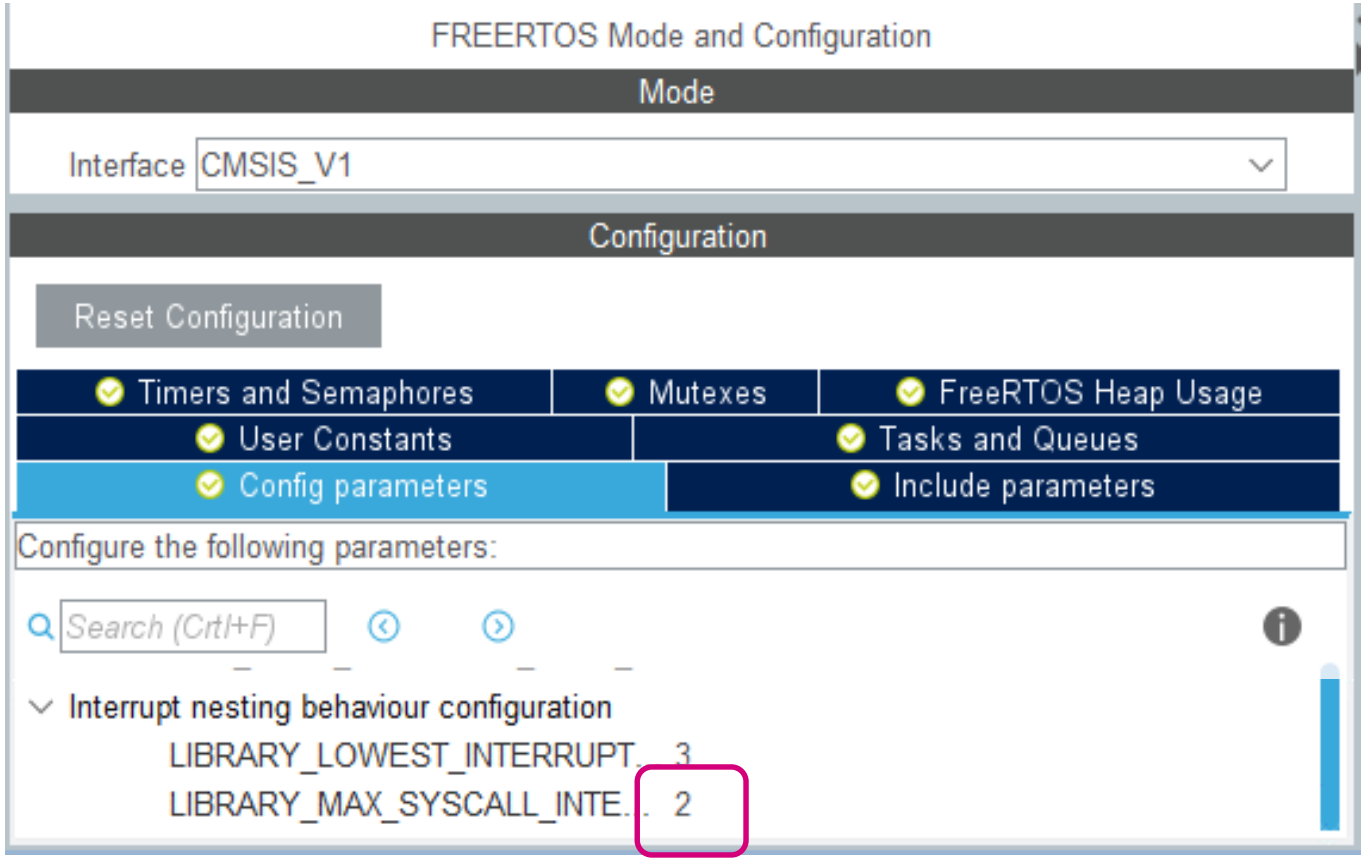
Add FreeRTOS middleware

- Select Middleware -> FREERTOS from peripherals list
- Select its interface as CMSIS_V1



Configure FreeRTOS middleware

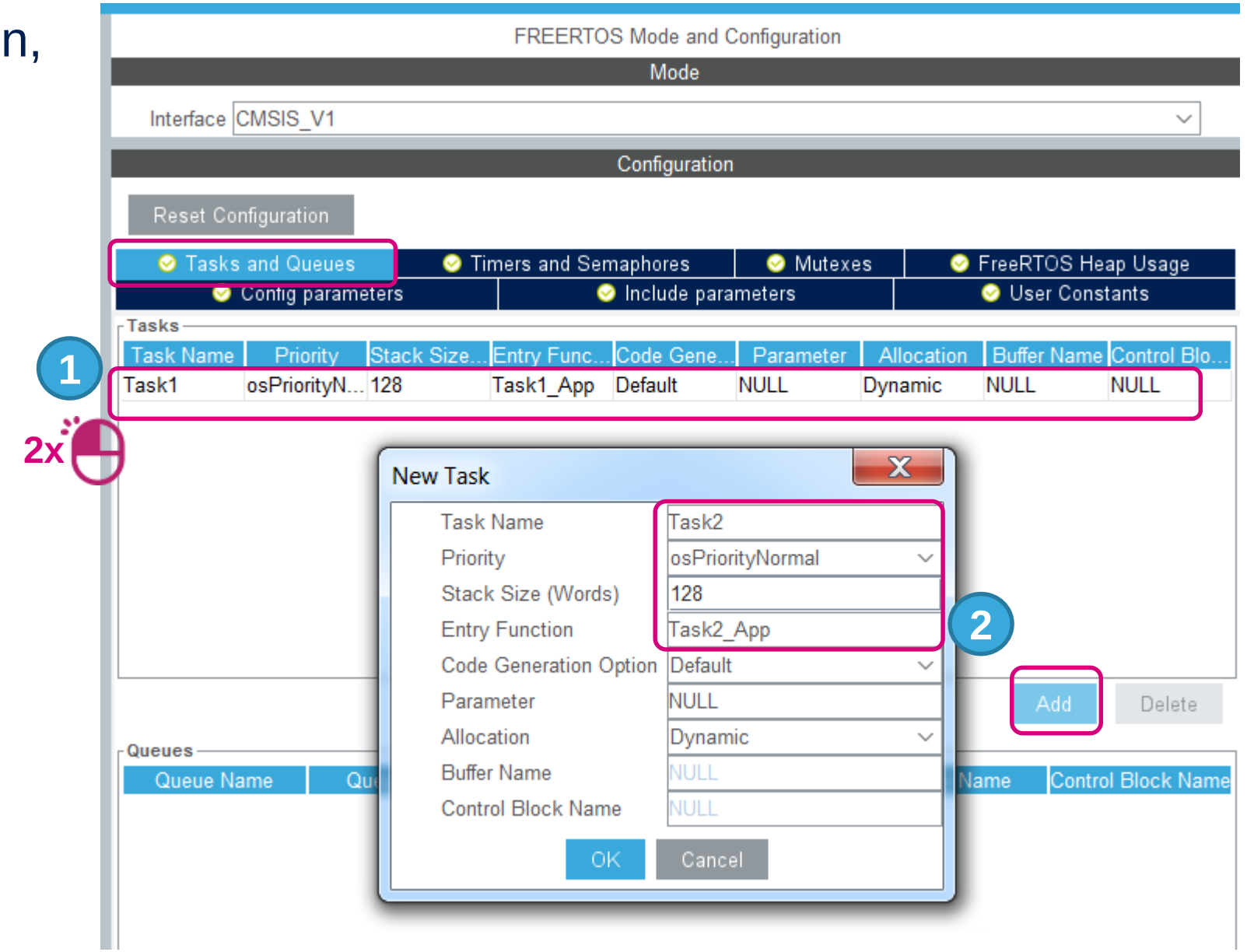
- Within FreeRTOS configuration modify LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY from 3 to 2



Add two tasks within FreeRTOS

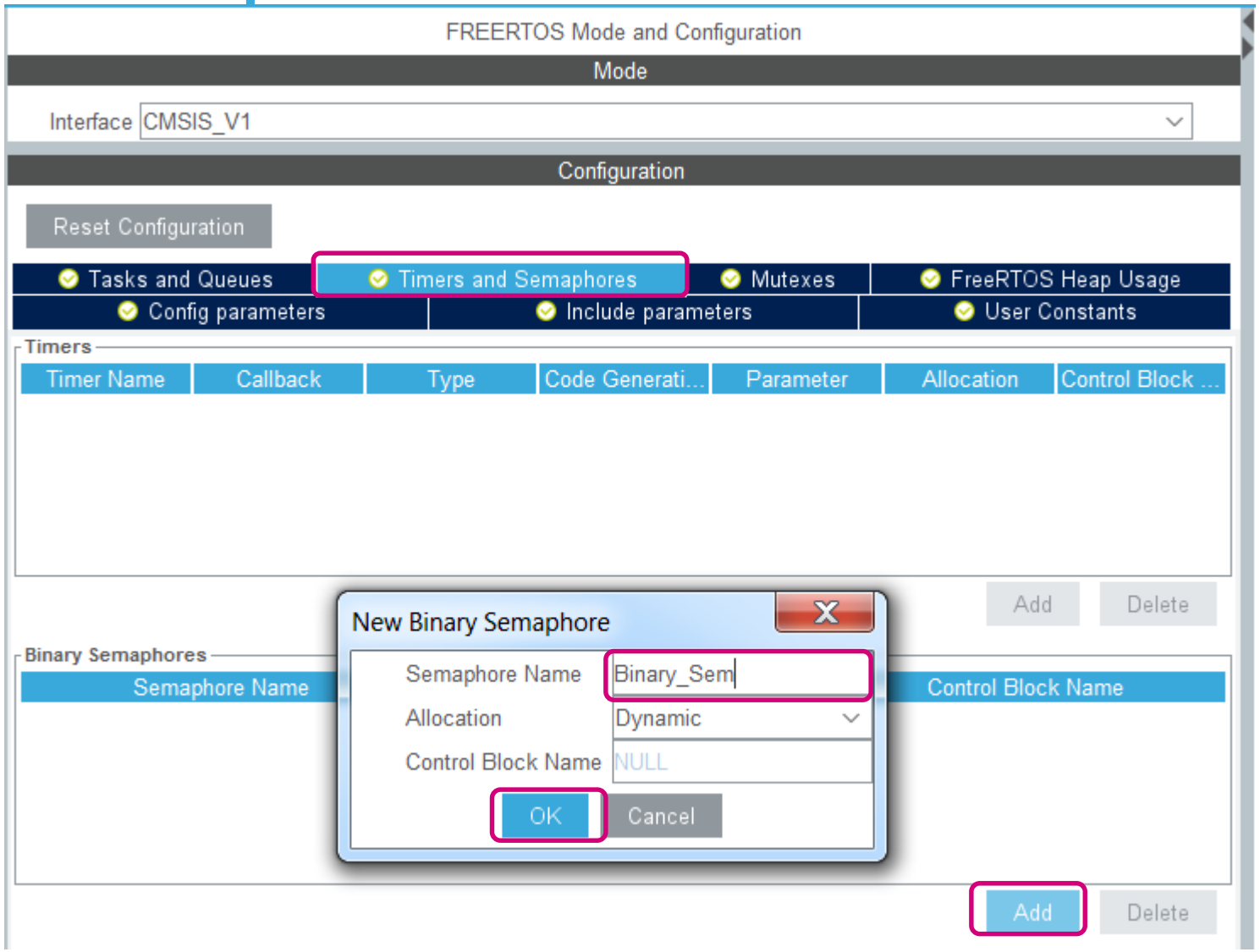
- Go to FreeRTOS Configuration, Tasks and Queues tab

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



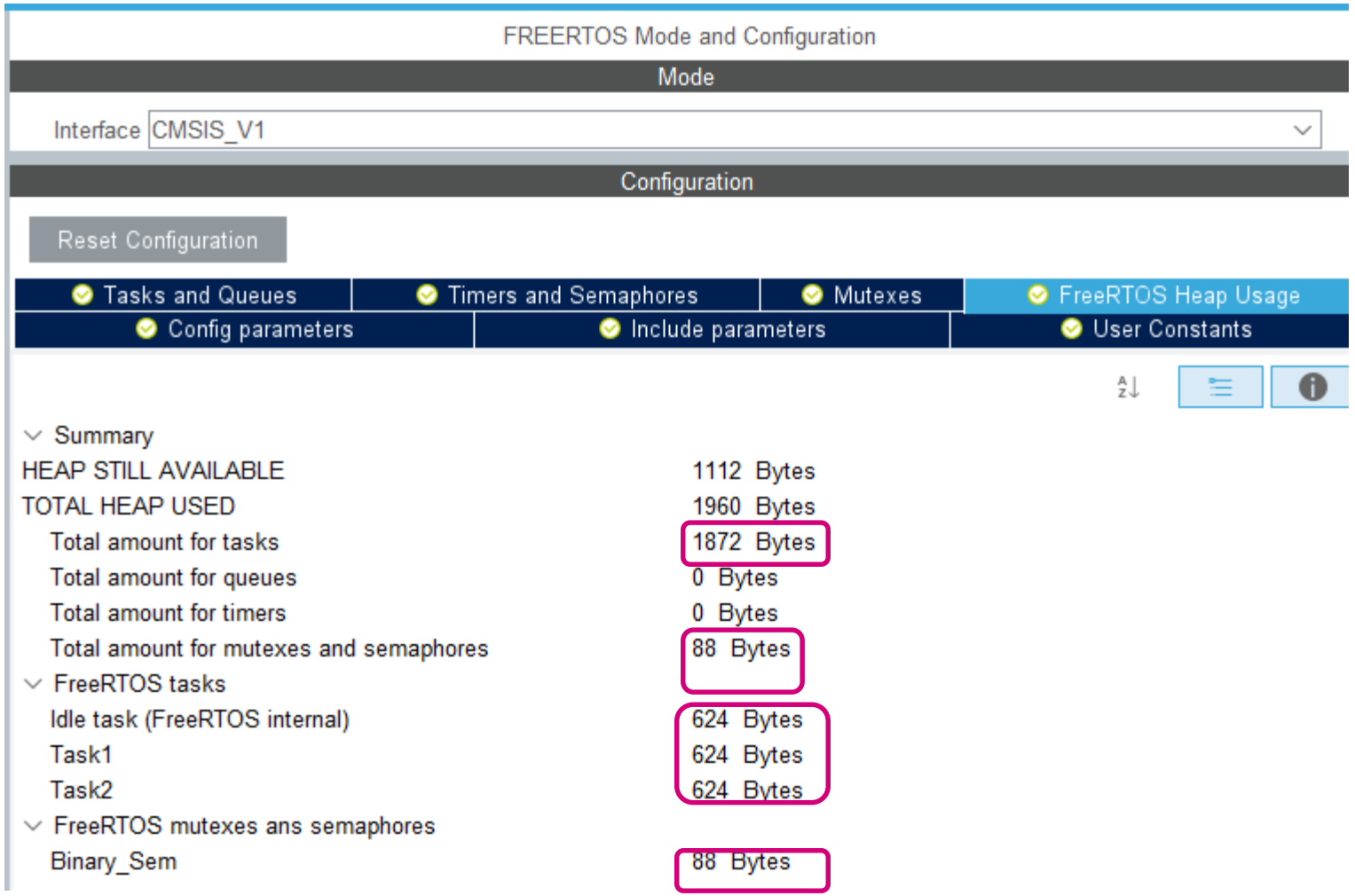
Add binary semaphore within FreeRTOS

- Go to FreeRTOS Configuration, Timers and Semaphores tab
- Add new Binary Semaphore:
  - Samphore Name: Binary_Sem



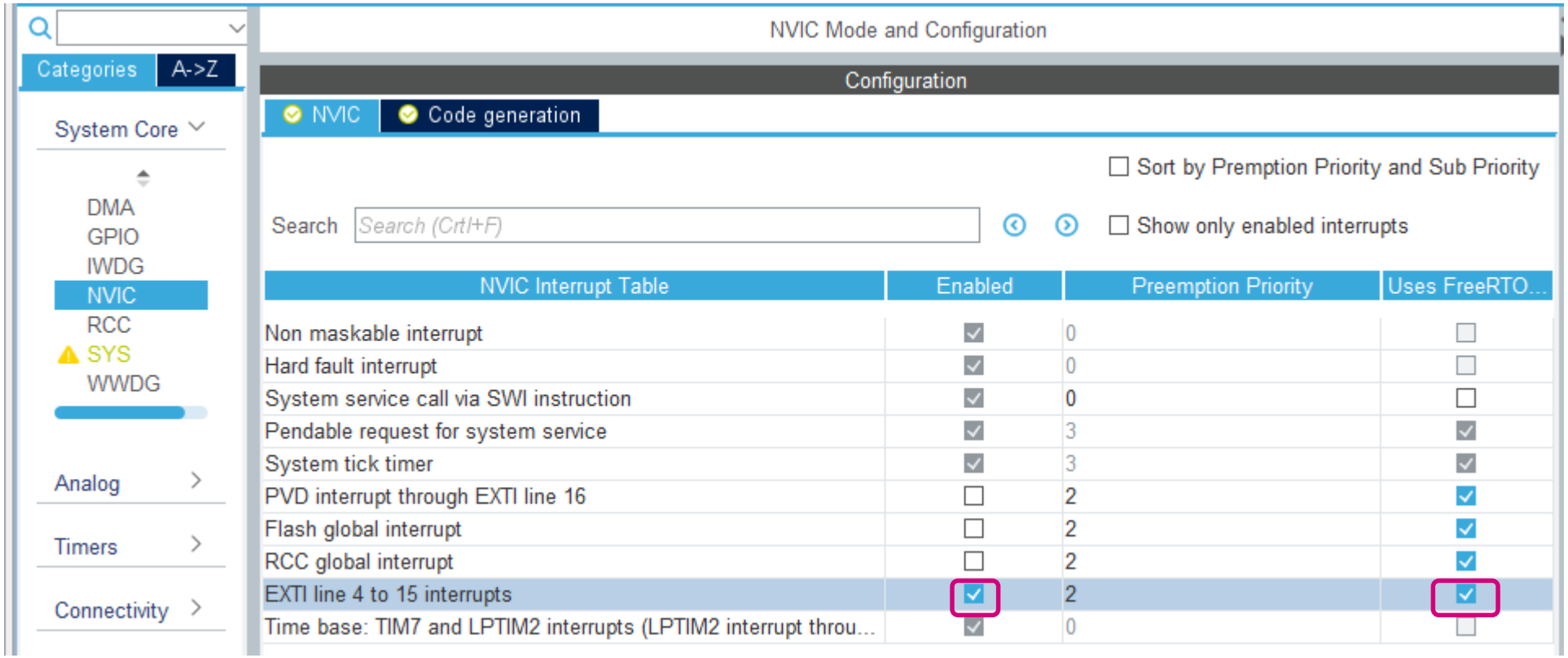
Verify how much memory OS has used

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

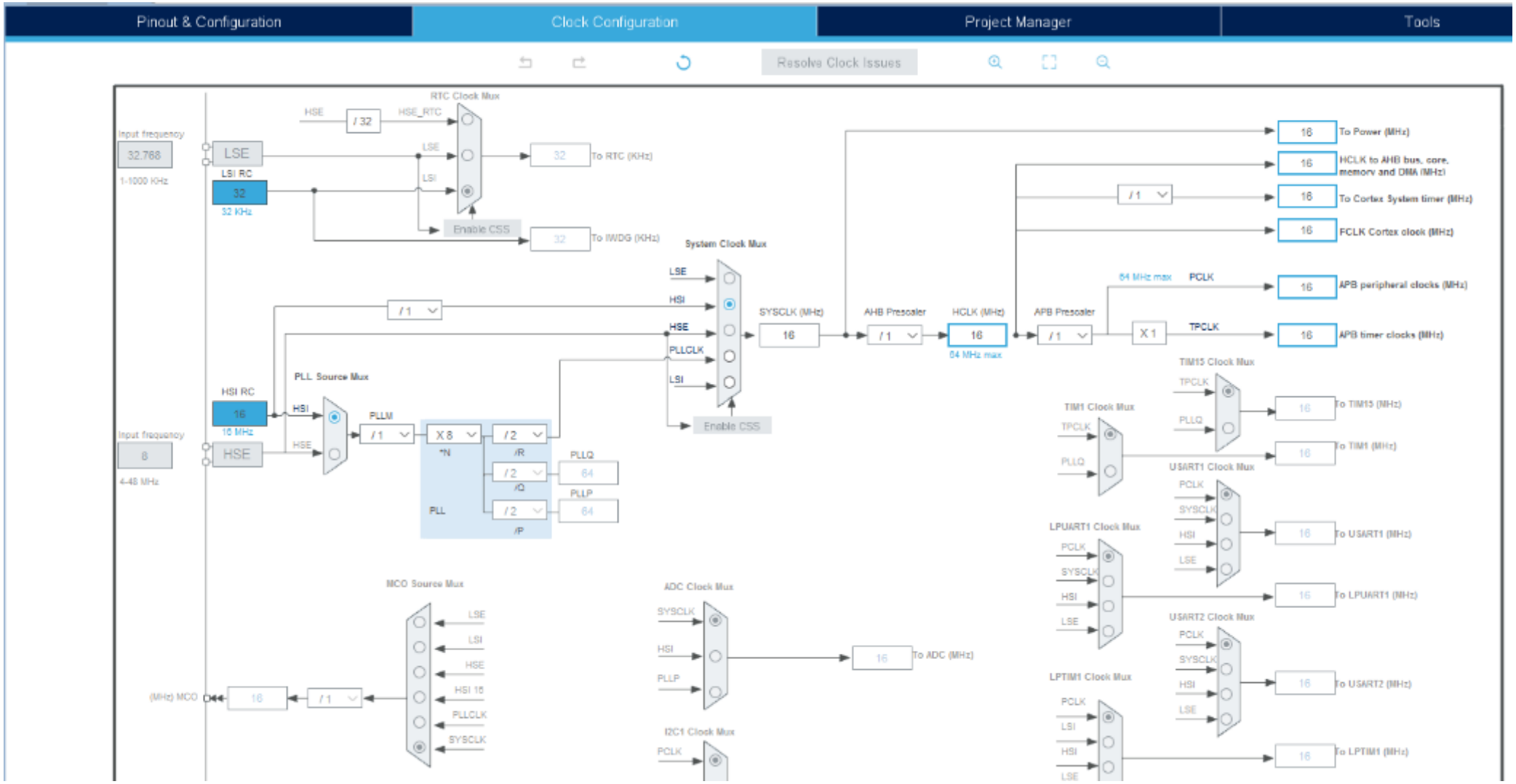


Configure NVIC

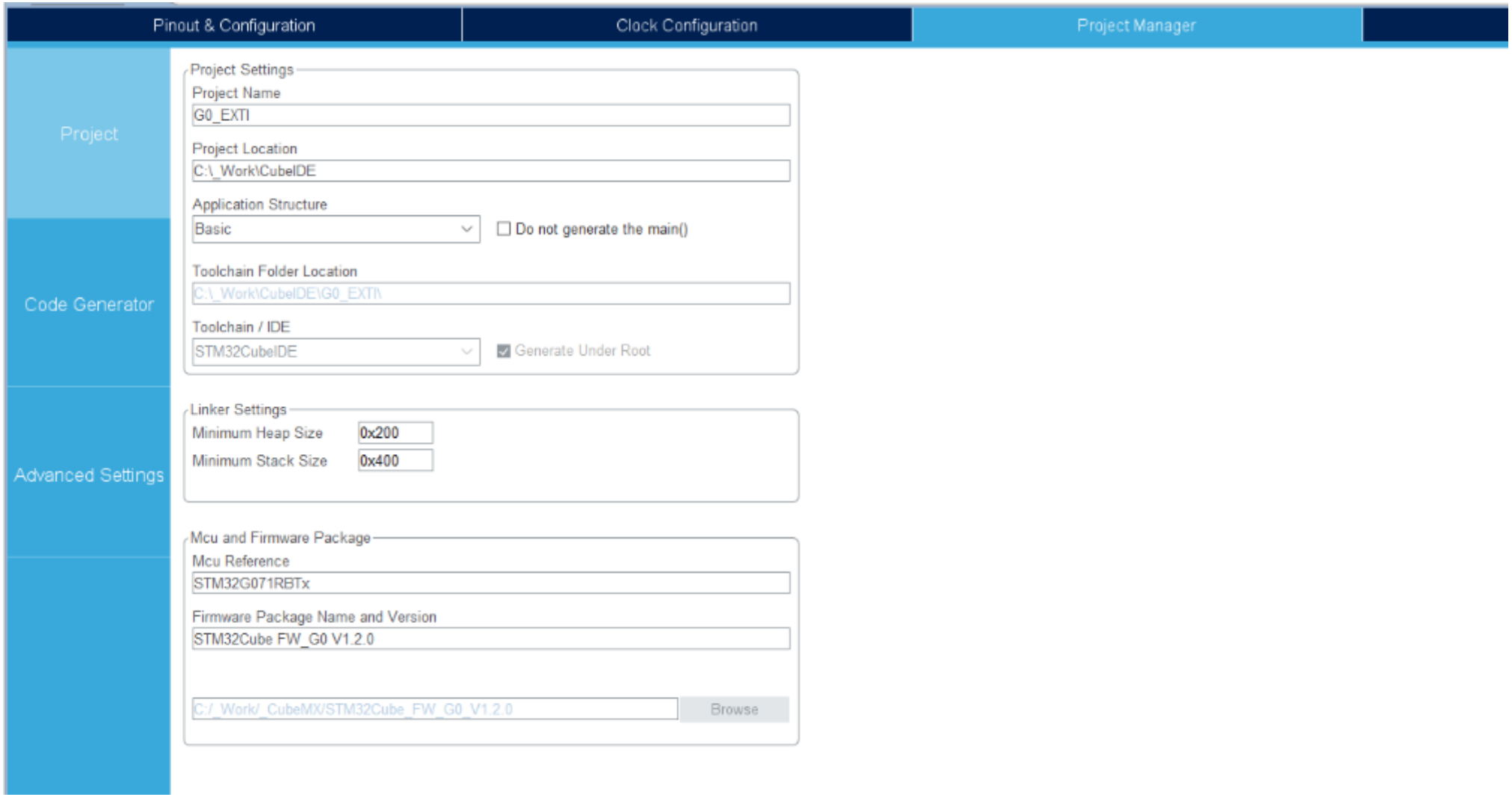
- Go to System Core -> NVIC
- Check whether EXTI line 4 to 15 interrupts is marked as “Uses FreeRTOS functions and is enabled



Default clock configuration – no change

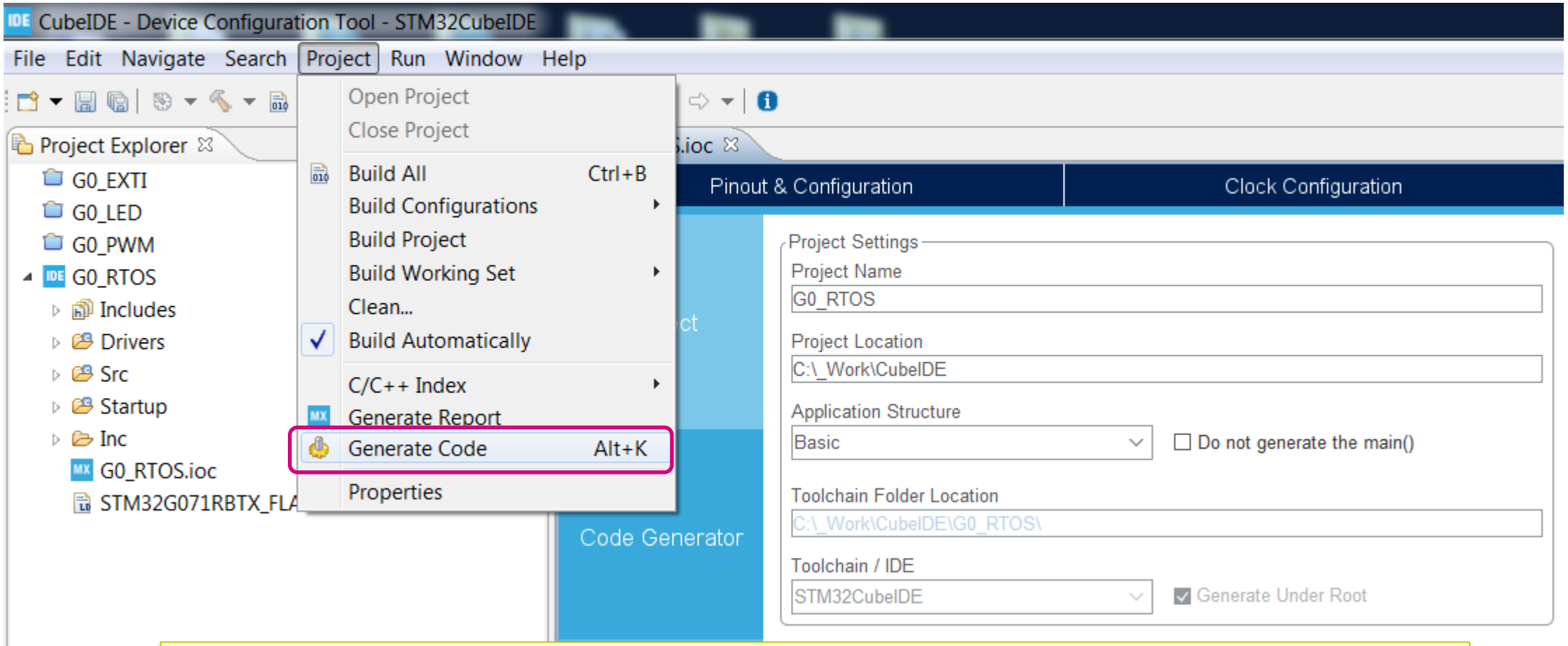


Basic project settings – no change



Code generation

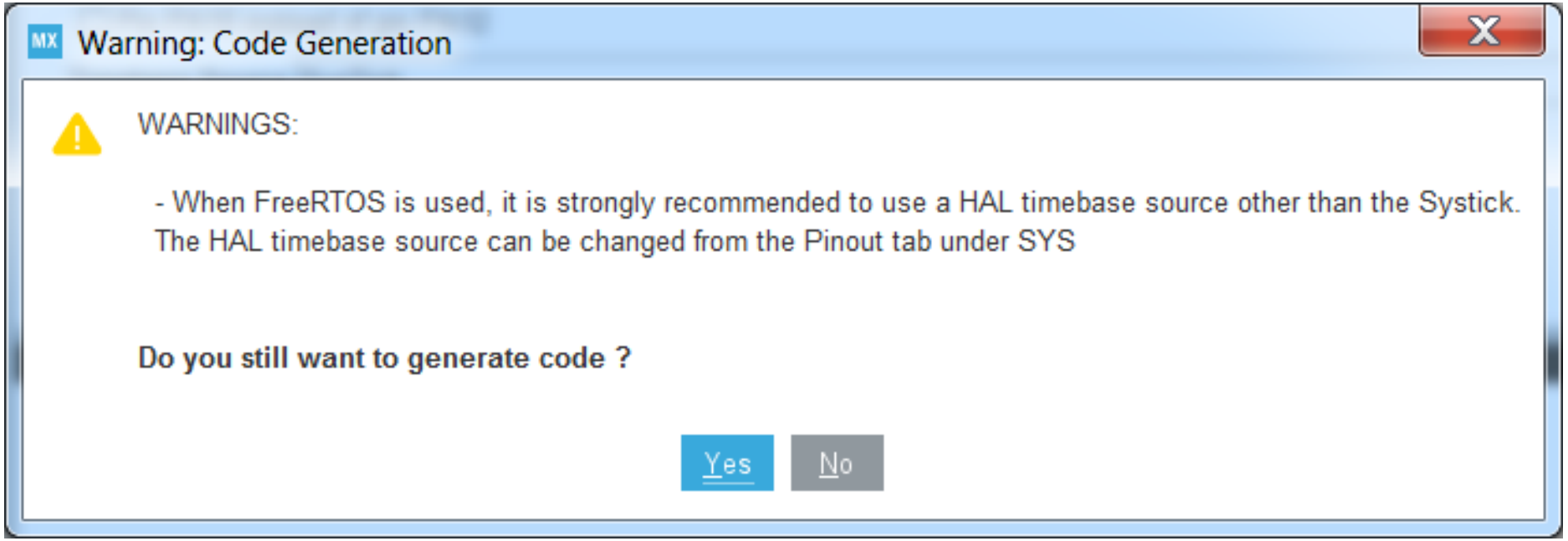
- It is necessary to add to an empty template our project we have just prepared



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

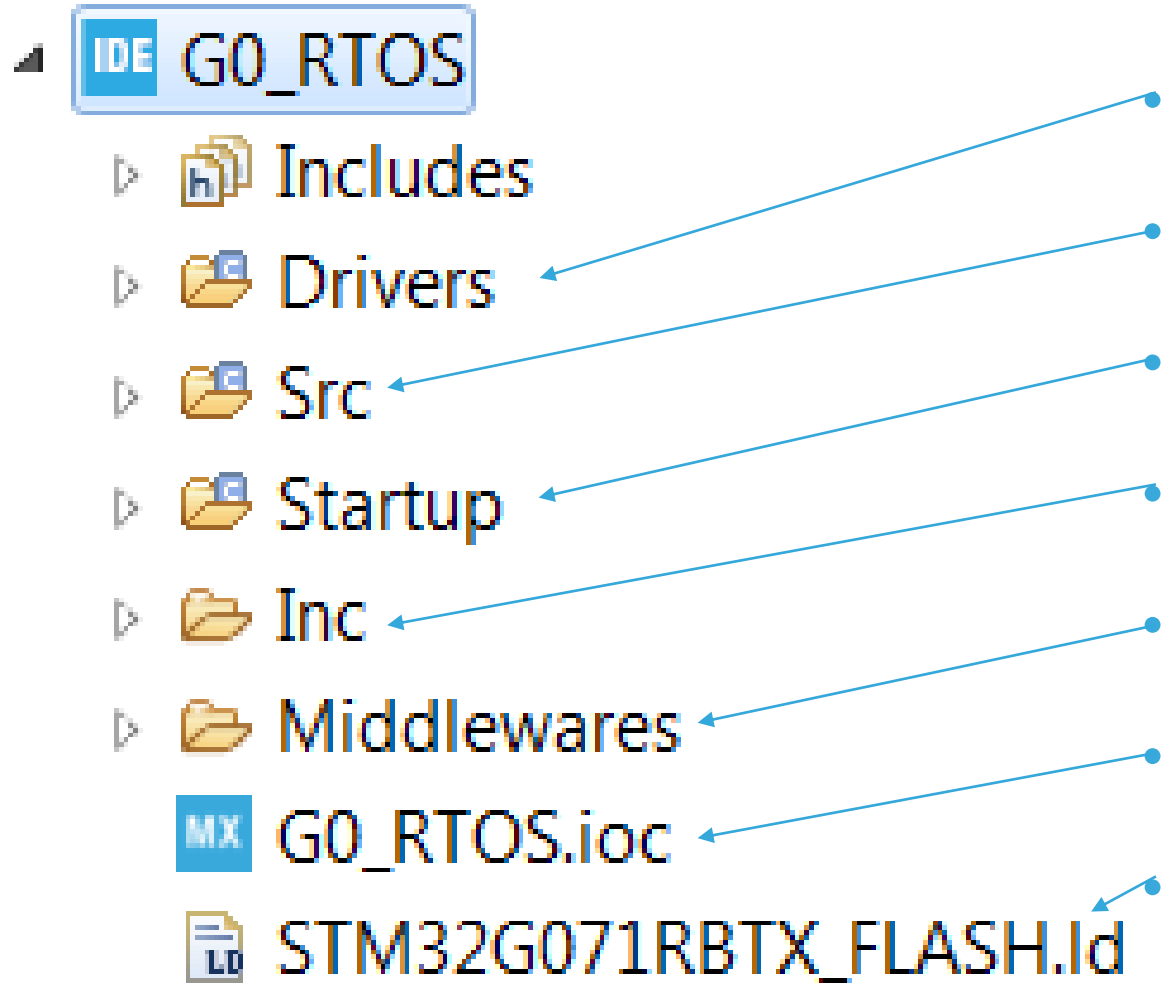
In case we forget about timebase timer

- 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



STM32 HAL library

User application code

STM32 startup assembly file

User application includes

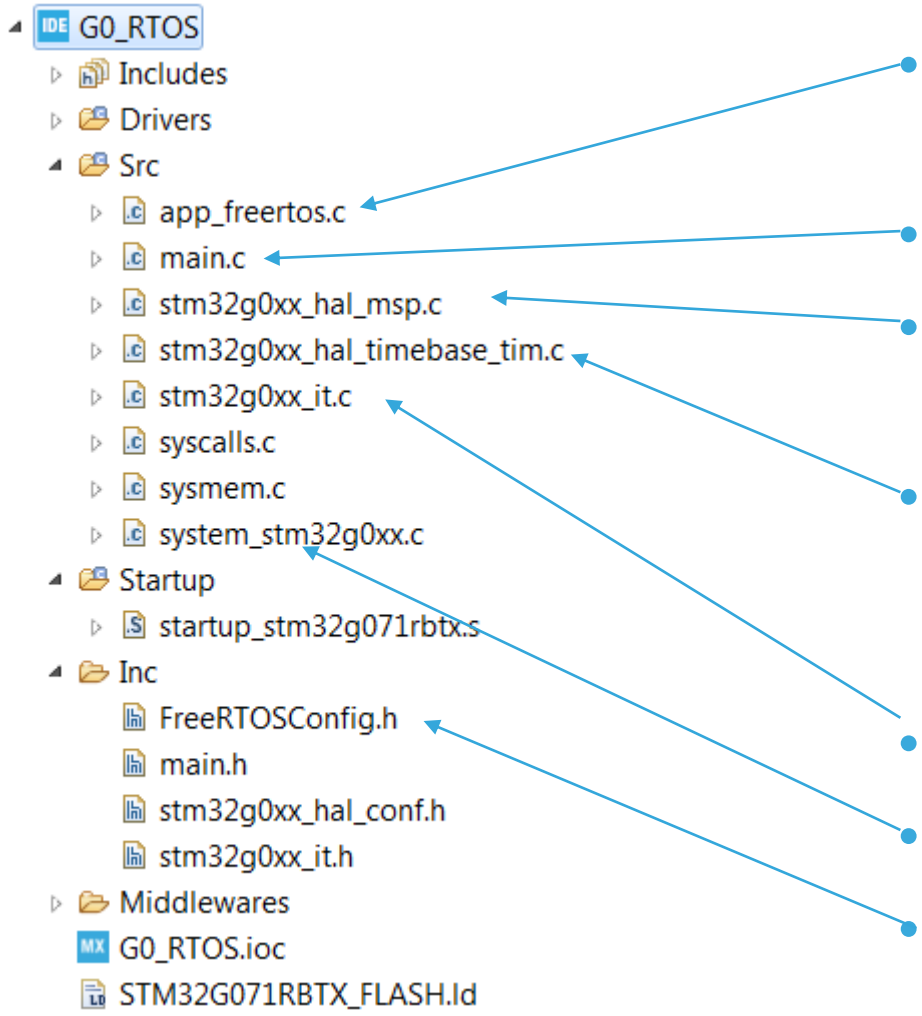
FreeRTOS code

Project configuration file (STM32CubeMX style)

STM32 linker file

User application files structure

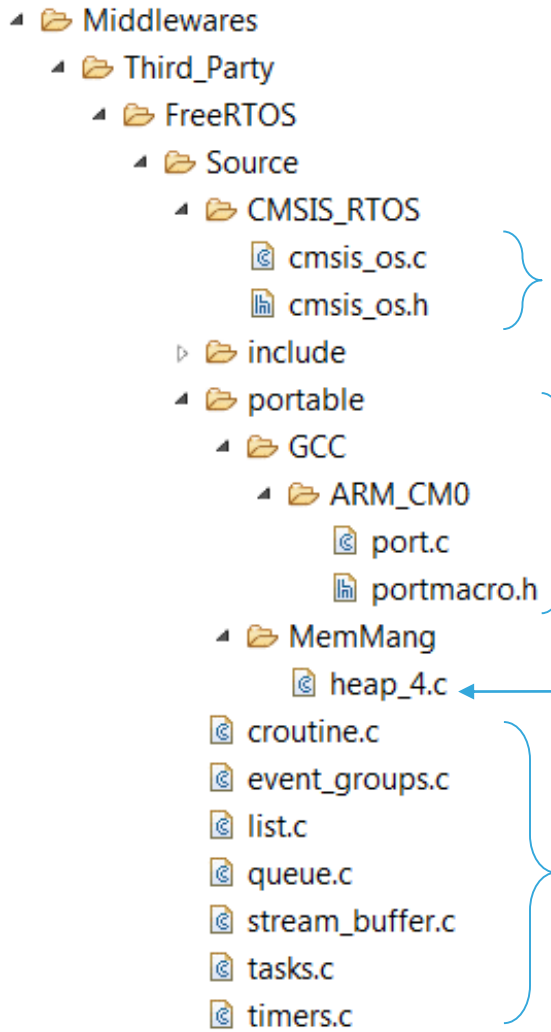
- Once we generate the project, we should see the following files structure



- 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



CMSIS_OS layer files – transforming FreeRTOS API functions into CMSIS_OS ones (defined by ARM)

Interface between FreeRTOS and used MCU (system interrupt vectors definitions)

Memory management functions to allocate and free the memory resources

FreeRTOS source files

STM32 CubeIDE

Code modification

Task1 application code – main.c file

- 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 */
}
```

life.augmented

Code modification

Task2 application code – main.c file

- 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 */
}
```

STM32 CubeIDE

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

life.augmented

STM32 CubeIDE

Code modification

Task1 application code – main.c file

- 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 */
}
```

life.augmented

Code modification

BLUE_BUTTON interrupt callback – main.c file

- 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 */
```

STM32 CubeIDE

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.

life.augmented

STM32 CubeIDE

Thank you

/STM32

@ST_World

community.st.com

life.augmented

www.st.com/stm32g0, www.st.com/stm32cubeide