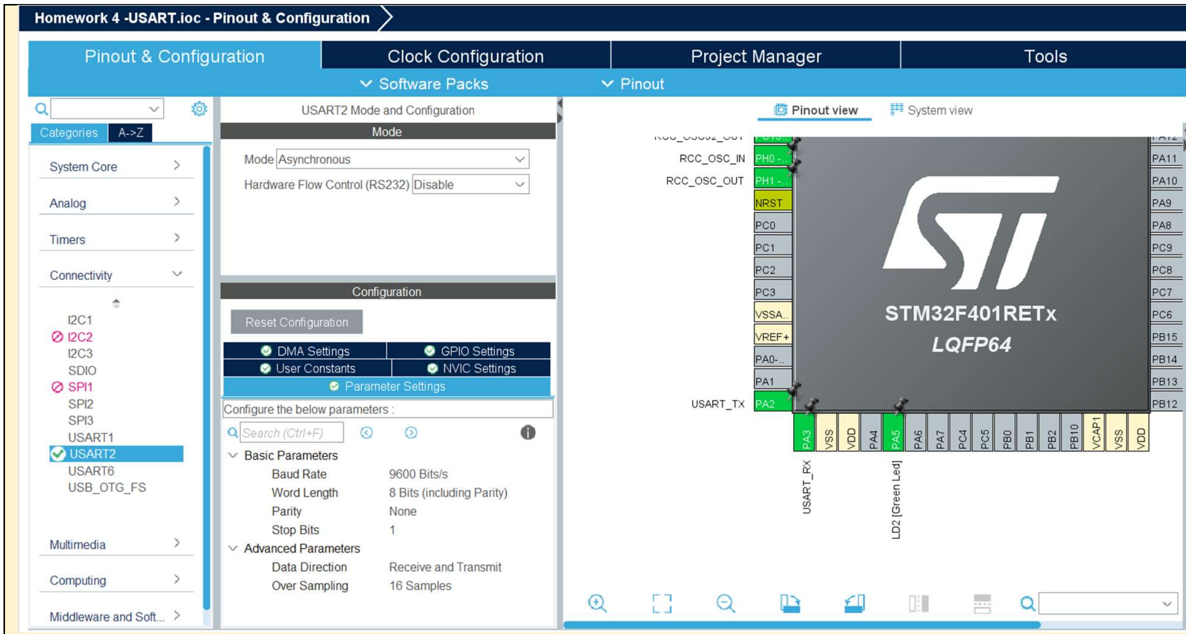
Mark	/11

Team name:	*A14*		
Homework number:	*HOMEWORK 04*		
Due date:	12/10/25		
Contribution	NO	Partial	Full
Mattia Di Mauro			x
Francesca Biondi			*x*
Lorenzo Castelli			x
Pietro Albrigi			*x*
Notes: none			

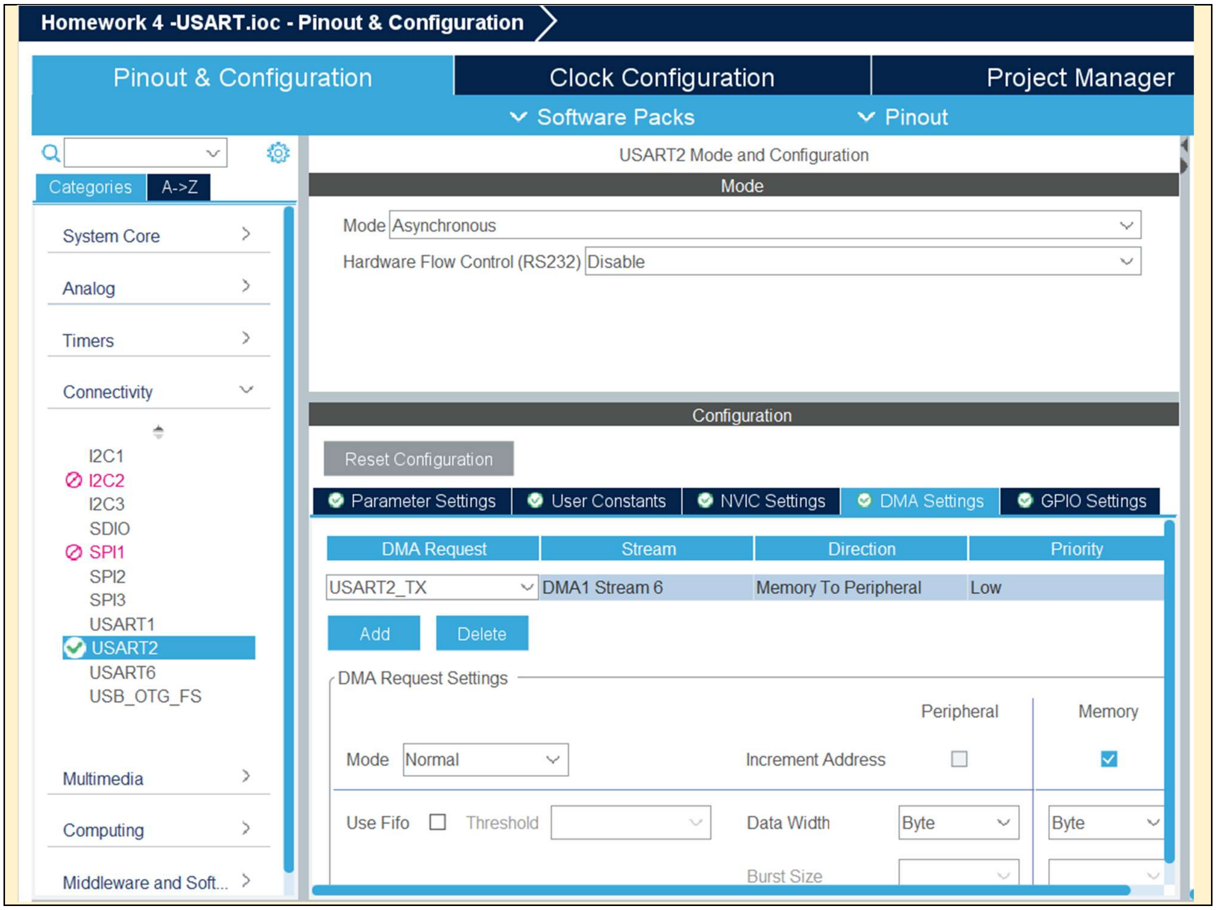
Project name	HOMEWORK 4		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

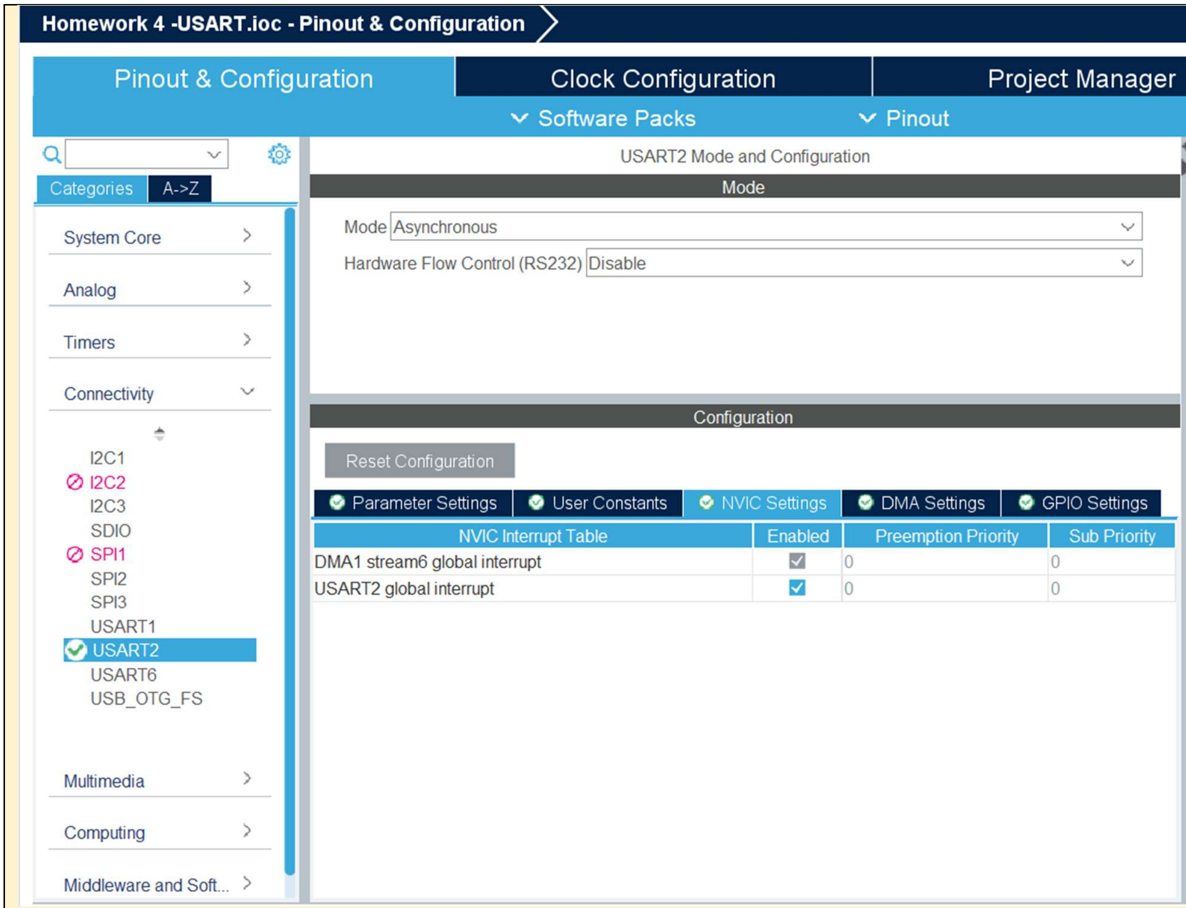
Part 1:

To send data from the microcontroller to the PC using the *USART interface*, after verifying that the TX and RX pins were correctly assigned, we configured *USART2* in the *Pinout & Configuration* tab under the *Connectivity* section. In the *Parameter Settings*, we set the *Baud Rate* to *9600 bits/s:*

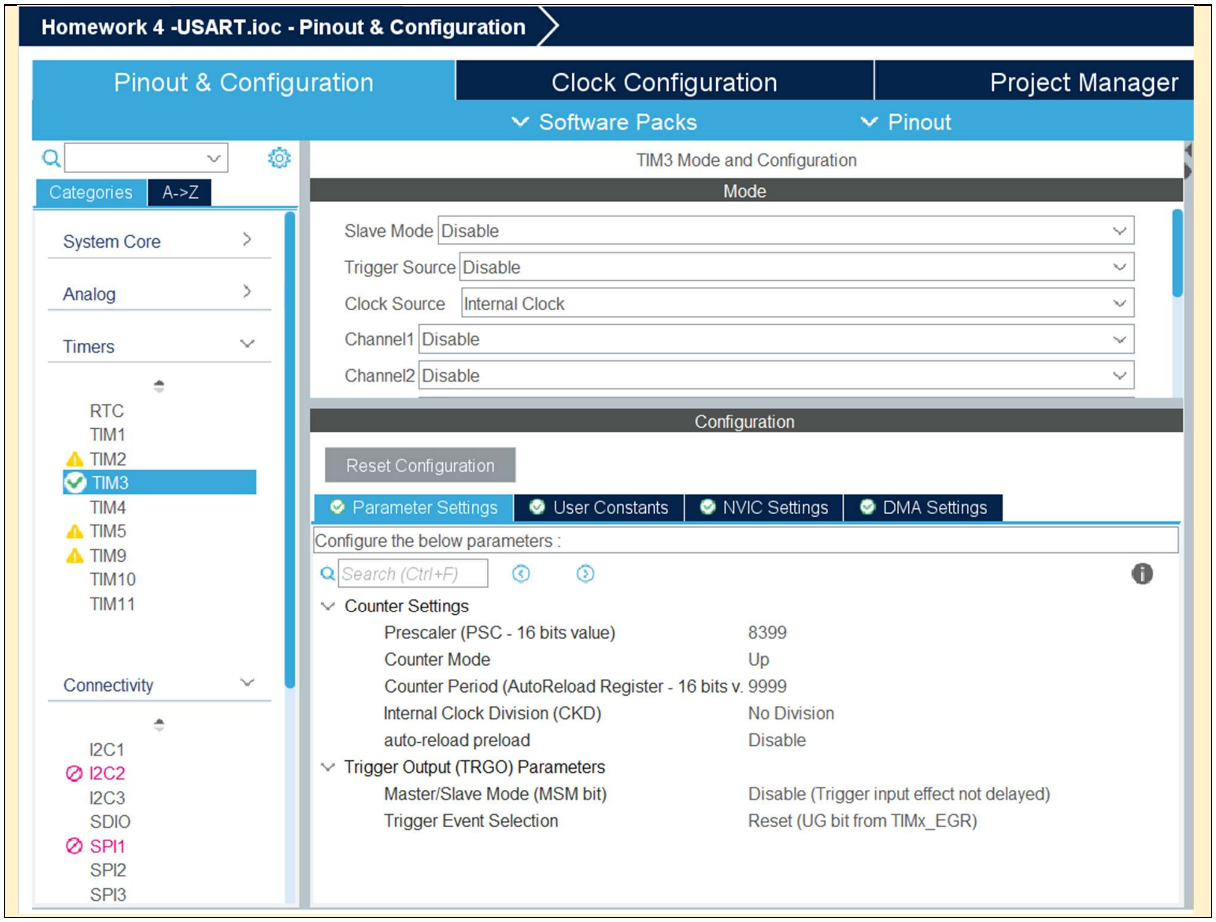


Under *DMA Settings*, we added a *DMA request* by selecting *USART2_TX* in *normal mode*, and finally, we enabled the *USART2 global interrupt* in the *NVIC Settings:*

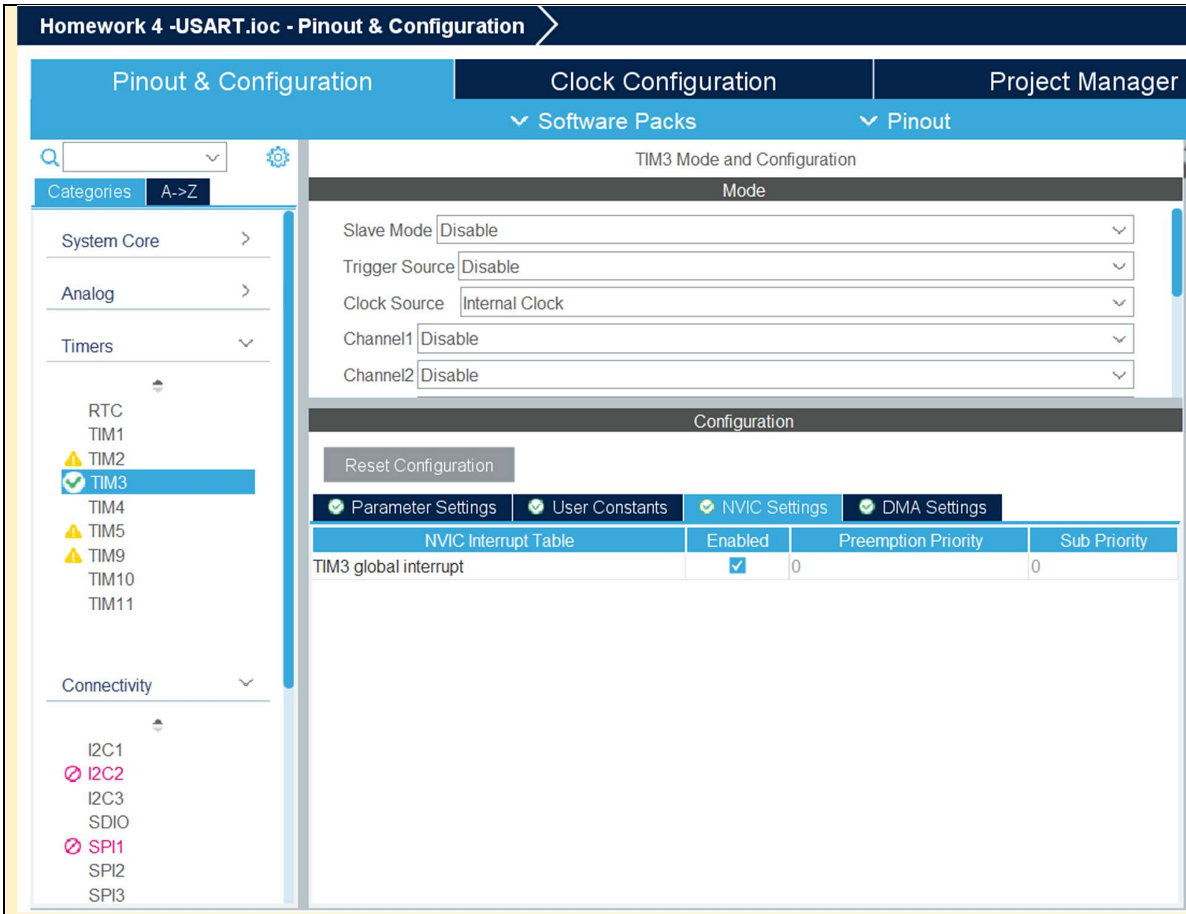

Homework 4 -USART.ioc - Pinout & Configuration
Pinout & Configuration
Clock Configuration
Project Manager
Software Packs
Pinout
Categories
A->Z
System Core
Analog
Timers
Connectivity
I2C1
I2C2
I2C3
SDIO
SPI1
SPI2
SPI3
USART1
USART2
USART6
USB_OTG_FS
Multimedia
Computing
Middleware and Soft...
USART2 Mode and Configuration
Mode
Mode Asynchronous
Hardware Flow Control (RS232) Disable
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
DMA Request
Stream
Direction
Priority
USART2_TX
DMA1 Stream 6
Memory To Peripheral
Low
Add
Delete
DMA Request Settings
Peripheral
Memory
Mode Normal
Increment Address
Use Fifo
Threshold
Data Width
Byte
Byte
Burst Size



To send data periodically, we used *TIM3* to generate an *interrupt every second*. To achieve this, we connected TIM3 to the internal clock, set the *prescaler* to *8399* and the *auto-reload register (ARR)* to *9999*, and enabled the *TIM3 global interrupt:*



Homework 4 -USART.ioc - Pinout & Configuration
Pinout & Configuration
Clock Configuration
Project Manager
Software Packs
Pinout
Categories
A->Z
System Core
Analog
Timers
RTC
TIM1
TIM2
TIM3
TIM4
TIM5
TIM9
TIM10
TIM11
Connectivity
I2C1
I2C2
I2C3
SDIO
SPI1
SPI2
SPI3
TIM3 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 Disable
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
Configure the below parameters :
Search (Ctrl+F)
Counter Settings
Prescaler (PSC - 16 bits value) 8399
Counter Mode Up
Counter Period (AutoReload Register - 16 bits v. 9999
Internal Clock Division (CKD) No Division
auto-reload preload Disable
Trigger Output (TRGO) Parameters
Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)
Trigger Event Selection Reset (UG bit from TIMx_EGR)



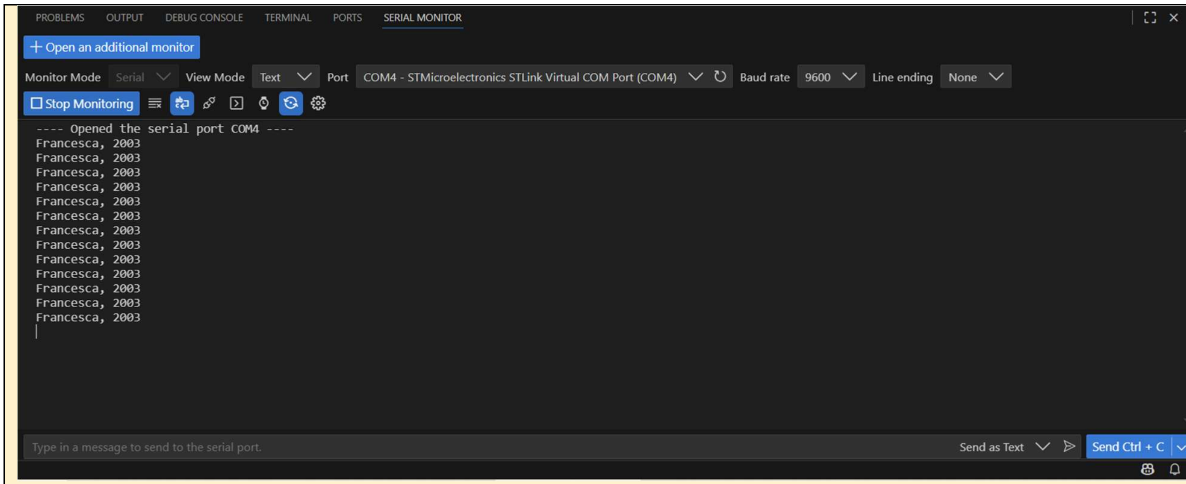
We then defined a *global flag variable* named `USART_send_flag`, which is set to `1` in the `HAL_TIM_PeriodElapsedCallback()` function when TIM3 overflows. This flag is used to determine when the data should be sent over USART. We also defined a *global character buffer* as `char string[50]`:

```c
/* USER CODE BEGIN PV */
char string[50];
int USART_send_flag = 0;
char name[] = "Francesca";
int birth_year = 2003;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim == &htim3) {
        USART_send_flag = 1;
    }
}
/* USER CODE END 0 */
```

In the `main()` function, after starting the timer, inside the infinite `while(1)` loop, we used an `if` statement to check if the `UART_send_flag` was set to `1`. If so, we formatted the message to be sent using `snprintf()`, storing its length in the `length` variable. We then transmitted the data using the `HAL_UART_Transmit_DMA()` function, which sends the string via *USART2* using *DMA*. The flag was then reset to 0.

```c
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      if(USART_send_flag){
          USART_send_flag = 0;
          int length = snprintf(string, sizeof(string), "%s, %d\r\n", name, birth_year);
          HAL_UART_Transmit_DMA(&huart2, string, length);
      }
    /* USER CODE END WHILE */
```

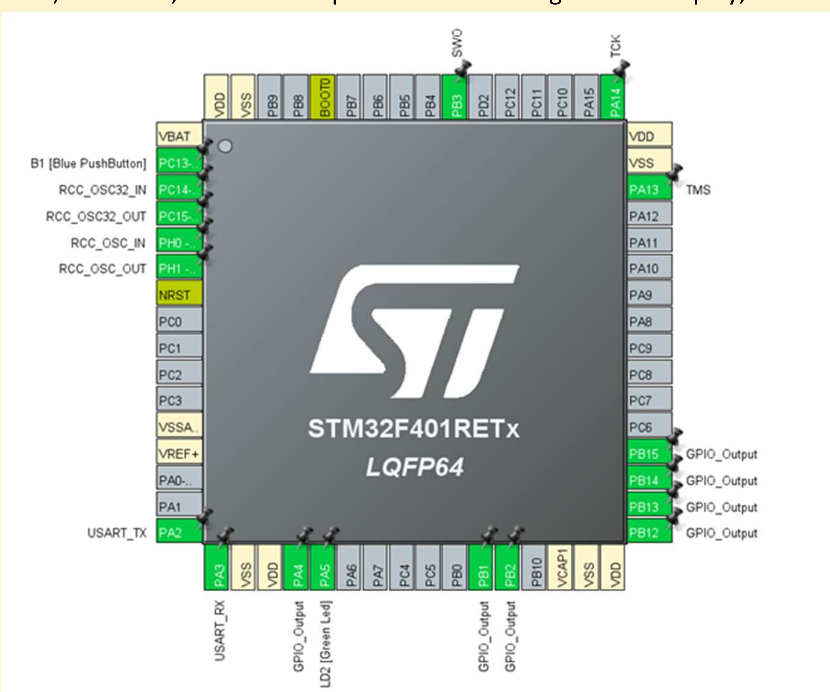
To monitor the data sent from the microcontroller, we used *Visual Studio Code* with the *Serial Monitor extension*. We selected the correct serial port associated with the STM32 board and set the *baud rate* to *9600 bits/s*, matching the *USART2* configuration. The successful reception and display of the expected string (for example, a name) in the serial monitor confirmed that our code was functioning as intended:



Part 2:

The aim of this part of the project was to implement a cyclic display algorithm on the LCD that shows the names of the group members in alphabetical order, with an automatic scroll every second.

Before proceeding with the code implementation, we configured the pins PA4, PB1, PB2, PB12, PB13, PB14, and PB15, which are required for controlling the LCD display, as GPIO output pins, as shown below:



To properly manage the LCD display using a high-level abstraction, we imported the "PMDB16_LCD" library into our STM32CubeIDE project.

Specifically, the file PMDB16_LCD.c was placed in the Core/Src folder, and the file PMDB16_LCD.h was placed in the Core/Inc folder.

In the header section of the main.c file, we then added the directive: #include "PMDB16_LCD.h" along with #include <string.h> which is required for string manipulation.

```c
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "PMDB16_LCD.h"
#include <string.h>

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
```

Next, we defined an array of string pointers containing the names of the group members, along with an integer variable that stores the total number of names, which is used to control the scrolling cycle.

```c
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */
    const char *nomi[] = {
        "Francesca",
        "Lorenzo",
        "Mattia",
        "Pietro"
    };
    int n_nomi = 4;
  /* USER CODE END 1 */

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

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
```

Before entering the main while(1) loop, we inserted the initialization calls for the LCD controller and for enabling the display backlight:

```c
  /* USER CODE BEGIN 2 */

  lcd_initialize();
  lcd_backlight_ON();
  /* USER CODE END 2 */
```

To implement the automatic scrolling every second, inside the infinite while(1) loop we created a for loop that iterates through all the names stored in the array.
For each iteration, the display is updated to simultaneously show two consecutive names: one on the top line and one on the bottom line.
The name corresponding to the current index i is displayed on the top row using the lcd_println() function.
For the bottom row, the logic depends on the position within the array: if the current index i equals 3 (the last element of the array), the first name (names[0]) is shown on the bottom line, thus ensuring a cyclic display.
Otherwise, the next name in the array (names[i+1]) is shown.

After each display update, the function HAL_Delay(1000) introduces a one second delay (1000 milliseconds), as specified in the project requirements.

```c
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      for (int i = 0; i < n_nomi; i++)
              {
                  lcd_println(nomi[i], 0);

                   if (i==3){
                     lcd_println(nomi[0], 1);
                     }
                   else {
                   lcd_println(nomi[i+1], 1);
                   }
                 HAL_Delay(1000);
              }
    /* USER CODE END WHILE */

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

/**
```

Finally, the code was tested on the board and confirmed to work as intended.

Professor comments: