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

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *B5* | | |
| Homework number: | *09* | | |
| Due date: | Tuesday, 5th December, 08:30 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| Ghidini Alessandro |  |  | *x* |
| Latino Francesco |  |  | *x* |
| Luppi Eleonora |  |  | *x* |
| Bravin Riccardo |  |  | *x* |
| Feltrin Elia |  |  | *x* |
| Notes: | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | *Keyboard readout* | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  | *x* |
| Explanation:  The project aims to enhance the functionality of the embedded system by incorporating a keyboard for user input. This involves configuring a timer, USART communication and managing the keyboard’s pins.  **Part 1a:**  In this first part of the project, we need to scan the different columns of the keyboard in polling mode. In the GUI we enable GPIO inputs PC2, PC3, PC12, and PC13 should be enabled and GPIO outputs PC8, PC9, PC10, and PC11. We generate the code.  We initialize a vector to store rows pins *row\_shift[4]* and one for columns pins *column\_shift[4]* addresses; a matrix to store the numbers and the letters associated to each button *matrix\_button[4][4]* and three matrix called *flag\_matrix[4][4]*, *tempo\_riga [4][4]* and *print [4][4]* used to store the status of each button, the press duration and to manage the print operation.  In the *while(1)* loop we use 2 nested for cycles to iterate on the column and on the rows. In particular, with the first for cycle we keep one column fixed and we enable the pin with the function *HAL\_GPIO\_WritePin(GPIOC,column\_shift[i], GPIO\_PIN\_SET)*. Next, with the second for cycle we iterate on the 4 rows.  For each row we do some controls. Firstly, if the button is pressed, we read a logic 0 in the pin with the function *HAL\_GPIO\_ReadPin(GPIOC, row\_shift[j])==GPIO\_PIN\_RESET)*. We control that the button is pressed for 3 consecutive cycles to raise the flag *print[4][4]*, in the specific position for that button, to print the character. To do this we use a counter *tempo\_riga[4][4]*. In this way we manage the debouncing without making the code stop in the same point to control that the button has been pressed for long enough.  Then, if we read that the button has been released, the *tempo\_riga[4][4]* is set to 0 and we set another flag *flag\_matrix[4][4]*: this flag is used to avoid that the character is printed multiple times when the button is kept pressed. We want that same character to be printed only if the button is released and pressed again. So, an if construct sends the character through UART communication if the flag *print* is 1 and the *flag\_matrix* is 0. Once we have printed, we raise the *flag\_matrix* to 1 and set the *print* one to 0. In this way, after all the iterations, the same button will print the character only if the *flag\_matrix* is back to zero and this happens only if the button has been released!  **Part 1b:**  To implement the desired functionality of the button matrix, a timer (TIM3) with interrupts at a frequency of about 100Hz needs to be set up, with a prescaler of 8400-1 and a counter of 100-1. To enable the use of a button matrix of the green board, GPIO inputs PC2, PC3, PC12, and PC13 should be enabled, while GPIO outputs PC8, PC9, PC10, and PC11 need to be configured.  Once the code is generated, two global arrays are created for input and output, respecting the correct column and row ordering: *{GPIO\_PIN\_3, GPIO\_PIN\_2, GPIO\_PIN\_13, GPIO\_PIN\_12}* and *{GPIO\_PIN\_8, GPIO\_PIN\_9, GPIO\_PIN\_10, GPIO\_PIN\_11}* respectively. Within the *HAL\_TIM\_PeriodElapsedCallback()*, an array of characters to be displayed (*"CDEF89AB45670123"*), an array of 16 values (initially set to 0) to track button states, and a column counter have been declared.  During each timer interrupt, the current column output pin is activated, and all four rows are read sequentially. For each row, it is checked whether the button was pressed during the previous fourth call using the states array. If the button was not pressed, the current character pressed is sent through UART inside a string. If the button was pressed and is now released, the button's status is reset. Finally, the current column pin is deactivated, and the column counter is incremented with a modulus 4 operation.  To initiate the timer and start generating interrupts, the *HAL\_TIM\_Base\_Start\_IT(&htim3)* function has been called just before the main's *while(1)* loop. This ensures that the timer is set up and ready to produce interrupts periodically. | | | |
| Professor comments:  The algorithm proposed in 1a is a bit cumbersome and it could be optimized just using one array of counters.  In 2a it is not very clear how you managed the debounching. Providing portions of code would help. | | | |