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

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A2* | | |
| Homework number: | *08* | | |
| Due date: | 17/11/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| La Barbera Marco |  |  | *x* |
| Lotto Giulio |  |  | *x* |
| Majocchi Tommaso |  |  | *x* |
| Maffezzini Andrea |  |  | *x* |
| Pompilio Matteo |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | MEMS Accelerometer | | |
| 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 1b:**  First of all, we configure the board pinout for the I2C transmission with the accelerometer (pins PB8 and PB9 as verified in the schematic file) and for the UART transmission with the PC, pin PA2:    Then, from “connectivity*”* we enable the I2C communication in this way:    And the UART for the remote transmission of the x y z acceleration values to the MATLAB console, in DMA mode:    Finally, we set the timer with the usual values (timeout parametrized by the constant TEMPO):    Then in the “main.c” file we initialize the TIM2 base generation and the I2C transmission in this way:      Here we setup the control registers of the sensor and we perform the identification of the specific accelerometer mounted on our board (LISDE or LISDE12), by expoiting the return of the HAL\_I2C\_Master\_Transmit function: if it returns HAL\_OK then we are sure that the accelerometer is one or the other. In fact the only difference between the two is the address as pointed out in the following picture.    MEMS\_WR\_ADDRESS (or MEMS12\_WR\_ADDRESS) is the address of the sensor and MEMS\_REGISTER\_X(/Y/Z) is the address of the register where the converted x (y/z) acceleration is stored as explained in the datasheet:    We defined the variables:   * *TEMPO* will dictate the frequency of the timer; * *size* is a variable that indicates the number of bytes received or transmitted during the I2C communication; * *timeout* is the timeout time in ms; * *string* and *string\_length* are used as usual to convert an integer value into a string; * *x, y and z*  are the variables that will store the respective acceleration values;     In the TIM2 callback function, triggered at 1 Hz rate, we arrange the 3 transmissions with the 3 different addresses of x, y and z registers. With the 3 functions *HAL\_I2C\_Master\_Receive* we read and store the 3 accelerations. Then the proper value is converted in the acc\_g\_x, acc\_g\_y, acc\_g\_z variables before converting it into a string using the *snprintf()* function. Finally, we send the string to MATLAB through the UART.  Switching to MATLAB, we can now run the script “UART\_read\_data.m” to read the voltage values at a baud rate of 115200 bps (as set on our board). We successfully receive the data on our console:    **Part 1c:**  The configuration of this project is exactly the same as the previous one except that we enable the DMA for the I2C communication and its interrupts:    Then in the “main.c” file we initialize the TIM2 base generation and the I2C transmission as in the part 1b: we are not required to use I2C DMA transmissions here, since the process is not interfering with any other:    The control registers and all the addresses are the same as part 1b. The only difference stands in the address of the register where the acceleration values are stored, and in the presence of a unique data structure for the whole reading (*xyz\_data*). In particular, we now use the auto-increment feature of the LIS2DE sensor:    In order to enable multiple reads utilizing DMA, the *MEMS\_REGISTER\_X\_AUTO\_INCREMENT* variable contains the address of the first register to be read (OUT\_X), with the MSB set to ‘1’ as expained in the datasheet:    Also in the datasheet, in the register mapping here reported, we can see that between OUT\_X, OUT\_Y and OUT\_Z there are 2 register reserved, so the number of register to be read with auto-increment is 5 (here the value of the global variable *multiple\_size*). The values of the register with address 0x2A and 0x2C will hence be ignored:    For a proper communication to be set, we’ve structured the callbacks in a three-layer cascade system:   * In the TIM2 callback function, triggered at 1 Hz rate, we transmit the address of the X register in auto-increment mode * In the second callback, we store the data (5 bytes) using *HAL\_I2C\_Master\_Receive\_DMA* * In the third callback we select only the useful values (index 0, 2 and 4) and at the same time we convert it on the acc\_g\_x, acc\_g\_y, acc\_g\_z variables. Then we convert the 3 accelerations into a string using the *snprintf()* function. Finally, we send the string to MATLAB through the UART:     Switching to MATLAB, we can now run the script “UART\_read\_data.m” to read the voltage values at a baud rate of 115200 bps (as set on our board). We successfully receive the data on our console: | | | |
| Professor comments: | | | |