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

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A2* | | |
| Homework number: | *05* | | |
| Due date: | 20/10/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| La Barbera Marco |  |  | *x* |
| Lotto Giulio |  |  | *x* |
| Majocchi Tommaso |  |  | *x* |
| Maffezzini Andrea |  |  | *x* |
| Pompilio Matteo |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | UART rx from pc and ADC potentiometer voltage reading | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  | *x* |
| We successfully completed the homework.  Next, we will explain all the steps for accomplishing our goals:  **Part 1:**  Firstly, we’ve configured the board pinouts from the graphical interface of the CUBE IDE.  For the LCD configuration we set PA4/5 and PB1/2/12/13/14/15 as GPIO\_Output and for the USART configuration we set PA2 and PA3 as follows.    Then we imported the PMDB16\_LCD library in our project in the following way: we copied 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 added this line in our “main.c” file:    Then, from “*Connectivity” -> “USART2”* we configure the parameters:      For this task we set direction from peripheral to memory:    We want to send 1 byte at each transmission, so we define an array *RxData* with LENGTH equal to 1 (we will use a different value for the advanced version of the assignment). Then we define two variables, one for the char conversion and the other to compose all the received data into a string.      In the *main()* we inizialize the LCD and then we use the *HAL\_UART\_Receive\_DMA* function to receive the first byte.    We implemented the UART receive callback function as follows:  The function sprintf converts RxData into a char, then using *strcat* we add the char to the string and we print the string on the LCD.  Finally *HAL\_UART\_Receive\_DMA* receives the new data.    With this implementation, the user sends from MATLAB one character per transmission.  In addition to this we implemented a version that requires only two transmissions to send the string. First, the user sends a number specifying the number of bytes of the string that has to be displayed on the LCD screen and then he can send the full string (up to 16, LCD maximum characters in one line).  • *length* is a constant that defines the received data dimension, it is set as the LCD maximum characters in a line.  • *size* is a constant that defines the number of characters that can be sent as information size (ex: min = 01, max = 99).  • *RxData[length]* is an array that contains the received data.  • *size[char\_size]* is an array that contains the ASCII information of the data transfer size.  • *packet\_size* is an integer that contains the size of the packet that has to be received, converted into an integer.  • *rx\_flag* is a flag that is used to tell if the packet size is been received or not.  • *c* is a char that stores the value of *RxData* in a certain moment of the execution.  • *string[length]* is a string that stores the entire string received.  • *row* is an integer that indicates the line of the LCD where we want to write.    In the *main()* we inizialize the LCD and then we use the *HAL\_UART\_Receive\_DMA()* function to receive the size data.    We implemented the UART receive callback function as follows:  If the *rx\_flag* is equal to zero, the size information has already been received; we convert that data into its corresponding integer value, we use it in the function *HAL\_UART\_Receive\_DMA()* to receive the string and we set the flag to one.  Otherwise we convert the *RxData* into a char string using a for loop and the same functions used in the first part of the part 1 and we empty *Rx\_Data*. Then we print the string on the LCD screen, we set the flag to zero and receive the new size data.    **Part 2b:**  Firstly, we configure the board pinouts for the ADC pin PA1 (connected to our board’s potentiometer as verified in the schematic file) as follows:    Then, from “*Analog” -> “ADC1”* we configure the parameters (after enabling IN1), setting the External Trigger Conversion Source as TRGO, that will be set soon from the Timer settings of the TIM2. Also we set the sampling time to 480 cycles:    And enabling the interrupt:    For the timer, we set the usual values (timeout parametrized by the constant TEMPO), plus we enable the TRGO to trigger the callback function of the ADC (Trigger Event Selection as *Update Event*):    Finally, we configure the UART, for remote transmission of the values read by the potentiometer to the remote MATLAB console, as done and explained in the previous homework (using DMA):    Then in the “main.c” file we initialized the TIM 2 base generation in interrupt mode and the ADC in interrupt mode with these functions:    We declared two global variables:   * *buffer* will contain the final string to send; * *voltage* will contain the voltage readed by the ADC after a proper conversion acted by the *reading* variable (from range 0/4096 to 0/3.3V).   The timer triggers the start of conversion at 1 Hz rate as required, then at the end of conversion the ADC generates an interrupt. We implemented the ADC callback function as follows:    The *snprintf()* function, given *buffer* and itssize*,* sets the buffer to contain the value of our measurement, truncated by 3 decimal positions. The 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\_read\_data.m” to read the voltage values at baud rate 115200 bps (as set on our board): as in the previous homework, we successfully receive the data on our console.  **Part 2c:**  The pinout configuration is very similar to the previous project (part 2b). We just disable the USART pins (PA2, PA3) and enable as GPIO\_Output the PA 4-5, PB 1-2-12-13-14-15 pins.    The ADC and timer settings are exactly the same as part 2b.  Then we initialize the LCD following the procedure of the previous homework: we copied 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 added this line in our “main.c” file:    We initialize the ADC, the timer and the LCD as well:    Then we modify the ADC interrupt callback as following:    With respect to the part 2b version we delete the UART communication line and we add a few others, aimed to implement the required LCD behavior. At the beginning we refresh the lcd screen with *lcd\_clear().* Then we define the *bar* variable that is needed to convert the ADC reading (0/4096 value) into a value compatible with the one required by the *lcd\_drawBar()* function (0/80 value). Since the division might generate a float result, a casting is also needed. In order to smoothen out the bar update rate, the *TEMPO* variable can be decreased. The *snprintf()* function parses the *voltage* in the *lcd\_buffer*: “Voltage: 1.789 V” for example. The *lcd\_println()* function prints the lcd buffer string on the display showing the exact voltage generated by our board’s potentiometer truncated by 3 decimal positions.  Immagine che contiene testo, elettronica, Ingegneria elettronica, luce  Descrizione generata automaticamente | | | |
| Professor comments: | | | |