|  |  |
| --- | --- |
| Mark |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A2* | | |
| Homework number: | *10* | | |
| Due date: | 1/12/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| La Barbera Marco |  |  | *x* |
| Lotto Giulio |  |  | *x* |
| Majocchi Tommaso |  |  | *x* |
| Maffezzini Andrea |  |  | *x* |
| Pompilio Matteo |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | SPI LED MATRIX | | |
| 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:  **First part (Keyboard):**  First of all, we configure the board pinout for the keyboard: PC2/3/12/13 for the rows to be read, as GPIO\_Input; PC8/9/10/11 for the columns to be scanned, and so written, as GPIO\_Output.    From “Timers*”*, we enable TIM2 and TIM3, as we’ll both need a timer to scan the next column, and one for the debouncing timeout (timeouts parameterized by the constants TEMPO and TEMPO2). Finally, the UART interface in DMA mode:    And here our interrupt table:    In the “main.c” file we declared the two timers (4ms for each column’s scan, 50ms as the debouncing allowed time), and a struct type to store all ports and pins relative to the keyboard connections: this, for easiness of use in the code, by allowing access to the data structures *ROWS* and *COLS* through indexing:      To conclude the declaration part, out global variables (all set to 0 except for *c\_old*, initialized to -1 to be different from *c*, and for *buttons[]*, array of characters set as indicated on our physical board):    In the **HAL\_TIM\_PeriodElapsedCallback** function, we handle the two timeouts.  At each TIM2 timeout, we activate one column (by writing on the relative GPIO pin) and scan all rows (by reading from the relative GPIO pin). We only perform this when no button has been kept pressed in the past *(TEMPO2 =) 50* milliseconds (via the *pressed\_flag* variable): when this is set to zero and a button is pressed (state GPIO\_PIN\_RESET) we enter in a state where only the relative column is active (all buttons in the same column can be pressed), and we save the row (*row* variable) to be checked by the debouncing routine; finally, the timer relative to this routine is started, and the *scan* variable incremented.  We also highlight an important defect of the keyboard, that is how **the second row has left-shifted scanning columns**: we solved this by saving the value of *scan* into *col* and decrementing its value, finally re-writing on the right column to activate, disabling the shifted one; now the index *c* can be correctly computed, by the formula *c = col + (4\*row).*  At each TIM3 timeout, we simply read again the value of the saved row, and check whether the last computed index *c* holds a different value from the previous one *c\_old* (here updated), so to avoid printing repetitions when holding the button pressed for more than TEMPO2 milliseconds.  In case the button is not pressed anymore, it can mean that either it has been pressed for a time lower than TEMPO2 milliseconds, or that it was pressed and released after the timeout, hence we can reset the global state *pressed\_flag* and set to -1 the index *c\_old.*  Finally, we can press all buttons on our keyboard and verify the relative printed chars on our MATLAB console*.* Note how the last three ‘0’s are printed in sequence: this is a wanted behavior, as the button was pressed and released for exactly three times:    **Second Part (Encoder):** | | | |
| Professor comments: | | | |