BE ON D

UART communication using LL libraries

rev1.0 24/03/2020

GOAL

Transmit data via UART protocol when the user button interrupt occours

PREREQUISITES

Software needed:

- STM32IDE
- CoolTerm

Hardware used in this example:

- **NUCLEO-F446ZE**

UART communication using LL libraries

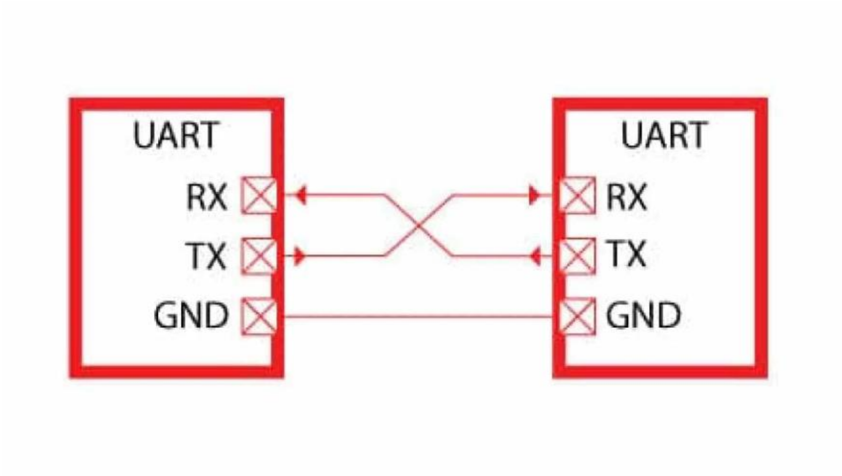
In this example:

- What’s UART
- Start a new project
- Configure the peripherals
- Generate code
- USART Configuration code
- GPIO Configuration code
- Run the program and troubleshooting

What’s UART

“UART” stands for Universal Asynchronous receiver-transmitter. It is a peripheral that is present inside a microcontroller. The **function of UART** is to convert the incoming and outgoing data into the serial binary stream.

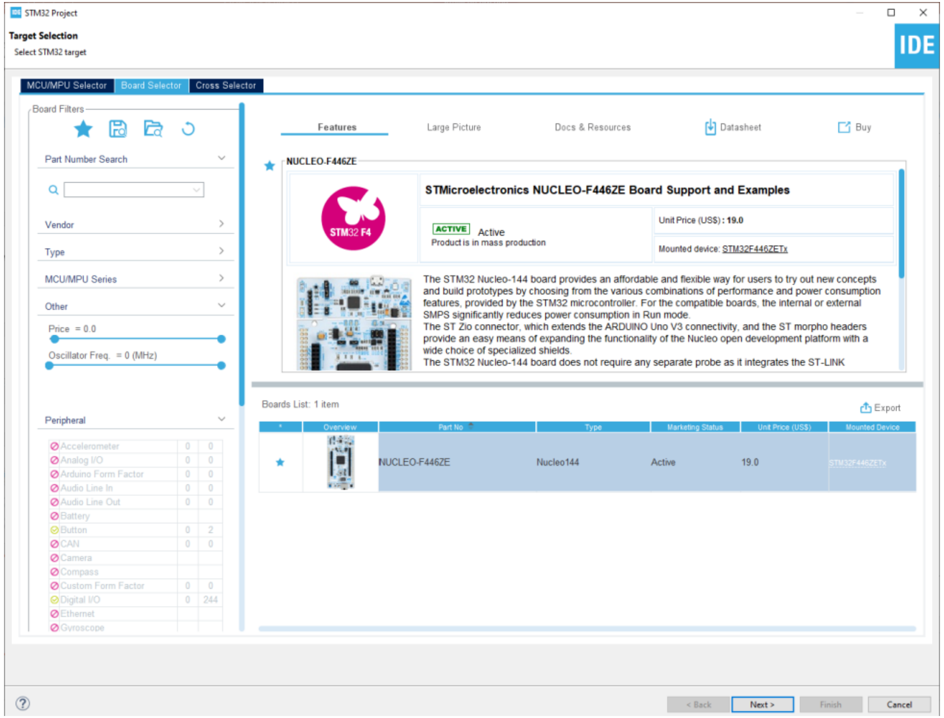
Notice that in UART communication the ***TX*** pin of the *transmitter* is connected to the ***RX*** pin of the *receiver* and viceversa.



Start a new project

From the stm32IDE software click on File -> New -> STM32 Project.

Select your board or your uC and click *next.*



Accelerometer	0	0
Analog I/O	0	0
Arduino Form Factor	0	0
Audio Line In	0	0
Audio Line Out	0	0
Battery		
Button	0	2
CAN	0	0
Camera		
Compass		
Custom Form Factor	0	0
Digital I/O	0	244
Ethernet		
Gyroscope		

NUCLEO-F446ZE

STMicroelectronics NUCLEO-F446ZE Board Support and Examples

ACTIVE Active
Product is in mass production

Unit Price (US$) : 19.0

Mounted device: STM32F446ZETx

The STM32 Nucleo-144 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller. For the compatible boards, the internal or external SMPS significantly reduces power consumption in Run mode.
The ST Zio connector, which extends the ARDUINO Uno V3 connectivity, and the ST morpho headers provide an easy means of expanding the functionality of the Nucleo open development platform with a wide choice of specialized shields.
The STM32 Nucleo-144 board does not require any separate probe as it integrates the ST-LINK

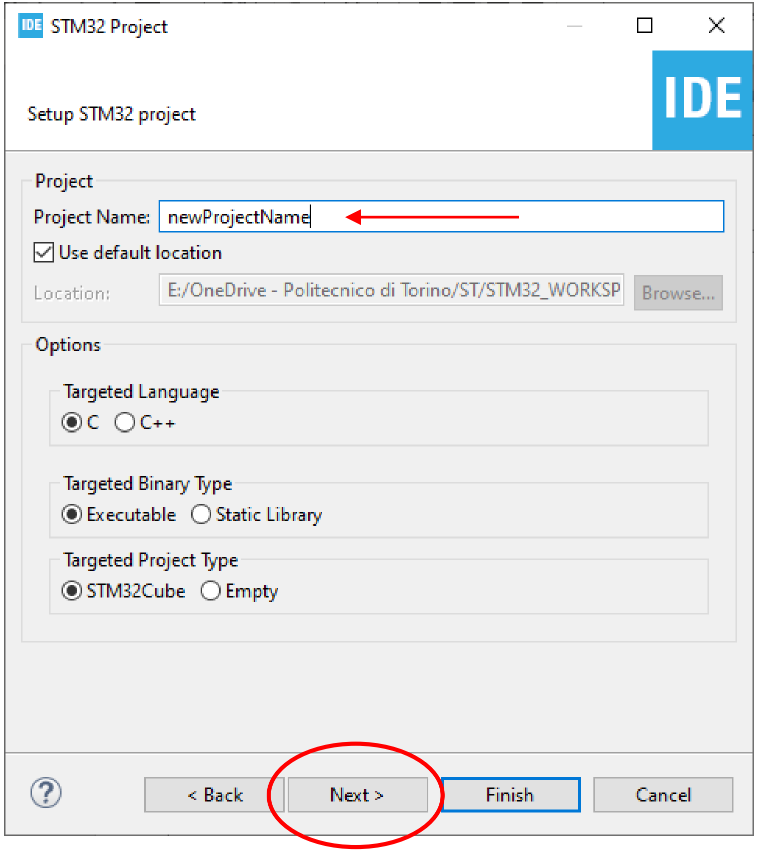
Boards List: 1 item

*	Overview	Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
★		NUCLEO-F446ZE	Nucleo144	Active	19.0	STM32F446ZETx

Start a new project

Type the name of your project and click next.

By default the project will be created in the workspace folder.



STM32 Project

Setup STM32 project

Project

Project Name: newProjectName

☑ Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP

Options

Targeted Language

● C ○ C++

Targeted Binary Type

● Executable ○ Static Library

Targeted Project Type

● STM32Cube ○ Empty

Start a new project

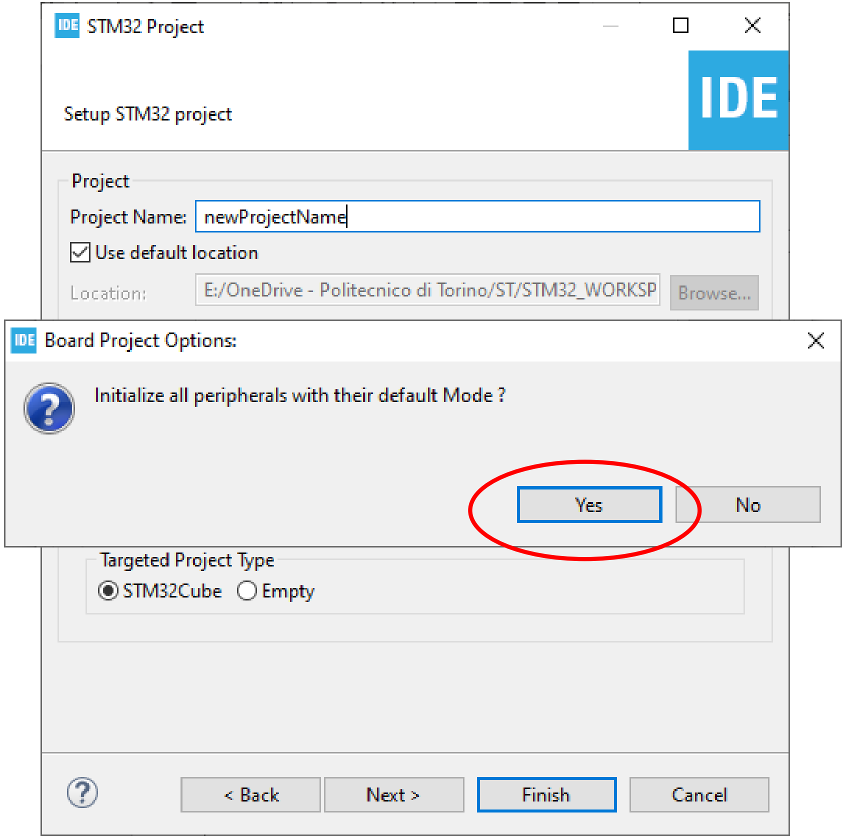
Type the name of your project and click next.

By default the project will be created in the *workspace* folder.

The *STM32IDE* has the option to initialize all the peripheral with their ***default*** mode:

Clicking *Yes* the *USART3*, all the *LEDs* and the blue *UserButton* will be configured as default.

Click *Yes*.



STM32 Project

Setup STM32 project

Project

Project Name: newProjectName

☑ Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP

Board Project Options:

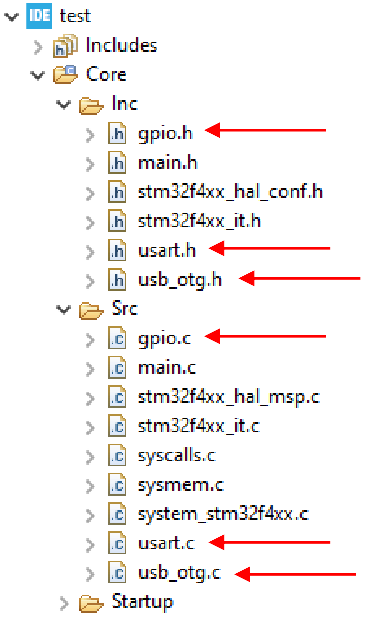
Initialize all peripherals with their default Mode ?

Targeted Project Type

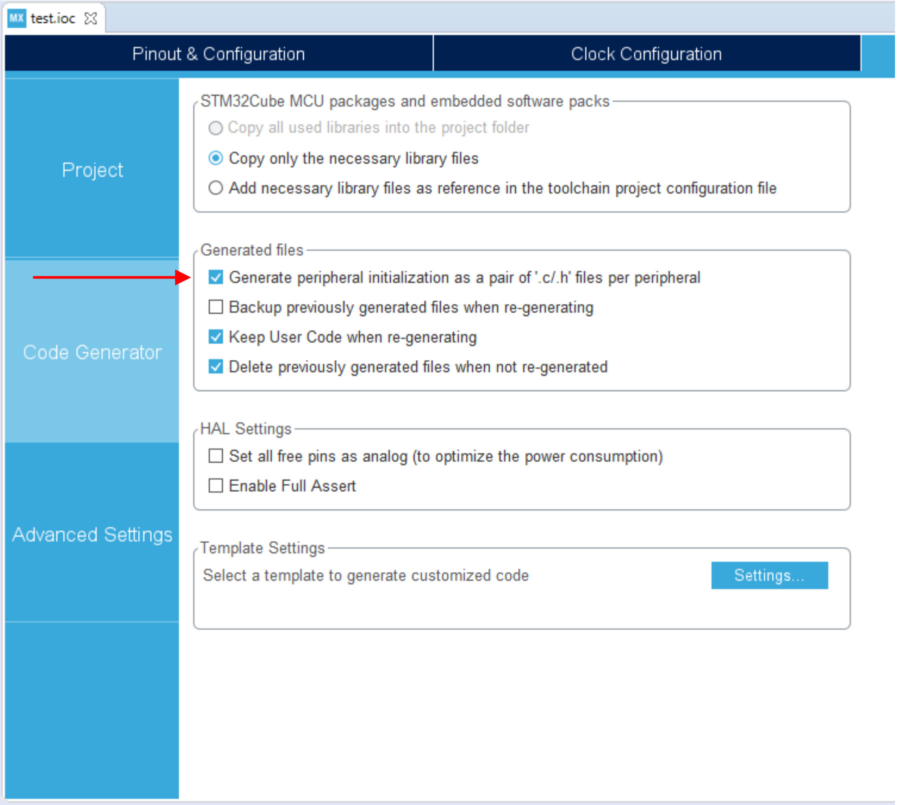
● STM32Cube ○ Empty

Project Manager

In the Code Generator Tab check the ***Generate peripheral initialization […]*** box: each periperhal will have a disting *periph.c* and *periph.h* files.



```
test
  Includes
  Core
    Inc
      gpio.h
      main.h
      stm32f4xx_hal_conf.h
      stm32f4xx_it.h
      usart.h
      usb_otg.h
    Src
      gpio.c
      main.c
      stm32f4xx_hal_msp.c
      stm32f4xx_it.c
      syscalls.c
      sysmem.c
      system_stm32f4xx.c
      usart.c
      usb_otg.c
    Startup
```



STM32Cube MCU packages and embedded software packs

○ Copy all used libraries into the project folder
● Copy only the necessary library files
○ Add necessary library files as reference in the toolchain project configuration file

Generated files

☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
□ Backup previously generated files when re-generating
☑ Keep User Code when re-generating
☑ Delete previously generated files when not re-generated

HAL Settings

□ Set all free pins as analog (to optimize the power consumption)
□ Enable Full Assert

Template Settings

Select a template to generate customized code

Configure the peripherals

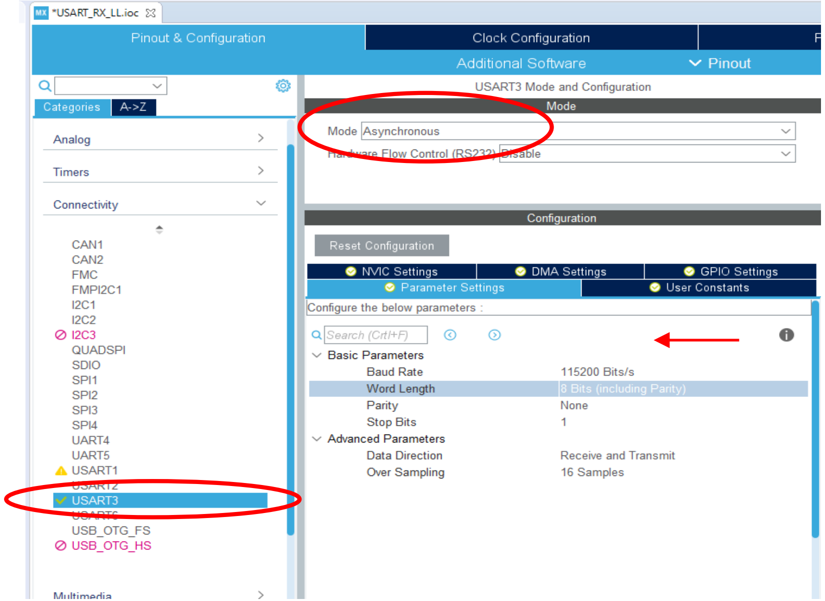
The main core of this project is the USART peripheral, let’s have a look in the USART page configuration.

For this example we will use the USART3:

- *Mode*: Asynchronous
- *Baud Rate*: 115200 Bits/s

The *Baud Rate* is the speed of the serial communication, 115200 Bits/s is a reasonable speed.

Don’t forget to enable the *interrupt* under the **NVIC** tab.



USART3 Mode and Configuration

Mode

Mode: Asynchronous

Hardware Flow Control (RS232): Disable

Configuration

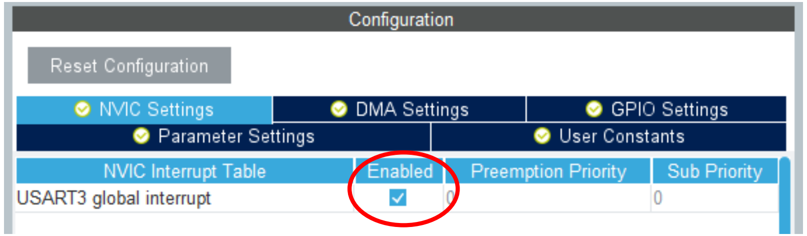
Configure the below parameters :

Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples



Configuration

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART3 global interrupt	☑	0	0

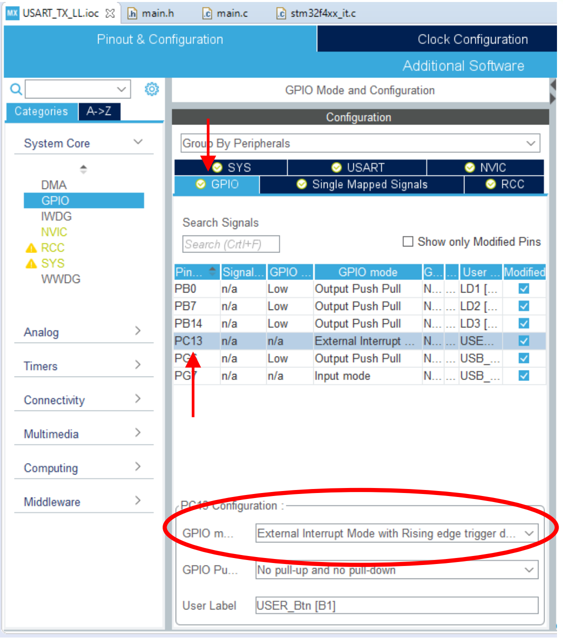
Configure the peripherals

For this example, we will send a character when an interrupt generated by a GPIO pins occours.

Let use the user button as our trigger source.

Under the **GPIO Tab** select the *GPIO pin* (the blu user button is connected to GPIO PC13) and check that the **GPIO mode** is setted as ***External Interrupt Mode***

Don’t forget to enable the interrupt under the **NVIC** tab.



Search Signals

Search (Crtl+F)

□ Show only Modified Pins

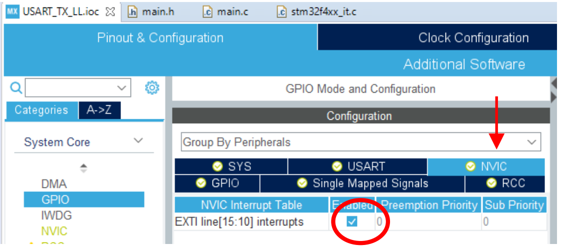
Pin...	Signal...	GPIO ...	GPIO mode	G...	...	User ...	Modified
PB0	n/a	Low	Output Push Pull	N...	...	LD1 [...	☑
PB7	n/a	Low	Output Push Pull	N...	...	LD2 [...	☑
PB14	n/a	Low	Output Push Pull	N...	...	LD3 [...	☑
PC13	n/a	n/a	External Interrupt ...	N...	...	USE...	☑
PG6	n/a	Low	Output Push Pull	N...	...	USB_...	☑
PG7	n/a	n/a	Input mode	N...	...	USB_...	☑

PC13 Configuration :

GPIO m... External Interrupt Mode with Rising edge trigger d...

GPIO Pu... No pull-up and no pull-down

User Label USER_Btn [B1]



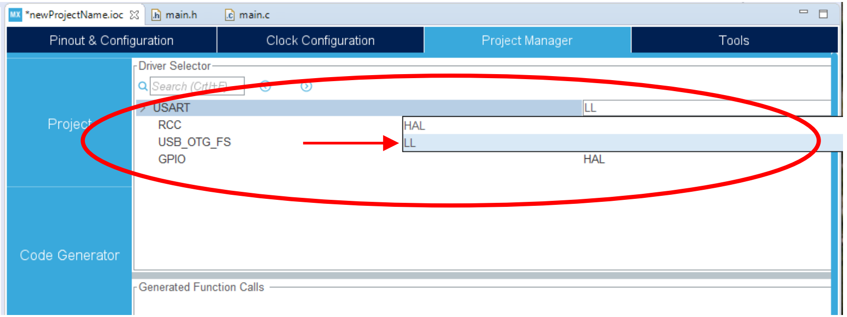
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line[15:10] interrupts	☑	0	0

Generate code

The last step before generating the code is to select LL libraries for manage the USART peripheral.

Go in *Project Manager -> Advanced Configuration* and select ***LL*** (USART and GPIO).

Then generate the code (click on the generate icon ⚙)



Driver Selector

USART	LL
RCC	HAL
USB_OTG_FS	LL
GPIO	HAL

Generated Function Calls

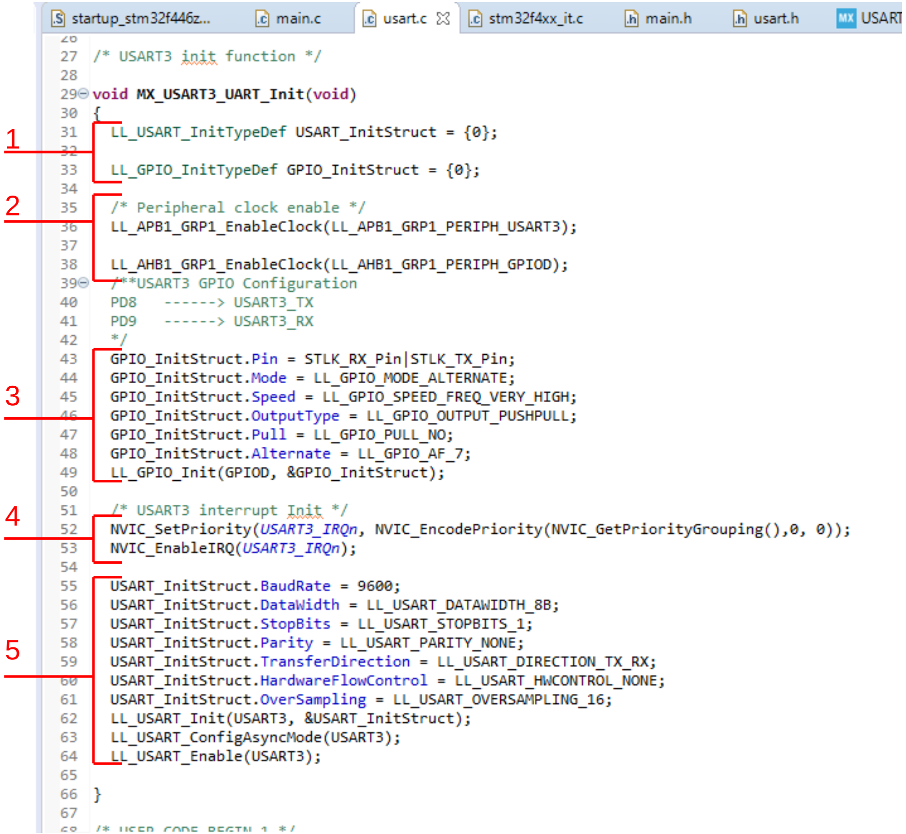
USART Configuration

Let’s have a look in the **USART** configuration function made by *CubeMX*.

As always we can find it in the **usart.c** file

We can subdivide this section in 5 parts:

1. Inizialization of the GPIO and USART structures, they contains all the configuration info for the peripherals.
2. Enable of the clock sources for the GPIO pins and for the USART bus.

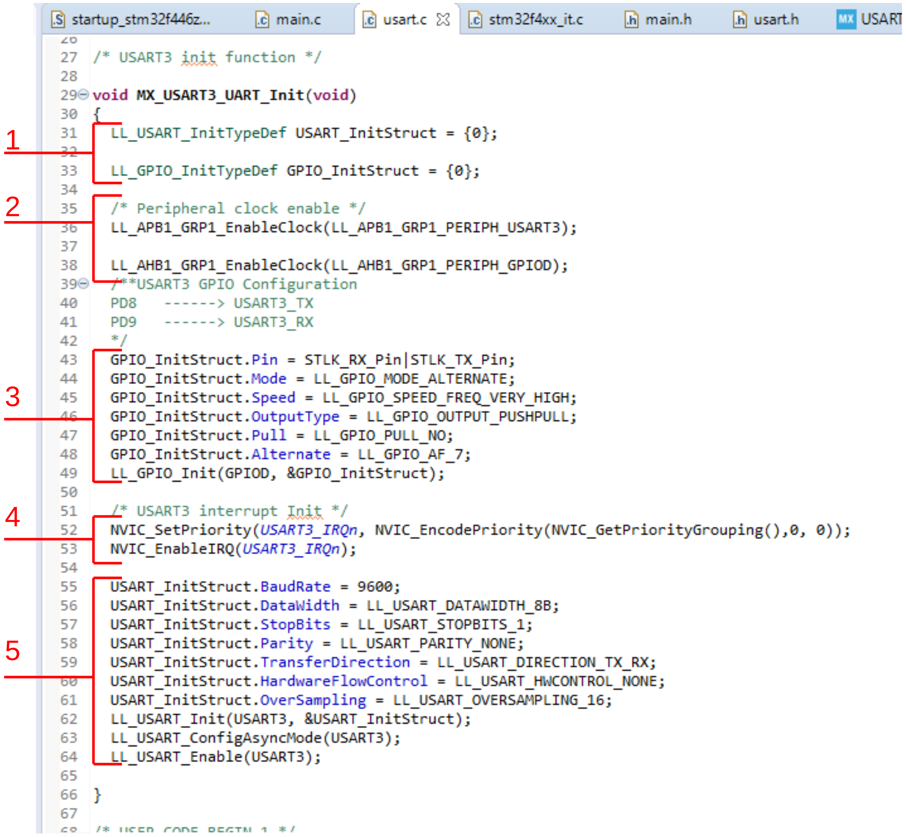


```
/* USART3 init function */

void MX_USART3_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_USART3);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  /**USART3 GPIO Configuration
  PD8   ------> USART3_TX
  PD9   ------> USART3_RX
  */
  GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* USART3 interrupt Init */
  NVIC_SetPriority(USART3_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(USART3_IRQn);

  USART_InitStruct.BaudRate = 9600;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
  LL_USART_Init(USART3, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART3);
  LL_USART_Enable(USART3);

}

/* USER CODE BEGIN 1 */
```

USART Configuration

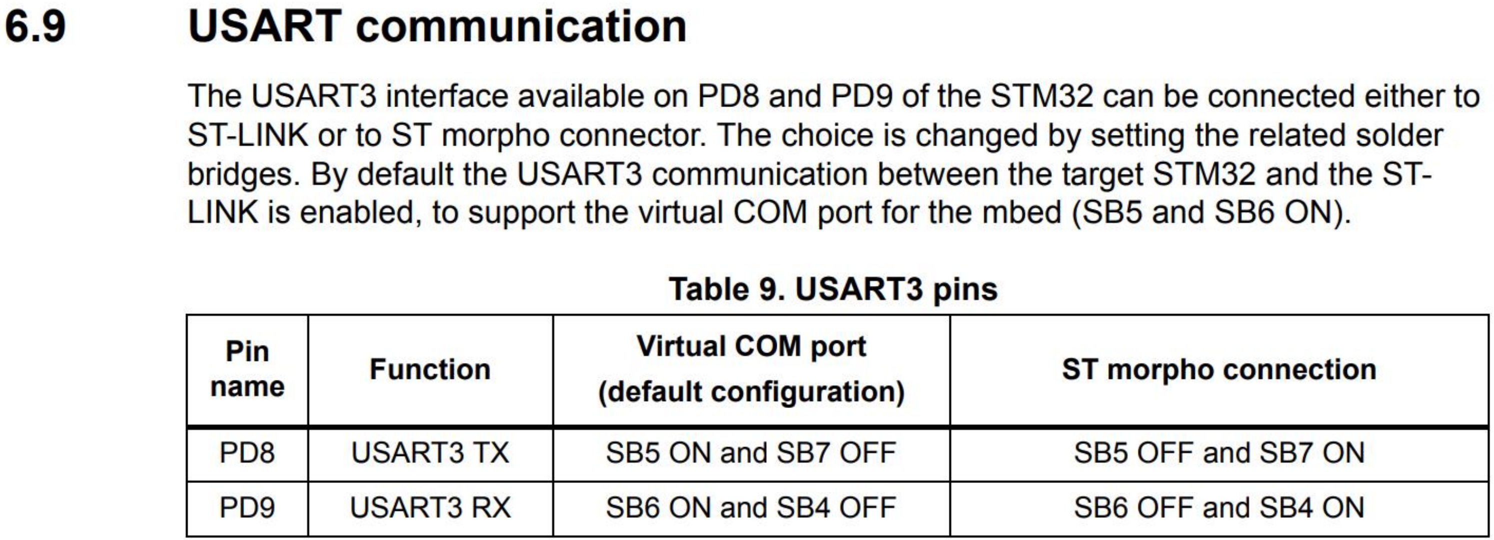
3. Configuration of the GPIO pins required for the communication.
   In this example we use USART3 since the RX and TX pins PD8 and PD9 are directly connected to the Nucleo ST-Link (the upper part of the board that allows the communication between the uC and the PC) by default. For other configurations have a look in the manual here.

4. Initialization of the interrupts and priorities.



```
/* USART3 init function */

void MX_USART3_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_USART3);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  /**USART3 GPIO Configuration
  PD8   ------> USART3_TX
  PD9   ------> USART3_RX
  */
  GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* USART3 interrupt Init */
  NVIC_SetPriority(USART3_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(USART3_IRQn);

  USART_InitStruct.BaudRate = 9600;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
  LL_USART_Init(USART3, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART3);
  LL_USART_Enable(USART3);

}

/* USER CODE BEGIN 1 */
```

Usart configuration



6.9 USART communication

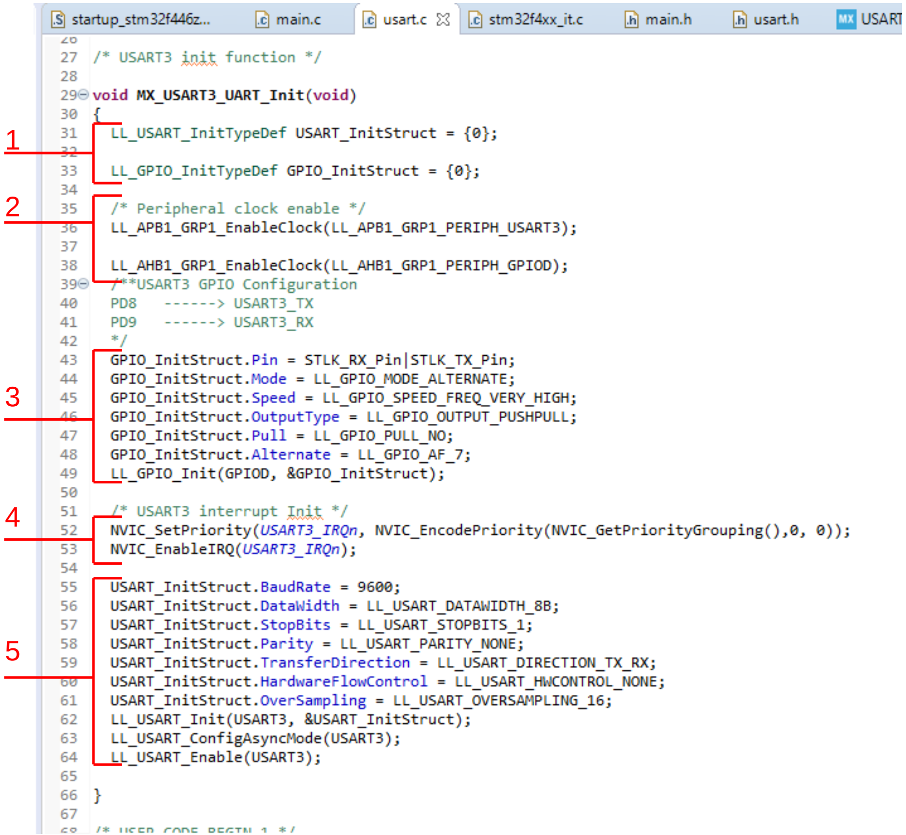
The USART3 interface available on PD8 and PD9 of the STM32 can be connected either to ST-LINK or to ST morpho connector. The choice is changed by setting the related solder bridges. By default the USART3 communication between the target STM32 and the ST-LINK is enabled, to support the virtual COM port for the mbed (SB5 and SB6 ON).

Table 9. USART3 pins

Pin name	Function	Virtual COM port (default configuration)	ST morpho connection
PD8	USART3 TX	SB5 ON and SB7 OFF	SB5 OFF and SB7 ON
PD9	USART3 RX	SB6 ON and SB4 OFF	SB6 OFF and SB4 ON

USART Configuration

5. Configuration of the USART peripheral. Here it’s possible to see some values like the *BaudRate*, *StopBits*, *Parity, ecc*.



```c
/* USART3 init function */

void MX_USART3_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_USART3);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  /**USART3 GPIO Configuration
  PD8   ------> USART3_TX
  PD9   ------> USART3_RX
  */
  GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* USART3 interrupt Init */
  NVIC_SetPriority(USART3_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(USART3_IRQn);

  USART_InitStruct.BaudRate = 9600;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
  LL_USART_Init(USART3, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART3);
  LL_USART_Enable(USART3);

}

/* USER CODE BEGIN 1 */
```

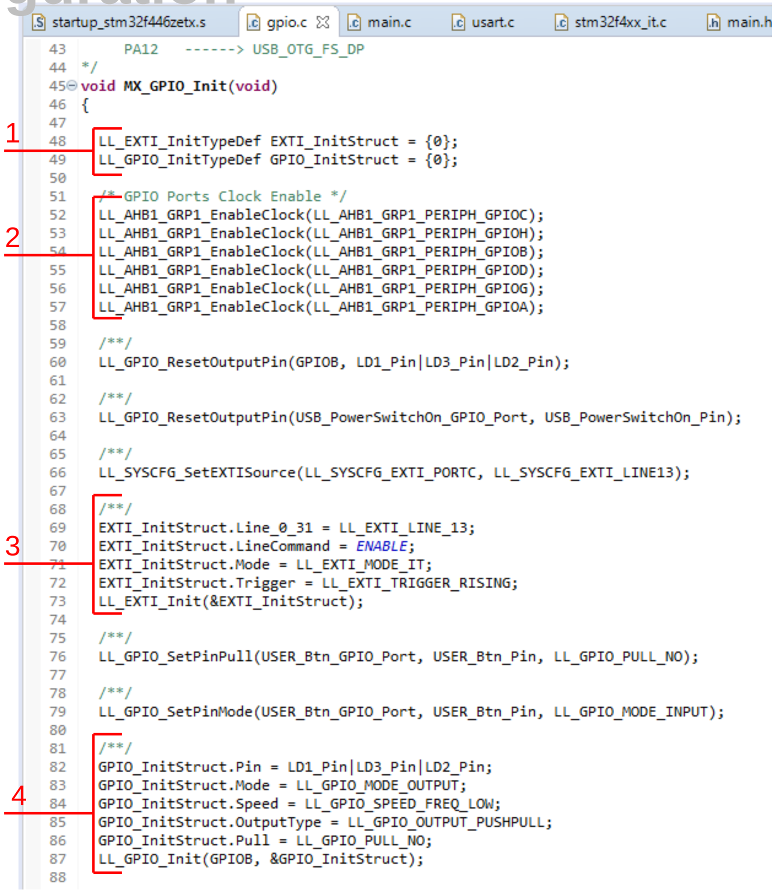
GPIO Configuration

Let’s have a look in the **GPIO** configuration function made by *CubeMX*.

As always we can find it in the ***gpio.c*** file.

The most important parts are the following:

1. Inizialization of the *GPIO* and *INTERRUPT* structures, they contains all the configuration info for the peripherals.
2. Enable of the clock sources for all the GPIO ports.

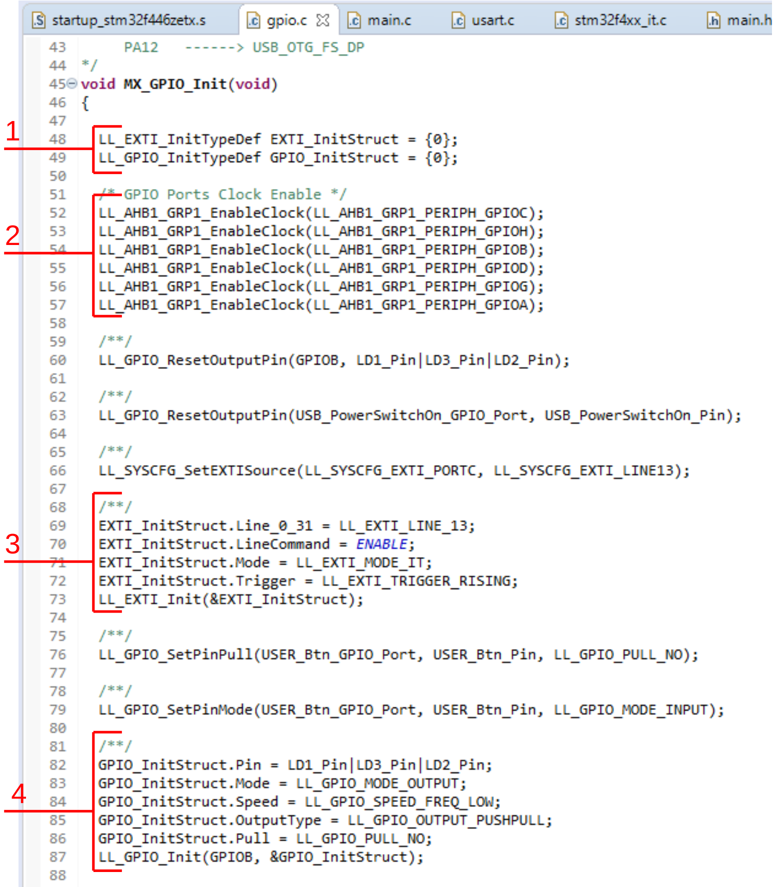


```
      PA12   ------> USB_OTG_FS_DP
*/
void MX_GPIO_Init(void)
{

  LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOC);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOH);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOG);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);

  /**/
  LL_GPIO_ResetOutputPin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin);

  /**/
  LL_GPIO_ResetOutputPin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin);

  /**/
  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTC, LL_SYSCFG_EXTI_LINE13);

  /**/
  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_13;
  EXTI_InitStruct.LineCommand = ENABLE;
  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
  LL_EXTI_Init(&EXTI_InitStruct);

  /**/
  LL_GPIO_SetPinPull(USER_Btn_GPIO_Port, USER_Btn_Pin, LL_GPIO_PULL_NO);

  /**/
  LL_GPIO_SetPinMode(USER_Btn_GPIO_Port, USER_Btn_Pin, LL_GPIO_MODE_INPUT);

  /**/
  GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin|LD2_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

```

GPIO Configuration

3. Configuration of the interrupt structure:

- *Line*: Line of the interrupt where the pin is connected (more info in the datasheet)
- *LineCommand*: State of the line, ENABLE means that the interrupt is enabled
- *Trigger*: Select the edge of triggering (see the button Interrupt report for more Info)

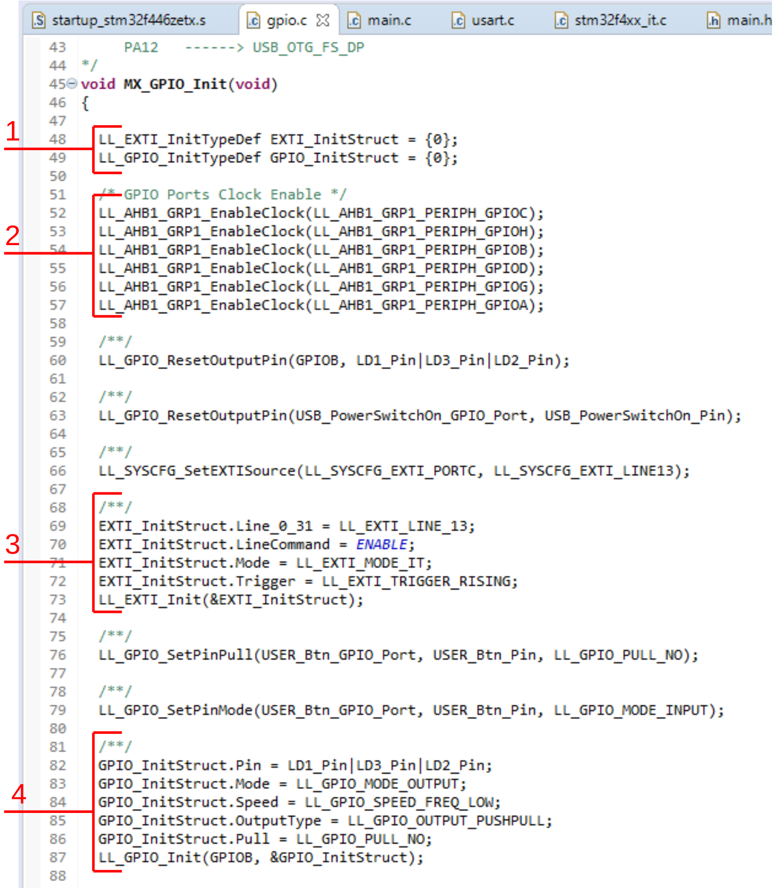


```
      PA12   ------> USB_OTG_FS_DP
*/
void MX_GPIO_Init(void)
{

  LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOC);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOH);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOG);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);

  /**/
  LL_GPIO_ResetOutputPin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin);

  /**/
  LL_GPIO_ResetOutputPin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin);

  /**/
  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTC, LL_SYSCFG_EXTI_LINE13);

  /**/
  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_13;
  EXTI_InitStruct.LineCommand = ENABLE;
  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
  LL_EXTI_Init(&EXTI_InitStruct);

  /**/
  LL_GPIO_SetPinPull(USER_Btn_GPIO_Port, USER_Btn_Pin, LL_GPIO_PULL_NO);

  /**/
  LL_GPIO_SetPinMode(USER_Btn_GPIO_Port, USER_Btn_Pin, LL_GPIO_MODE_INPUT);

  /**/
  GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin|LD2_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

```

GPIO Configuration

4. Configuration of the GPIO structure for the LED’s:

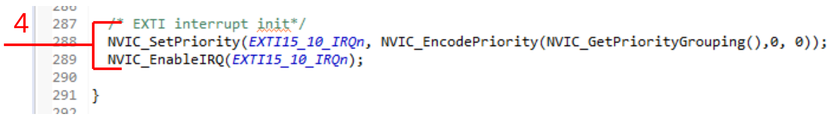
   - *Pin*: GPIO Pins selected
   - *Mode*: set the pin as Output, Input, Analog Input, etc
   - *Speed*: Control the speed of GPIO pins.
   - *Pull*: Type of pull, choose between Pull-down, Pull-up or no-pull (see report on GPIO inputs for more info)



```
       PA12   ------> USB_OTG_FS_DP
*/
void MX_GPIO_Init(void)
{

  LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOC);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOH);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOG);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);

  /**/
  LL_GPIO_ResetOutputPin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin);

  /**/
  LL_GPIO_ResetOutputPin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin);

  /**/
  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTC, LL_SYSCFG_EXTI_LINE13);

  /**/
  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_13;
  EXTI_InitStruct.LineCommand = ENABLE;
  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
  LL_EXTI_Init(&EXTI_InitStruct);

  /**/
  LL_GPIO_SetPinPull(USER_Btn_GPIO_Port, USER_Btn_Pin, LL_GPIO_PULL_NO);

  /**/
  LL_GPIO_SetPinMode(USER_Btn_GPIO_Port, USER_Btn_Pin, LL_GPIO_MODE_INPUT);

  /**/
  GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin|LD2_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

```

GPIO Configuration

4. Set the interrupt priorities.



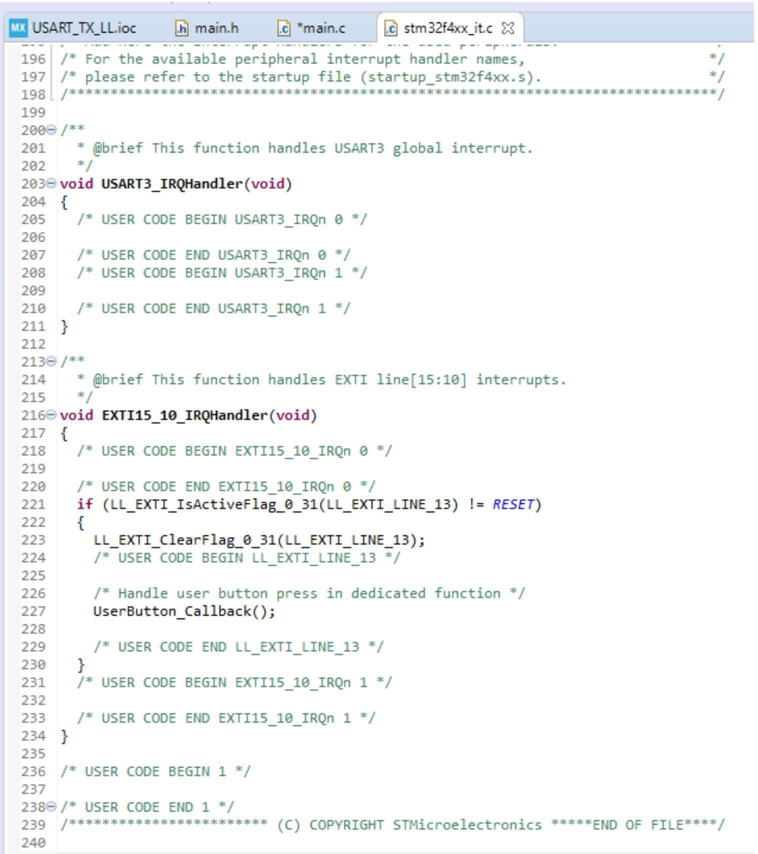
```
  /* EXTI interrupt init*/
  NVIC_SetPriority(EXTI15_10_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(EXTI15_10_IRQn);

}
```

USART Handler

When the interrupt is triggered, the program calls the USART handler, defined in the ***stm32f4xx_it.c*** file.

At this point we check if interrupt has been generated correctly and eventually calls the ***UserButton_Callback()*** that we now define in the gpio.c file.

As you can see we check only the interrupt generated from the button, the usart handler should remain empty in this case.



```c
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f4xx.s).                    */
/******************************************************************************/

/**
  * @brief This function handles USART3 global interrupt.
  */
void USART3_IRQHandler(void)
{
  /* USER CODE BEGIN USART3_IRQn 0 */

  /* USER CODE END USART3_IRQn 0 */
  /* USER CODE BEGIN USART3_IRQn 1 */

  /* USER CODE END USART3_IRQn 1 */
}

/**
  * @brief This function handles EXTI line[15:10] interrupts.
  */
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_13) != RESET)
  {
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_13);
    /* USER CODE BEGIN LL_EXTI_LINE_13 */

    /* Handle user button press in dedicated function */
    UserButton_Callback();

    /* USER CODE END LL_EXTI_LINE_13 */
  }
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

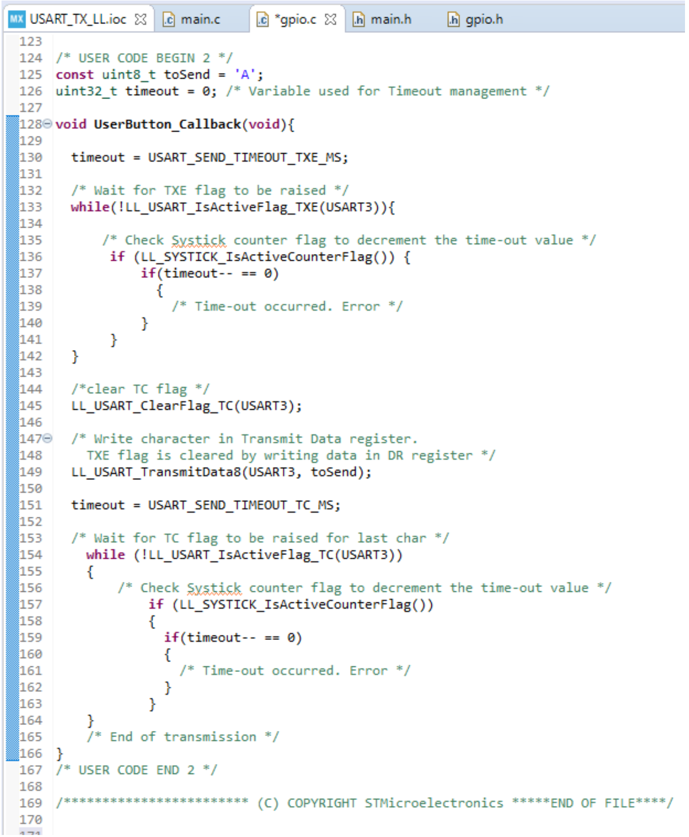
UserButton CallBack

Now go to the *gpio.c* file and check the ***UserButton_Callback()*** function.

At first we wait that the **Transmit Data Register Empty** (TXE) flag raises, so we know that there are no messages that have to be sent, then we clear the **Transmission Complete Flag** (TC).

Now we are ready to send data, so we send it using the ***LL_USART_TransmirData8()*** function and then we check the TC flag for the end of the trasmission.

Notice that we used some timeout to check for communication errors.

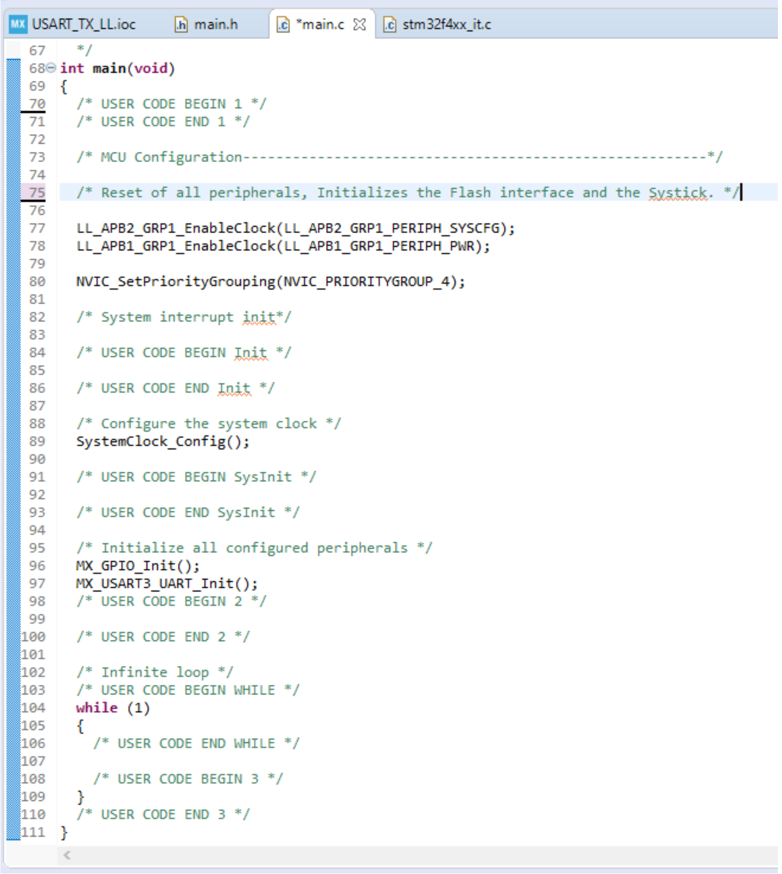


```c
/* USER CODE BEGIN 2 */
const uint8_t toSend = 'A';
uint32_t timeout = 0; /* Variable used for Timeout management */

void UserButton_Callback(void){

  timeout = USART_SEND_TIMEOUT_TXE_MS;

  /* Wait for TXE flag to be raised */
  while(!LL_USART_IsActiveFlag_TXE(USART3)){

      /* Check Systick counter flag to decrement the time-out value */
       if (LL_SYSTICK_IsActiveCounterFlag()) {
           if(timeout-- == 0)
             {
               /* Time-out occurred. Error */
           }
       }
  }

  /*clear TC flag */
  LL_USART_ClearFlag_TC(USART3);

  /* Write character in Transmit Data register.
    TXE flag is cleared by writing data in DR register */
  LL_USART_TransmitData8(USART3, toSend);

  timeout = USART_SEND_TIMEOUT_TC_MS;

  /* Wait for TC flag to be raised for last char */
    while (!LL_USART_IsActiveFlag_TC(USART3))
    {
        /* Check Systick counter flag to decrement the time-out value */
            if (LL_SYSTICK_IsActiveCounterFlag())
            {
              if(timeout-- == 0)
              {
                /* Time-out occurred. Error */
              }
            }
    }
    /* End of transmission */
}
/* USER CODE END 2 */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Empty Loop

Since our programs manage all the functions via the interrupts, the main loop is empty.

The coding part ends here, now simply download the program to your board.



```
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */

  LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_SYSCFG);
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);

  NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* System interrupt init*/

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Receive data via UART

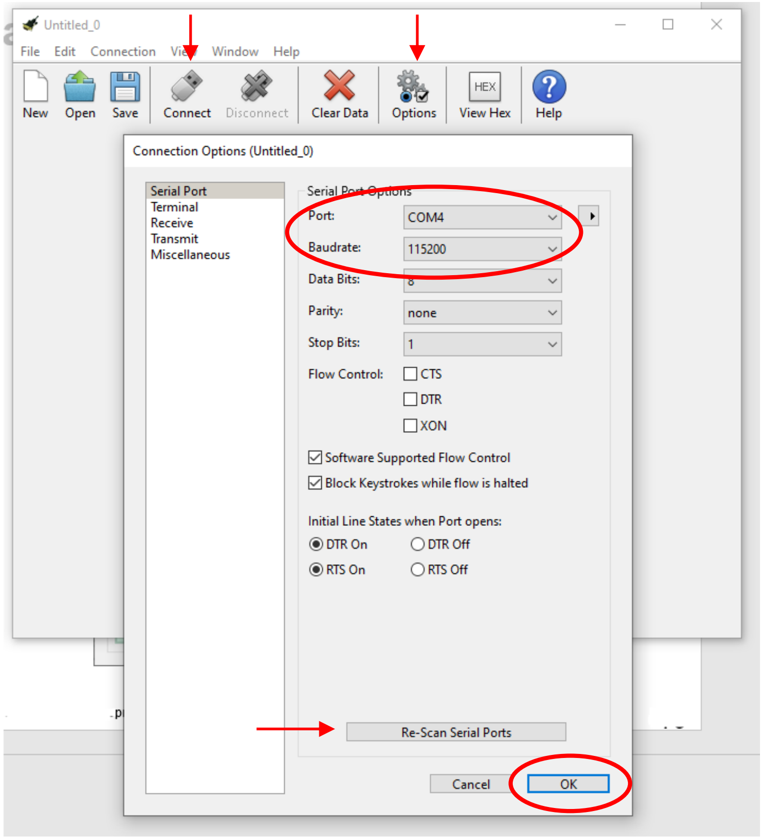
In this example we send the *UART* data via *PC*. You can use any Serial interface program you want, in this case we will use CoolTerm (link).

Run the program and select the correct settings in order to communicate with the board.

Remeber to select the correct *COM Port* and *BaudRate.*

Once done, click on *Connect.*

If you have troubles finding your board COM Port in the COM Port list try to click *Re-Scan Serial Ports* and check again.



Connection Options (Untitled_0)

Serial Port Options

Port: COM4

Baudrate: 115200

Data Bits: 8

Parity: none

Stop Bits: 1

Flow Control: □ CTS
□ DTR
□ XON

☑ Software Supported Flow Control

☑ Block Keystrokes while flow is halted

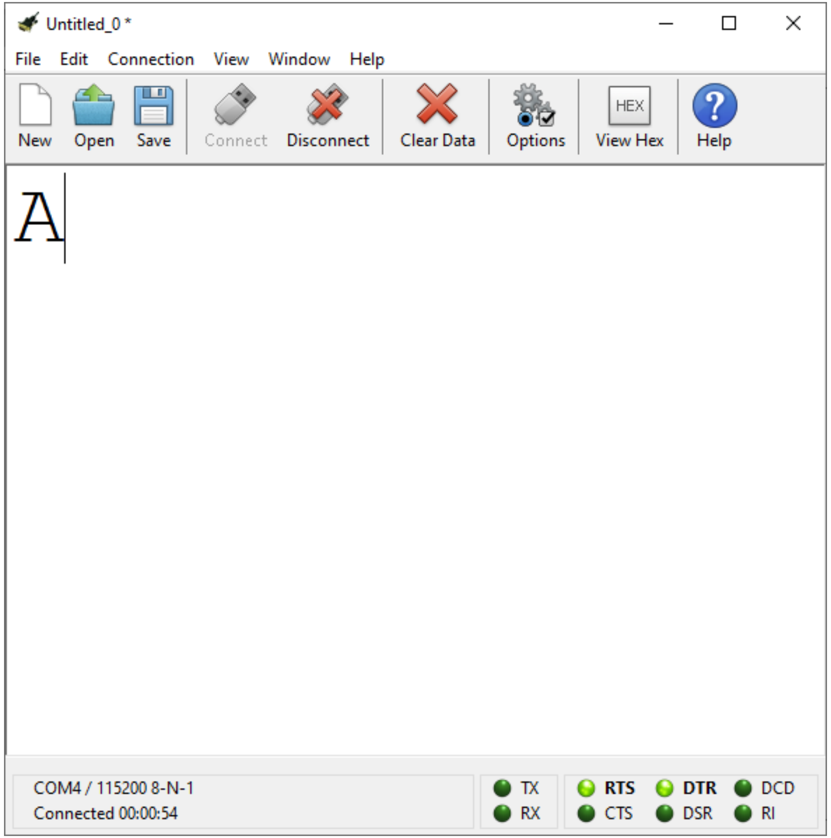
Initial Line States when Port opens:

● DTR On ○ DTR Off

● RTS On ○ RTS Off

Receive data via UART

If it’s all correct the character *‘A’* will appear at each press of the blue button on the board.



```
A
```

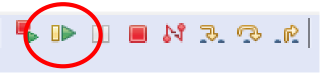
COM4 / 115200 8-N-1
Connected 00:00:54

Troubleshooting

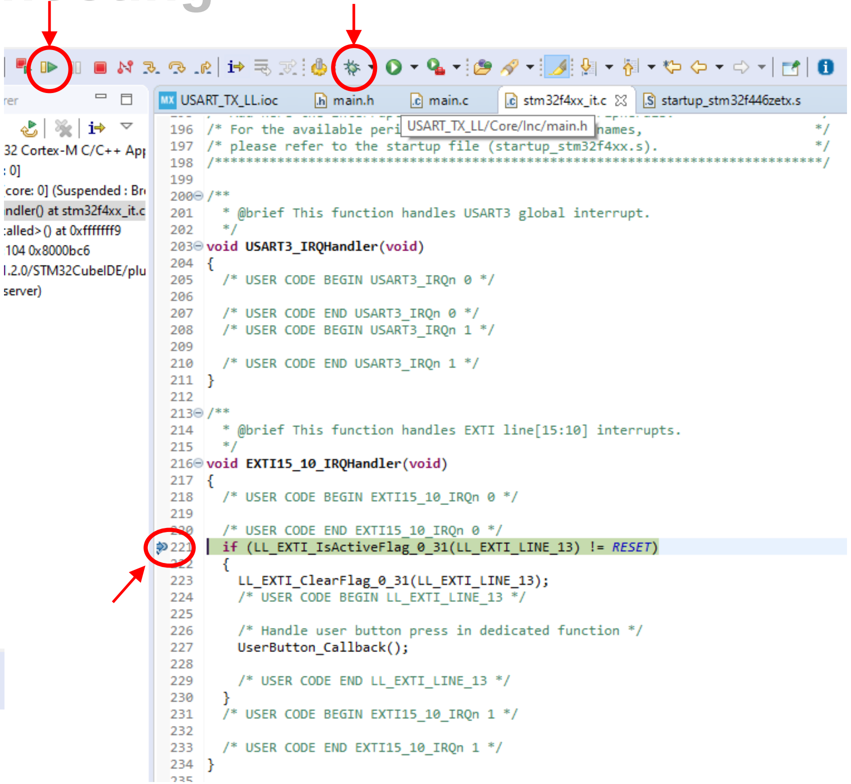
If you have any trouble getting the program to run correctly try to debug your application in the IDE:

1. Download the program to your board in *debug mode* , the debug view will open.

2. Place a *breakpoint* where the interrupt should be triggered (to place a breakpoint double click on the number of line where you want to place it).

3. Click on the resume icon. 

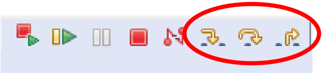
4. Try to press the blue button on the board.

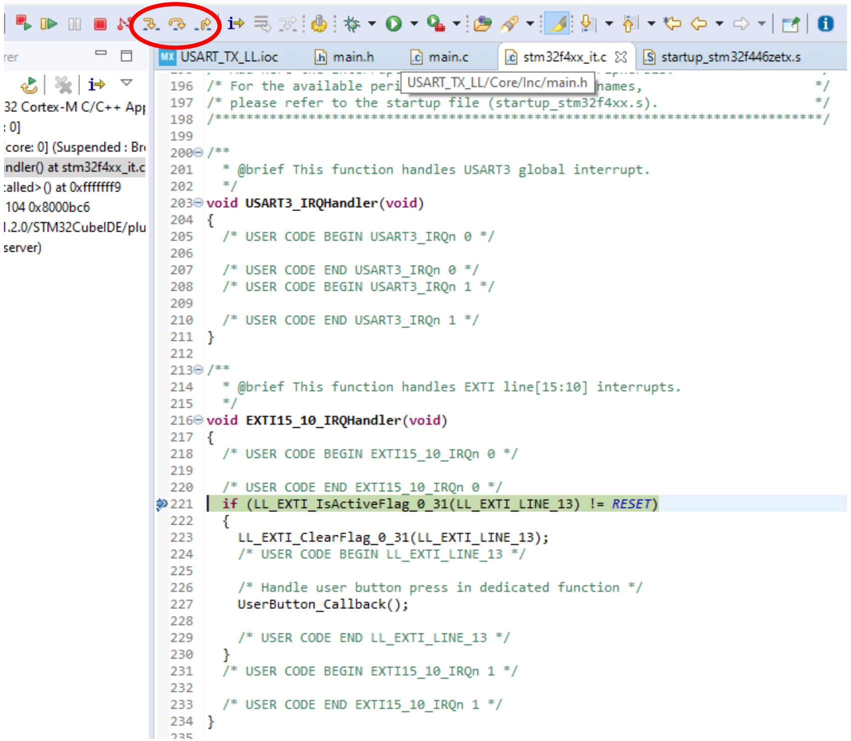


```
/* For the available peri                                names,      */
/* please refer to the startup file (startup_stm32f4xx.s).          */
/******************************************************************************/

/**
  * @brief This function handles USART3 global interrupt.
  */
void USART3_IRQHandler(void)
{
  /* USER CODE BEGIN USART3_IRQn 0 */

  /* USER CODE END USART3_IRQn 0 */
  /* USER CODE BEGIN USART3_IRQn 1 */

  /* USER CODE END USART3_IRQn 1 */
}

/**
  * @brief This function handles EXTI line[15:10] interrupts.
  */
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_13) != RESET)
  {
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_13);
    /* USER CODE BEGIN LL_EXTI_LINE_13 */

    /* Handle user button press in dedicated function */
    UserButton_Callback();

    /* USER CODE END LL_EXTI_LINE_13 */
  }
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

Troubleshooting

5. If it’s all correct your program will jump to the breakpoint and the line will be highlighted.

6. At this point you can resume again or move inside your program with the arrows  .



```
/* For the available peri                          names,                                  */
/* please refer to the startup file (startup_stm32f4xx.s).                                 */
/******************************************************************************/

/**
  * @brief This function handles USART3 global interrupt.
  */
void USART3_IRQHandler(void)
{
  /* USER CODE BEGIN USART3_IRQn 0 */

  /* USER CODE END USART3_IRQn 0 */
  /* USER CODE BEGIN USART3_IRQn 1 */

  /* USER CODE END USART3_IRQn 1 */
}

/**
  * @brief This function handles EXTI line[15:10] interrupts.
  */
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_13) != RESET)
  {
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_13);
    /* USER CODE BEGIN LL_EXTI_LINE_13 */

    /* Handle user button press in dedicated function */
    UserButton_Callback();

    /* USER CODE END LL_EXTI_LINE_13 */
  }
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

For more info in how the UART bus works see: https://www.youtube.com/watch?v=ZzRXKDkMBhA