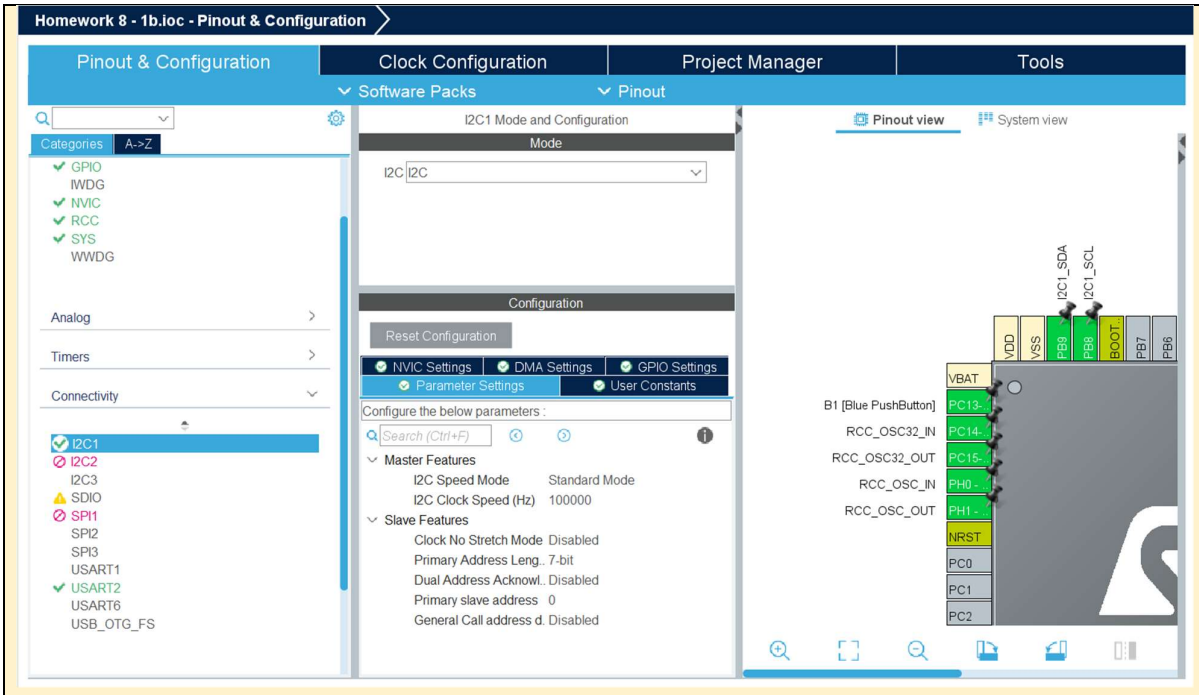
Mark	/11

Team name:	*A14*		
Homework number:	*HOMEWORK 08*		
Due date:	09/11/25		
Contribution	NO	Partial	Full
Mattia Di Mauro			*x*
Francesca Biondi			*x*
Lorenzo Castelli			*x*
Carmen Maria Niro			*x*
Pietro Albrigi			*x*
Notes: none			

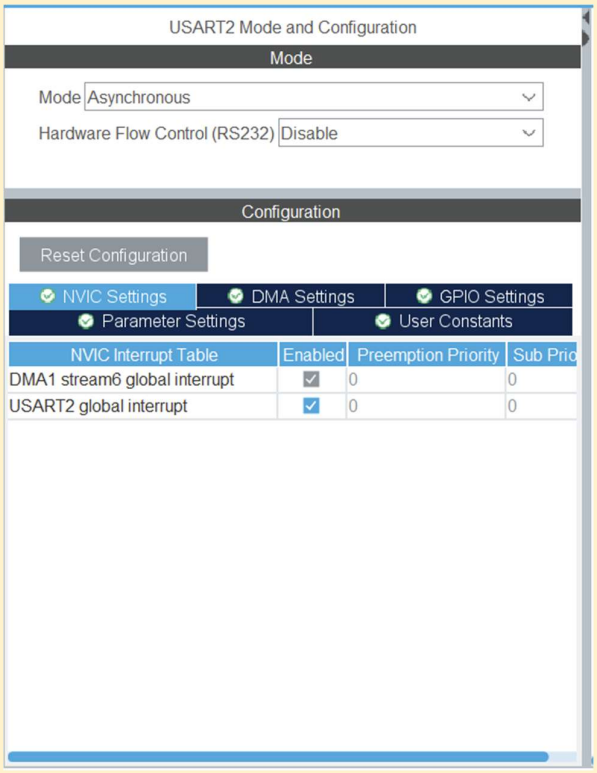
Project name	HOMEWORK 8		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Project 1b:

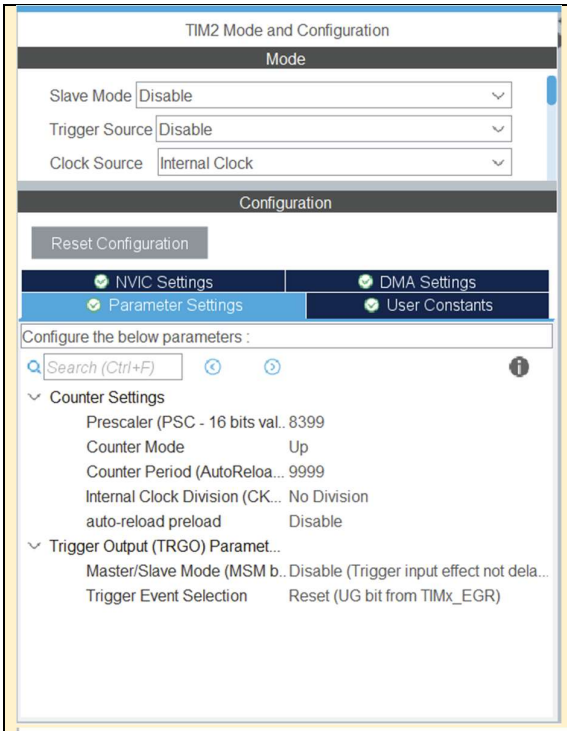
To acquire data from the accelerometer, the I2C1 was enabled in the Connectivity section of the .ioc configuration. Pins PB8 and PB9 were configured as SCL and SDA, respectively. The I2C interface was set to Standard Mode with 7-bit addressing:

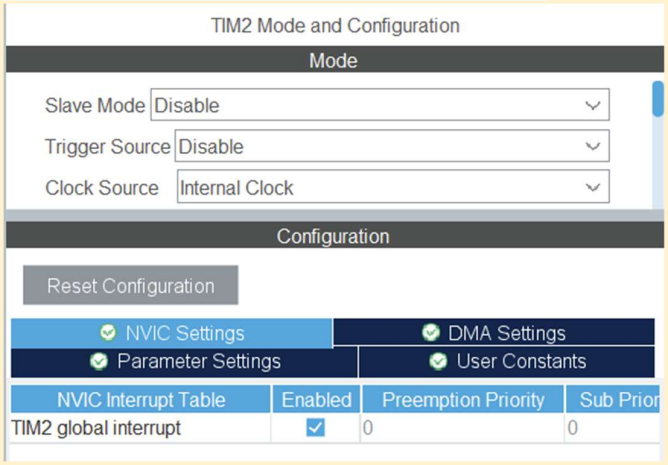


To transmit the acquired data to a remote terminal, the USART2 was configured in the .ioc file. A DMA request was added to enable efficient data transfer, and both the DMA and USART2 global interrupts were enabled:



To ensure that data acquisition and transmission occur once per second, TIM2 was configured to generate an interrupt every one second:





In the private variables section of the code, several registers and configuration parameters for the accelerometer were defined. Specifically, the control registers CTRL_REG1, CTRL_REG2, and CTRL_REG4 were initialized to configure the device in normal mode operation with a 1 Hz update rate, no high-pass filtering, and a ±2 g Full Scale range.

The output register addresses for the X, Y, and Z axes were also defined, along with the possible I²C addresses of the two possible accelerometer models (LIS2DE and LIS2DW):

```
/* USER CODE BEGIN PV */
//reg address, 1Hz + normal mode + XYZ enabled
uint8_t CTRL_REG1[] = {0x20, 0b00010111};
//reg address, no HPF (default value at startup)
uint8_t CTRL_REG2[] = {0x21, 0b00000000};
//reg address, continuous update + 2g FSR + self test disabled (default)
uint8_t CTRL_REG4[] = {0x23, 0b00000000};

uint8_t OUT_X_ADD = 0x29;
uint8_t OUT_Y_ADD = 0x2B;
uint8_t OUT_Z_ADD = 0x2D;
uint8_t ACCEL_ADD;
uint8_t LIS2DE_ADDRESS = 0b01010000; // 0x28 << 1 = 0x50
uint8_t LIS2DW_ADDRESS = 0b00110000; // 0x18 << 1 = 0x30
```

In the main() function, the accelerometer is configured and initialized as seen in class. The return value of the HAL_I2C_Master_Transmit() function is used to determine which accelerometer model is connected (LIS2DE or LIS2DW). This identification result is then transmitted to the terminal via UART using DMA. Finally, the TIM2 timer is started to generate interrupts every 1 second:

```
148   /* USER CODE BEGIN 2 */
149   //Initialize accelerometer
150   ACCEL_ADD = LIS2DE_ADDRESS;
151   if(HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, CTRL_REG1, sizeof(CTRL_REG1), 100)== HAL_OK)
152   {
153       len = snprintf(str, sizeof(str), "LIS2DE found\r\n");
154   }else{
155       ACCEL_ADD = LIS2DW_ADDRESS;
156       if(HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, CTRL_REG1, sizeof(CTRL_REG1), 100) == HAL_OK)
157       {
158           len = snprintf(str, sizeof(str), "LIS2DW found\r\n");
159       }else{
160           len = snprintf(str, sizeof(str), "Accelerometer error\r\n");
161       }
162    }
163    HAL_UART_Transmit_DMA(&huart2, str, len);
164
165    // Not strictly needed (already in default config)
166    HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, CTRL_REG2, sizeof(CTRL_REG2), 100);
167    HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, CTRL_REG4, sizeof(CTRL_REG4), 100);
168    HAL_TIM_Base_Start_IT(&htim2);
169   /* USER CODE END 2 */
```

The timer interrupt callback function is responsible for acquiring acceleration data from all three axes (X, Y, and Z) of the sensor. The raw data are read via I2C, converted into acceleration values expressed in g, and then transmitted to the terminal using UART with DMA:

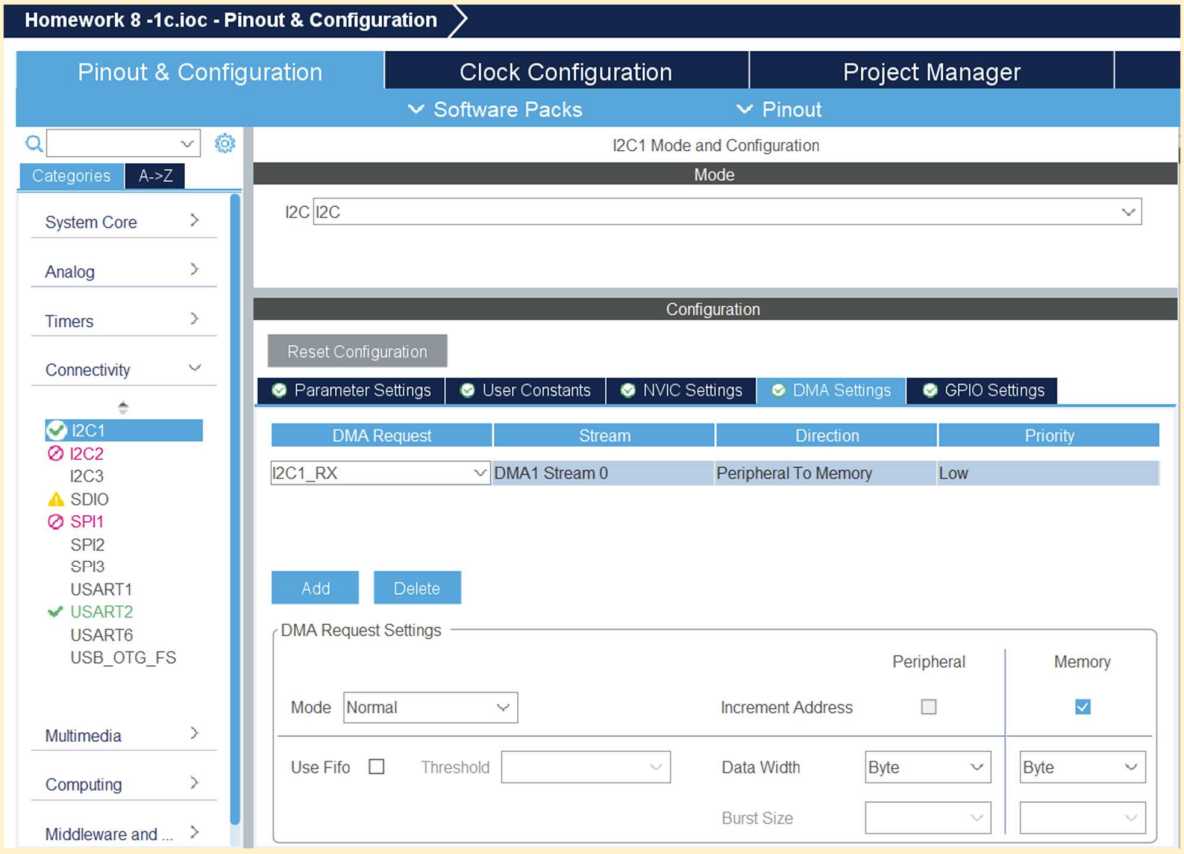
```
 82 /* USER CODE BEGIN 0 */
 83
 84 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
 85 {
 86     if(htim == &htim2)
 87     {
 88         // Get X
 89         int8_t acc_x = 0;
 90         HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, &OUT_X_ADD, 1, 50);
 91         HAL_I2C_Master_Receive(&hi2c1, ACCEL_ADD+1, &acc_x, 1, 50);
 92
 93         //Get Y
 94         int8_t acc_y = 0;
 95         HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, &OUT_Y_ADD, 1, 50);
 96         HAL_I2C_Master_Receive(&hi2c1, ACCEL_ADD+1, &acc_y, 1, 50);
 97
 98         //Get Z
 99         int8_t acc_z = 0;
100         HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, &OUT_Z_ADD, 1, 50);
101         HAL_I2C_Master_Receive(&hi2c1, ACCEL_ADD+1, &acc_z, 1, 50);
102
103         float acc_g_x = acc_x / 64.0;
104         float acc_g_y = acc_y / 64.0;
105         float acc_g_z = acc_z / 64.0;
106
107         len = snprintf(str, sizeof(str), "X: %+.2f g\r\nY: %+.2f g\r\nZ: %+.2f g\r\n\r\n", acc_g_x, acc_g_y, acc_g_z);
108         HAL_UART_Transmit_DMA(&huart2, str, len);
109     }
110 }
111
112 /* USER CODE END 0 */
```

The code was then tested on the board and confirmed to work as intended. Using a serial terminal configured with the same baud rate and the corresponding COM port, the accelerometer data were successfully displayed:

```
LIS2DE found
X: -0.11 g
Y: +0.03 g
Z: +0.98 g

X: -0.14 g
Y: +0.03 g
Z: +0.72 g

X: -0.44 g
Y: -0.02 g
Z: +0.98 g

X: -0.44 g
Y: -0.02 g
Z: +0.98 g

X: -1.00 g
Y: +0.11 g
Z: +0.23 g

X: -0.95 g
Y: +0.22 g
Z: -0.38 g

X: -0.75 g
Y: +0.20 g
Z: -0.67 g

X: -0.84 g
Y: +0.00 g
Z: +0.62 g
```

Project 1c:

To enable data transfers via I2C using DMA, the previous project was modified to include an I2C1_RX DMA request with memory address incrementation enabled:



In addition, the I2C interrupts were enabled:

Parameter Settings | User Constants | NVIC Settings | DMA Settings | GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA1 stream0 global interrupt	☑	0	0
I2C1 event interrupt	☑	0	0
I2C1 error interrupt	☑	0	0

In the TIM2 callback (triggered every second), acceleration data is acquired from the sensor using the auto-increment feature of the sub-address:

```
86 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
87 {
88     if (htim == &htim2)
89     {
90         uint8_t SUB = 0x29 | 0x80; //0x29 = OUT_X_ADD, 0x80 sets the MSB to 1
91         HAL_I2C_Master_Transmit(&hi2c1, ACCEL_ADD, &SUB, 1, 50);
92         HAL_I2C_Master_Receive_DMA(&hi2c1, ACCEL_ADD+1, values, 5);
93     }
94 }
```

This feature allows consecutive registers to be read automatically after each byte transfer. According to the datasheet, setting the MSB of the sub-address to 1 enables this behavior.

Using I2C with DMA, a single read operation retrieves data from the OUT_X, OUT_Y, and OUT_Z registers. The results are stored in the values[5] array, since five bytes must be transferred (one for each axis and two reserved bytes between them, as indicated in the register map on page 27 of the LIS2DE datasheet).

Within the HAL_I2C_MasterRxCpltCallback function, the acquired acceleration data is converted to values expressed in g. The formatted data string is then transmitted via UART using DMA:

```
96 void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c){
97     if (hi2c == &hi2c1){
98         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0); // or LED blink
99         char str[74];
100        int len = 0;
101
102        float acc_g_x = values[0] / 64.0;
103        float acc_g_y = values[2] / 64.0;
104        float acc_g_z = values[4] / 64.0;
105
106        len = snprintf(str, sizeof(str), "X: %+.2f g\r\nY: %+.2f g\r\nZ: %+.2f g\r\n\r\n",acc_g_x, acc_g_y, acc_g_z);
107        HAL_UART_Transmit_DMA(&huart2, str, len);
108    }
109 }
```

The code was then tested on the board and confirmed to work as intended.

Professor comments: