

NEEEICUM

núcleo de estudantes de engenharia  
eletrônica industrial e computadores  
da universidade do minho

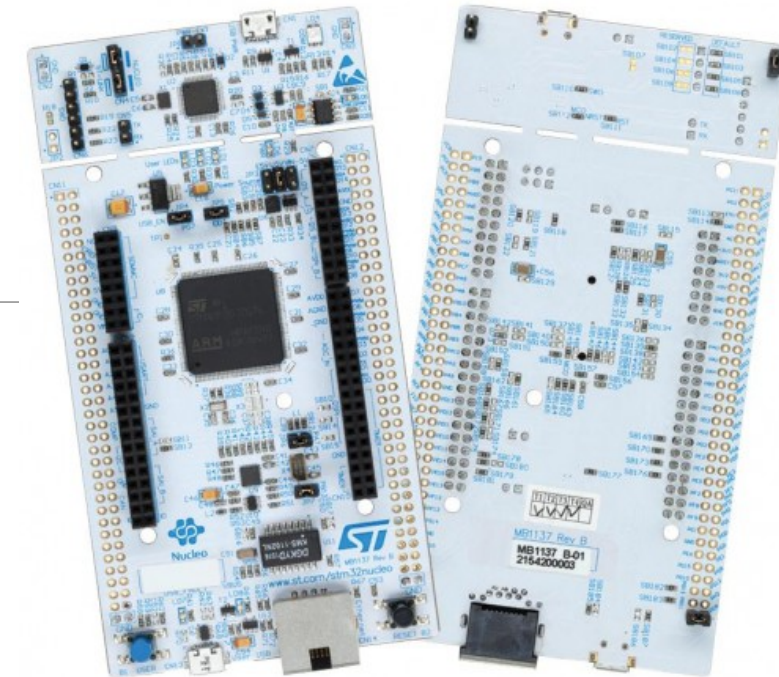
ESRG

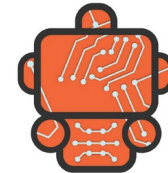
EMBEDDED SYSTEMS  
RESEARCH  
GROUP

# STM 32 F767ZI

GPIO, INTERRUPTS AND UART

ÁLVARO CASTRO LEITE  
DEC 2021





# Requirements

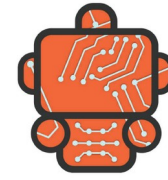
---

Micro-USB cable

STM32 development board

PC with all the tools installed

Serial Port terminal



# Agenda

---

**Skill level: Beginner**

Introduction to HAL API

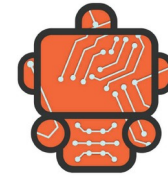
STMCubeMX walkthrough

GPIO

Interrupt management

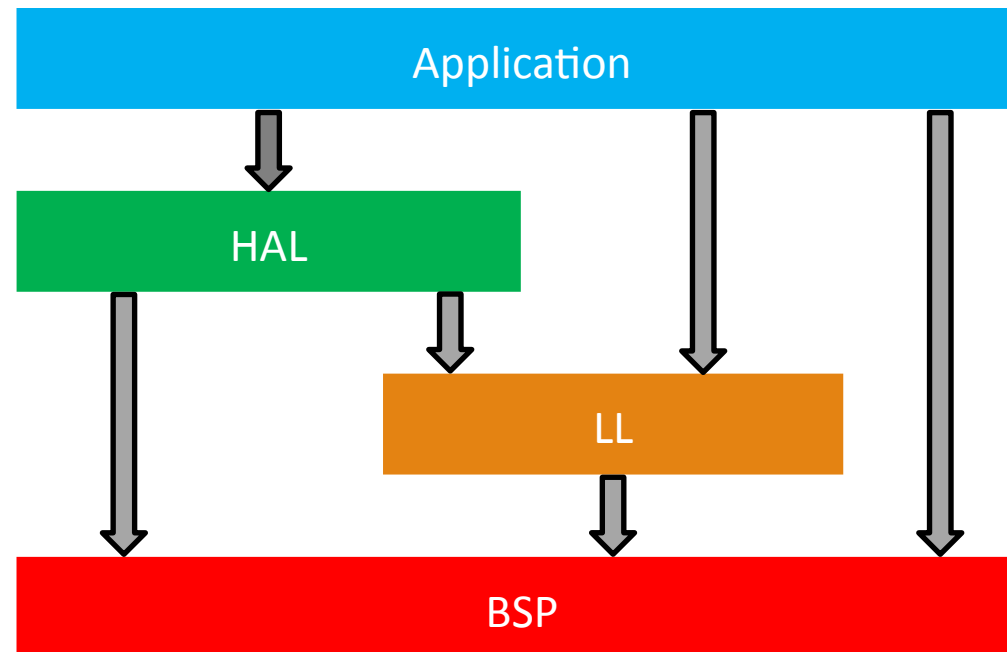
On-chip debug

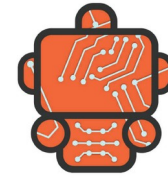
UART



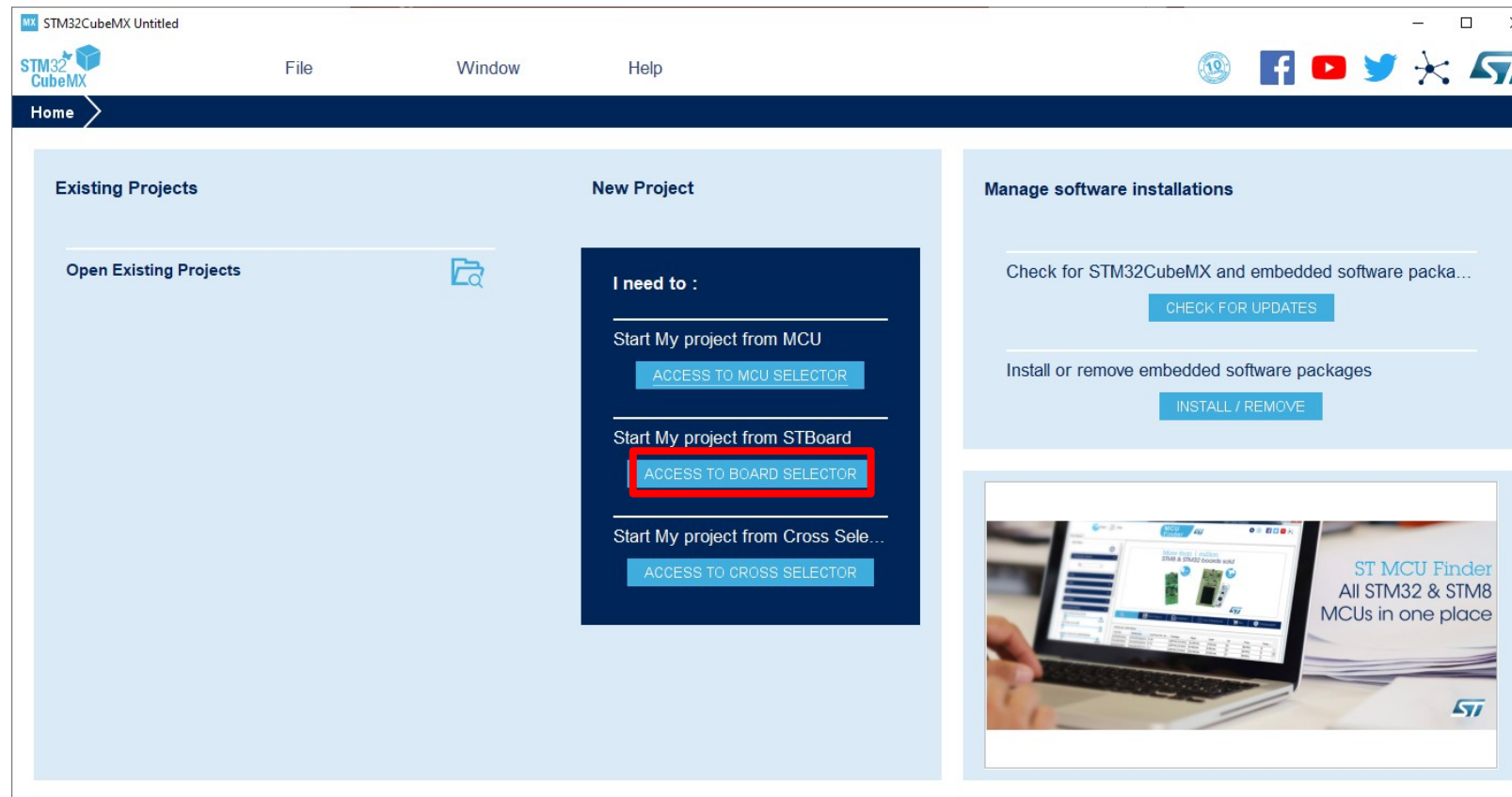
# Introduction HAL API

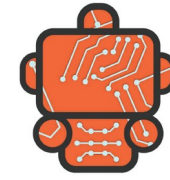
---





# STMCubeMX walkthrough





# STMCubeMX walkthrough

New Project from a Board

MCU/MPU Selector | Board Selector | Cross Selector

Board Filters

Part Number Search

Vendor

Type

MCU/MPU Series

Other

Price = 23.0

Oscillator Freq. = 0 (MHz)

Peripheral

- Accelerometer
- Analog I/O
- Arduino Form Factor
- Audio Line In
- Audio Line Out
- Battery
- Button
- CAN
- Camera
- Compass
- Custom Form Factor
- Digital I/O
- Ethernet
- Gyroscope
- IDA
- Joystick
- LCD Display (Graphics)

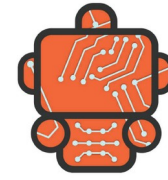
New multicore STM32MP1 Series  
for Industrial and IoT applications

STM32MP1

OpenSTLinux Distribution

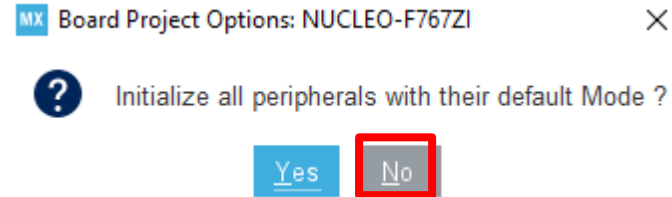
Boards List: 1 item

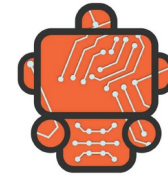
	Overview	Part No.	Type	Marketing Status	Unit Price (US\$)	Mounted Device
☆		NUCLEO-F767ZI	Nucleo144	Active	23.0	<a href="#">STM32F767ZITx</a>



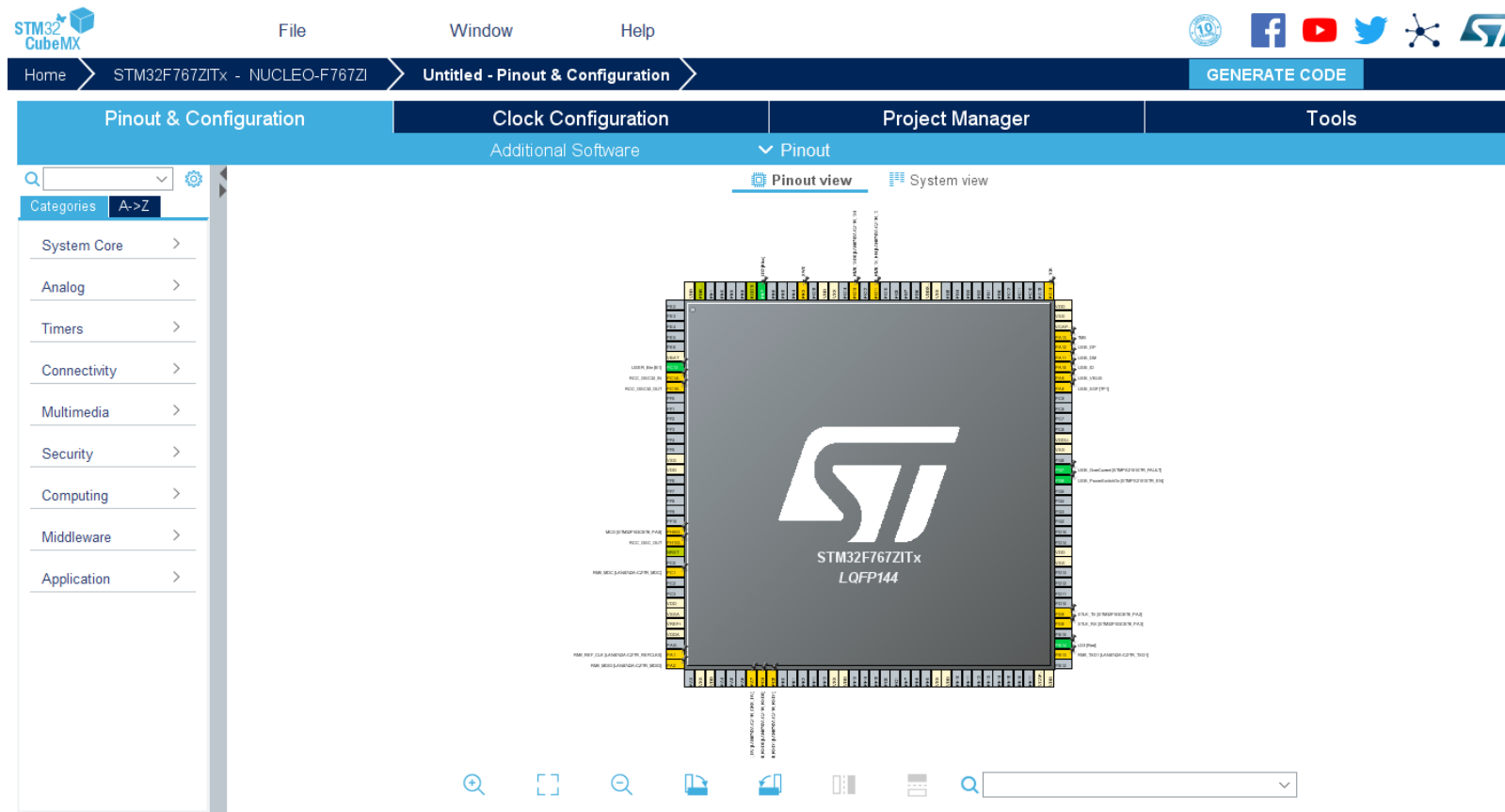
# STMCubeMX walkthrough

---

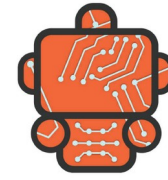




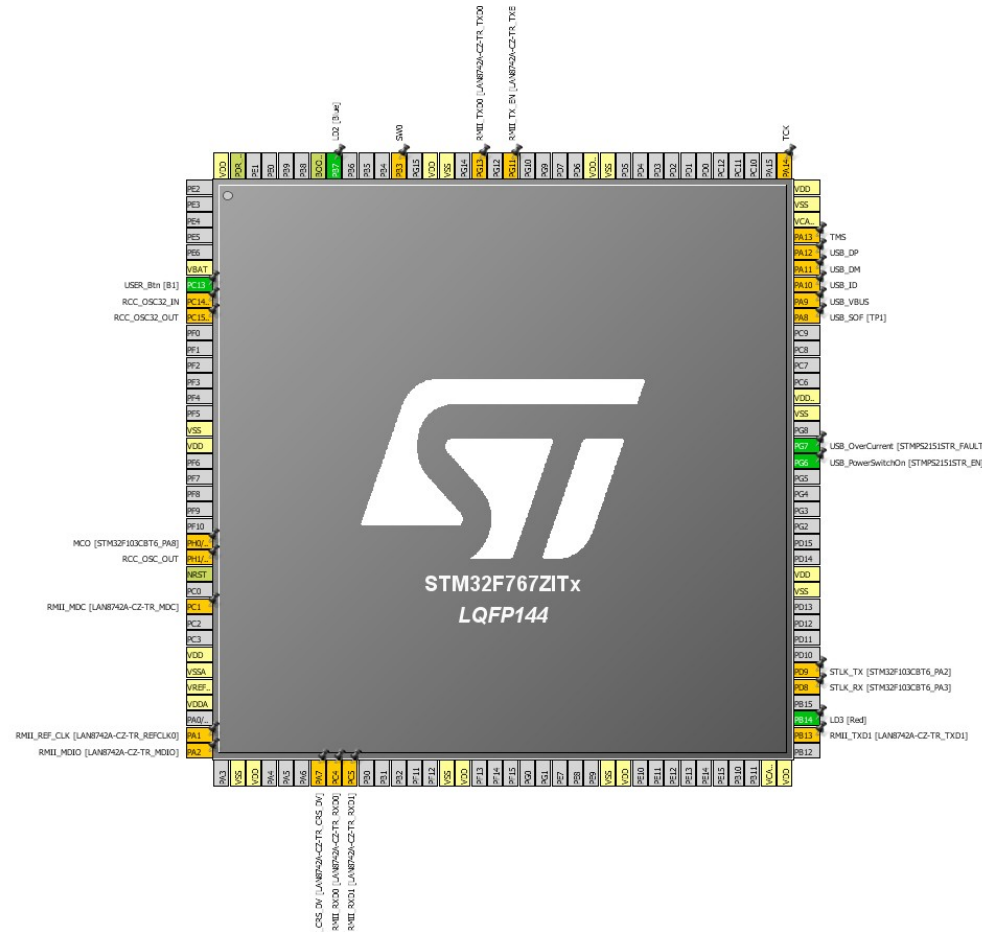
# STMCubeMX walkthrough

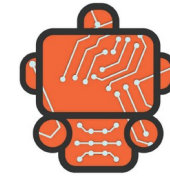




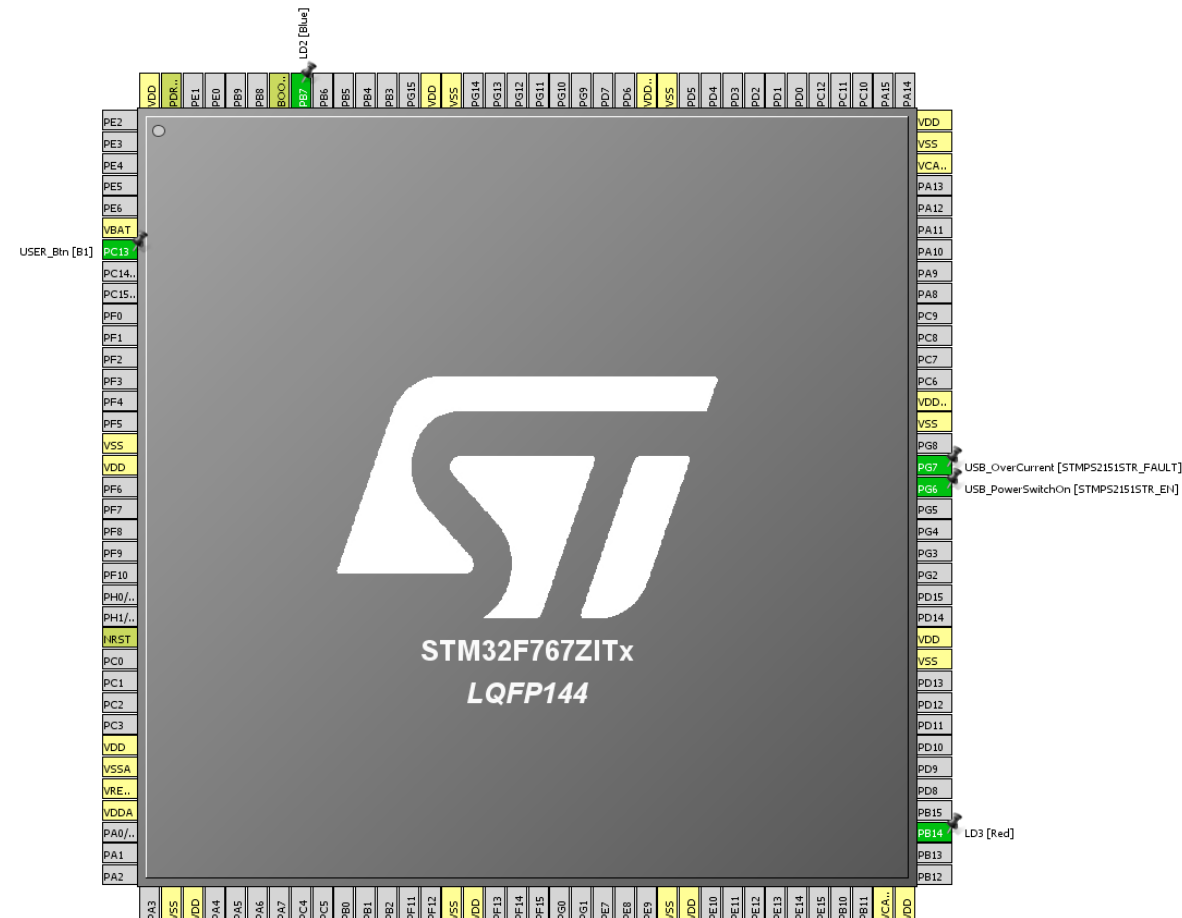


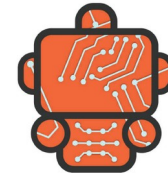
# STMCubeMX walkthrough





# STMCubeMX walkthrough





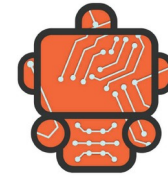
# STMCubeMX walkthrough

---

Question: How many user LED's the board has? 3 right? Where's the green LED?

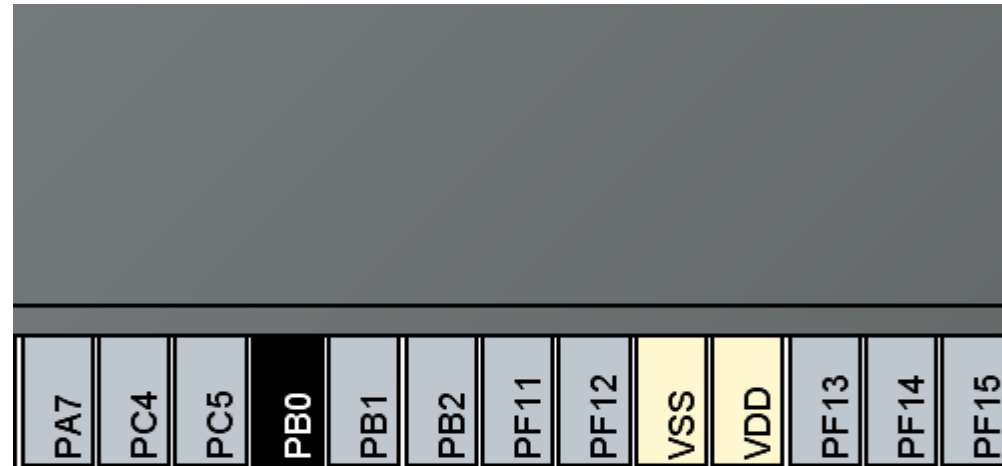
Tip: Open UM1974 and go to page 24.

Answer: PB0

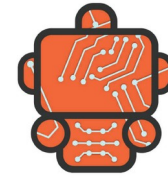


# STMCubeMX walkthrough

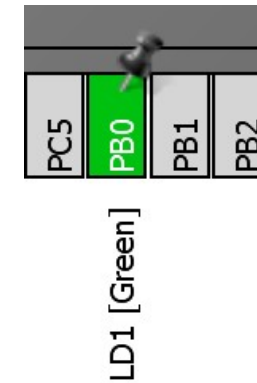
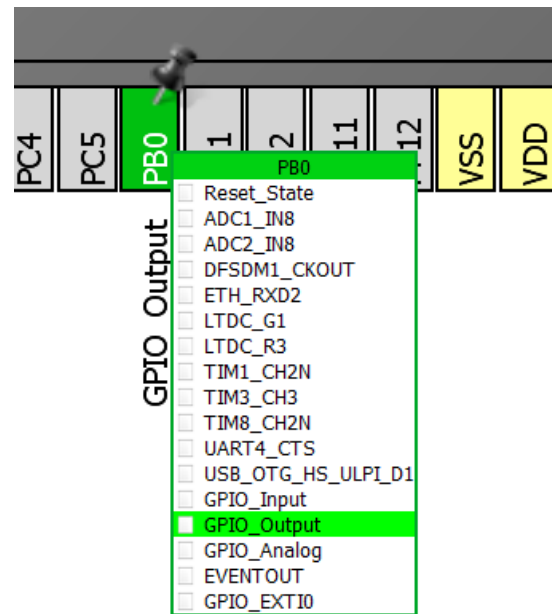
---



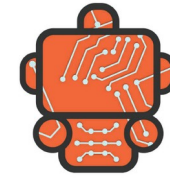
PB0



# STMCubeMX walkthrough

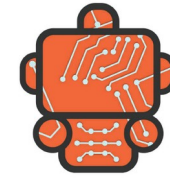






# STMCubeMX walkthrough

Pinout & Configuration	Clock Configuration	Project Manager	Tools
Project	<p>Project Settings</p> <p>Project Name GPIOandUART</p> <p>Project Location C:\Users\alvar\Documents\STM32\code</p> <p>Application Structure Advanced <input type="checkbox"/> Do not generate the main()</p>		
Code Generator	<p>Toolchain Folder Location C:\Users\alvar\Documents\STM32\code\GPIOandUART\</p> <p>Toolchain / IDE MDK-ARM V5 <input type="checkbox"/> Generate Under Root</p>		
Advanced Settings	<p>Linker Settings</p> <p>Minimum Heap Size <input type="text" value="0x200"/></p> <p>Minimum Stack Size <input type="text" value="0x400"/></p> <p>Mcu and Firmware Package</p> <p>Mcu Reference STM32F767ZITx</p> <p>Firmware Package Name and Version STM32Cube FW_F7 V1.15.0</p> <p><input checked="" type="checkbox"/> Use Default Firmware Location <input type="text" value="C:/Users/alvar/STM32Cube/Repository/STM32Cube_FW_F7_V1.15.0"/> <input type="button" value="Browse"/></p>		

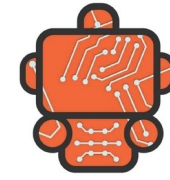


# STMCubeMX walkthrough

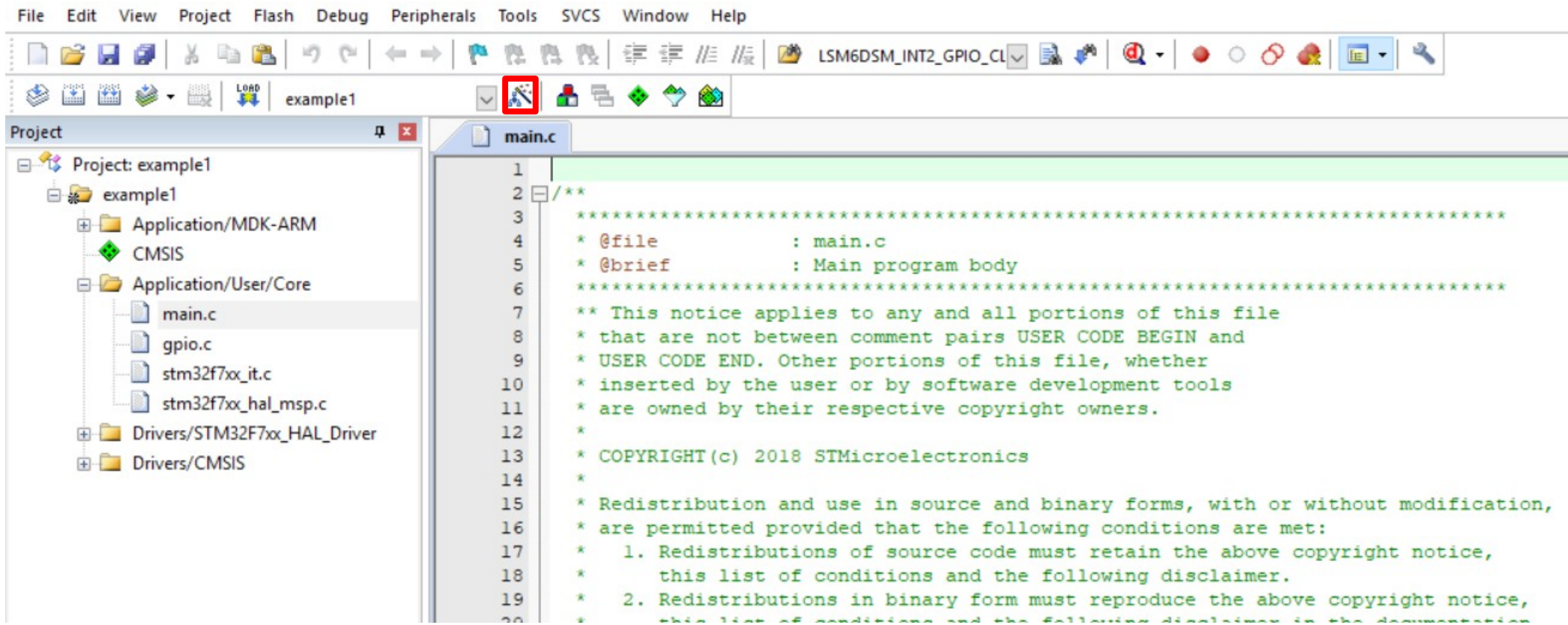
Home > STM32F767ZITx - NUCLEO-F767ZI > Untitled - Project Manager > **GENERATE CODE**

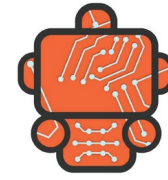
Pinout & Configuration	Clock Configuration	Project Manager	Tools
<b>Project</b>	<p>STM32Cube MCU packages and embedded software packs</p> <p><input type="radio"/> Copy all used libraries into the project folder</p> <p><input checked="" type="radio"/> Copy only the necessary library files</p> <p><input type="radio"/> Add necessary library files as reference in the toolchain project configuration file</p>		
<b>Code Generator</b>	<p>Generated files</p> <p><input checked="" type="checkbox"/> Generate peripheral initialization as a pair of '.c/.h' files per peripheral</p> <p><input type="checkbox"/> Backup previously generated files when re-generating</p> <p><input checked="" type="checkbox"/> Keep User Code when re-generating</p> <p><input checked="" type="checkbox"/> Delete previously generated files when not re-generated</p>		
<b>Advanced Settings</b>	<p>HAL Settings</p> <p><input type="checkbox"/> Set all free pins as analog (to optimize the power consumption)</p> <p><input type="checkbox"/> Enable Full Assert</p> <p>Template Settings</p> <p>Select a template to generate customized code <span>Settings...</span></p>		



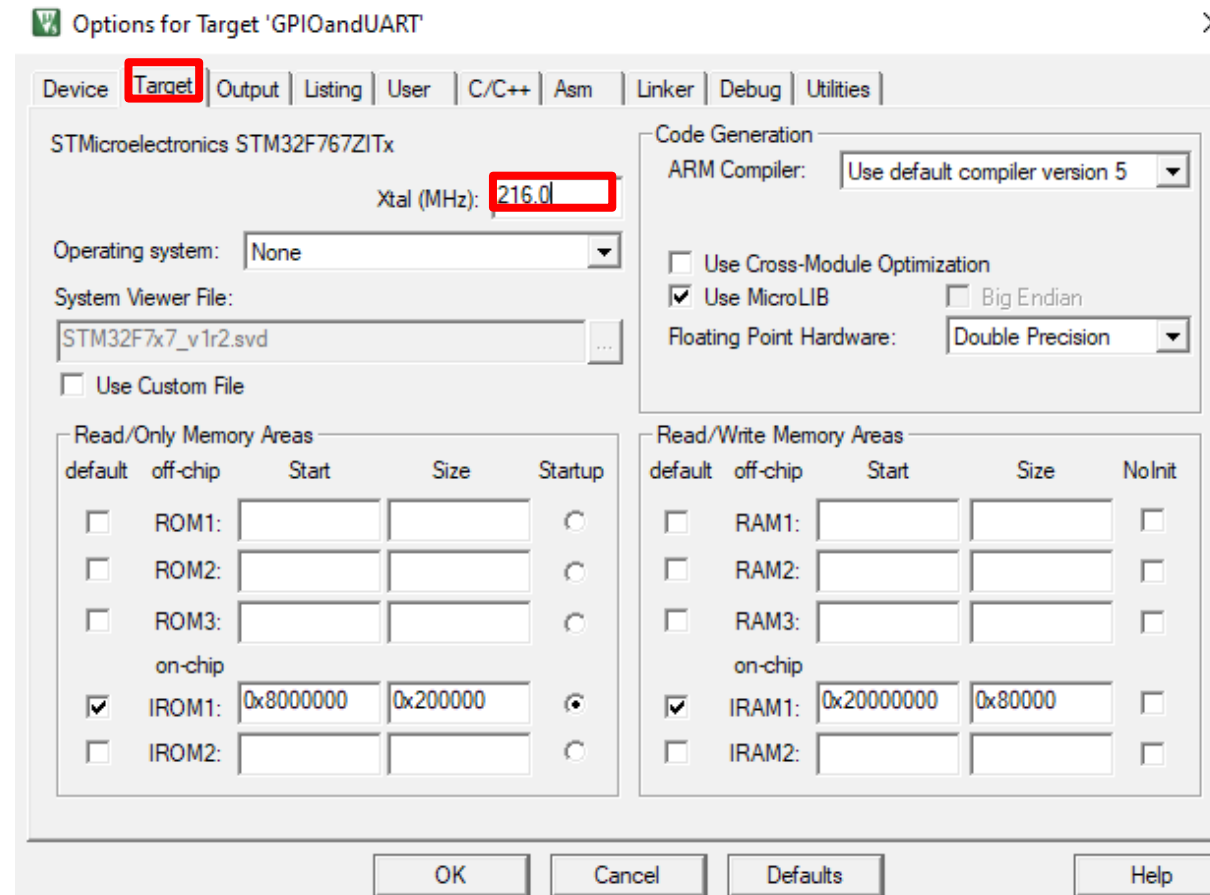


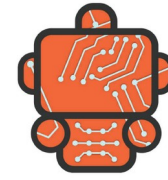
# STMCubeMX walkthrough



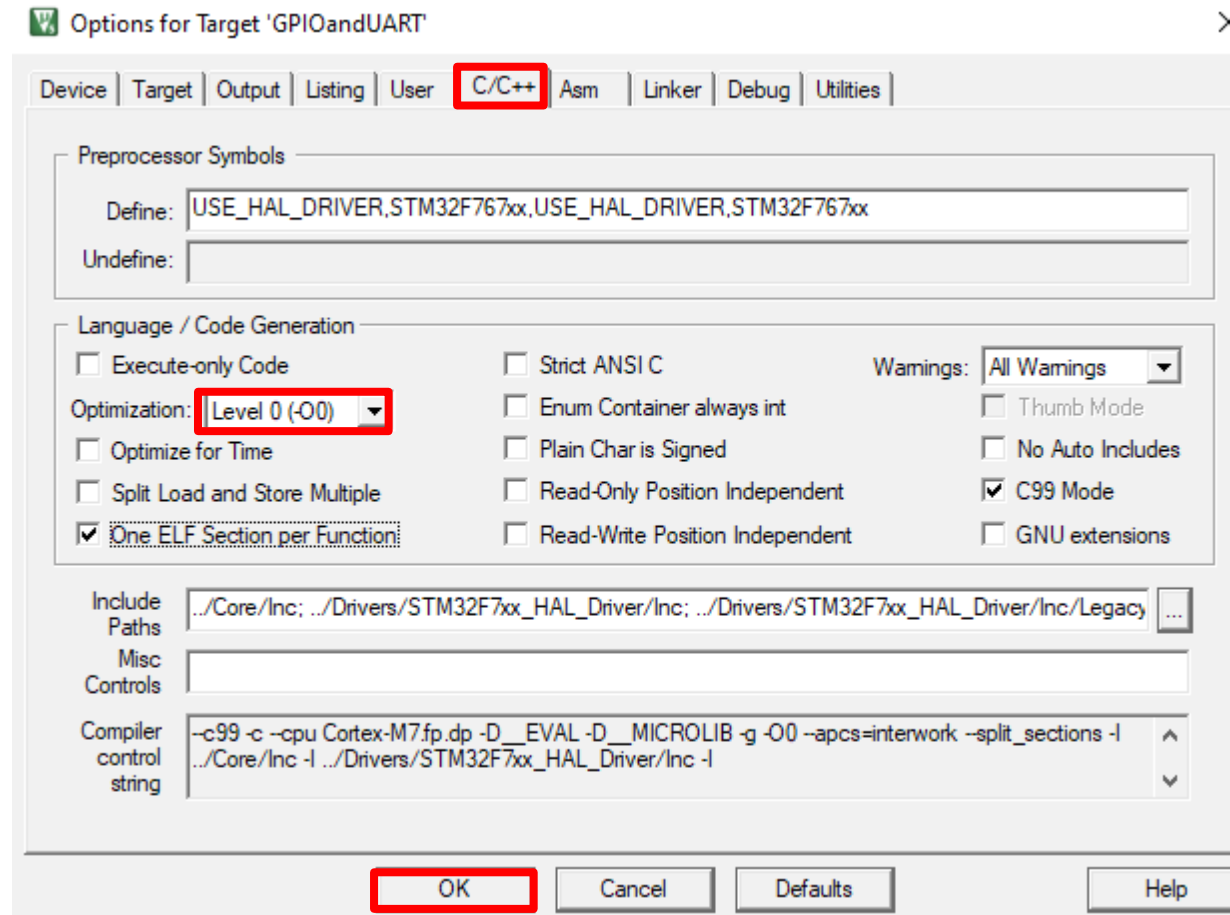


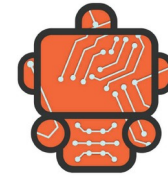
# STMCubeMX walkthrough





# STMCubeMX walkthrough





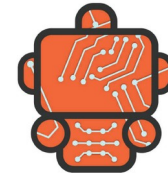
# GPIO

---

In UM1905 User Manual, on Chapter 26 (page 372), you can find the information about the GPIO HAL API.

It's important that you understand the structure associated with each I/O port and the API (set of functions to handle all GPIO).

HAL provides four manipulation routines to read, change and lock the state of an I/O.



# GPIO

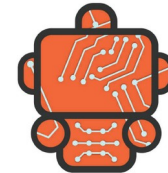
---

HAL provides four manipulation routines to read, change and lock the state of an I/O and two for configuration.

To read the status of an I/O we can use the function:

- `GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

which accepts the GPIO descriptor and the pin number. It returns `GPIO_PIN_RESET` when the I/O is low or `GPIO_PIN_SET` when high.



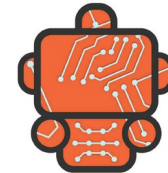
# GPIO

---

Conversely, to change the I/O state, we have the function:

- `void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)`

which accepts the GPIO descriptor, the pin number and the desired state.

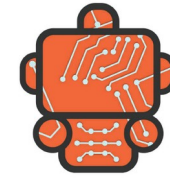


# GPIO

---

If we want to simply invert the I/O state, then we can use this convenient routine:

- `void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx,  
uint16_t GPIO_Pin)`



# GPIO

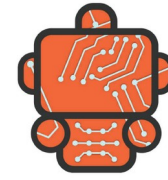
---

Example 1: Make a blinking led with the three LEDs reconfigured before.

```
97  while (1)
98  {
99      HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
100     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
101     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
102     HAL_Delay(500);
103     /* USER CODE END WHILE */
```

After writing the following lines, compile and flash, and the leds should blink.





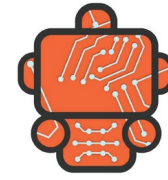
# GPIO

Example 2: Make the green LED to change the state when you push the user button.

```
97     while (1)
98     {
99         if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13))
100             HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
101             HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
102             HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
103             HAL_Delay(500);
104         /* USER CODE END WHILE */
```

The problem: the system isn't analyzing the button all the time, most of it is doing other stuff (in the case: the delay).

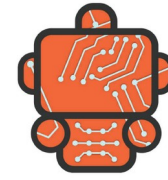
The solution could be, use *interrupts*.



# Interrupt management

---

ARM architecture defines events as something that can be originated both by the hardware and the software itself. ARM architecture events are distinguished between two types: interrupts originated by the hardware, exceptions by the software (e.g., invalid access to the memory location).

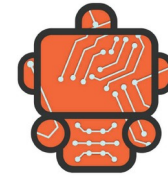


# Interrupt management

---

Cortex-M processors provide a unit dedicated to event management. This is called Nested Vectored Interrupt Controller (NVIC). NVIC is a dedicated hardware unit inside the Cortex-M based microcontrollers that is responsible for handling exceptions. You can find more information about NVIC in RM0410, chapter 10.

A dedicated programmable controller, named External Interrupt/Event Controller (EXTI), is responsible for the interconnection between the external I/O signals and the NVIC controller, as we will see next.



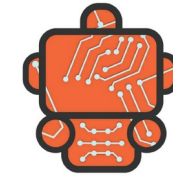
# Interrupt management

---

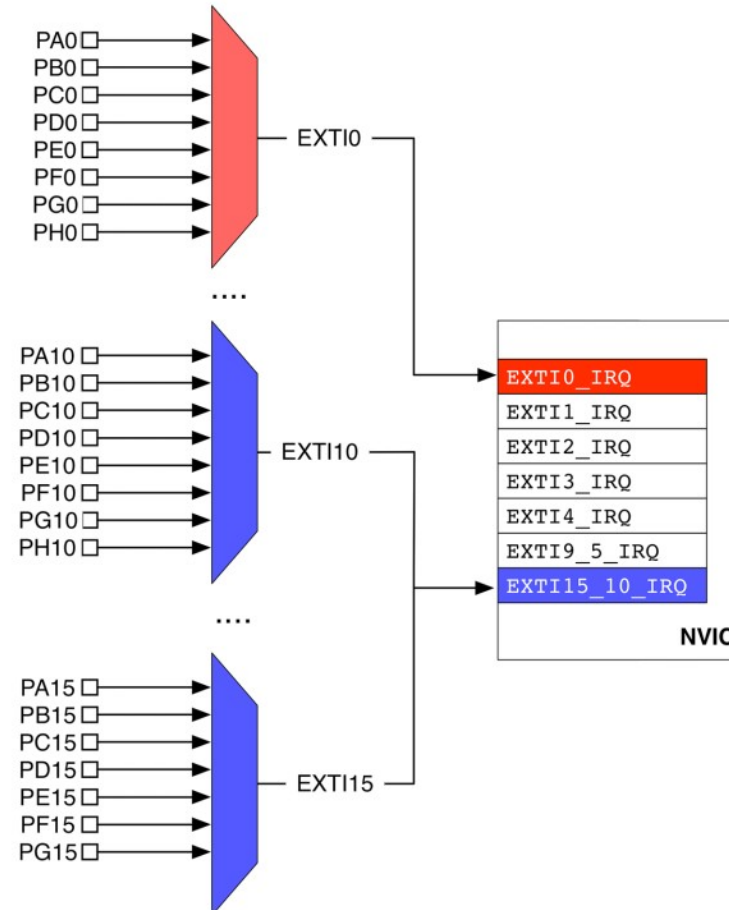
STM32 microcontrollers provide a variable number of external interrupt sources connected to the NVIC through the EXTI controller, which in turn is capable of managing several EXTI lines. The number of interrupt sources and lines depends on the specific STM32 family.

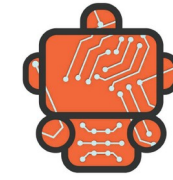
GPIO is connected to the EXTI lines, and it is possible to enable interrupts for every MCU GPIO, even if most of them share the same interrupt line. For example, for an STM32F7 MCU, up to 168 GPIOs are connected to 16 EXTI lines. However, there are only 11 independent interrupt in the EXTI.

You can find more information about EXTI in RM0410, chapter 11.

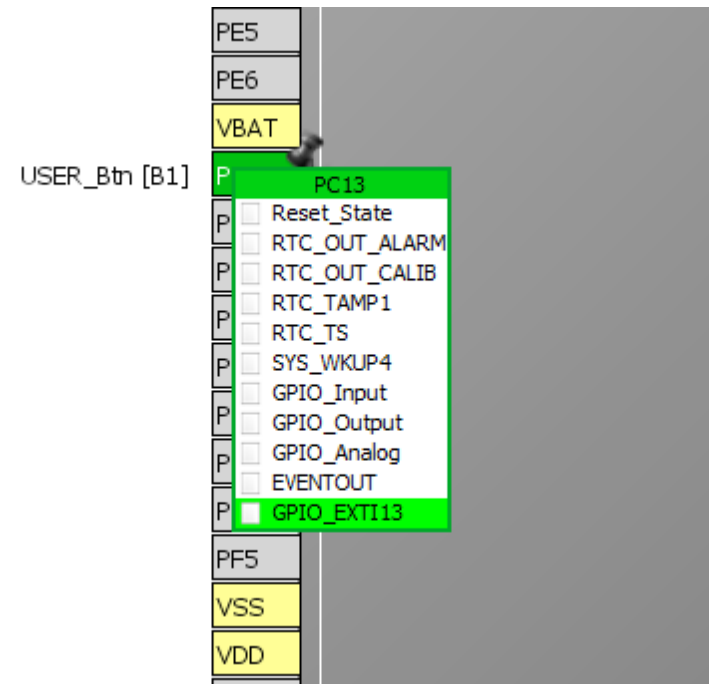


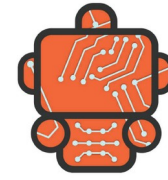
# Interrupt management





# Interrupt management





# Interrupt management

Pinout & Configuration

Clock Configuration

Additional Software

Categories

A->Z

System Core

CORTEX\_M7

DMA

**GPIO**

IWDG

NVIC

RCC

▲ SYS

WWDG

Analog

>

USER\_Btn [B1]

VDD

PDR\_...

PE1

PE0

PB9

PB8

BOOT0

**PB7**

PB6

PB5

PB4

PB3

PG15

PE2

PE3

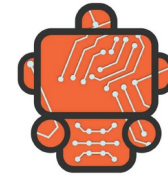
PE4

PE5

PE6

VBAT

**PC13**



# Interrupt management

GPIO Mode and Configuration

Configuration

☐ Group By Peripherals

☒ GPIO ☒ NVIC

Search Signals

☐ Show only Modified Pins

Pin Name	Signal on Pin	GPIO output...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PB0	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	LD1 [Green]	<input checked="" type="checkbox"/>
PB7	n/a	Low	Output Pus...	No pull-up ...	Low	Disable	LD2 [Blue]	<input checked="" type="checkbox"/>
PB14	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	LD3 [Red]	<input checked="" type="checkbox"/>
PC13	n/a	n/a	External Int...	No pull-up ...	n/a	n/a	USER_Btn ...	<input checked="" type="checkbox"/>
PG6	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	USB_Powe...	<input checked="" type="checkbox"/>
PG7	n/a	n/a	Input mode	No pull-up ...	n/a	n/a	USB_OverC...	<input checked="" type="checkbox"/>

PC13 Configuration :

GPIO mode: External Interrupt Mode with Rising edge trigger detection

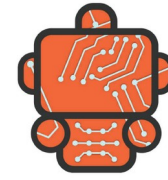
GPIO Pull-up/Pull-down: No pull-up and no pull-down

User Label: USER\_Btn [B1]

??

UM1974  
page 75





# Interrupt management

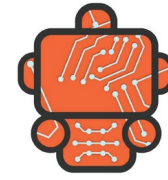
GPIO Mode and Configuration

Configuration

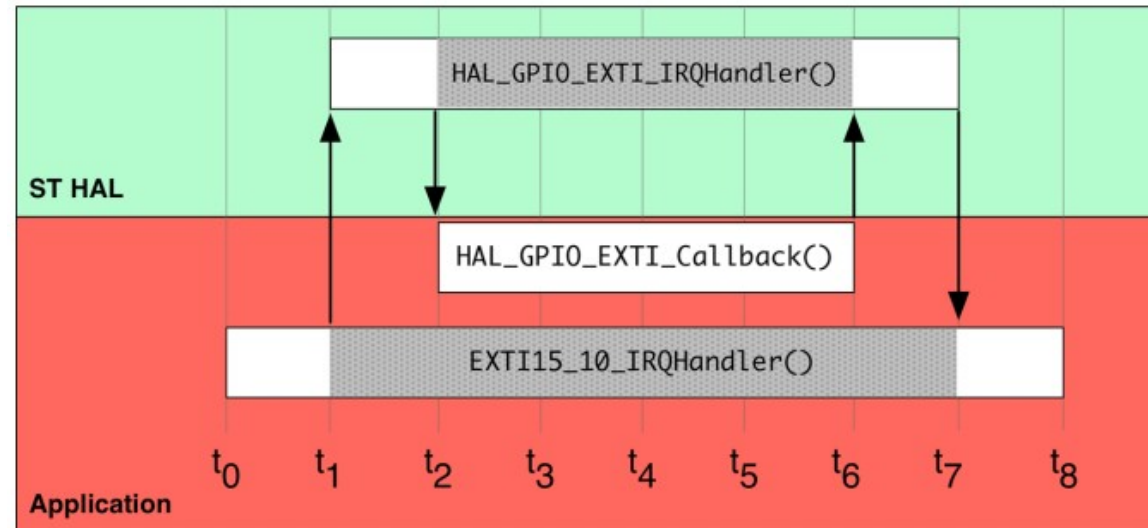
☐ Group By Peripherals

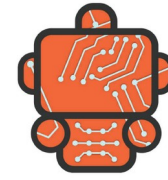
☒ GPIO ☒ NVIC

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	0	0



# Interrupt management



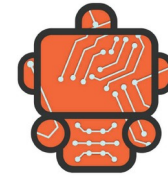


# Interrupt management

---

The STM32CubeMX already added implementation code of the EXTI15\_10\_IRQHandler function, this code is on stm32f7xx\_it.c file.

```
203 void EXTI15_10_IRQHandler(void)
204 {
205     /* USER CODE BEGIN EXTI15_10_IRQn 0 */
206
207     /* USER CODE END EXTI15_10_IRQn 0 */
208     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
209     /* USER CODE BEGIN EXTI15_10_IRQn 1 */
210
211     /* USER CODE END EXTI15_10_IRQn 1 */
212 }
213
```

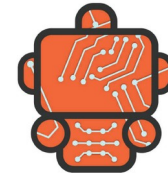


# Interrupt management

---

You must implement the HAL\_GPIO\_Callback function on GPIO file.

```
89  /* USER CODE BEGIN 2 */
90
91  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
92      if (GPIO_Pin == GPIO_PIN_13)
93          HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
94  }
95
96  /* USER CODE END 2 */
```

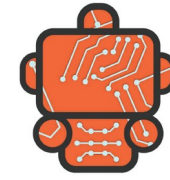


# On-chip debug

---

In order to debug our code we need to prepare the peripherals for the communication.

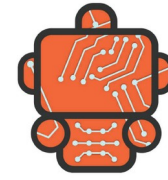
To do this it's necessary to enable the Single Wire Debug in the SYS(tem) peripheral.



# On-chip debug

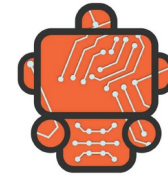
The screenshot shows the STM32CubeIDE configuration window. The 'Pinout & Configuration' tab is active. On the left, under 'System Core', the 'SYS' component is highlighted. The main configuration area shows 'SYS Mode and Configuration'. The 'Debug' dropdown menu is set to 'Serial Wire'. Below it, 'System Wake-Up 1', 'System Wake-Up 2', and 'System Wake-Up 3' are unchecked, while 'System Wake-Up 4' is checked. The 'Timebase Source' is set to 'SysTick'.

Generate the code again.

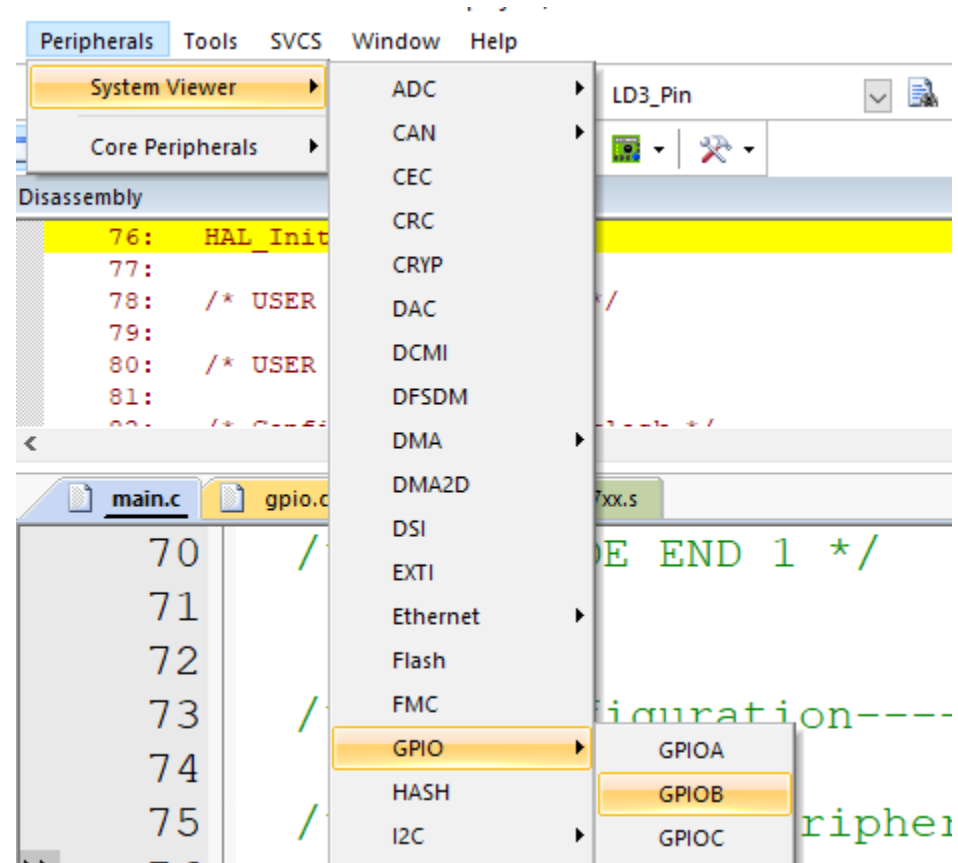


# On-chip debug

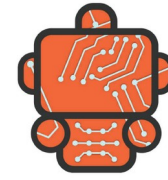
```
Peripherals  Tools  SVCS  Window  Help
[Navigation Icons] [D3_Pin] [@] [Run/Debug Icons]
ART
main.c  gpio.c  startup_stm32f767xx.s
95  /* Infinite loop */
96  /* USER CODE BEGIN WHILE */
97  while (1)
98  {
99  HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
100  HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
101  HAL_Delay(500);
102  /* USER CODE END WHILE */
103
```



# On-chip debug







# On-chip debug

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

LD3\_Pin

Registers

Register	Value
Core	
R0	0x00000028
R1	0x00000028
R2	0x00000100
R3	0xE000E004
R4	0x0800115C
R5	0x0800115C
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000410
R14 (LR)	0x08000F68
R15 (PC)	0x08001102
xPSR	0x01000000

Disassembly

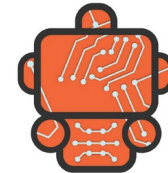
```
0x08001100 E00C B 0x0800111C
99: HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
0x08001102 2180 MOVS r1,#0x80
0x08001104 4806 LDR r0,[pc,#24] ; @0x08001120
0x08001106 F7FFFA4B BL.W HAL_GPIO_TogglePin (0x080005A0)
100: HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
0x08001108 F7FFFA4B BL.W HAL_GPIO_TogglePin (0x080005A0)
```

main.c gpio.c startup\_stm32f767xx.s

```
93 /* USER CODE END 2 */
94
95 /* Infinite loop */
96 /* USER CODE BEGIN WHILE */
97 while (1)
98 {
99 HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
100 HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
101 HAL_Delay(500);
102 /* USER CODE END WHILE */
103
104 /* USER CODE BEGIN 3 */
105
```

GPIOB

Property	Value
MODER	0x10004281
OTYPER	0x00000000
GPIO_OSPEEDR	0x000000C0
PUPDR	0x00000100
IDR	0x0000343C
ODR	0x00000000
ODR15	<input type="checkbox"/>
ODR14	<input type="checkbox"/>
ODR13	<input type="checkbox"/>
ODR12	<input type="checkbox"/>
ODR11	<input type="checkbox"/>
ODR10	<input type="checkbox"/>
ODR9	<input checked="" type="checkbox"/>
ODR8	<input type="checkbox"/>
ODR7	<input checked="" type="checkbox"/>
ODR6	<input type="checkbox"/>
ODR5	<input type="checkbox"/>
ODR4	<input type="checkbox"/>
ODR3	<input type="checkbox"/>
ODR2	<input type="checkbox"/>
ODR1	<input type="checkbox"/>
ODR0	<input checked="" type="checkbox"/>



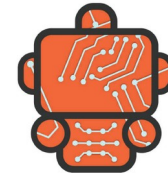
# UART

---

In order to use UART and USART peripherals, HAL API offers a set of functions.

To transmit a sequence of bytes over the USART in polling mode the HAL provides the function:

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef  
*huart, uint8_t *pData, uint16_t Size, uint32_t  
Timeout);
```

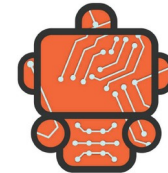


# UART

---

To receive a sequence of bytes over the USART in polling mode the HAL provides the function:

```
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef  
*huart, uint8_t *pData, uint16_t Size, uint32_t  
Timeout);
```

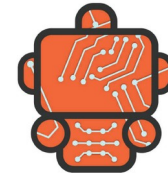


# UART

---

To transmit a sequence of bytes in interrupt mode, the HAL defines the function:

```
HAL_StatusTypeDef  
HAL_UART_Transmit_IT(UART_HandleTypeDef *huart,  
uint8_t *pData, uint16_t Size);
```

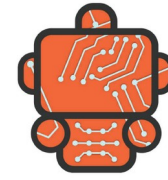


# UART

---

To receive a sequence of bytes over the USART in interrupt mode the HAL provides the function:

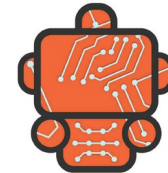
```
HAL_StatusTypeDef  
HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t  
*pData, uint16_t Size);
```



# UART

---

Example 3: Create a program that communicates with the computer through the RS232 protocol, echoing whatever the computer sent, using an UART (or USART) peripheral in the STM32 and a terminal program in the computer. The program may send a message on startup and should change the state of the red led every time a message is received.



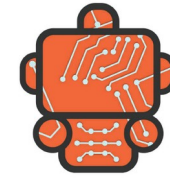
# UART

---

Question: The STM32F767ZI has 8 UART/USART peripherals, what we should use?

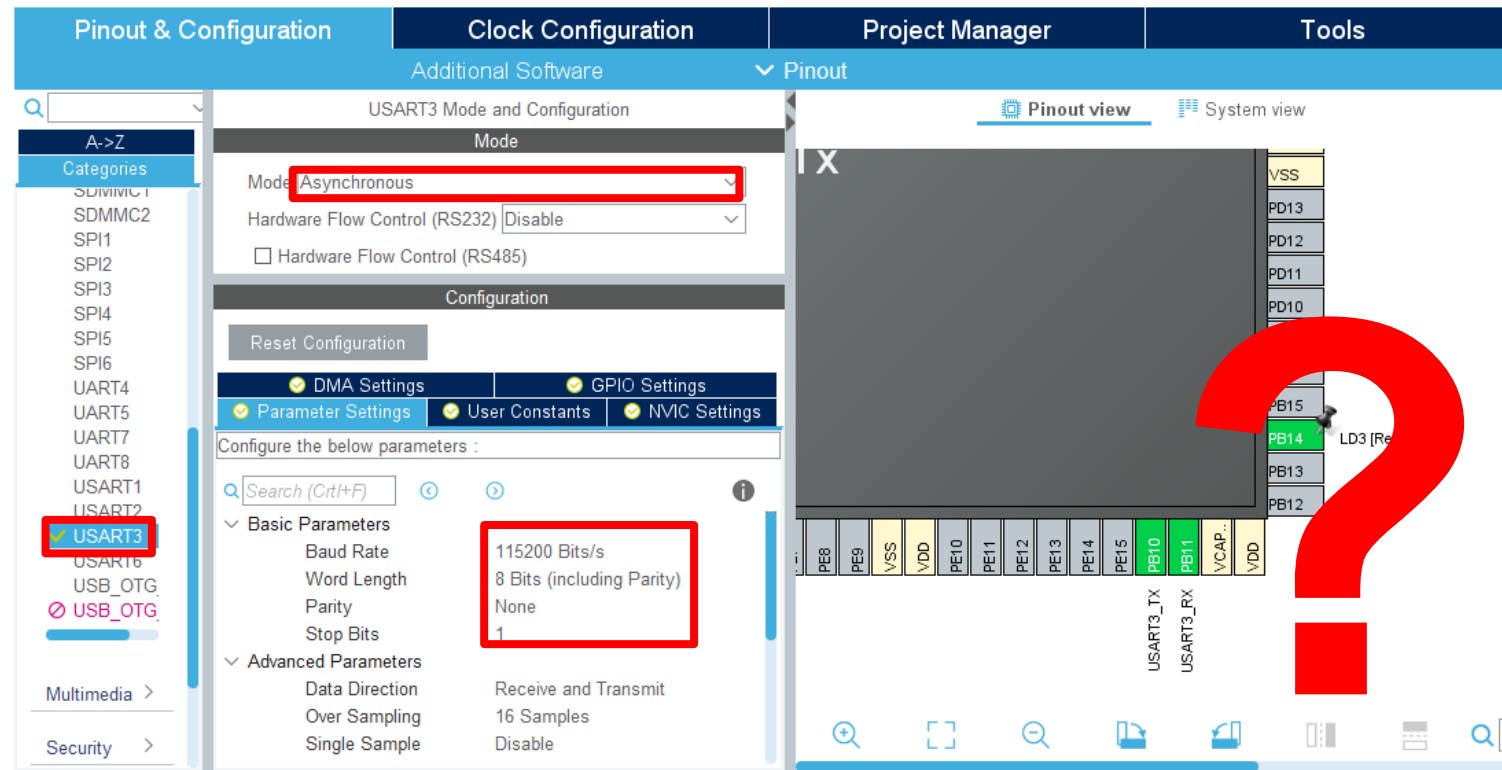
Tip: Open UM1974, go to chapter 6 (page 11), try to understand Figure 3, and after go to section 6.9 (page 26).

Answer: UART3

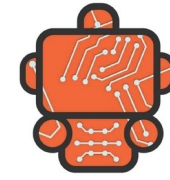


# UART

Let's edit the project in STM32CubeMX.



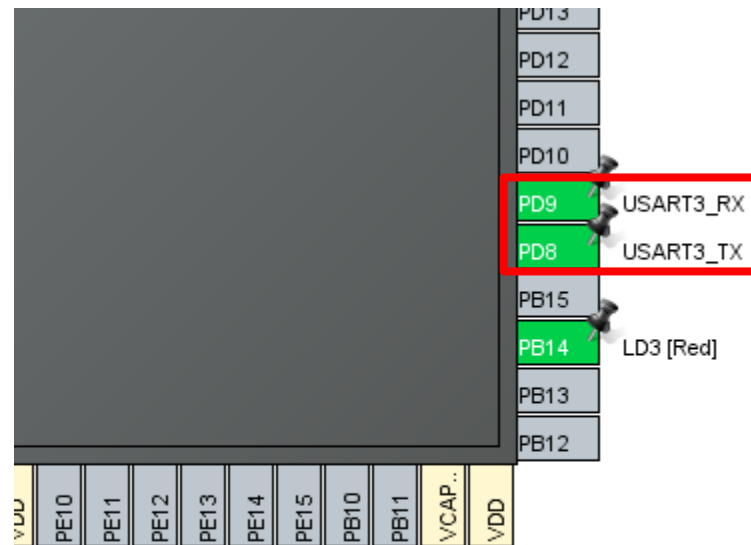


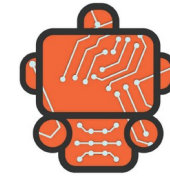


# UART

Let's correct the GPIO Pins.

Tip: To discover alternatives pins you can press CTRL and click over the pin.





# UART

Do not forget to enable the USART3 interrupt

USART3 Mode and Configuration

Mode

Mode: Asynchronous

Hardware Flow Control (RS232): Disable

☐ Hardware Flow Control (RS485)

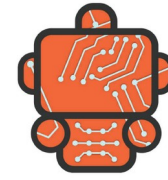
Configuration

Reset Configuration

✓ NVIC Settings    ✓ DMA Settings    ✓ GPIO Settings

✓ Parameter Settings    ✓ User Constants

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART3 global interrupt	<input checked="" type="checkbox"/>	0	0

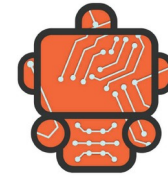


# UART

---

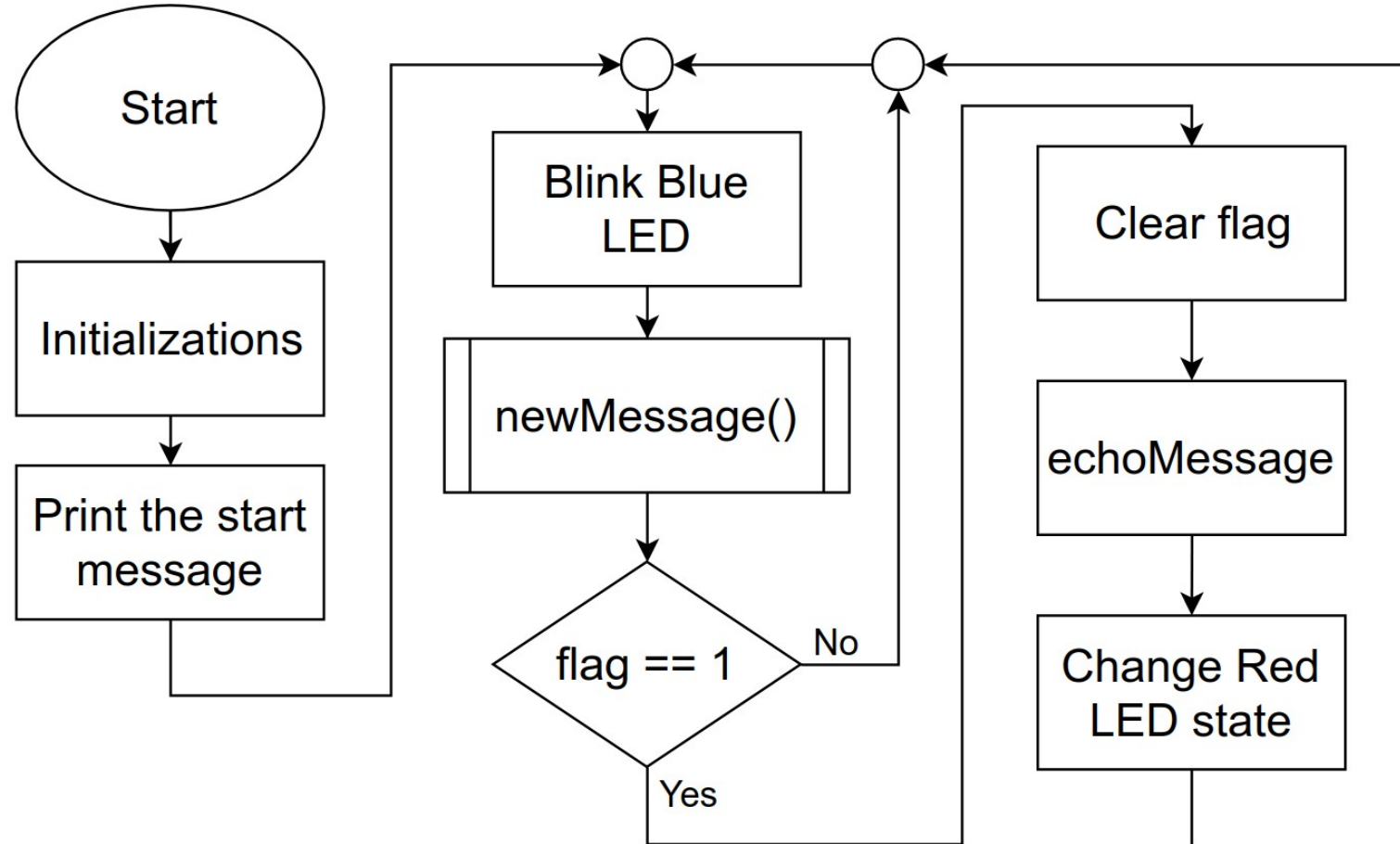
Question: How to design the application?

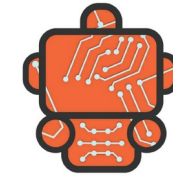
Tip: Use some flowchart or algorithm



# UART

main():

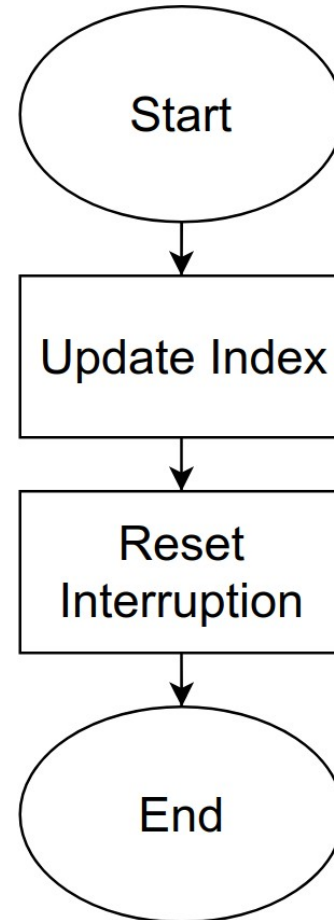


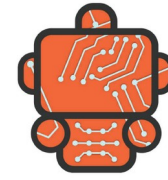


# UART

---

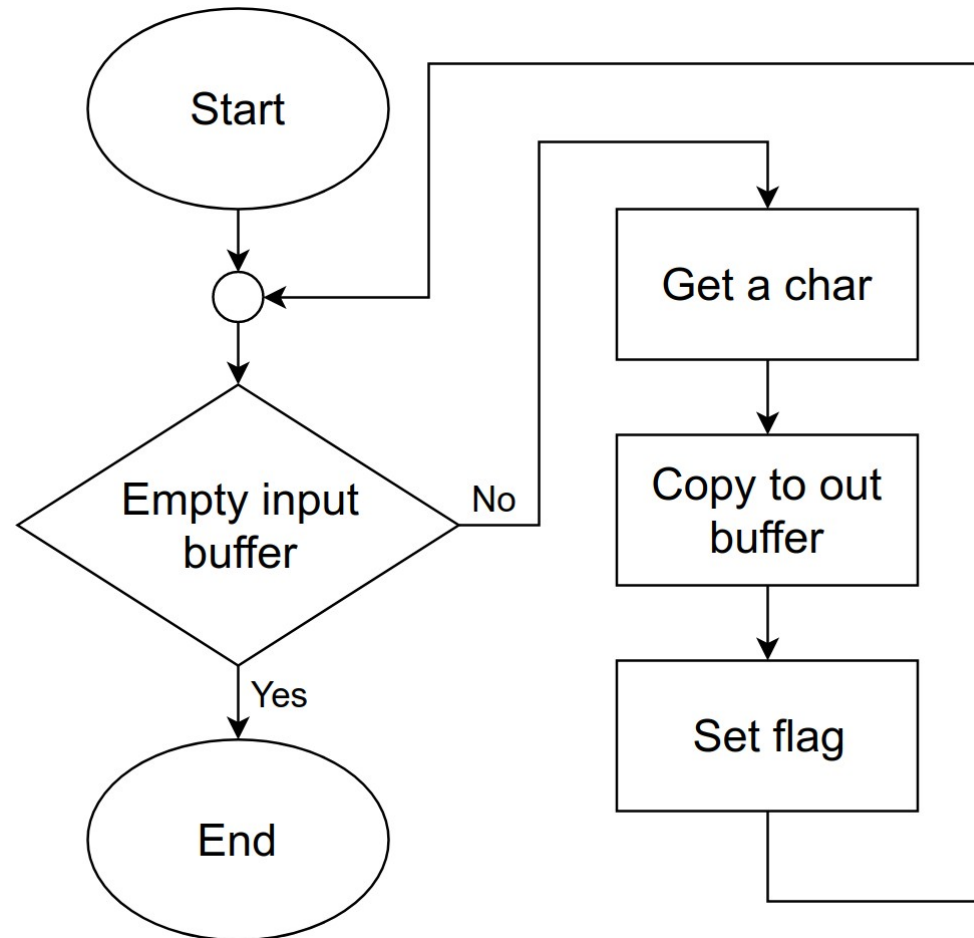
USART Callback:

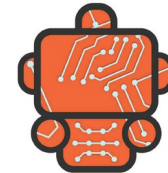




# UART

newMessage():





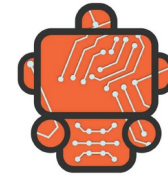
# UART

---

Let's implement...

Uart.c

```
23  /* USER CODE BEGIN 0 */
24
25  //UART3 RX data structures
26  uint8_t UART3Rx_Buffer[128];
27  uint8_t Rx_Buffer[128];
28  int receve_flag = 0;
29  volatile uint8_t UART3Rx_index = 0;
```



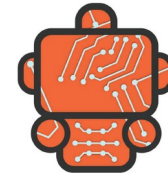
# UART

---

## Usart.h

```
35  /* USER CODE BEGIN Private defines */
36  extern uint8_t Rx_Buffer[128];
37  extern int receve_flag;
38  /* USER CODE END Private defines */
39
40  void MX_USART3_UART_Init(void);
41
42  /* USER CODE BEGIN Prototypes */
43
44  void newMessage(void);
45  void init_UART3(void);
46
47  /* USER CODE END Prototypes */
```



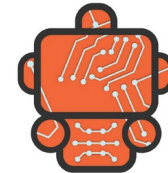


# UART

---

At usart.c file you must add some functions.

```
115 /* USER CODE BEGIN 1 */
116
117 void init_UART3() {
118     // set the interrupt for UART3 Rx
119     HAL_UART_Receive_IT(&huart3, &UART3Rx_Buffer[UART3Rx_index], 1);
120 }
```

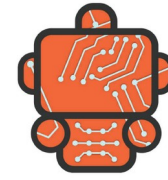


# UART

---

## The Usart Callback

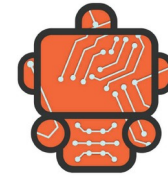
```
122 //implemantation of UART ISR
123 void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart) {
124     if (huart->Instance == USART3) { //current UART?
125         UART3Rx_index++;
126         UART3Rx_index &= ~(1<<7); //keep index inside the limits
127         // set the interrupt for UART3 Rx again
128         HAL_UART_Receive_IT(&huart3, &UART3Rx_Buffer[UART3Rx_index], 1);
129     }
130 }
```



# UART

## newMessage()

```
132 void newMessage() {  
133     static int local_index = 0;  
134     int out_index = 0;  
135     while(local_index != UART3Rx_index) {  
136         Rx_Buffer[out_index] = UART3Rx_Buffer[local_index];  
137         out_index++;  
138         local_index++;  
139         local_index &= ~(1<<7);  
140         receve_flag = 1;  
141     }  
142     Rx_Buffer[out_index] = '\0';  
143 }
```

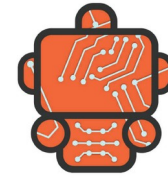


# UART

---

To make the printf work you must change the output of the function by redefining the fputc function in usart.c file.

```
145 //redefine the stdout
146 int fputc(int ch, FILE *f) {
147     HAL_UART_Transmit(&huart3, (uint8_t*)&ch, 1, 100);
148     return ch;
149 }
150
151 /* USER CODE END 1 */
```

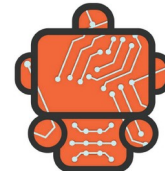


# UART

---

At main.c implement the main function

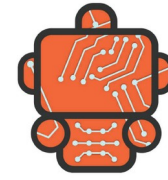
```
90  /* Initialize all configured peripherals */
91  MX_GPIO_Init();
92  MX_USART3_UART_Init();
93  /* USER CODE BEGIN 2 */
94
95  init_UART3();
96  printf("Hello STM32!!\r\n");
97  printf("Type a message and press enter\r\n");
98
99  /* USER CODE END 2 */
```



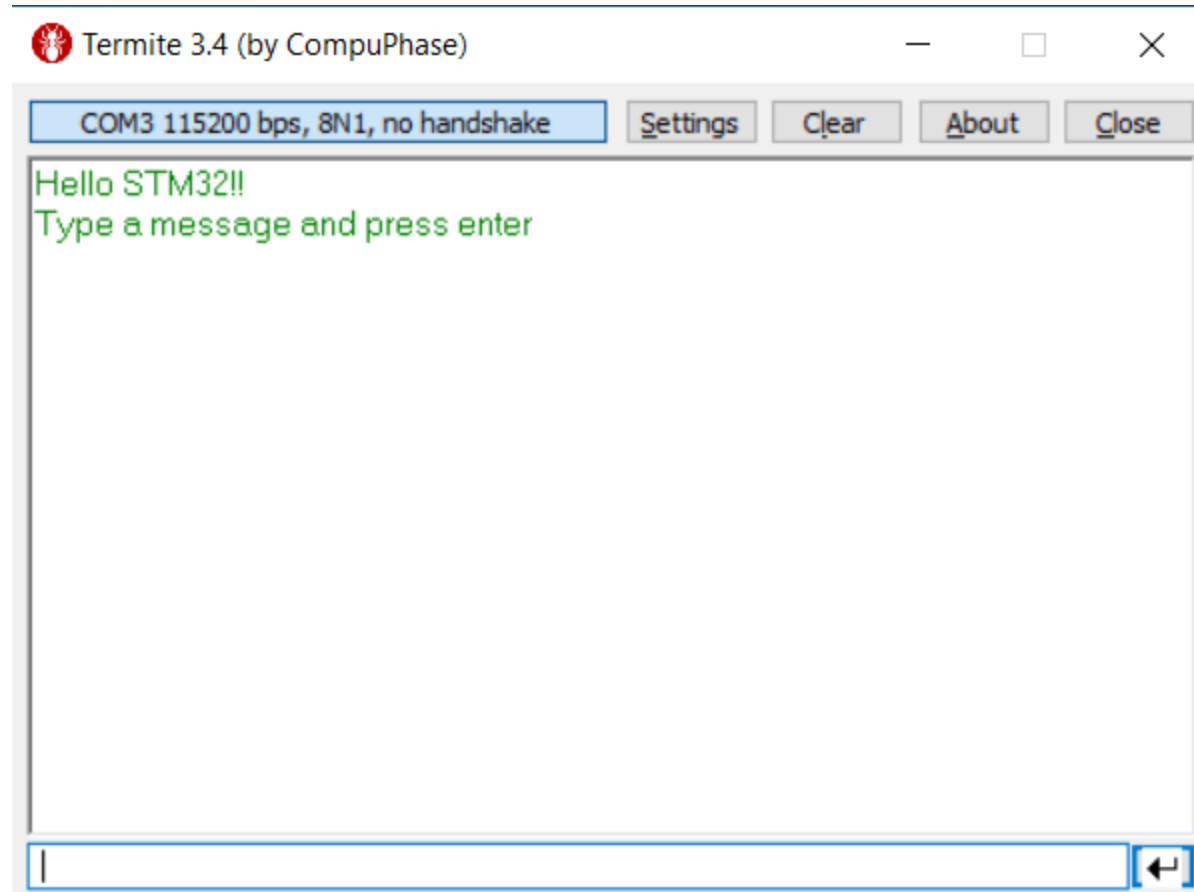
# UART

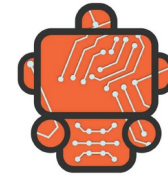
---

```
101  /* Infinite loop */
102  /* USER CODE BEGIN WHILE */
103  while (1)
104  {
105      HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
106      newMessage();
107      if(receive_flag){
108          receive_flag = 0;
109          printf("The message was: [\"%s\"]\r\n", Rx_Buffer);
110          printf("Type another message and press enter\r\n");
111          HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
112      }
113      HAL_Delay(500);
114  /* USER CODE END WHILE */
```

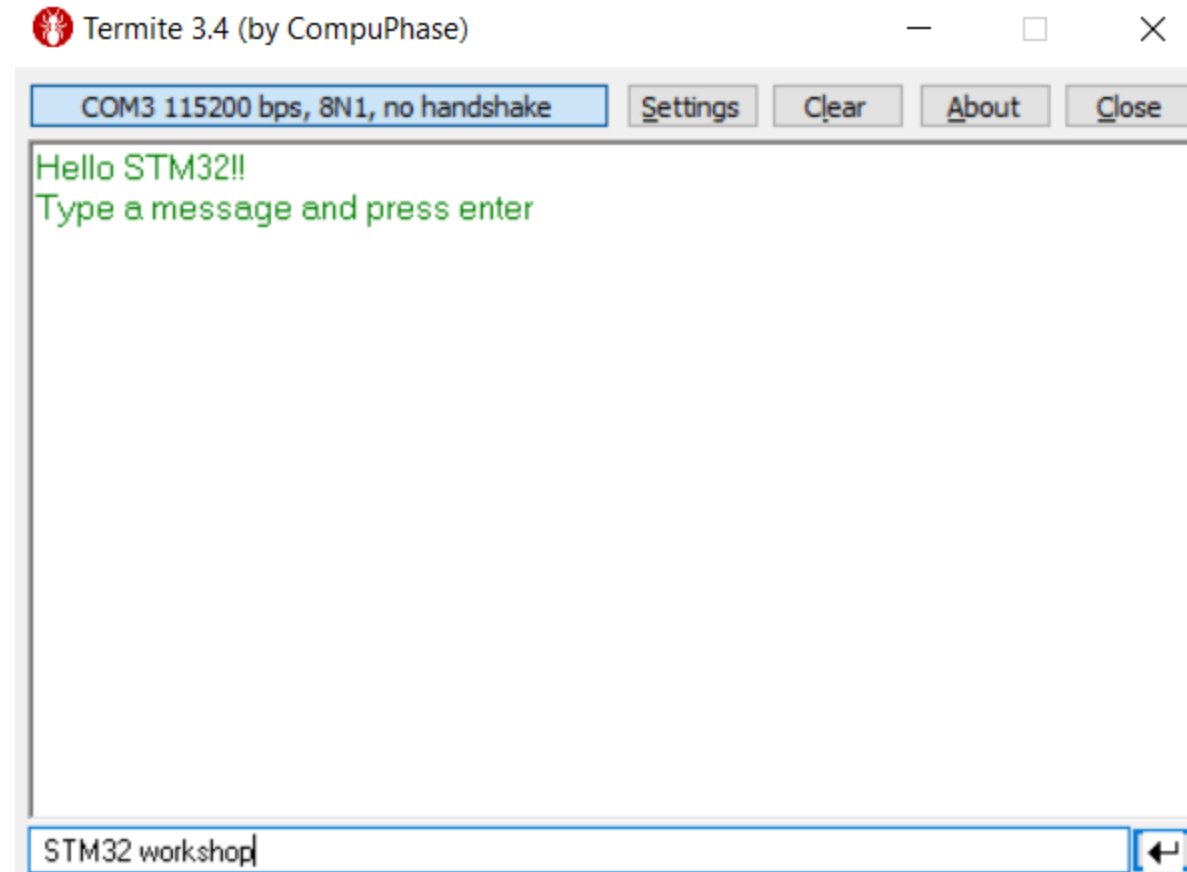


# UART

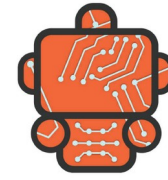




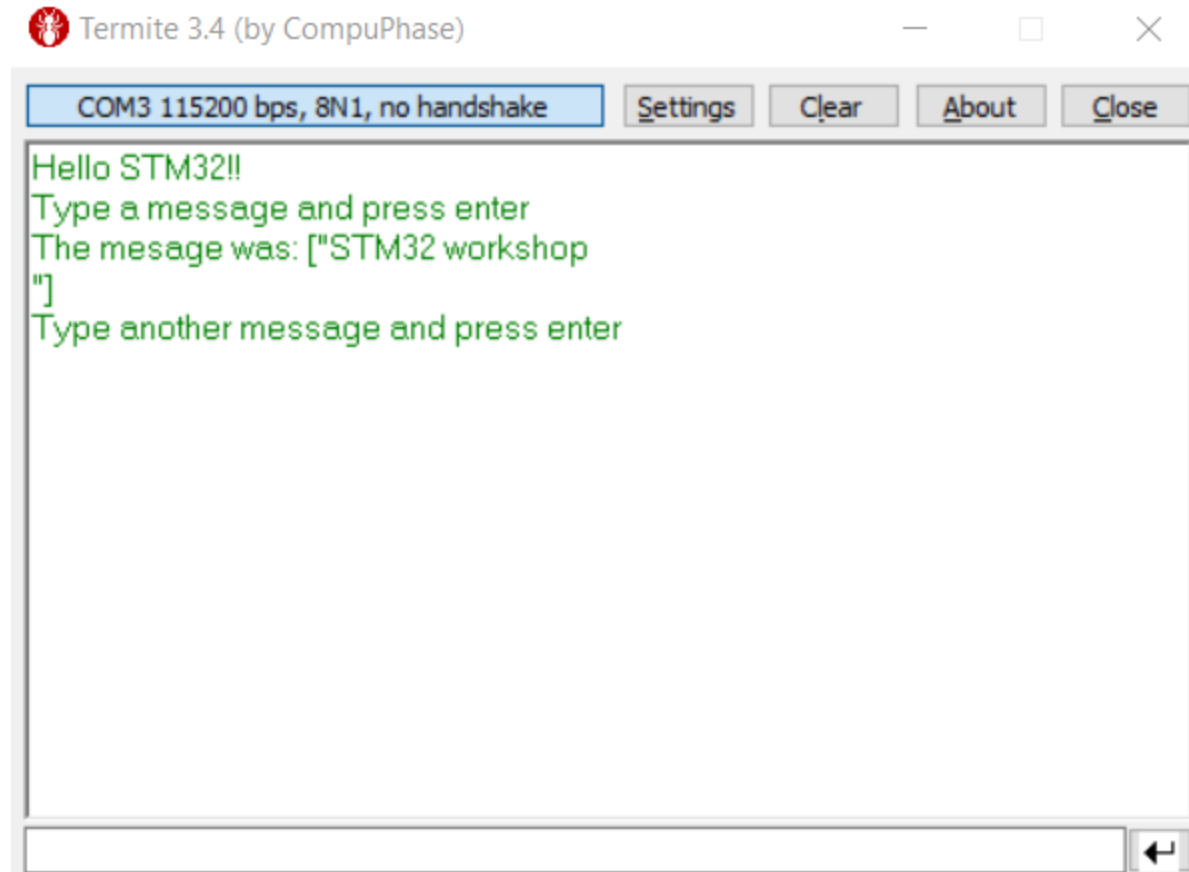
# UART

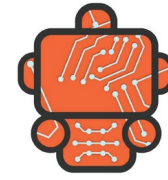






# UART





# Conclusion

---

With this you've learned enough to continue your study of the GPIO, interrupt management and the USART peripherals.

The tools shouldn't be a problem now.

Keep in mind, there is much more to learn, this is just the beginning and the only way to improve is doing it yourselves.