|  |  |
| --- | --- |
| 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 | Keyboard and Encoder | | |
| 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 enabled, and so written, as GPIO\_Output.    From “Timers*”*, we enable TIM2 and TIM3, as we’ll both need a timer to scan the columns, and one for the debouncing timeout (timeouts respectively 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, 20ms as the debouncing allowed time, that we’ve established to be appropriate for cases of fast pressings), 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, a few other global variables all set to 0 except for *c\_old*, initialized to -1 so to differ from *c*, and for *buttons[16]*, array of characters (set accordingly to the silkscreen of 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 *50* milliseconds *(TEMPO2 )*  checking the *pressed\_flag* variable: when it is zero and a button is pressed (state GPIO\_PIN\_RESET) we enter in a state where only the related column is active (all buttons in the same column are enabled), and we save the row index of the pressed button (*row* variable) to be later checked by the debouncing routine; finally, the timer 3 is started, and the *scan* variable incremented.  We also highlight an important defect of the keyboard: **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 and 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 very short 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):**  First of all, we configure the board pinout for TIM3 Channel 1 (PC6) and TIM3 Channel 2 (PC7).    We enabled the TIM3 in encoder mode (counter period parameterized by the constant STEPS) just as in figure:    Then we also set the TIM2 parameters in the ordinary way (counter period parameterized by the constant TEMPO):    We enable the UART2 and the DMA1 as shown in figure:    We go on the NVIC window to set all the flags just as in the figure:    In the “main.c” we declared the following variables:   * *new\_position* to sample the current value of the encoder position; * *old\_position* is the variable that contains the old value of the encoder position (used to calculate the rpm); * *rpm* is the variable that contains the speed value calculated in the timer callback function; * *string[]* and *string\_length* are the variables used to communicate the data via Uart     The TIM2 triggers the timer callback every second. It starts by storing the value of the *new\_position* sampled in the old callback in the *old\_position* variable. Then it reads the current position value using the function \_\_*HAL\_TIM\_GET\_COUNTER().* Then we check if an overflow or an underflow occurred supposing that the biggest number of steps the potentiometer can do in a second is half of the value of STEPS (border condition). Then we calculate the rpm value depending on the condition we are in (overflow, underflow, or normal condition). Then we send the rpm value to the PC through a UART.    In the *main()* function we started the two timers and the TIM3 encoder mode.    This is a screenshot taken from our UART MATLAB console with some measurements. We can notice how the rpm value is positive with a clockwise rotation and how the value is negative with a counterclockwise rotation: | | | |
| Professor comments: | | | |