Mark	**/11**

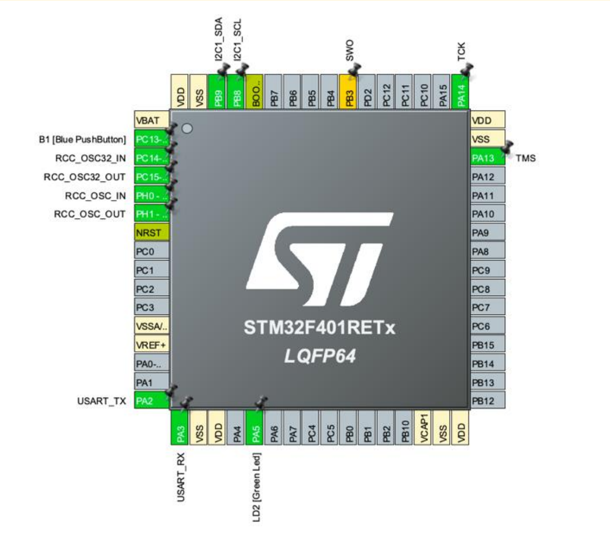
Team name:	*A1*		
Homework number:	*HOMEWORK 08*		
Due date:	19/11/24		
Contribution	NO	Partial	Full
Piombo			*x*
Fumagalli			*x*
Pierfederici			*x*
Zenoni			*x*
Ferraro			*x*
Notes:			

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

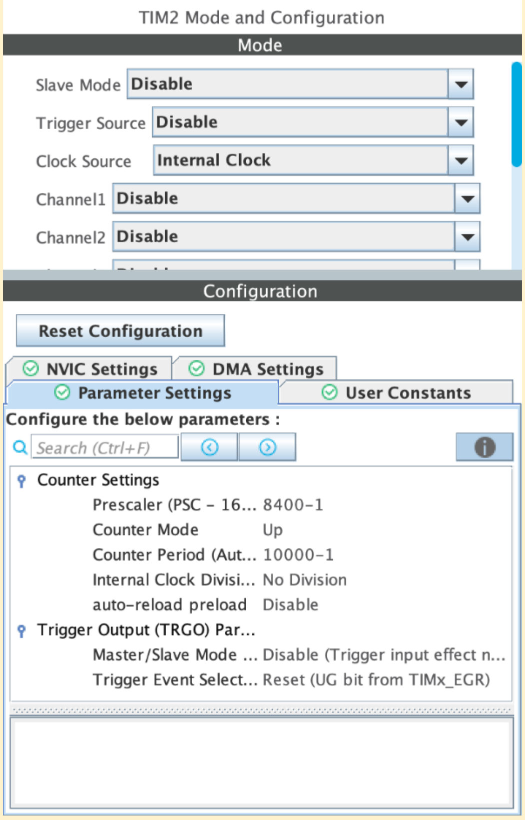
Part 1:

Starting from the “.ioc” we enabled the I2C1 and we set the “I2C Clock Speed” to 100KHz.

Then, we configured our pin of interest (PB8 - SCL, PB9 - SDA) to communicate with the I2C to the accelerometer sensor and we checked if the pins we exploit to communicate with UART protocol are correctly set.



We used the timer 2 in interrupt mode to read data from the sensor each second.



TIM2 Mode and Configuration

Mode

Slave Mode: Disable
Trigger Source: Disable
Clock Source: Internal Clock
Channel1: Disable
Channel2: Disable

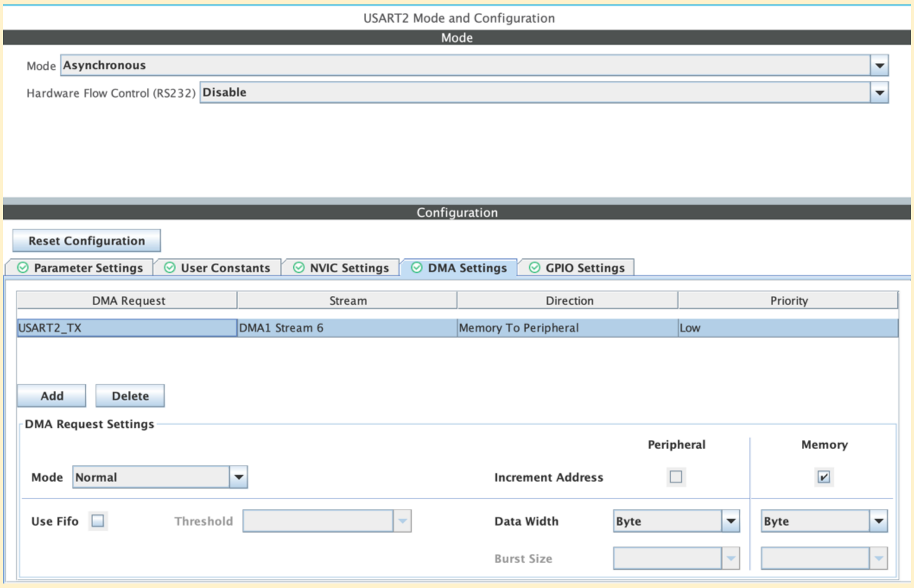
Configuration

Configure the below parameters :

Counter Settings
- Prescaler (PSC - 16... 8400-1
- Counter Mode Up
- Counter Period (Aut... 10000-1
- Internal Clock Divisi... No Division
- auto-reload preload Disable

Trigger Output (TRGO) Par...
- Master/Slave Mode ... Disable (Trigger input effect n...
- Trigger Event Select... Reset (UG bit from TIMx_EGR)

We configured the UART (enabling also its interrupt) in DMA mode to send the coordinates to the pc.



USART2 Mode and Configuration

Mode

Mode: Asynchronous
Hardware Flow Control (RS232): Disable

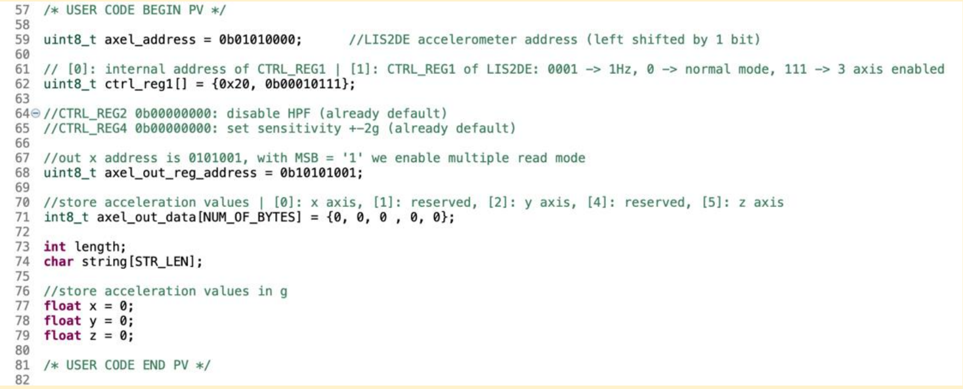
Configuration

DMA Request	Stream	Direction	Priority
USART2_TX	DMA1 Stream 6	Memory To Peripheral	Low

DMA Request Settings

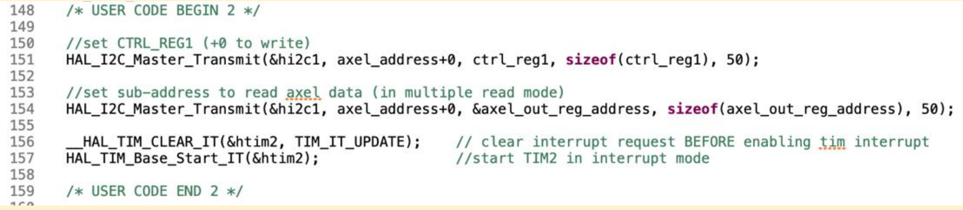
		Peripheral	Memory
Mode: Normal	Increment Address	□	☑
Use Fifo □ Threshold	Data Width	Byte	Byte
	Burst Size		

In the “main.c” we defined the following variables:



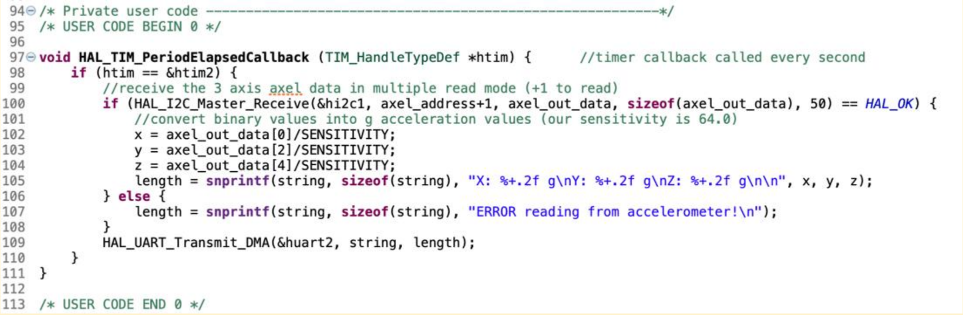
```c
/* USER CODE BEGIN PV */

uint8_t axel_address = 0b01010000;      //LIS2DE accelerometer address (left shifted by 1 bit)

// [0]: internal address of CTRL_REG1 | [1]: CTRL_REG1 of LIS2DE: 0001 -> 1Hz, 0 -> normal mode, 111 -> 3 axis enabled
uint8_t ctrl_reg1[] = {0x20, 0b00010111};

//CTRL_REG2 0b00000000: disable HPF (already default)
//CTRL_REG4 0b00000000: set sensitivity +-2g (already default)

//out x address is 0101001, with MSB = '1' we enable multiple read mode
uint8_t axel_out_reg_address = 0b10101001;

//store acceleration values | [0]: x axis, [1]: reserved, [2]: y axis, [4]: reserved, [5]: z axis
int8_t axel_out_data[NUM_OF_BYTES] = {0, 0, 0 , 0, 0};

int length;
char string[STR_LEN];

//store acceleration values in g
float x = 0;
float y = 0;
float z = 0;

/* USER CODE END PV */

```

In the main function we set the CTRL_REG1 of the sensor (as you can see in the comment). Then we set the sub-address in order to read in multiple-read mode starting from the OUT_X REG. Notice that multiple-read mode exploits the auto-increment of the sub-address, thus we have to read 5 bytes according to page 27 of the datasheet. Finally we started TIM2 in interrupt mode.



```c
    /* USER CODE BEGIN 2 */

    //set CTRL_REG1 (+0 to write)
    HAL_I2C_Master_Transmit(&hi2c1, axel_address+0, ctrl_reg1, sizeof(ctrl_reg1), 50);

    //set sub-address to read axel data (in multiple read mode)
    HAL_I2C_Master_Transmit(&hi2c1, axel_address+0, &axel_out_reg_address, sizeof(axel_out_reg_address), 50);

    __HAL_TIM_CLEAR_IT(&htim2, TIM_IT_UPDATE);    // clear interrupt request BEFORE enabling tim interrupt
    HAL_TIM_Base_Start_IT(&htim2);                //start TIM2 in interrupt mode

    /* USER CODE END 2 */
```

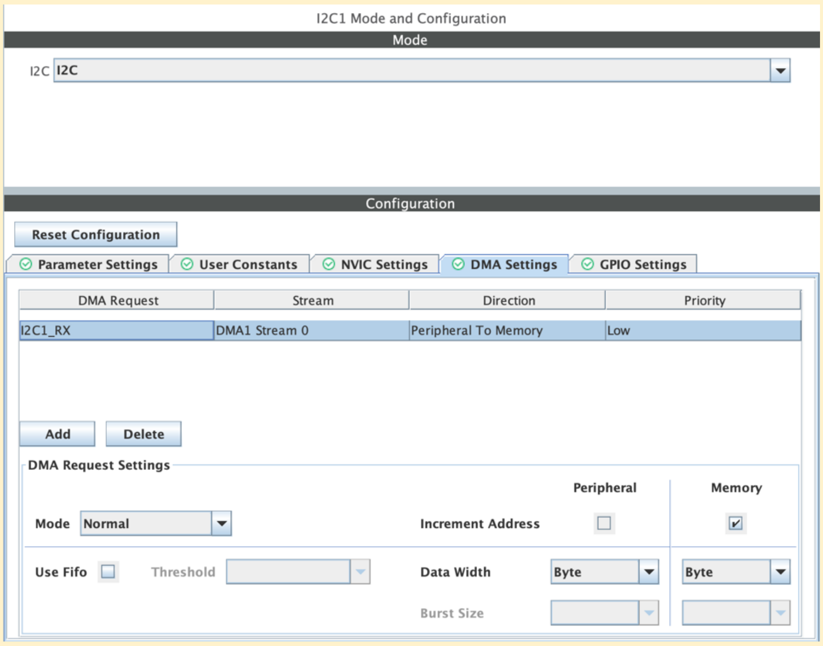
In the timer callback we actually receive the coordinates, then we convert them into g (gravitational acceleration unit). If the reception is successful, we send the string through UART with DMA to the terminal. The SENSITIVITY has been derived by reading the output z data of the accelerometer as integer while the board is placed on the table (x = 0; y = 0; z = 64 = 1g).



```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) {      //timer callback called every second
    if (htim == &htim2) {
        //receive the 3 axis axel data in multiple read mode (+1 to read)
        if (HAL_I2C_Master_Receive(&hi2c1, axel_address+1, axel_out_data, sizeof(axel_out_data), 50) == HAL_OK) {
            //convert binary values into g acceleration values (our sensitivity is 64.0)
            x = axel_out_data[0]/SENSITIVITY;
            y = axel_out_data[2]/SENSITIVITY;
            z = axel_out_data[4]/SENSITIVITY;
            length = snprintf(string, sizeof(string), "X: %+.2f g\nY: %+.2f g\nZ: %+.2f g\n\n", x, y, z);
        } else {
            length = snprintf(string, sizeof(string), "ERROR reading from accelerometer!\n");
        }
        HAL_UART_Transmit_DMA(&huart2, string, length);
    }
}

/* USER CODE END 0 */
```

Part 2:

Starting from the previous project we only needed to configure the DMA for the I2C reception



I2C: I2C

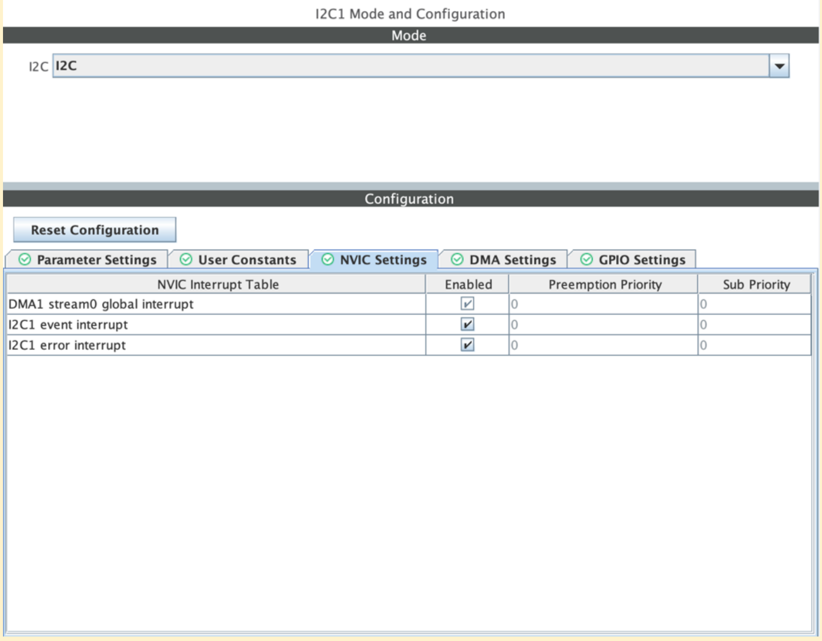
DMA Request	Stream	Direction	Priority
I2C1_RX	DMA1 Stream 0	Peripheral To Memory	Low

Mode: Normal

	Peripheral	Memory
Increment Address	□	☑
Data Width	Byte	Byte
Burst Size		

Use Fifo □ Threshold

We also needed to enable the interrupt for the DMA reception

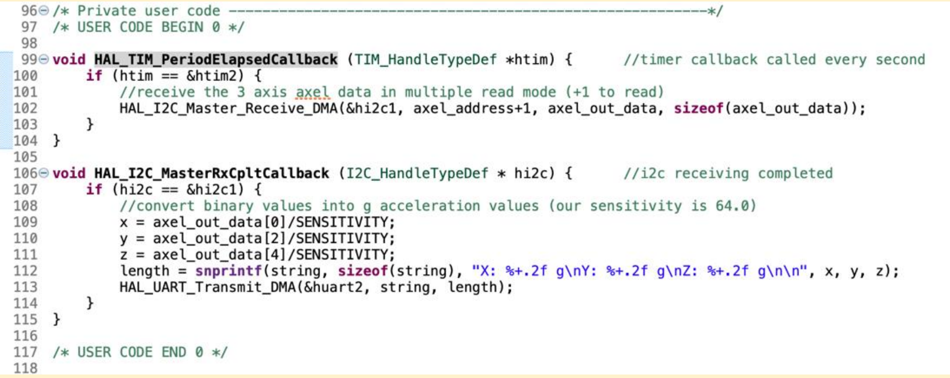


I2C: I2C

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

The main function and the variables are the same as the previous project.

The changes regard only callbacks. Every second, the timer enter in its interrupt routine and data reception starts. Once the reading is complete, we enter in the I2C callback where we process and send our data.



```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) {      //timer callback called every second
    if (htim == &htim2) {
        //receive the 3 axis axel data in multiple read mode (+1 to read)
        HAL_I2C_Master_Receive_DMA(&hi2c1, axel_address+1, axel_out_data, sizeof(axel_out_data));
    }
}

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c) {      //i2c receiving completed
    if (hi2c == &hi2c1) {
        //convert binary values into g acceleration values (our sensitivity is 64.0)
        x = axel_out_data[0]/SENSITIVITY;
        y = axel_out_data[2]/SENSITIVITY;
        z = axel_out_data[4]/SENSITIVITY;
        length = snprintf(string, sizeof(string), "X: %+.2f g\nY: %+.2f g\nZ: %+.2f g\n\n", x, y, z);
        HAL_UART_Transmit_DMA(&huart2, string, length);
    }
}

/* USER CODE END 0 */

```

P.S.: About the transmission of the sub-address:

We can correctly collect data from the accelerometer by sending the configuration and the sub-address only at the startup, so we did not need to perform the I2C transmission to the sub-address for every reading.

Thus, we decided to do not configure the DMA for I2C transmission because it would not improve considerably the performance.

Every time a new reception starts, the accelerometer automatically begins reading from the x axis register, in multiple reading mode.