

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 Albrigì			x
Notes: none			

Project name	HOMEWORK 4		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x
Part 1:			
<p>To send data from the microcontroller to the PC using the <i>USART interface</i>, after verifying that the TX and RX pins were correctly assigned, we configured <i>USART2</i> in the <i>Pinout & Configuration</i> tab under the <i>Connectivity</i> section. In the <i>Parameter Settings</i>, we set the <i>Baud Rate</i> to 9600 bits/s:</p>			

Homework 4 -USART.ioc - Pinout & Configuration >

Pinout & Configuration Clock Configuration Project Manager Tools

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: Asynchronous
Hardware Flow Control (RS232): Disable

Configuration
Reset Configuration
DMA Settings GPIO Settings
User Constants NVIC Settings
Parameter Settings

Configure the below parameters:
 Search (Ctrl+F)
 Basic Parameters
 Baud Rate: 9600 Bits/s
 Word Length: 8 Bits (including Parity)
 Parity: None
 Stop Bits: 1
 Advanced Parameters
 Data Direction: Receive and Transmit
 Over Sampling: 16 Samples

Pinout view System view

RCC_OSC_IN PH0-
RCC_OSC_OUT PH1-
NRST PC0
PC1
PC2
PC3
VSSA VREF+
PA0- PA1
USART_TX PA2
PA3-
VSS VDD PA4 PA5
LD2 [Green Led] PA6
PB1 PB2
PB10 PB11
VCAP1 VSS VDD
PB12 PB13
PB14 PB15
PB16 PB17
PB18 PB19
PB20 PB21
PB22 PB23
PB24 PB25
PB26 PB27
PB28 PB29
PB30 PB31

STM32F401RETx
LQFP64

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: 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

Mode: Normal	Increment Address <input type="checkbox"/>	Peripheral	Memory <input checked="" type="checkbox"/>
Use Fifo <input type="checkbox"/>	Threshold	Data Width: Byte	Byte
		Burst Size	

Homework 4 -USART.ioc - Pinout & Configuration

The screenshot shows the Pinout & Configuration software interface for a project titled "Homework 4 -USART.ioc". The main window is divided into three tabs: Pinout & Configuration (selected), Clock Configuration, and Project Manager.

Pinout & Configuration Tab:

- Categories:** A-Z
- System Core:** I2C1, I2C2, I2C3, SDIO, SPI1, SPI2, SPI3, USART1, USART2 (selected), USART6, USB_OTG_FS.
- Multimedia:**
- Computing:**
- Middleware and Soft...:**

Clock Configuration Tab:

Software Packs: USART2 Mode and Configuration

Mode: Asynchronous

Hardware Flow Control (RS232): Disable

Configuration Tab:

Reset Configuration:

Parameter Settings: NVIC Interrupt Table

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA1 stream6 global interrupt	<input checked="" type="checkbox"/>	0	0
USART2 global interrupt	<input checked="" type="checkbox"/>	0	0

User Constants:

NVIC Settings:

DMA Settings:

GPIO Settings:

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**

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 :

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)

Search (Ctrl+F) **?** **?**

Categories **A-Z**

System Core **>**

Analog **>**

Timers **>**

RTC
TIM1
▲ TIM2
TIM3
TIM4
▲ TIM5
▲ TIM9
TIM10
TIM11

Connectivity **>**

I2C1
I2C2
I2C3
SDIO
SPI1
SPI2
SPI3

Homework 4 -USART.ioc - Pinout & Configuration

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM3 global interrupt	<input checked="" type="checkbox"/>	0	0

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]:

```

48 /* USER CODE BEGIN PV */
49 char string[50];
50 int USART_send_flag = 0;
51 char name[] = "Francesca";
52 int birth_year = 2003;
53 /* USER CODE END PV */
54
55 /* Private function prototypes -----*/
56 void SystemClock_Config(void);
57 static void MX_GPIO_Init(void);
58 static void MX_DMA_Init(void);
59 static void MX_USART2_UART_Init(void);
60 static void MX_TIM3_Init(void);
61 /* USER CODE BEGIN PFP */
62
63 /* USER CODE END PFP */
64
65/* Private user code -----*/
66 /* USER CODE BEGIN 0 */
67 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
68 {
69     if (htim == &htim3) {
70         USART_send_flag = 1;
71     }
72 }
73 /* 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 `USART_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.

```

102 /* Initialize all configured peripherals */
103 MX_GPIO_Init();
104 MX_DMA_Init();
105 MX_USART2_UART_Init();
106 MX_TIM3_Init();
107 /* USER CODE BEGIN 2 */
108 HAL_TIM_Base_Start_IT(&htim3);
109 /* USER CODE END 2 */
110
111 /* Infinite loop */
112 /* USER CODE BEGIN WHILE */
113 while (1)
114 {
115     if(USART_send_flag){
116         USART_send_flag = 0;
117         int length = sprintf(string, sizeof(string), "%s, %d\r\n", name, birth_year);
118         HAL_UART_Transmit_DMA(&huart2, string, length);
119     }
120 /* USER CODE END WHILE */
121

```

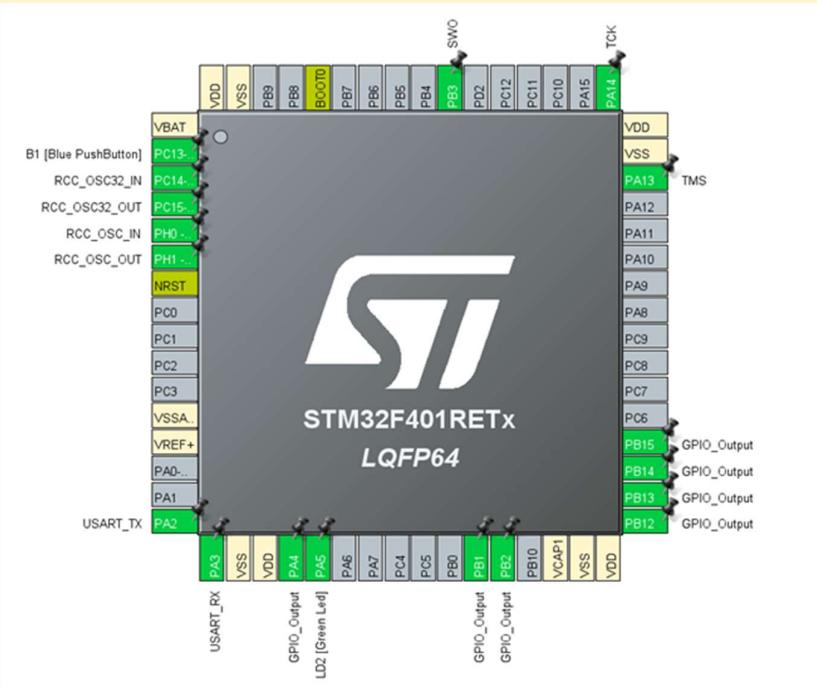
To monitor the data sent from the microcontroller, we used *Visual Studio Code* with the *Serial Monitor extension*. We selected the correct serial port associated with the STM32 board and set the *baud rate* to *9600 bits/s*, matching the `USART2` configuration. The successful reception and display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

The screenshot shows the STM32CubeIDE Serial Monitor window. At the top, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and SERIAL MONITOR. The SERIAL MONITOR tab is selected. Below the tabs, there are dropdown menus for Monitor Mode (Serial), View Mode (Text), Port (COM4 - STMicroelectronics STLink Virtual COM Port (COM4)), Baud rate (9600), and Line ending (None). A message in the monitor area says "---- Opened the serial port COM4 ----" followed by ten lines of the text "Francesca, 2003". At the bottom of the monitor area, there is a text input field with placeholder "Type in a message to send to the serial port." and a "Send as Text" button.

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.

```
17  */
18 /* USER CODE END Header */
19 /* Includes -----*/
20 #include "main.h"
21 #include "PMD816_LCD.h"
22 #include <string.h>
23
24 /* Private includes -----*/
25 /* USER CODE BEGIN Includes */
26
27 /* USER CODE END Includes */
28
29 /* 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.

```
62 /* USER CODE END 0 */
63
64 /**
65 * @brief The application entry point.
66 * @retval int
67 */
68 int main(void)
69 {
70
71     /* USER CODE BEGIN 1 */
72     const char *nomi[] = {
73         "Francesca",
74         "Lorenzo",
75         "Mattia",
76         "Pietro"
77     };
78     int n_nomi = 4;
79     /* USER CODE END 1 */
80
81     /* MCU Configuration-----*/
82
83     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
84     HAL_Init();
85 }
```

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

```
100    /* USER CODE BEGIN 2 */
101
102    lcd_initialize();
103    lcd_backlight_ON();
104    /* USER CODE END 2 */
105
```

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.

```
105  /* Infinite loop */
106  /* USER CODE BEGIN WHILE */
107  while (1)
108  {
109      for (int i = 0; i < n_nomi; i++)
110      {
111          lcd.println(nomi[i], 0);
112
113          if (i==0){
114              lcd.println(nomi[0], 1);
115          }
116          else {
117              lcd.println(nomi[i+1], 1);
118          }
119          HAL_Delay(1000);
120      }
121  /* USER CODE END WHILE */
122  /* USER CODE BEGIN 3 */
123
124  /* USER CODE END 3 */
125 }
126 /* End of function main */
127 }
128
129/* End of file main.c */
```

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

Professor comments: