|  |  |
| --- | --- |
| **Mark** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A2* | | |
| Homework number: | *04* | | |
| Due date: | 13/10/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| La Barbera Marco |  |  | *x* |
| Lotto Giulio |  |  | *x* |
| Majocchi Tommaso |  |  | *x* |
| Maffezzini Andrea |  |  | *x* |
| Pompilio Matteo |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name |  | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  | *x* |
| We successfully completed the homework.  Next, we explain all the steps for accomplishing our goals:  **Part 1:**  Firstly, we’ve configured the board pinouts and the DMA settings from the graphical interface of the CUBE IDE as follows, for the UART:  Immagine che contiene testo, schermata, Carattere, diagramma  Descrizione generata automaticamente  Then, from Connectivity -> USART2 we configure the parameters:        Where the DMA transfer complete interrupt (USART global interrupt) is required to signal the end of the transmission, otherwise subsequent transmissions might not be triggered properly, as experienced on our boards disabling it.  Secondly, for the Timer interrupt:    Then we pass to the “main.c” file. We defined the constant TEMPO to personalize the speed of UART transmissions (TEMPO of 1000 results in a htim2.Init.Period of (TEMPO\*10) – 1, hence every second an interrupt is generated from the TIM). Also we initialize the TIM 2 base generation in interrupt mode with this function:    Two global variables: *buffer* will contain the final string to send, *voltage* will simulate an increasing behaviour of a float value:    We can now implement the timer callback function as follows:    Where the *snprintf* function, given *buffer* and its size, sets it to contain the value of our voltage increasing simulator (exponentially by factor 1.2), truncated by 3 decimal positions; *length* will contain the number of characters parsed in our buffer; finally, we transmit the buffer with Direct Memory Access through our *uart2* interface.  Switching to MATLAB, we can now run the script “UART\_plot\_data.m” to plot, in a window of 10 seconds and with baud rate 115200 bps (as set on our board), the received voltage values:    where a reset has been forced on the 5th second.  We can state to have successfully accomplished our goals:   * To avoid the intervention of the CPU to retrieve data from the memory, instead performing DMA: this will also allow us to ignore the setting of a Timeout period during a normal transmission (with the *HAL\_UART\_Transmit\_DMA*function, which ensures that the CPU doesn't wait indefinitely); now, there’s no worry to block the CPU, hence no timeout is needed * To have a non-blocking behavior, by the usage of timers (instead of the *HAL\_Delay* function) * To finally successfully receive data to our UART receiver interface via MATLAB   **Part 2:** Firstly, we’ve configured the board pinouts from the graphical interface of the CUBE IDE as follows. We set PA4/5 and PB1/2/12/13/14/15 as GPIO\_Output.    We have also set the TIMER2 as we did for the **Part 1** project. Then we import the PMDB16\_LCD library in our project in the following way: we copy the “PMDB16\_LCD.c” file into the project/Core/Src folder and the “PMDB16\_LCD.h” file into the project/Core/Inc folder. Then we add this line in our main.c file:    Then we declare:   * a pointer to an array of char that contains the names of the members of the group; * an integer representing the number of members; * an integer that indicates the number of the row of the LCD where we want to write; * an integer representing a flag that will be used when the first name of the list is written for the first time (better explained later); * an integer containing the index of the array of char;     Then we use the *HAL\_TIM\_PeriodElapsedCallback()* function to write new lines on the LCD every second. We use the function *lcd\_clear()* to clear the lines written on the LCD. Then we check if the flag that we declared before, *flag\_first\_round* (initialized to the value of ‘0’) is ‘0’ or ‘1’:   * If the value is ‘0’ then we proceed to write on the LCD the name of the first member of the group in the second row of the LCD (this case happens only once at the beginning of the execution) and we set flag\_first\_round to ‘1’ ; * If the value is ‘1’ we write in the first row the member's name at the position of the name\_index. Then we check if the name\_index has arrived at the maximum value, corresponding to the number of members of the list and if so we change its value to ‘-1’ otherwise we do nothing. Lastly, we write the person's name corresponding to the position of name\_index+1 on the second row and increase the index (name\_index++) for the next cycle. In this way when the last member's name is written in the first row the name of the first member is written on the second one.     Last of all we initialize the LCD and turn it on in the *main()* function using the following lines: | | | |
| Professor comments: | | | |