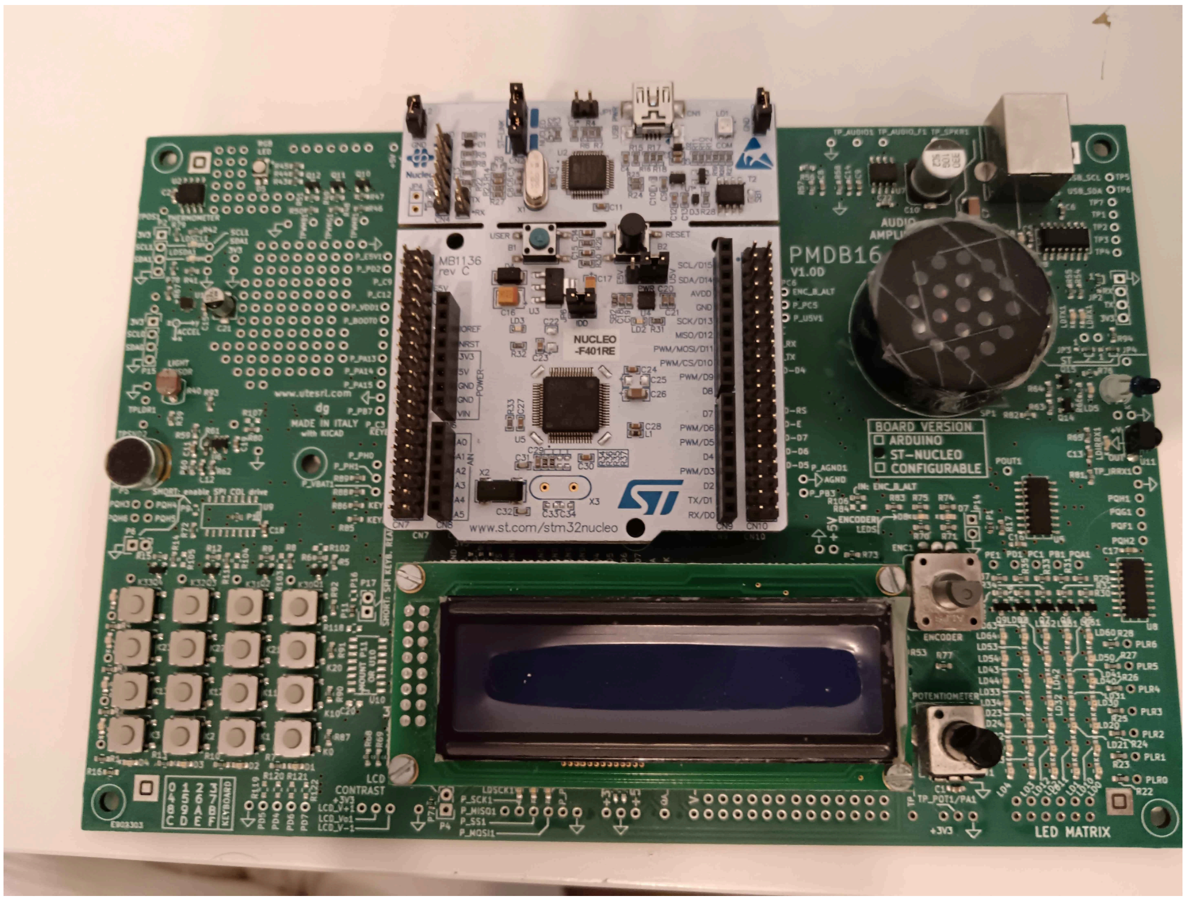
Computer Science Team B4

Sensor system

Homework Report



HOMEWORK NUMBER 8

2024 - 2025

POLITECNICO
MILANO 1863

Summary

I. Members work

Members	Exercise 1	General
Cazin Némo	✓	Reviewed report
Adrien Paliferro	✓	Reviewed report
Heikki Leveelahti	✓	Reviewed report
Osmo Kieksi	✓	Reviewed report
Constantijn Coppers	✓	Wrote the report

II. Exercise 1

General functionality:
The aim of this project is to sequentially read the x, y, and z accelerometer data every second and transmit these values to the computer via UART. To achieve this, a timer is configured to trigger the DMA controller every second, which sets the accelerometer registers for sequential data reading. Subsequently, a callback function initiates the reading process using DMA. Once the data is read, another callback formats and transmits the data to the computer via UART. In this project, we made intensive usage of the DMA controller for I2C and UART communication.

Implementation:
To obtain the desired functionality, the following steps are followed:

1. **PIN lookup**

The I2C protocol requires two lines (SDA and SCL), hence two PINs must be configured:

PIN	**I/O**
PB9	SDA
PB10	SCL

2. **Board configuration**

I2C configuration
In the ***connectivity > I2C1*** tab, we enable the I2C communication. Furthermore, we set up the DMA RX and TX in the ***DMA Settings*** tab and we also enable the IC2 event and error interrupt, under the ***NVIC Settings*** tab.

UART and TIM configuration
Finally, we configure UART transmission via DMA and the timer (1 second frequency), as demonstrated in previous projects.

3. Code implementation

First, we include some header files (LIS2DE contains all the required addresses for the accelerometer) and we define some private variables (variables to store the correct accelerometer addresses, strings for UART transmission and a buffer to store the accelerometer data).

```
. . .

/* USER CODE BEGIN Includes */
#include "stdio.h"
#include "string.h"
#include "LIS2DE.h"
/* USER CODE END Includes */

. . .

/* USER CODE BEGIN PV */
// accelerometer read/write addresses
uint8_t LIS2DE_AccAddr_READ;
uint8_t LIS2DE_AccAddr_WRITE;

// characters
char str1[32];
char str2[32];
char message[64];  // the message is send 1 time since the DMA needs time to transmit it

// ACC buffer
uint8_t accBuffer[6];
/* USER CODE END PV */

. . .
```

As explained above, we first make a timer callback that triggers the DMA to set the control registers of the accelerometer (by I2C) such that we are able to read the x, y and z data sequentially.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){

        if (htim->Instance == htim1.Instance) {

                // sequentially read data (1 byte)
                uint8_t ctrData = LIS2DE_OUT_X_H | 0x80;

                // set the registers in the ACC
                if (HAL_I2C_Master_Transmit_DMA(&hi2c1, LIS2DE_AccAddr_WRITE, &ctrData, 1) != HAL_OK) {
                        Error_Handler();
                }
        }
}
```

When the MCU receives an ACK from the accelerometer (by I2C protocol), the HAL_I2C_MasterTxCpltCallback will be triggered. In this callback, we simply receive the x,y and z accelerometer data and store this in the predefined buffer.

```
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c) {

        if (HAL_I2C_Master_Receive_DMA(&hi2c1, LIS2DE_AccAddr_READ, (uint8_t *) accBuffer, 6) != HAL_OK) {
                Error_Handler();
        }
}
```

Afterwards, when the data is received, the HAL_I2C_MasterRxCpltCallback will be called. In this callback we get the x, y and z values that were stored in the buffer, we format those values and finally, we transmit it to the computer by the UART via the DMA controller.

```
void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c) {

        // get values from buffer
        float x = accBuffer[0]/64.0;
        float y = accBuffer[2]/64.0;
        float z = accBuffer[4]/64.0;

        // format result
        char str3[32];
        snprintf(str3, sizeof(str3), "x: %.2f\r\ty: %.2f\r\tz: %.2f\r\n", x, y, z);

        // transmit to remote terminal
        HAL_UART_Transmit_DMA(&huart2, (uint8_t *)str3, strlen(str3));
}
```

Lastly, the remaining task is to write the code in the **main**-function. We must do three things (1) detect and set the correct address of the accelerometer (LISDE or LISDE12), (2) we must configure the correct accelerometer settings by setting the internal registers of the accelerometer and (3) we must start the timer in interrupt mode.

First, the correct accelerometer address is set by *trial and error*. We both try reading from the LISDE and LISDE12 addresses. However, HAL_I2C_Master_Transmit will only return HAL_OK for the valid address. If no for both addresses, HAL_I2C_Master_Transmit, doesn't return HAL_OK, something went wrong and no address will be selected.

```
int main(void)
{

    . . .

    /* automatically select the correct ADR for the ACC */
    if (HAL_I2C_Master_Transmit(&hi2c1, LIS2DE_ADDR_READ, 0, 1, 100) == HAL_OK)
    {
          // format the message
          snprintf(str1, sizeof(str1), "LISDE ADR SELECTED\n");
          // set ADR
          LIS2DE_AccAddr_READ  = LIS2DE_ADDR_READ;
          LIS2DE_AccAddr_WRITE = LIS2DE_ADDR_WRITE;
    }
    else if (HAL_I2C_Master_Transmit(&hi2c1, LIS2DE12_ADDR_READ, 0, 1, 100) == HAL_OK)
    {
          // format the message
          snprintf(str1, sizeof(str1), "LISDE12 ADR SELECTED\n");
          // set ADR
          LIS2DE_AccAddr_READ  = LIS2DE12_ADDR_READ;
          LIS2DE_AccAddr_WRITE = LIS2DE12_ADDR_WRITE;
    }
    else {
                // format the message
                snprintf(str1, sizeof(str1), "NO ADR SELECTED\n");
                // Handle error
                Error_Handler();
    }

    . . .

}
```

In order to select the correct defaults for the accelerometer we must only write to control register 1 (1 Hz + Normal mode + x,y,z active → 0b00011111), since the other required specifications (in the assignment) are ok by default. Additionally, we send an update to the computer about the accelerometer address selection and the configuration of the accelerometer defaults.

```
int main(void)
{

   . . .

   uint8_t CTRL_REG1_MODE[2];
   CTRL_REG1_MODE[1] = LIS2DE_CTRL_REG1; // write to CTRL REG 1
   CTRL_REG1_MODE[2] = 0b00011111;       // Set the options
   if (HAL_I2C_Master_Transmit(&hi2c1, LIS2DE_AccAddr_WRITE,  CTRL_REG1_MODE, 2, 100) == HAL_OK) {

          // format message
          snprintf(str2, sizeof(str2), "OPTIONS SET\n");

   } else {

          // format message
          snprintf(str2, sizeof(str2), "ERROR\n");
   }

   // format message from ADR selection + options configuration ACC
   // they are combined since DMA can’t handle them sequentially
  strcpy(message, str1);
  strcat(message, str2);

  // Send an update
  HAL_UART_Transmit_DMA(&huart2, (uint8_t *)message, strlen(message));

  . . .

}
```

Finally, we set start the timer in interrupt mode

```
int main(void)
{
    . . .

    HAL_TIM_Base_Start_IT(&htim1);

    . . .
}
```

Now we are ready to run the code!

Note that ‘. . .’ is a replacement for irrelevant code.