Mark	

Team name:	*A1*		
Homework number:	*HOMEWORK 6*		
Due date:	23/11/2023		
Contribution	NO	Partial	Full
Pietro Monti			*x*
Alessia Moretto			*x*
Francesco Pallotto			*x*
Alessandro Perna			*x*
Ludovico Ventura			*x*
Notes:			

Project name	I2C thermometer and MEMS Accelerometer		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Explanation:
We successfully completed the homework.

Part 6a:
The aim of this part was to read both bytes of the temperature sensor LM75 (or LM75B).
To do this the first step was to modify the HAL instruction for the I2C transmission of data from the sensor to the microcontroller specifying in the 4th argument that we want to read **2** bytes:

```
if (HAL_I2C_Master_Receive(&hi2c1, LM75_ADDRESS+1, (uint8_t *)received_value_1, 2, 20) == HAL_OK)
```

We put this instruction, as requested, inside an interrupt routine related to the timer 2 peripheral, after activating in the GUI the timer 2 global interrupt and setting the prescaler to 8400-1 and the period to 10000-1, in order to execute the code every 1 second:

```
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) {
```

After the read of the first two bytes, then stored in an array of 8 bits signed int called “received_value_1” of 2 elements, we computed the temperature value with the following expression:

```
temperature_1 = received_value_1[0] + (float)((uint8_t)(received_value_1[1])) / 256;
```

We made the sum of the integer part, stored in “received_value_1[0]” (the first byte acquired) and the value inside “received_value_1[1]”, with a first cast to unsigned, since the fractional part of a number has no sign, and a second to float, to allow the precise division by 256, necessary to give this data the correct weight (LSB). In this way we were able to correctly read both the negative and the positive values of temperature measured by the sensor.

Then we noticed, as suggested at lecture, that, reading data from a LM75B sensor, there was a bug: it sometimes happened that the displayed fractional part was not coherent with the integer one in the considered flow of data. After discussing the problem, we understood that it was due to the ADC, which, even during the read of the data, doesn’t stop the conversion. So it might happen that the measured data change during the time in which we are reading and transmitting the data through the I2C bus.

A possible solution to this problem could be to compare a pair of data sequentially measured and read: if they are equal, there's no problem, so we can take the actual value;

```
// 2nd read
if (HAL_I2C_Master_Receive(&hi2c1, LM75_ADDRESS+1, (uint8_t *)received_value_2, 2, 20) == HAL_OK) {

    // bug check
    if (received_value_1[0] == received_value_2[0] && received_value_1[1] == received_value_2[1]) {
        // no bug
        temperature_1 = received_value_1[0] + (float)((uint8_t)(received_value_1[1])) / 256;
```

otherwise, it's necessary to take into account a third data, which is for sure coherent, since the ADC takes a time to convert sufficiently long not to compromise also this last value.

```
else {
  // bug, 3rd read
  if (HAL_I2C_Master_Receive(&hi2c1, LM75_ADDRESS+1, (uint8_t *)received_value_3, 2, 20) == HAL_OK) {
      temperature_3 = received_value_3[0] + (float)((uint8_t)(received_value_3[1])) / 256;
```

Note_1

We made the comparison directly between, respectively, the first and the second bytes of the two acquisitions, in order not to waste time and power in the computation of a possible useless value of temperature.

Note_2

We used "nested reads" with the aim of saving the third one in case of no bug. Another possible solution could be to read 6 bytes all together with a single instruction.

Part 6b:

As seen at lecture, we configured the PB8 and PB9 pins as SCL and SDA, respectively. In addition, we set the USART2 global interrupt to be able to exploit DMA, so we enabled it and we configured the timer TIM2 to generate an interrupt every second.

As done in the initial part of the accelerometer project, we initialize two variables, ACCEL_ADD and LIS2DE12_ADD, with the LIS2DE and the LIS2DE12 accelerometers slave addresses, respectively. After that we store the control register in dedicated arrays enabling the 1Hz mode for x, y, z. CTRL_REG2 and CTRL_REG4 are both initialized at their default values, the latter to make the sensor operate in normal mode and access linear acceleration data.

With the following callback we read the accelerometer data:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if (htim==&htim2){
        //Get X
        int8_t acc_x=0;
        HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, &OUT_X_ADD, 1, 50);//I tell the variable i want to rea
        HAL_I2C_Master_Receive(&hi2c1, ACCEL_ADD+1, (uint8_t *)&acc_x, 1, 50);

        //Get Y
        int8_t acc_y=0;
        HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, &OUT_Y_ADD, 1, 50);//I tell the variable i want to rea
        HAL_I2C_Master_Receive(&hi2c1, ACCEL_ADD+1, (uint8_t *)&acc_y, 1, 50);

        //Get Z
        int8_t acc_z=0;
        HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, &OUT_Z_ADD, 1, 50);//I tell the variable i want to rea
        HAL_I2C_Master_Receive(&hi2c1, ACCEL_ADD+1, (uint8_t *)&acc_z, 1, 50);

        //the resolution is 4/256
        float acc_g_x=acc_x/64.0;
        float acc_g_y=acc_y/64.0;
        float acc_g_z=acc_z/64.0;

        len=snprintf(str,sizeof(str),"X:%+.2f g\r\nY:%+.2f g\r\nZ:%+.2f g\r\n",acc_g_x,acc_g_y,acc_g_z);

        HAL_UART_Transmit_DMA(&huart2, (uint8_t *) str, len);
    }
}
```

For each of the 3 axes, with the first HAL_I2C transmit instruction we inform the sensor that we want to read the desired coordinates, which are respectively situated in address 0x29 (x axis), 0x2B (y axis) and 0x2D (z axis).
Then, with the second HAL_I2C receive instruction, we receive the desired data and store them in dedicated variables.
These are so converted in float values and divided by 64, to be in the –2 and 2 range, being the resolution 4/256.
In the end we format the string “str” including the accelerometer data with their signs and send it to the remote terminal using the USART through DMA.

We need to understand which sensor is soldered on the board. Firstly we check if there is a device with the LIS2DE address: if there isn’t we change the address to the one of the LIS2DE12 and try again; otherwise we send an error message.

```c
if(HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, CTRL_REG1, sizeof(CTRL_REG1), 50)==HAL_OK){
    len=snprintf(str,sizeof(str),"LIS2DE FOUND\r\n");
    //there is a device with LIS2DE address
}
else{
    //if not we change the address to the one of LIS2DE12 and try again
    ACCEL_ADD=LIS2DE12_ADD;
    if(HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, CTRL_REG1, sizeof(CTRL_REG1), 50)==HAL_OK){
        len=snprintf(str,sizeof(str),"LIS2DE12 FOUND\r\n");
        //there is a device with LIS2DE12 address
    }
    else{
        len=snprintf(str,sizeof(str),"Accelerometer ERROR found\r\n");
        //nothing found
    }
}
```

Note

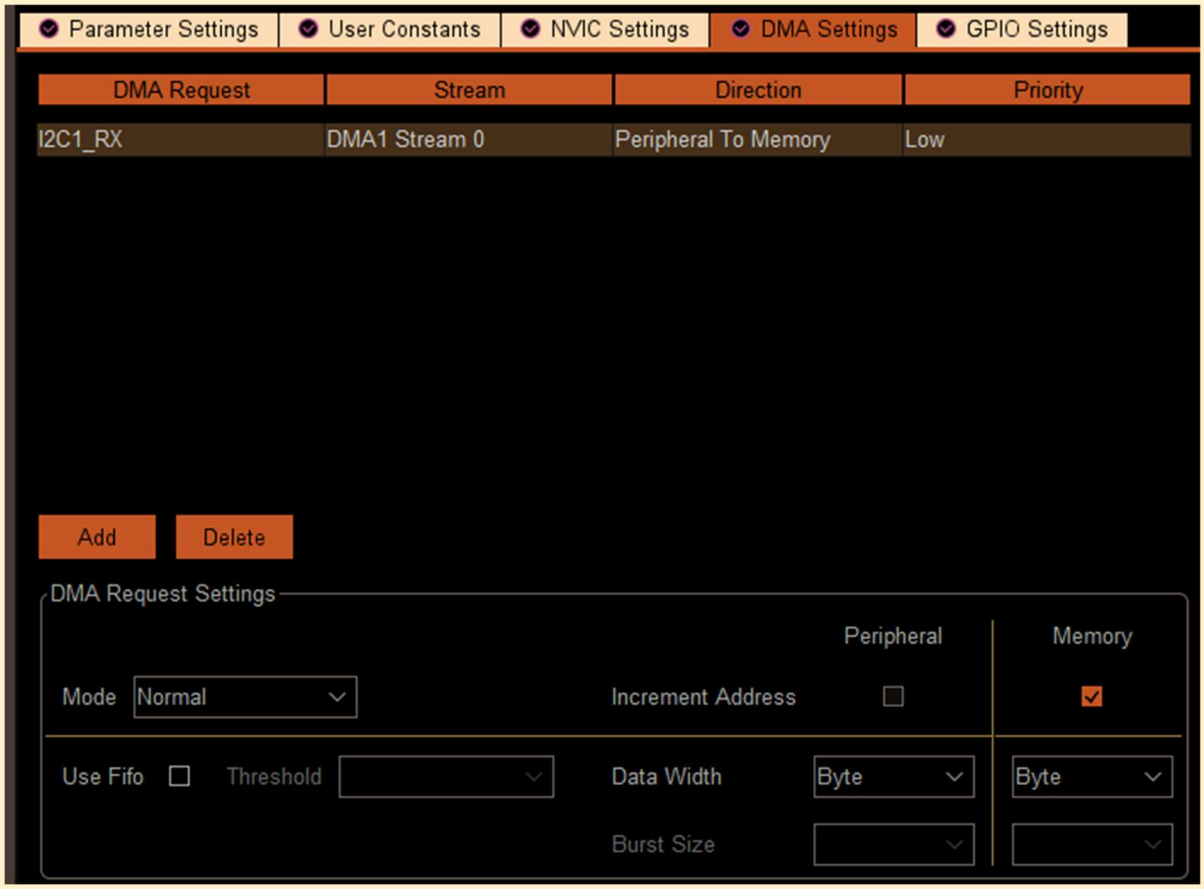
Watching the recording of the lecture, on Saturday we noticed that the professor said to use the multiple read protocol. However, it is not written as a requirement or hint in the homework slides. We asked him for clarifications, and he told us it is not mandatory. So we didn't use that mode.

Part 6c:

With this final project, the objective is to achieve substantially the same goals as Project 6b, with the exception of incorporating I2C DMA.

1. Setting up I2C DMA & Auto-Increment:

- Establish proper configuration for the I2C peripheral, ensuring that Direct Memory Access (DMA) is appropriately configured for efficient data transfer. This involves configuring both the I2C peripheral and the DMA controller, specifying the DMA settings to handle data from the I2C peripheral as follows:



- For the I2C we exploit the auto-increment feature of the sub-address by setting the MSB of the initial address to 1:

```
1  uint8_t INITIAL_ADDR = 0x29 | 0x80; // address of OUT_X, auto-increment enabled
```

2. Modifying TIM Callback:

- Adjust the Timer (TIM) callback to align with the DMA-based I2C data transfer. Confirm the correct configuration of the Timer, ensuring synchronization with the I2C DMA operation.
- acc_data is the int8_t array of dimension 5 where we store all the x,y,z data.

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  if (htim==&htim2){

    // Specify where to start reading
    HAL_I2C_Master_Transmit(&hi2c1, ACC_ADDRESS, &INITIAL_ADDR, 1, timeout_ms);
    HAL_I2C_Master_Receive_DMA(&hi2c1, ACC_ADDRESS+1, (uint8_t *)&acc_data, sizeof(acc_data));
  }
}
```

3. Handling I2C Callback & UART Data Transmission:

- Implement the **HAL_I2C_MasterRxCpltCallback** callback function, triggered upon completion of the I2C DMA transfer.
- Within this callback:
  - Perform the conversion of received data to float values. This involves interpreting the raw data stored in the **acc_data** array.
  - Store the converted x, y, and z values in designated float variables such as **acc_g_x**, **acc_g_y**, and **acc_g_z**.
- Assemble a string containing the converted float values.
- Implement the UART DMA transmission protocol to send the constructed string to the remote endpoint.

```c
void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c) {
  if (hi2c==&hi2c1){

    acc_g_x = acc_data[0] / 64.0;
    acc_g_y = acc_data[2] / 64.0;
    acc_g_z = acc_data[4] / 64.0;

    length = snprintf(string, sizeof(string), "Acceleration [g]:\r\n\n X = %+.2f g\r\n Y = %+.2f g\r\n Z = %+.2f g\r\n\n\n", acc_g_x, acc_g_y, acc_g_z);
    // the + sign ensures the sign is always printed

    HAL_UART_Transmit_DMA(&huart2, (uint8_t *)string, length);
  }
}
```

Professor comments: