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

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | I2C temperature sensor with interrupt | | |
| Not done | Partially done   (major problems) | Partially done   (minor problems) | Completed |
|  |  |  | *x* |

|  |
| --- |
| We have done the whole homework, here is our explanation:  **Homework 6a:**  Firstly, we have configured the board as shown below:  We have set the Pins PB8 and PB9 to I2C1\_SCL, I2C1\_SDA respectively.(by default, the USART2 is already configured).  Then we enabled the I2C1:  After doing that, we enabled the timer2 as shown below:  Where Prescaler and Counter Period are set to 8399 and 9999 respectively as (8399+1)\*(9999+1)/83e6 = 1  Later, we went to the NVIC table in order to enable the TIM2 global interrupt:  Now we have finished all the configurations in the GUI, then we went to main.c, firstly, we added a symbol to distinguish between the LM75 and LM75B sensor models (for reasons that will be clear later):  this is defined when the board mounts the LM75B sensor.  Then, we defined two global variables in order to write the two HAL I2C Master functions easily:    Then in the main, we started the I2C and Timer2 in a safe way:  Then, we created the callback function as shown below:  In the callback function, we first define a series of variables that we will use them later(and we will explain some of them later). Then we call the HAL\_I2C\_Master\_Receive. But you may notice in the function the number of bytes we are going to receive is 6 instead of 2. The reason we do this is to resolve the bug that is proponed in the slides:  In order to explain this bug (that happens with the LM75B version only) and how we solve it, we need to clarify one thing:  When the LM75B is accessed the conversion in process is not interrupted (that is, the I2C-bus section is totally independent of the Sigma-Delta converter section) and accessing the LM75B continuously without waiting at least one conversion time between communications will not prevent the device from updating the Temp register with a new conversion result. The new conversion result will be available immediately after the Temp register is updated.  The above paragraph we copied from the LM75B.pdf, the important thing is when the sensor is updating the result in the register, the I2C can still access the Temp register. Therefore, if we access the Temp register when the sensor is updating the result, we may encounter something weird like in the slides, the microcontroller read 26,26.875,25.875: where we noticed that the second one has a integer part equal to the integer part of the first one but the decimal part of the second part is equal to the second part of the third one, thus when the microcontroller was reading the second one via the I2C, the sensor is doing the conversion, and just have changed the decimal part but not the integer part and when the microcontroller was reading the third one, the conversion has done. That is the bug we need to solve.  In order to solve the bug, we read 3 values instead of 1. We now back to the code: if the first value read and the second value read is the same, means we don’t meet the bug, so we just transfer the read value in the right form: first, the variable temperature\_final is int16\_t so the compiler can recognize the two’s complement. Then we just move the first element of the datas\_temperature(the integer part of the fisrt temperature read) 8 positions to right(igual to datas\_temperature[0] 00000000) and put the second element of the datas\_temperature(the decimal part of the fisrt temperature read) to the least 8 significant bits of the variable temperature\_final using the logic operation “or”. Otherwise, the fisrt and second read temperature are different which means we encounter the bug. So we just put the third read temperature in the temperature\_final. Which according to the conversion time description:  The LM75B performs the temperature-to-data conversions with a much higher speed than the LM75A. While the LM75A takes almost the whole of conversion period (Tconv) time of about 100 ms to complete a conversion, the LM75B takes only about 1⁄ 10 of the period, or about 10 ms. Therefore, the conversion period (Tconv) is the same, but the temperature conversion time (tconv(T)) is different between the two parts. A shorter conversion time is applied to significantly reduce the device’s average power dissipation. During each conversion period, when the conversion is completed, the LM75B becomes idled and the power is reduced, resulting in a lesser average power consumption.  The third read temperature is for sure the correct new result. Then we divided by 256.0 in order to change it to degree.  Finally, if something wrong with the HAL\_I2C\_Master\_Receive, we send a message related to this using the UART.  And the code works as we expected:  Where we have put our finger on the sensor for a while.  **Homework 6b:**  Project 1b:  Firstly, we have configured the board as shown below:  Exactly the same configuration as the previous project.  Then,we enabled the I2C1:  After doing that, we configured the timer2 in order to control the sampling frecuency to 1s as shown below:  Once again: Prescaler and Period to 8399 and 9999 respectively as (8399+1)\*(9999+1) /84e6 = 1  Then we added a DMA configuration to the UART2:  Before we went to the main.c, we enebled the folowing interruptions in the NVIC table:  In main.c, we first define a series of global variables:  Then in the main, we first check the model of the sensor on the board and Initialize the accelerometer setting the correct registers to enable normal mode operation, with all 3 channels active, 1 Hz update rate, no high pass filter and ± 2g Full Scale range and start the timer 2:  Then, we created our callback function:  Where we first read datas from the sensor in a safe way then we transfrom the data in the correct form and send it to our PC using the UART2 in DMA.  Finally, the code works as we expected:  Project 1c:  Firstly, we have configured the board in the same way as the previous project:  Then we enabled the I2C1 and added a DMA:  And enabled the I2C1 event interrupt:    After doing that, we added a DMA to UART2 as the project required:  And we also configured the timer 2 in order to set the sampling frequency to 1Hz:  And we finally went to the NVIC table to enebled the following interrupts before going to the main.c:  In the main.c, as the previous project, we defined a series of global variables:  we will explain the variables Autoincrement\_Address and datas when we reach the appropriate point.  In the main, we did exactly the same thing as the previous project:  Then we established the callback function for the timer 2:  Where every time the timer 2 reaches its maximum value thus every 1s, we acquire the datas in autoincrement form as we have put the most significant bit of the variable Autoincrement\_Address to 1 according to the manual(copied from the page 22):  The I2C embedded inside the LIS2DE behaves like a slave device and the following protocol must be adhered to. After the START condition (ST) a slave address is sent, once a slave acknowledge (SAK) has been returned, an 8-bit sub-address (SUB) is transmitted: the 7 LSB represent the actual register address while the MSB enables address auto increment. If the MSB of the SUB field is ‘1’, the SUB (register address) is automatically increased to allow multiple data read/write.  But in the HAL\_I2C\_Master\_Receive\_DMA function, we need to read 6 bytes instead of 3 bytes(we have tried, if put it to 3 bytes, didn’t work correctly as we will explain now):    (from the page 27)According to our code, the first register we will read is the OUT\_X, as we are in autoincrement form, the next register we will read is the register at the address 2A, NOT the register OUT\_Y. Also we can see from the above list, the three registers OUT\_X, OUT\_Y, OUT\_Z are’t located in consecutive position, therefore, we read 6 bytes instead of 3 bytes also the size of the variable datas is 6 instead of 3 for the same reason.    As the function description says, once the transfer is completed, the callback function is called. So in this callback function we just process the datas as the previous project and send them to our PC using the UART.  Finally, the code works as we expected: |
| Professor comments: |