

As shown in the code snippets given below, the SysTick interrupt handler is updated to use CMSIS-os osSystickHandler function.

- main.c file

```
/* SysTick_IRQn interrupt configuration */
HAL_NVIC_SetPriority(SysTick_IRQn, 15, 0);
```

- stm32f4xx\_it.c file

```
/**
 * @brief This function handles System tick timer.
 */
void SysTick_Handler(void)
{
    /* USER CODE BEGIN SysTick_IRQn 0 */

    /* USER CODE END SysTick_IRQn 0 */
    HAL_IncTick();
    osSystickHandler();
    /* USER CODE BEGIN SysTick_IRQn 1 */

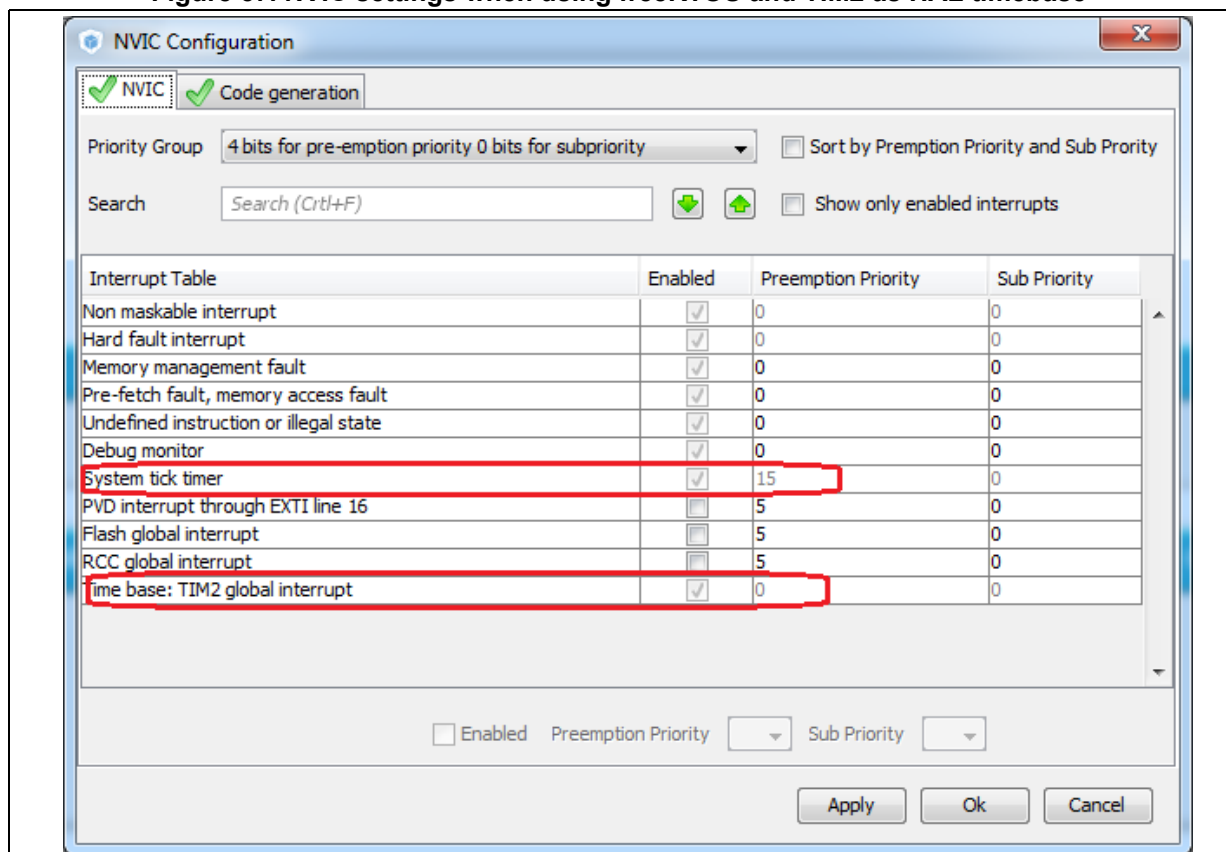
    /* USER CODE END SysTick_IRQn 1 */
}
```

### Example of configuration using TIM2 as HAL timebase source

When TIM2 is used as HAL timebase source, a new `stm32f4xx_hal_timebase_TIM.c` file is generated to overload the HAL timebase related functions, including the `HAL_InitTick` function that configures the TIM2 as the HAL time-base source.

The priority of TIM2 timebase interrupts is set to 0 (High). The SysTick priority is set to 15 (Low) if FreeRTOS is used, otherwise it is set to 0 (High).

**Figure 57. NVIC settings when using freeRTOS and TIM2 as HAL timebase**



The `stm32f4xx_it.c` file is generated accordingly:

- `SysTick_Handler` calls `osSysTickHandler` when FreeRTOS is used, otherwise it calls `HAL_SYSTICK_IRQHandler`.
- `TIM2_IRQHandler` is generated to handle TIM2 global interrupt.

## 4.12 Configuration view

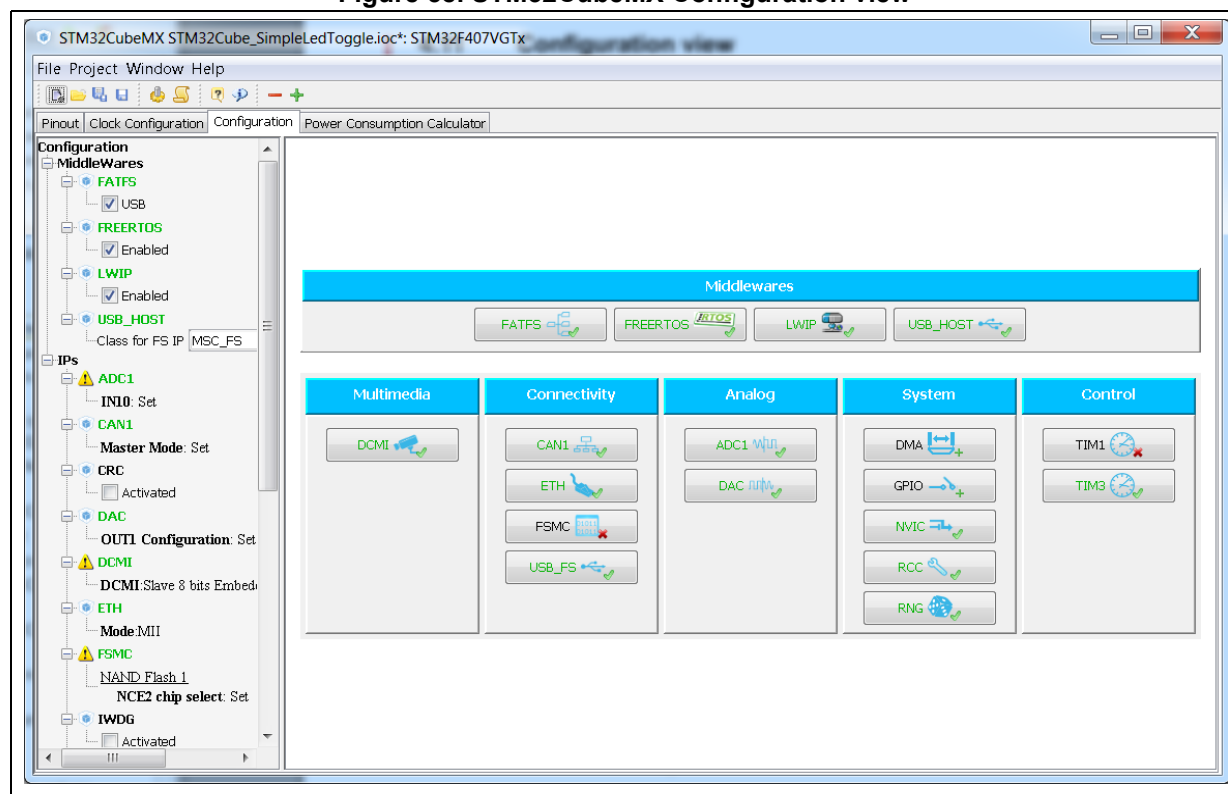
STM32CubeMX **Configuration** window (see [Figure 58](#)) gives an overview of all the software configurable components: GPIOs, peripherals and middleware. Clickable buttons allow selecting the configuration options of the component initialization parameters that will be included in the generated code. The button icon color reflects the configuration status:

- Green checkmark: correct configuration
- Warning sign: incomplete but still functional configuration
- Red cross: for invalid configuration.

**Note:** *GPIO and Peripheral modes that influence the pinout can be set only from the **Pinout** view. They are read-only in the Configuration view.*



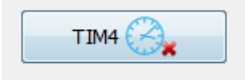
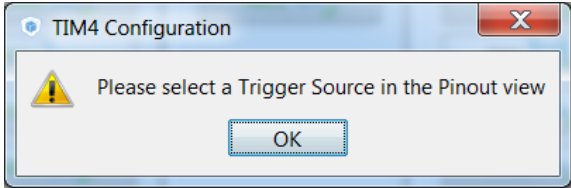
In this view, the MCU is shown on the left pane by its IP tree and on the right pane, by the list of IPs organized in Middleware, Multimedia, Connectivity, Analog, System and Control categories. Each peripheral instance has a dedicated button to edit its configuration: as an example, TIM1 and TIM3 TIM instances are shown as dedicated buttons in [Figure 58](#).

**Figure 58. STM32CubeMX Configuration view**



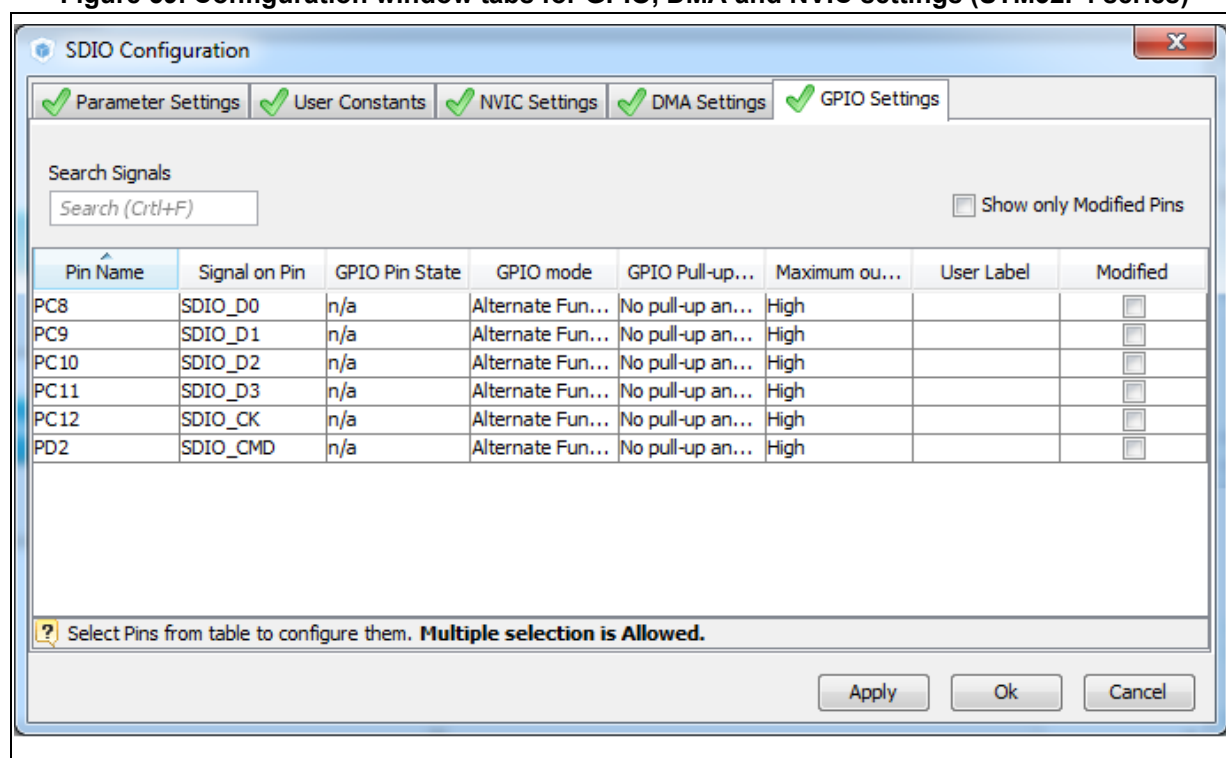
An IP configuration button is associated to each peripheral in the **Configuration** window (see [Table 10](#)).

Table 10. IP configuration buttons

Format	Peripheral Instance configuration status
	Available but not fully configured yet. Click to open the configuration window.
	Well configured with default or user-defined settings that allows proceeding with the generation of corresponding initialization C code. Click to open the configuration window.
	Badly configured with some wrong parameter values. Click to display the errors highlighted in red. Other example (UART): Baud Rate <span style="color: red;">✖ 1000000 Bits/s</span>
	Dialog box that explains source of error. It shall be fixed in another view.

**GPIO, DMA and NVIC** settings can be accessed either via a dedicated button like other IPs or via a tab in the other configuration windows of the IPs which use them (see [Figure 59](#)).

Figure 59. Configuration window tabs for GPIO, DMA and NVIC settings (STM32F4 series)



### 4.12.1 IP and Middleware Configuration window

This window is open by clicking the IP instance or Middleware name from the **Configuration** pane. It allows to configure the functional parameters that are required for initializing the IP or the middleware in the selected operating mode. This configuration is used to generate the corresponding initialization C code. Refer to [Figure 60](#) for an IP Configuration windows example.

The configuration window includes several tabs:

- **Parameter settings** to configure library dedicated parameters for the selected peripheral or middleware,
- **NVIC, GPIO and DMA settings** to set the parameters for the selected peripheral (see [Section 4.12.5: NVIC Configuration window](#), [Section 4.12.3: GPIO Configuration window](#) and [Section 4.12.4: DMA Configuration window](#) for configuration details).
- **User constants** to create one or several user defined constants, common to the whole project (see [Section 4.12.2: User Constants configuration window](#) for user constants details).

Invalid settings are detected and are either:

- Reset to minimum valid value if user choice was smaller than minimum threshold,
- Reset to maximum valid value if user choice was greater than maximum threshold,
- Reset to previous valid value if previous value was neither a maximum nor a minimum threshold value,
- Highlighted in red: ✖ 1000000 Bits/s

[Table 11](#) describes IP and middleware configuration buttons and messages.

**Figure 60. IP Configuration window (STM32F4 series)**

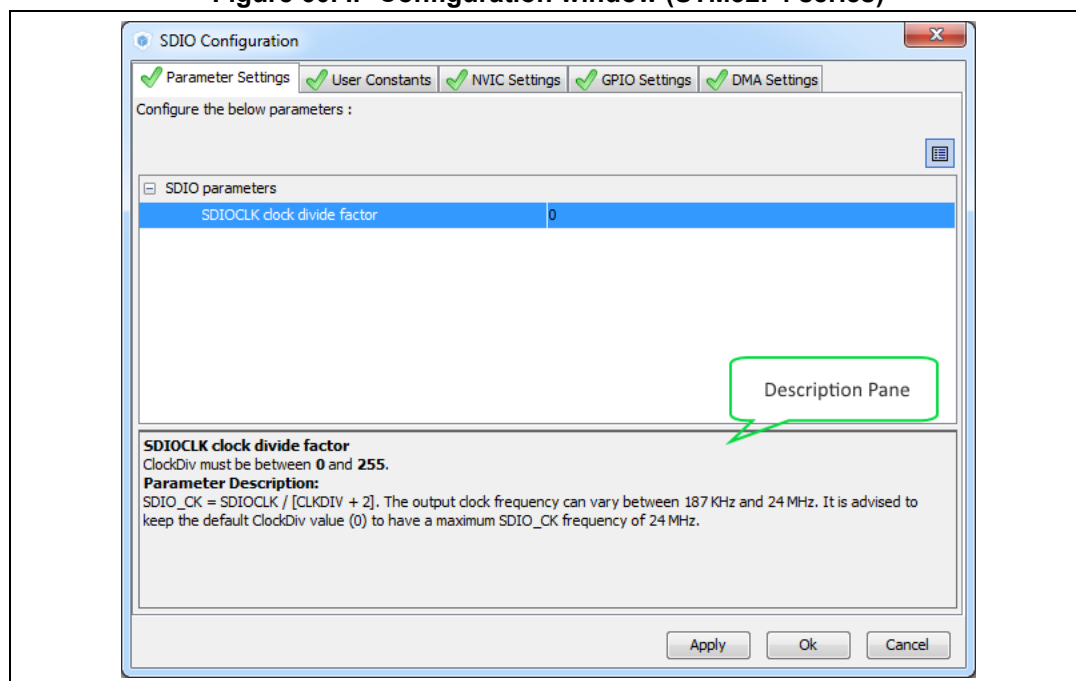

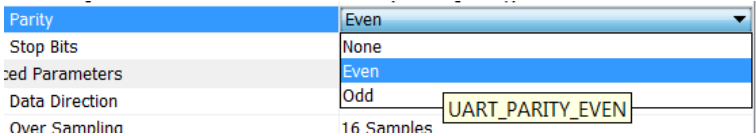
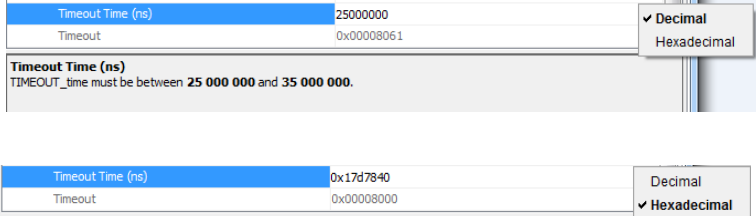


Table 11. IP Configuration window buttons and tooltips

Buttons and messages	Action
Apply	Saves the changes without closing the window
OK	Saves and closes the window
Cancel	Closes and resets previously saved parameter settings
	Shows and Hides the description pane
Tooltip	<p>Guides the user through the settings of parameters with valid min-max range.</p> <p>To display it, moves the mouse over a parameter value from a list of possible values.</p> 
Hexadecimal vs decimal values	<p>Choose to display the field as an hexadecimal or a decimal value by clicking the arrow on the right:</p> 

### 4.12.2 User Constants configuration window

A **User Constants** window is available to define user constants (see [Figure 61](#)). Constants are automatically generated in the STM32CubeMX user project within the mxconstants.h file (see [Figure 62](#)). Once defined, they can be used to configure peripheral and middleware parameters (see [Figure 63](#)).

Figure 61. User Constants window

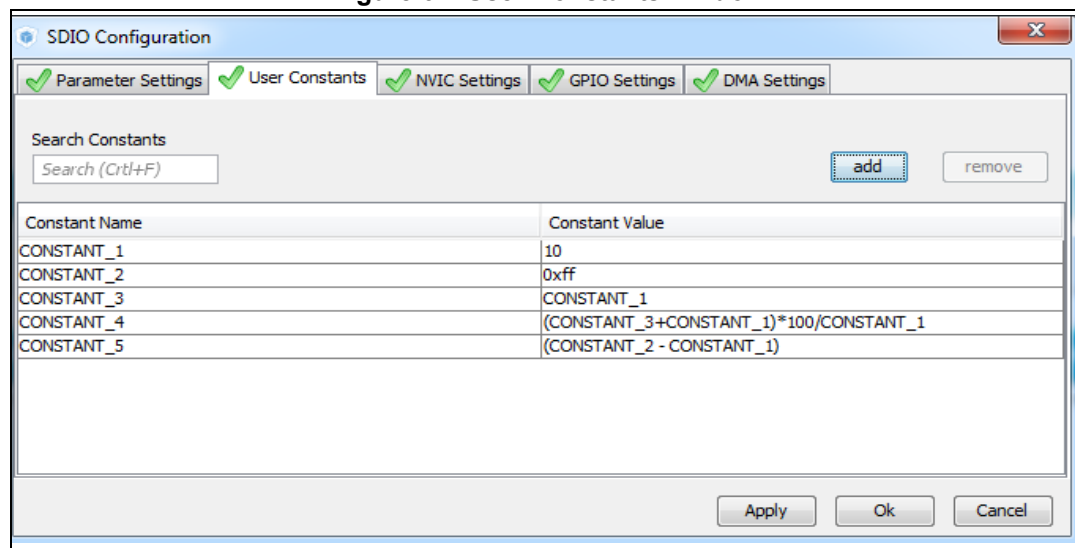


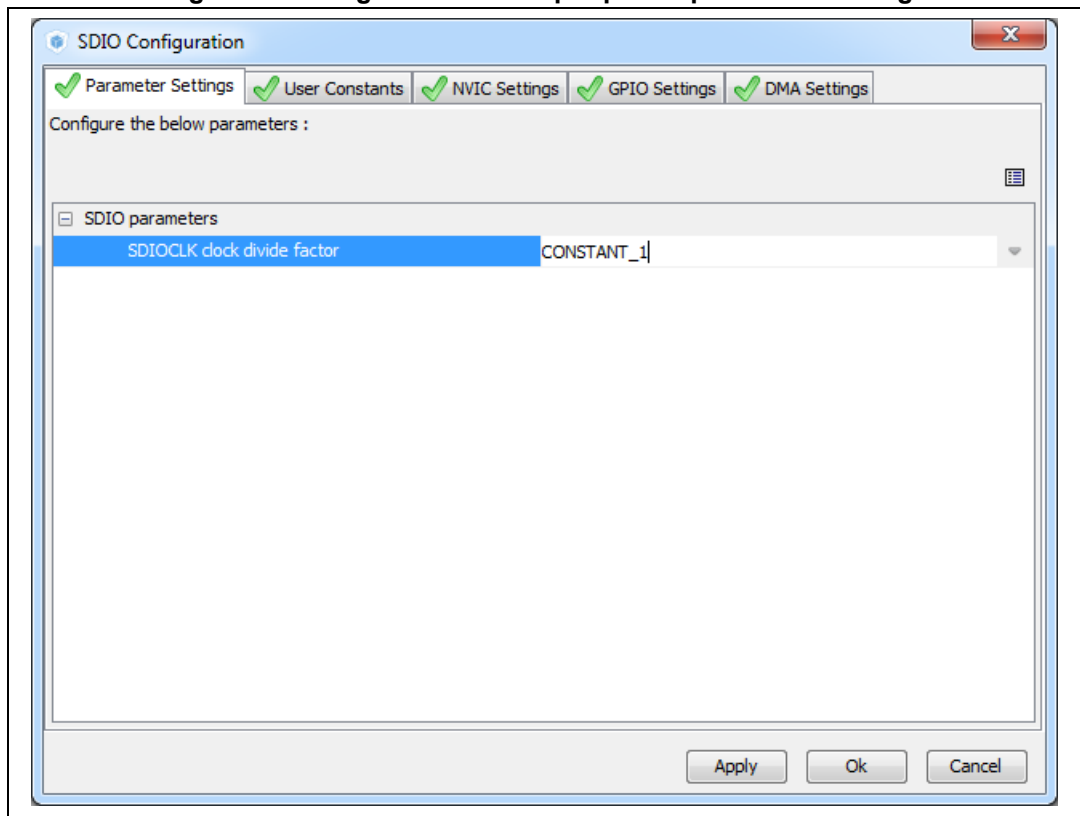
Figure 62. Extract of the generated mxconstants.h file

```
/* Includes -----*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private define -----*/
#define CONSTANT_1 10
#define CONSTANT_2 0xff
#define CONSTANT_3 CONSTANT_1
#define CONSTANT_4 (CONSTANT_3+CONSTANT_1)*100/CONSTANT_1
#define CONSTANT_5 (CONSTANT_2 - CONSTANT_1)

/* USER CODE BEGIN Private defines */
/* USER CODE END Private defines */
```

Figure 63. Using constants for peripheral parameter settings



### Creating/editing user constants

Click the **Add** button to open the **User Constants** window and create a new user-defined constant (see [Figure 64](#)).

A constant consists of:

- A name that must comply with the following rules:
  - It must be unique.
  - It shall not be a C/C++ keyword.
  - It shall not contain a space.
  - It shall not start with digits.
- A value
 

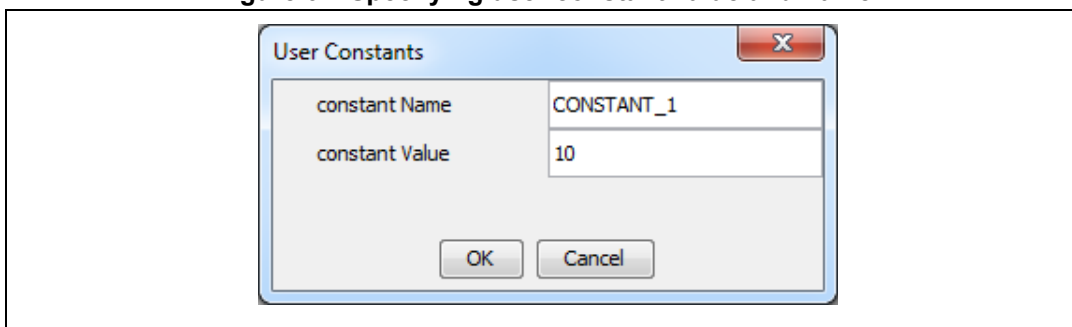
The constant value can be: (see [Figure 61](#) for examples):

  - a simple decimal or hexadecimal value
  - a previously defined constant
  - a formula using arithmetic operators (subtraction, addition, division, multiplication, and remainder) and numeric value or user-defined numeric constants as operands.
  - a character string: the string value must be between double quotes (example: "constant\_for\_usart").



Once a constant is defined, its name and/or its value can still be changed: double click the row that specifies the user constant to be modified. This opens the **User Constants** window for edition. The change of constant name is applied wherever the constant is used. This does not affect the peripheral or middleware configuration state. However changing the constant value impacts the parameters that use it and might result in invalid settings (e.g. exceeding a maximum threshold). Invalid parameter settings will be highlighted in red with a red cross.

**Figure 64. Specifying user constant value and name**



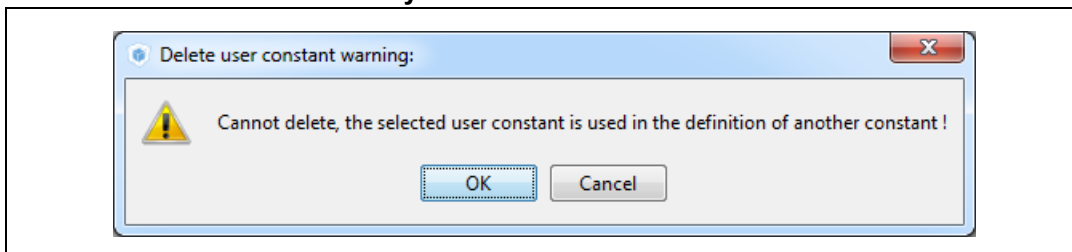
### Deleting user constants

Click the **Remove** button to delete an existing user-defined constant.

The user constant is then automatically removed except in the following cases:

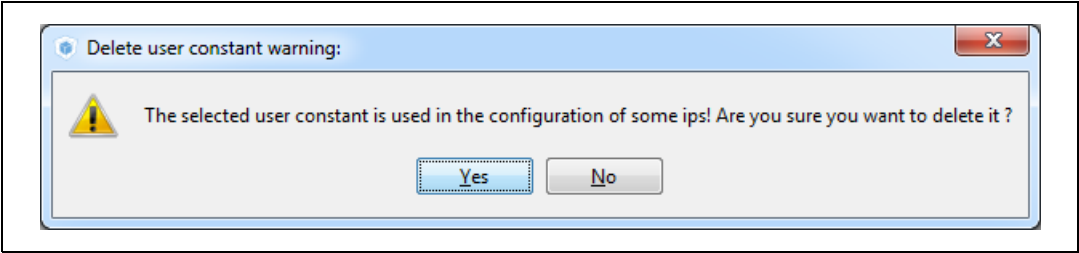
- When the constant is used for the definition of another constant. In this case, a popup window displays an explanatory message (see [Figure 65](#)).

**Figure 65. Deleting user constant not allowed when constant already used for another constant definition**



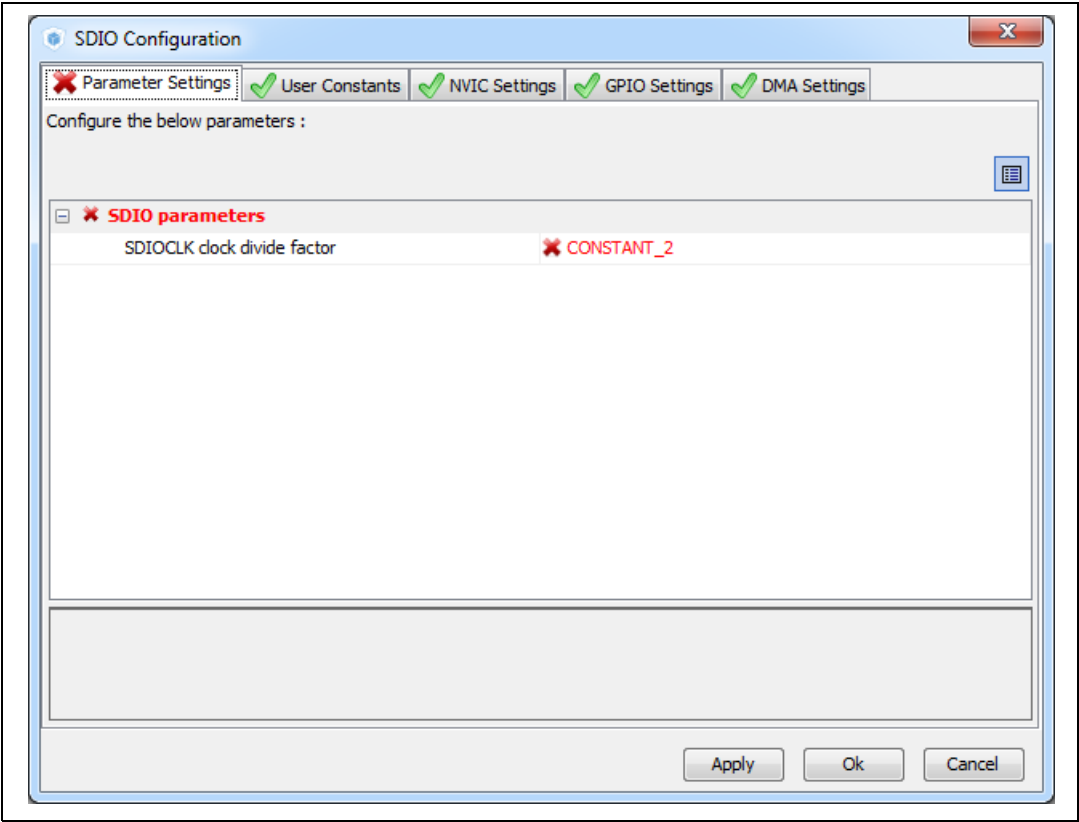
- When the constant is used for the configuration of a peripheral or middleware library parameter. In this case, the user is requested to confirm the deletion since the constant removal will result in a invalid peripheral or middleware configuration (see [Figure 66](#)).

**Figure 66. Deleting a user constant used for parameter configuration- Confirmation request**



Clicking Yes leads to an invalid peripheral configuration (see [Figure 67](#))

**Figure 67. Deleting a user constant used for peripheral configuration - Consequence on peripheral configuration**



Searching for user constants

The **Search Constants** field allows searching for a constant name or value in the complete list of user constants (see [Figure 68](#) and [Figure 69](#)).

Figure 68. Searching user constants list for name

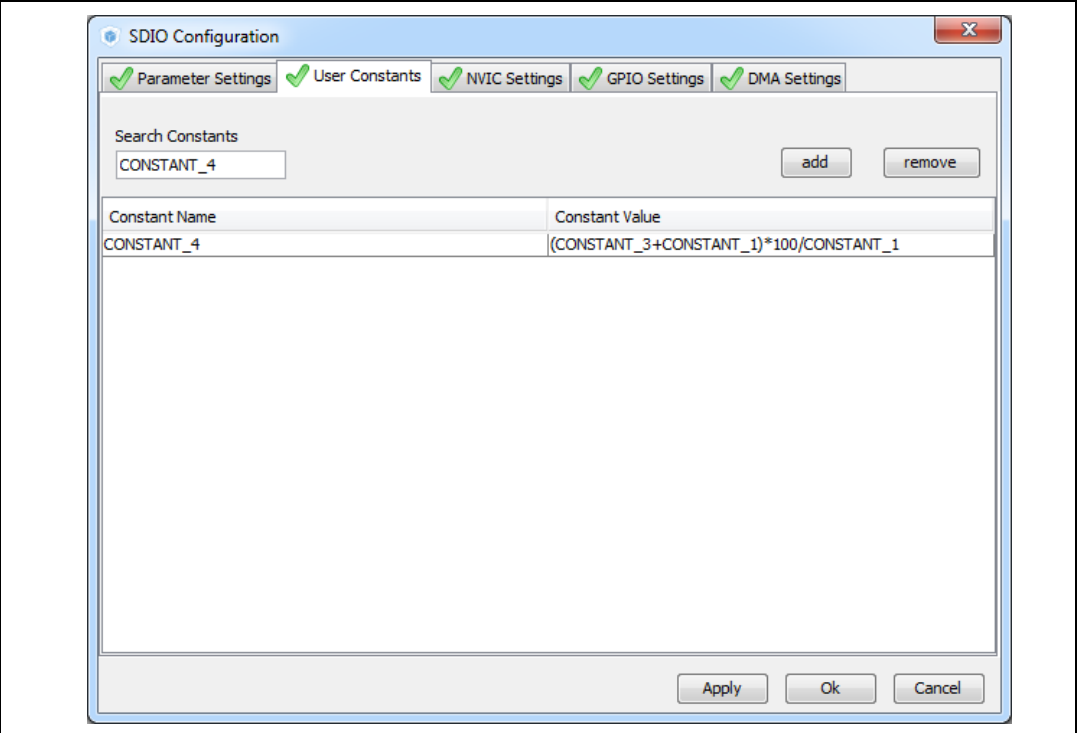
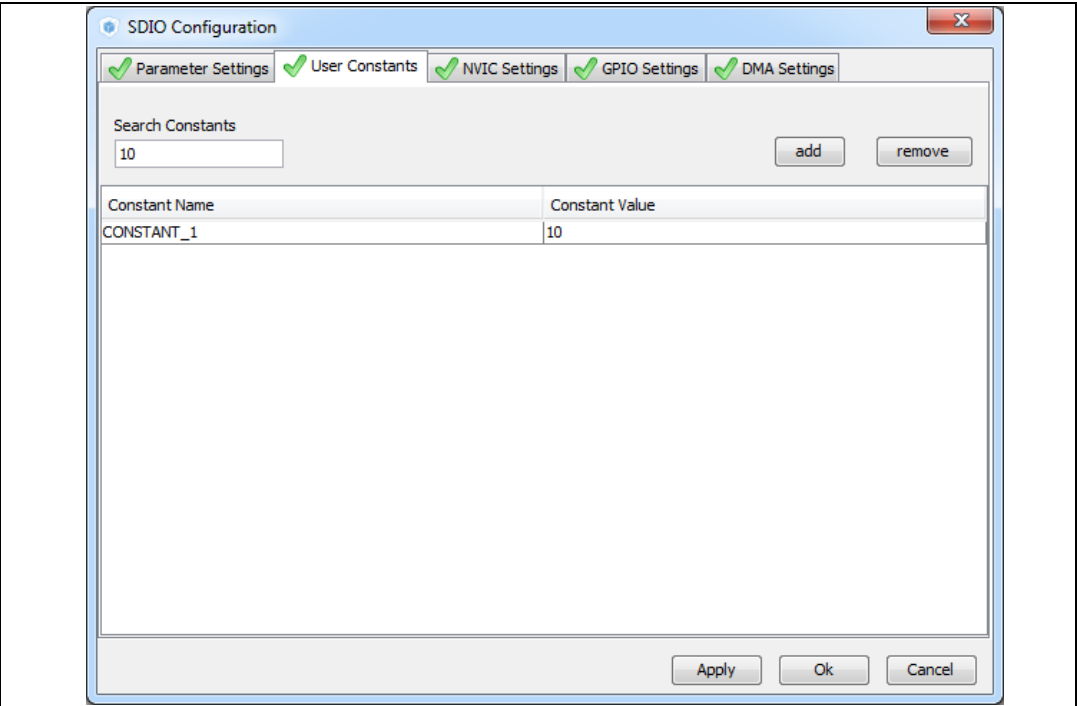


Figure 69. Searching user constants list for value



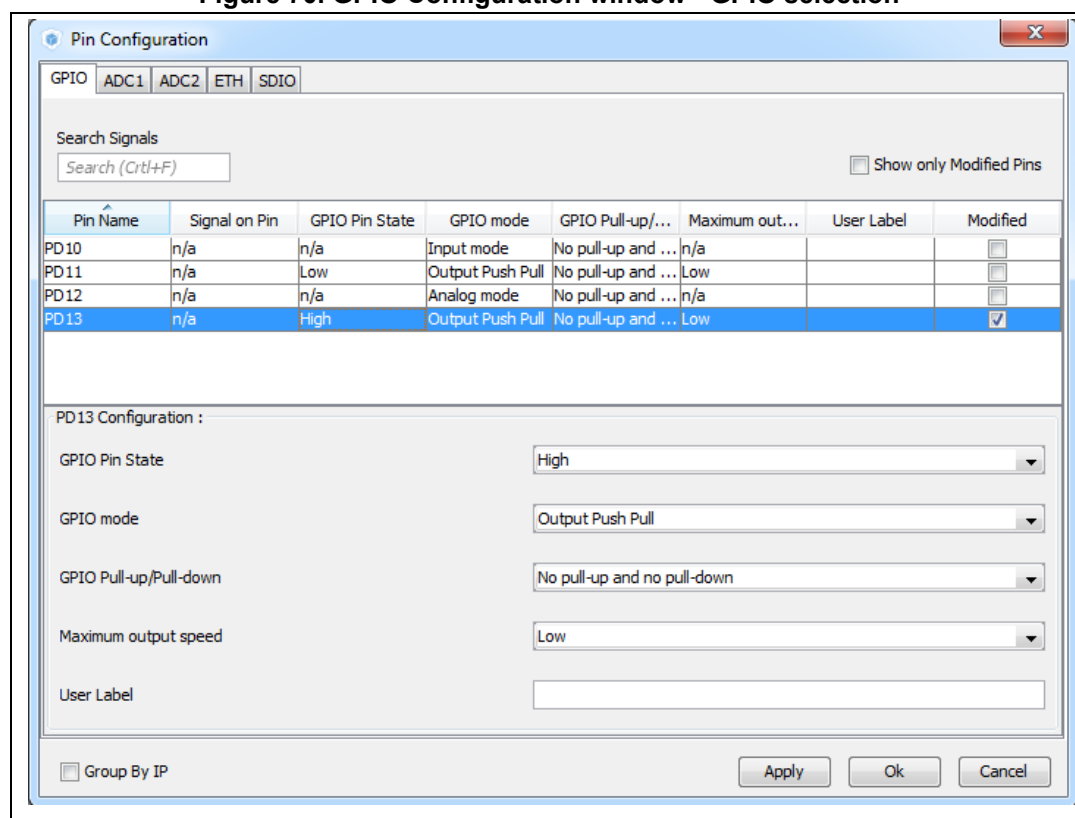
### 4.12.3 GPIO Configuration window

Click **GPIO** in the **Configuration** pane to open the **GPIO configuration** window that allows configuring the GPIO pin settings (see [Figure 70](#)). The configuration is populated with default values that might not be adequate for some peripheral configurations. In particular, check if the GPIO speed is sufficient for the peripheral communication speed and select the internal pull-up whenever needed.

*Note:* **GPIO settings can also be accessed for a specific IP instance via the dedicated GPIO window in the IP instance configuration window.**

*In addition, GPIOs can be configured in output mode (default output level). The generated code will be updated accordingly.*

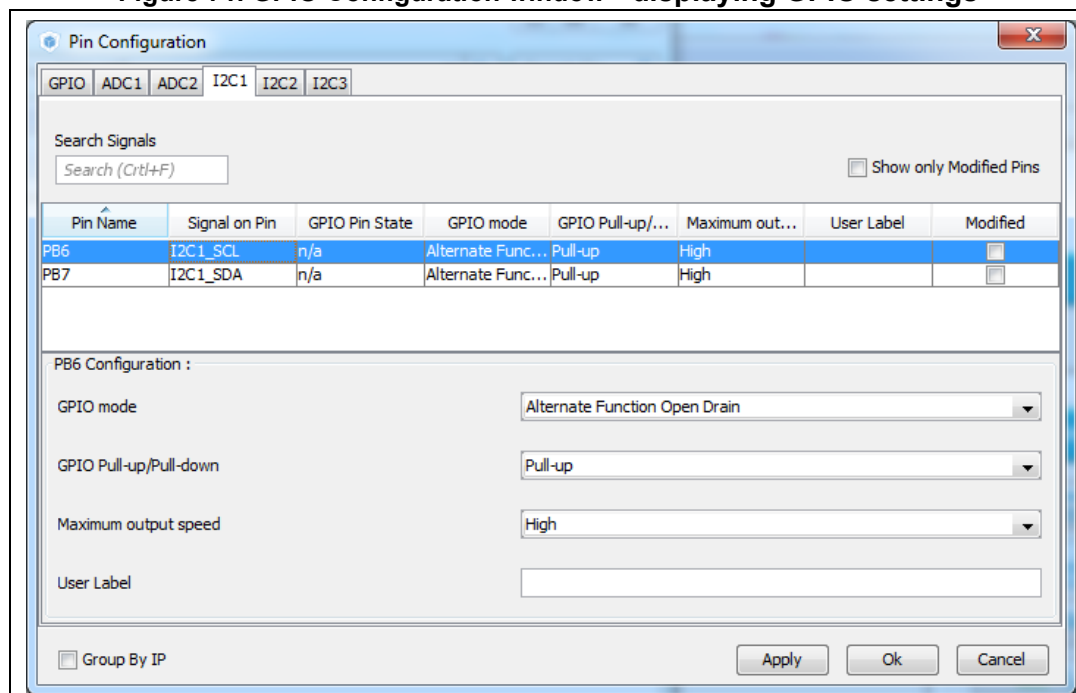
**Figure 70. GPIO Configuration window - GPIO selection**



Click a row or select a set of rows to display the corresponding GPIO parameters (see [Figure 71](#)):

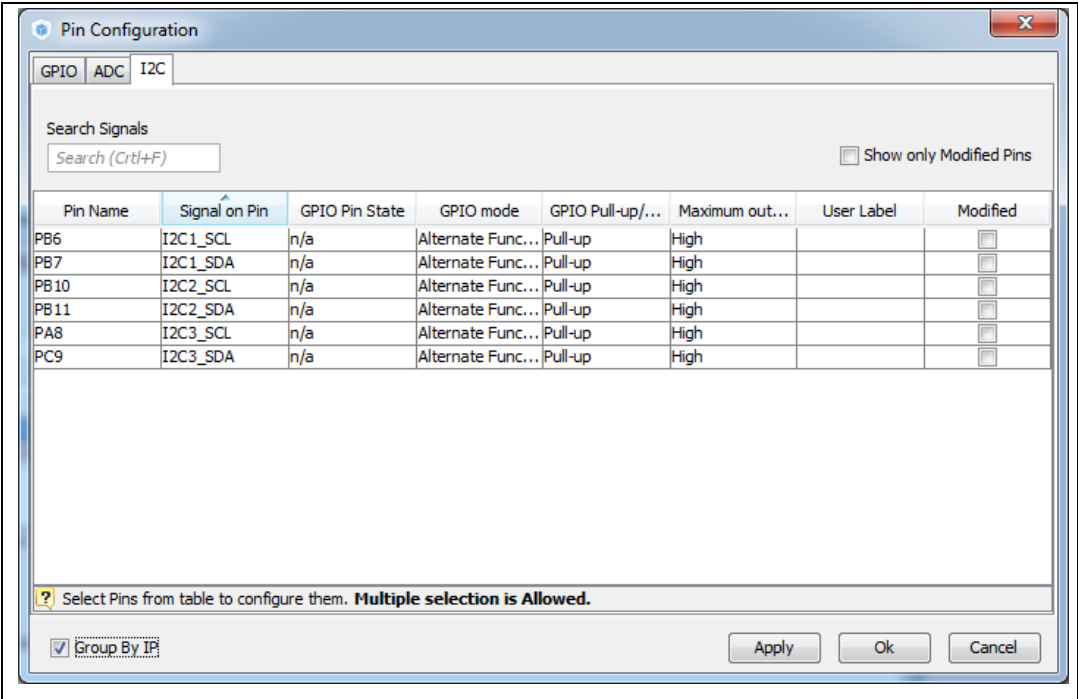
- GPIO PIN state**  
 It changes the default value of the GPIO Output level. It is set to low by default and can be changed to high.
- GPIO mode** (analog, input, output, alternate function)  
 Selecting an IP mode in the **Pinout** view automatically configures the pins with the relevant alternate function and GPIO mode.
- GPIO pull-up/pull-down**  
 It is set to a default value and can be configured when other choices are possible.
- GPIO maximum output speed** (for communication IPs only)  
 It is set to Low by default for power consumption optimization and can be changed to a higher frequency to fit application requirements.
- User Label**  
 It changes the default name (e.g. GPIO\_input) into a user defined name. The **Chip** view is updated accordingly. The GPIO can be found under this new name via the Find menu.

**Figure 71. GPIO Configuration window - displaying GPIO settings**



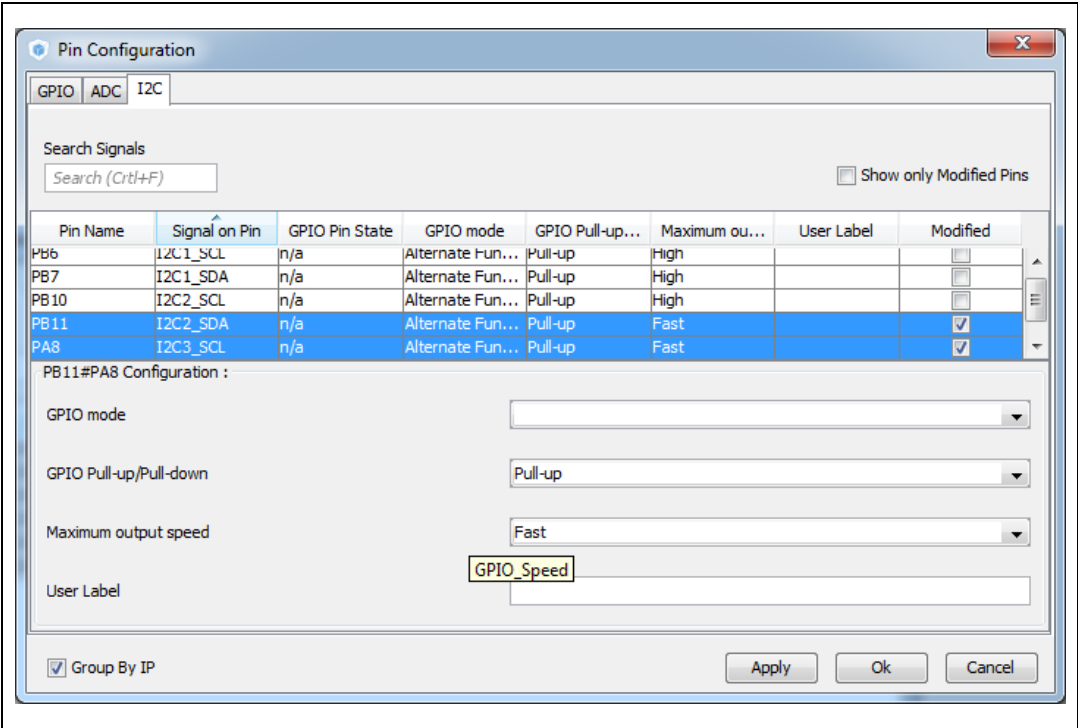
A **Group by IP** checkbox allows to group all instances of a peripheral under a same window (see [Figure 72](#)).

Figure 72. GPIO configuration grouped by IP



As shown in [Figure 73](#), row multi-selection can be performed to change a set of pins to a given configuration at the same time.

Figure 73. Multiple Pins Configuration



#### 4.12.4 DMA Configuration window

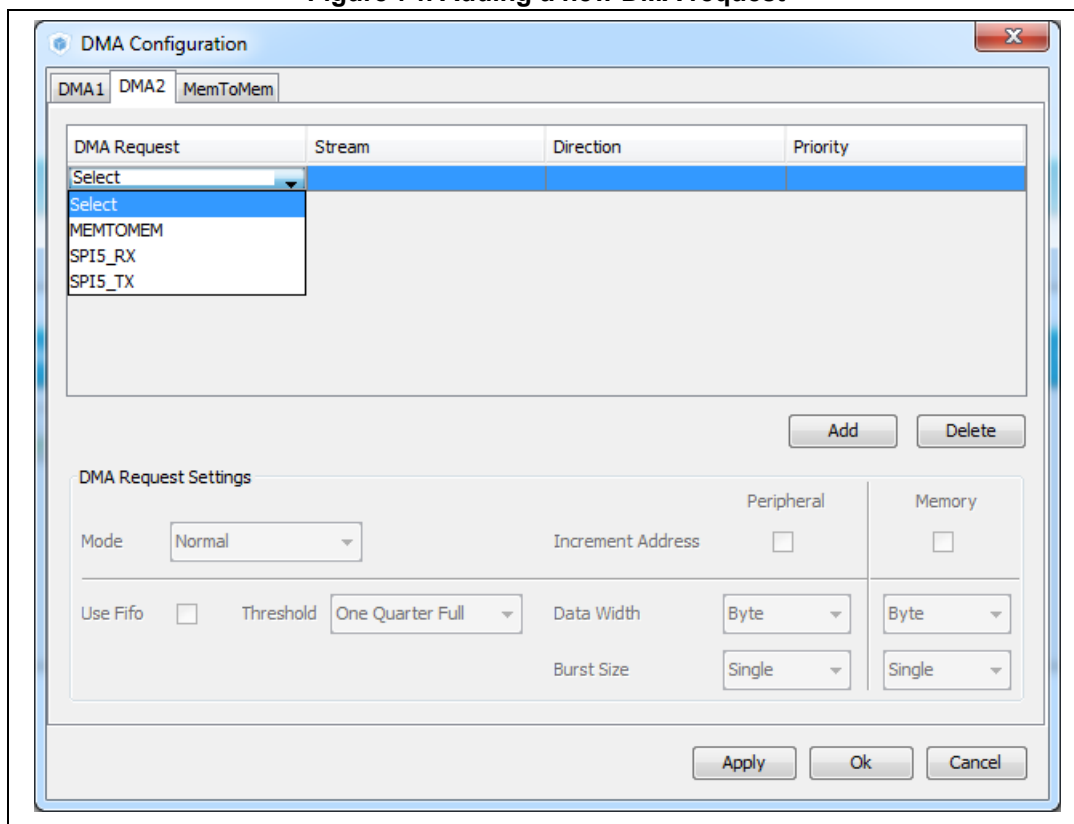
Click **DMA** in the **Configuration** pane to open the **DMA configuration** window.

This window allows to configure the generic DMA controllers available on the MCU. The DMA interfaces allow to perform data transfers between memories and peripherals while the CPU is running, and memory to memory transfers (if supported).

*Note:* Some IPs such as **USB** or **Ethernet**, have their own DMA controller, which is enabled by default or via the IP configuration window.

Clicking **Add** in the **DMA configuration** window adds a new line at the end of the DMA configuration window with a combo box proposing to choose between possible **DMA requests** to be mapped to peripherals signals (see [Figure 74](#)).

**Figure 74. Adding a new DMA request**

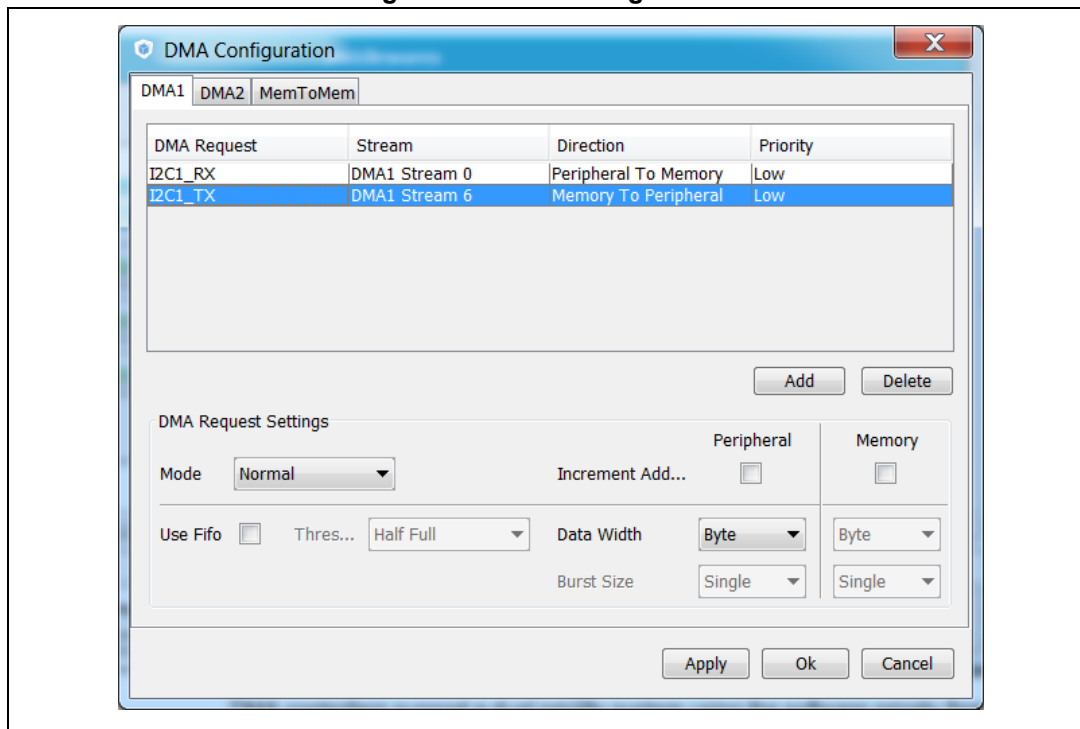


Selecting a DMA request automatically assigns a stream among all the streams available, a direction and a priority. When the DMA channel is configured, it is up to the application code to fully describe the DMA transfer run-time parameters such as the start address, etc....

The DMA request (called channel for STM32F4 MCUs) is used to reserve a stream to transfer data between peripherals and memories (see [Figure 75](#)). The stream priority will be used to decide which stream to select for the next DMA transfer.

DMA controllers support a dual priority system using the software priority first, and in case of equal software priorities, a hardware priority that is given by the stream number.

Figure 75. DMA Configuration



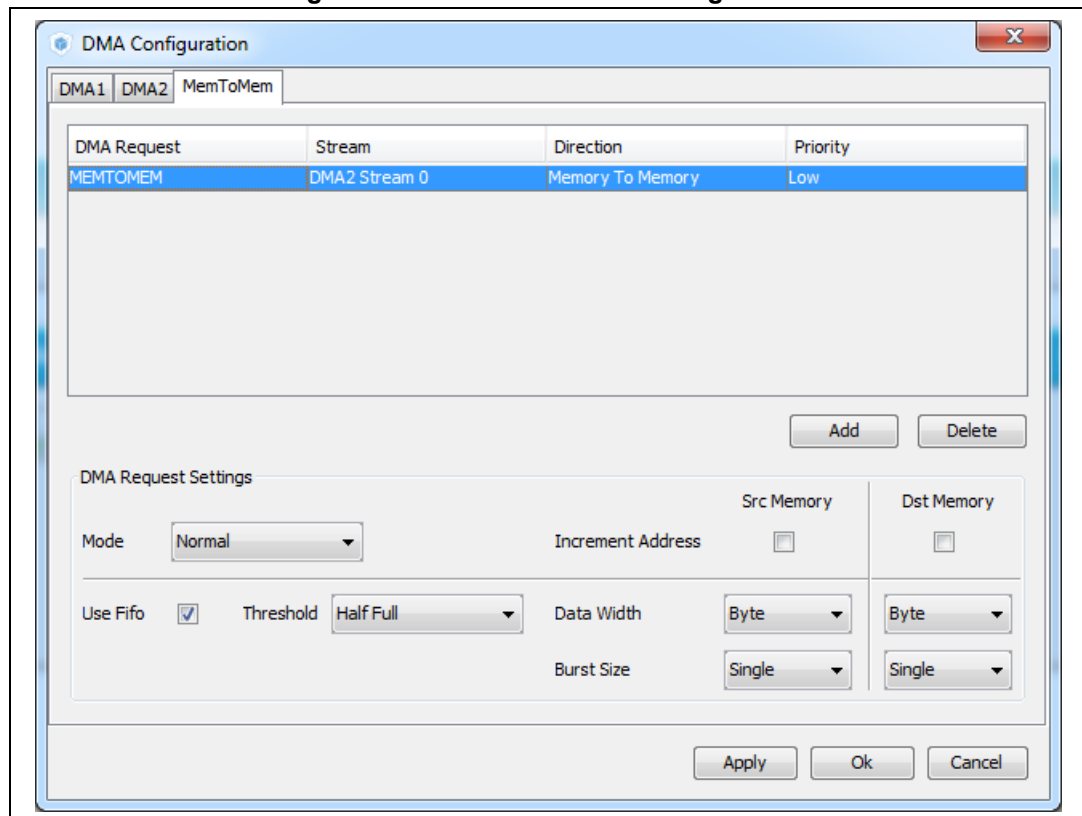
Additional DMA configuration settings can be done through the **DMA configuration** window:

- **Mode:** regular mode, circular mode, or peripheral flow controller mode (only available for the SDIO IP).
- **Increment Add:** the type of peripheral address and memory address increment (fixed or post-incremented in which case the address is incremented after each transfer). Click the checkbox to enable the post-incremented mode.
- **Peripheral data width:** 8, 16 or 32 bits
- Switching from the default direct mode to the *FIFO mode* with programmable *threshold*:
  - a) Click the **Use FIFO** checkbox.
  - b) Then, configure the **peripheral and memory data width** (8, 16 or 32 bits).
  - c) Select between **single transfer** and **burst transfer**. If you select burst transfer, choose a burst size (1, 4, 8 or 16).

In case of memory-to-memory transfer (MemtoMem), the DMA configuration applies to a source memory and a destination memory.



Figure 76. DMA MemToMem configuration



#### 4.12.5 NVIC Configuration window

Click **NVIC** in the Configuration pane to open the Nested Vector interrupt controller configuration window (see [Figure 77](#)).

Interrupt unmasking and interrupt handlers are managed within 2 tabs:

- The **NVIC** tab allows enabling peripheral interrupts in the NVIC controller and setting their priorities.
- The **Code generation** tab allows selecting options for interrupt related code generation.

##### Enabling interruptions using the NVIC tab view

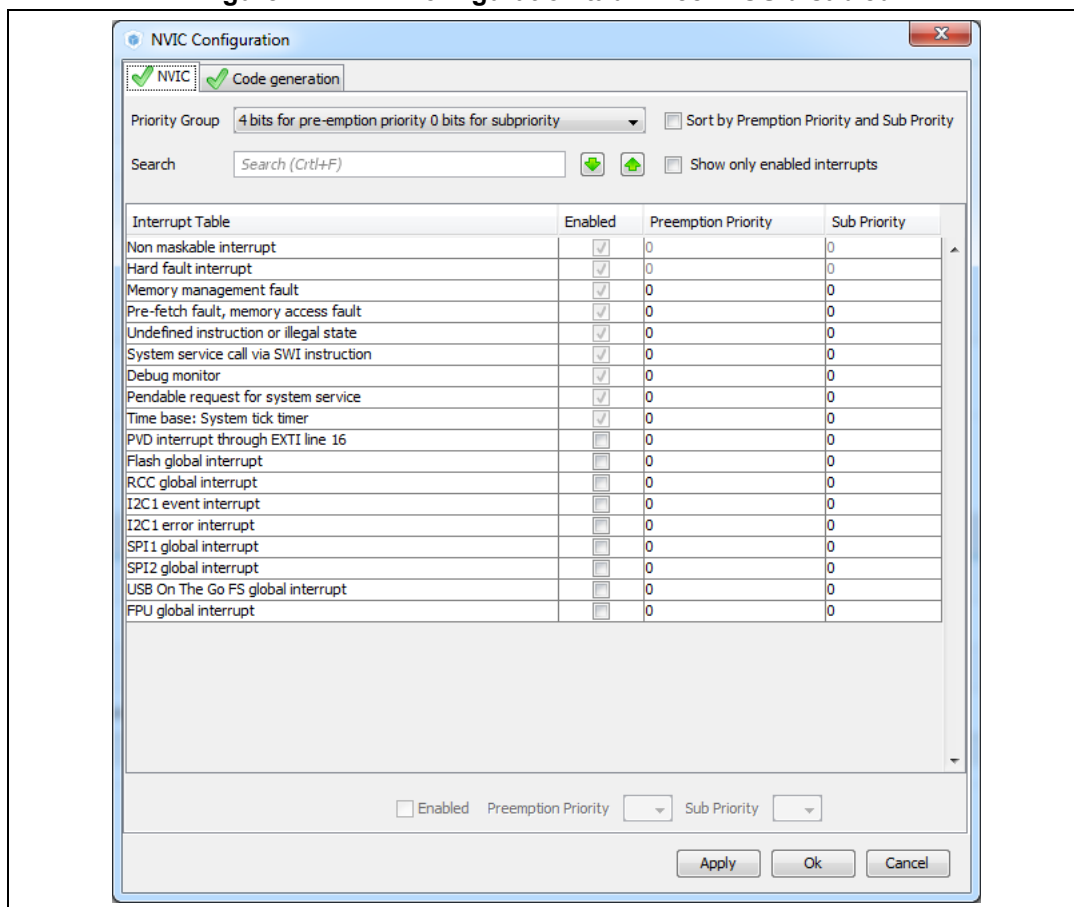
The **NVIC** view (see [Figure 77](#)) does not show all possible interrupts but only the ones available for the IPs selected in the **Pinout and Configuration** panes. System interrupts are displayed but can never be disabled.

Check/Uncheck the **Show only enabled interrupts** box to filter or not enabled interrupts.

Use the **search field** to filter out the interrupt vector table according to a string value. As an example, after enabling UART IPs from the **Pinout** pane, type UART in the NVIC search field and click the green arrow close to it: all UART interrupts are then displayed.

Enabling a **peripheral interrupt** will generate of NVIC function calls **HAL\_NVIC\_SetPriority** and **HAL\_NVIC\_EnableIRQ** for this peripheral.

Figure 77. NVIC Configuration tab - FreeRTOS disabled

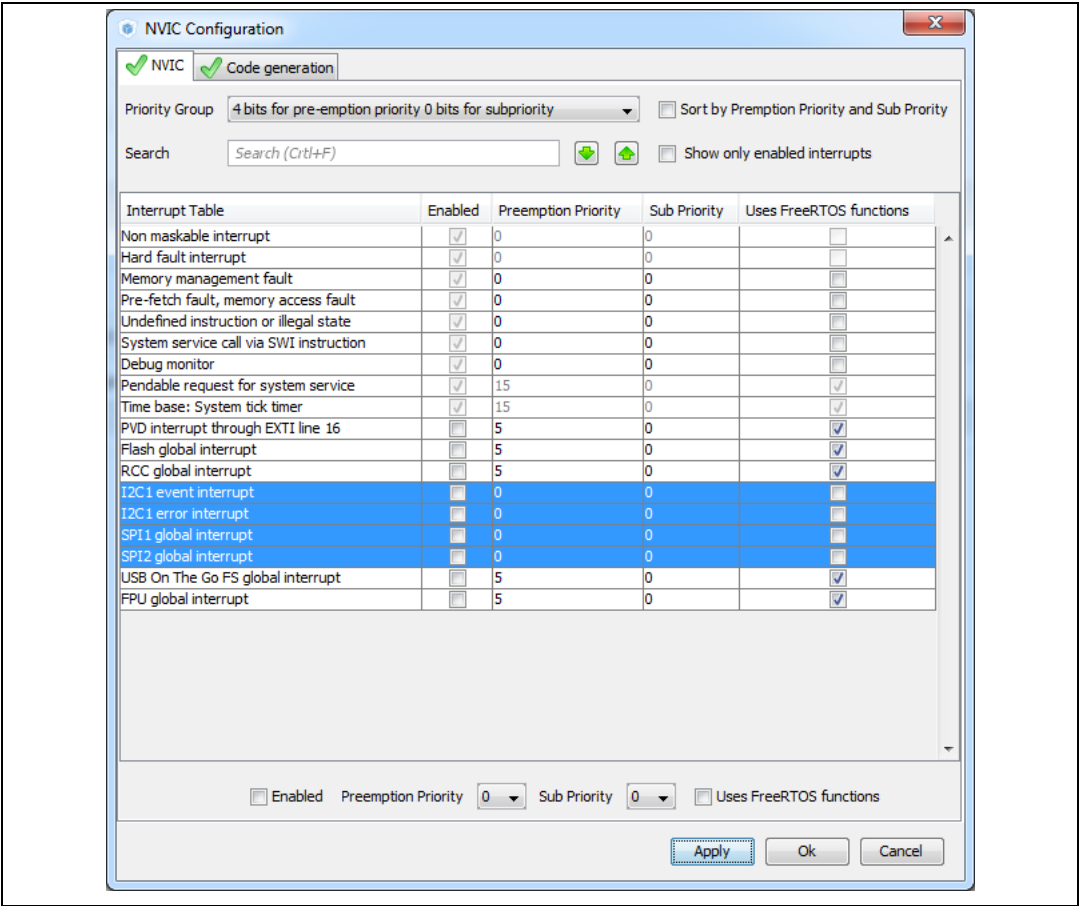


When FreeRTOS is enabled, an additional column is shown (see [Figure 78](#)). In this case, all the interrupt service routines (ISRs) that are calling the interrupt safe FreeRTOS APIs, should have a priority lower than the priority defined in the `LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY` parameter (the highest the value, the lowest the priority). The check in the corresponding checkbox guarantees that the restriction is applied.

If an ISR does not use such functions, the checkbox can be unchecked and any priority level can be set.

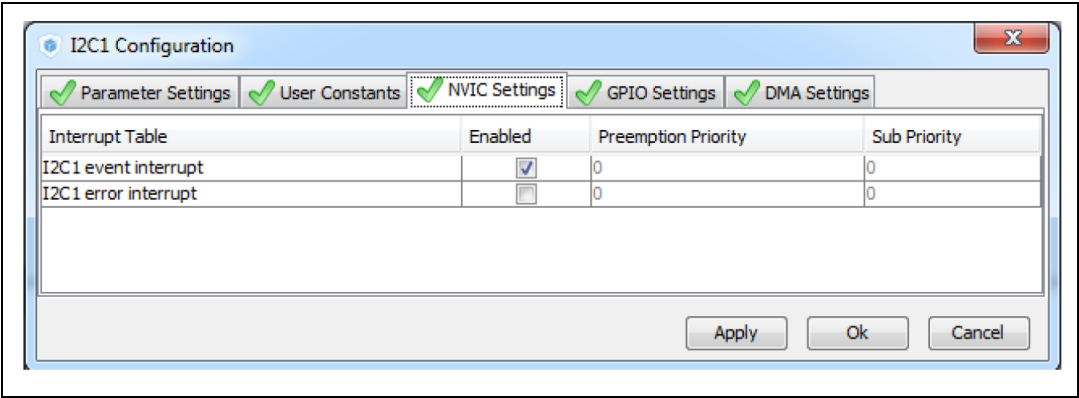
It is possible to check/uncheck multiple rows at a time (see rows highlighted in blue in [Figure 78](#)).

Figure 78. NVIC Configuration tab - FreeRTOS enabled



IP dedicated interrupts can also be accessed through the NVIC window in the IP configuration window (see [Figure 79](#)).

Figure 79. I2C NVIC Configuration window



STM32CubeMX NVIC configuration consists in selecting a priority group, enabling/disabling interrupts and configuring interrupts priority levels (preemption and sub-priority levels):

1. Select a **priority group**

Several bits allow to define NVIC priority levels. These bits are divided in two priority groups corresponding to two priority types: preemption priority and sub-priority. For example, in the case of STM32F4 MCUs, the NVIC priority group 0 corresponds to 0-bit preemption and 4-bit sub-priority.

2. In the interrupt table, click one or more rows to select one or more interrupt vectors. Use the widgets below the interrupt table to configure the vectors one by one or several at a time:

- **Enable checkbox:** check/uncheck to enable/disable the interrupt.
- **Preemption priority:** select a priority level. The preemption priority defines the ability of one interrupt to interrupt another.
- **Sub-priority:** select a priority level. The sub-priority defines the interrupt priority level.
- Click **Apply** to save changes, and **OK** to close the window.

### Code generation options for interrupt handling

The **Code Generation** view allows customizing the code generated for interrupt initialization and interrupt handlers:

- **Selection/Unselection of all interrupts for sequence ordering and IRQ handler code generation**

Use the checkboxes in front of the column names to configure all interrupts at a time (see [Figure 80](#)). Note that system interrupts are not eligible for init sequence reordering as the software solution does not control it.