|  |  |
| --- | --- |
| **Mark** | **/11** |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A5* | | |
| Homework number: | *HOMEWORK 07* | | |
| Due date: | 3/11/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| Alessio Spineto |  |  | *x* |
| Riccardo Lamarca | *x* |  |  |
| Sofia Cecchetto |  |  | *x* |
| Annamaria De Togni |  |  | *x* |
| Emma Crespi |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name |  | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  |  |
| **Complete the I2C thermometer project 1b:** read the temperature measured by the LM75 and send it to a remote terminal every 1 second.  The LM75 temperature sensor uses the I2C protocol on PB8 for the clock (SCL) and PB9 for the data (SDA) lines. We enabled I2C1 for the two pins using the pinout view and set I2C1 mode to I2C.    Since the I2C bus is connected to different peripherals, we set two defines with the address of the LM75. Both addresses are left-shifted by 1: *LM75\_ADDRESS\_W* has the R/W bit equal to 0, while *LM75\_ADDRESS\_R* has the R/W bit equal to 1.  We defined two other constants that depend on the version of the sensor (LM75 or LM75B), that can be changed to allow the code to work with each version.    We also set *LM75\_TEMP\_ADDRESS* as the address of the Pointer register of the sensor, which will allow us to select the temperature register.    We also need to include <string.h>.    We decided to use I2C in DMA mode, so we enabled the DMA requests to transmit and receive data.    We also enabled the event and error interrupts.    We used USART in DMA mode with transfer mode USART2\_TX and high priority.  We also set TIM2 to 1Hz and enabled its global interrupt, along with USART global interrupt.  Inside the main(), we initialize TIM2 in interrupt mode. We set the pointer register of the LM75 to the temperature register by writing its address (0x00) to the temperature sensor. This allows us to declare that we want to receive temperature values.    Our solution uses two callback functions, one for the I2C communication and one for the timer.    Every second, TIM2 initiates data reception from the sensor. Data is stored in the uint8 array *buffer.*    On the first callback, the two elements of the array are stored in the uint16 variable *previous* by left shifting the MSByte and adding the LSByte. The flag *call\_count* is set to 1 and another data reception is initiated.  The second value read from the sensor is stored in the uint16 variable *current.*    This allows us to compare the two consequent readings. If the integer part of two successive readings is different but the decimal part is the same, we can suppose that we performed the reading at a time where the LSB are updated but not the MSB\*. When this happens, we discard the *previous* and *current* values and start another reading. Since the *call\_count* flag has been set to 0, this starts the entire process again, checking the next two consequent readings.    If the bug check goes well, we convert the value *current* to a temperature.  We use *previous* as a temporary variable where we store the reading except from the sign bit.  After that, we do a right shift to exclude the bits of the LSB part that don’t carry any information (5 in the LM75B and 7 in the LM75) and cast to float.  We check the first bit (sign bit) of the variable *current* and multiply the value by –1 if necessary.  In the output string we multiply the value by the correct precision (0.125 in the LM75B and 0.5 in the LM75) and send it to the remote terminal.    The conversion was done following the suggested method on the LM75B data sheet.    \* Comment on our approach to the bug  This solution simply detects the behaviour described during the laboratory where the temperature seems to spike/drop suddenly during an otherwise decreasing/increasing trend.  We believe that this is due to the LSB and MSB not being updated at the same time by the sensor, so performing a reading between the two updates shows an incorrect value.  Our approach detects this kind of discrepancy and discards the problematic value, substituting it with a new reading.  An improvement could be to keep track of the overall trend of the temperature value. This would allow us to make sure that our target behaviour (same decimal part, different integer part) is not due to a fast change in temperature but is really due to a bug and does not fit the overall curve. If the measurements are stored and not only shown in real time, interpolation could be used to fix incorrect values in the overall curve. | | | |
| Professor comments: | | | |