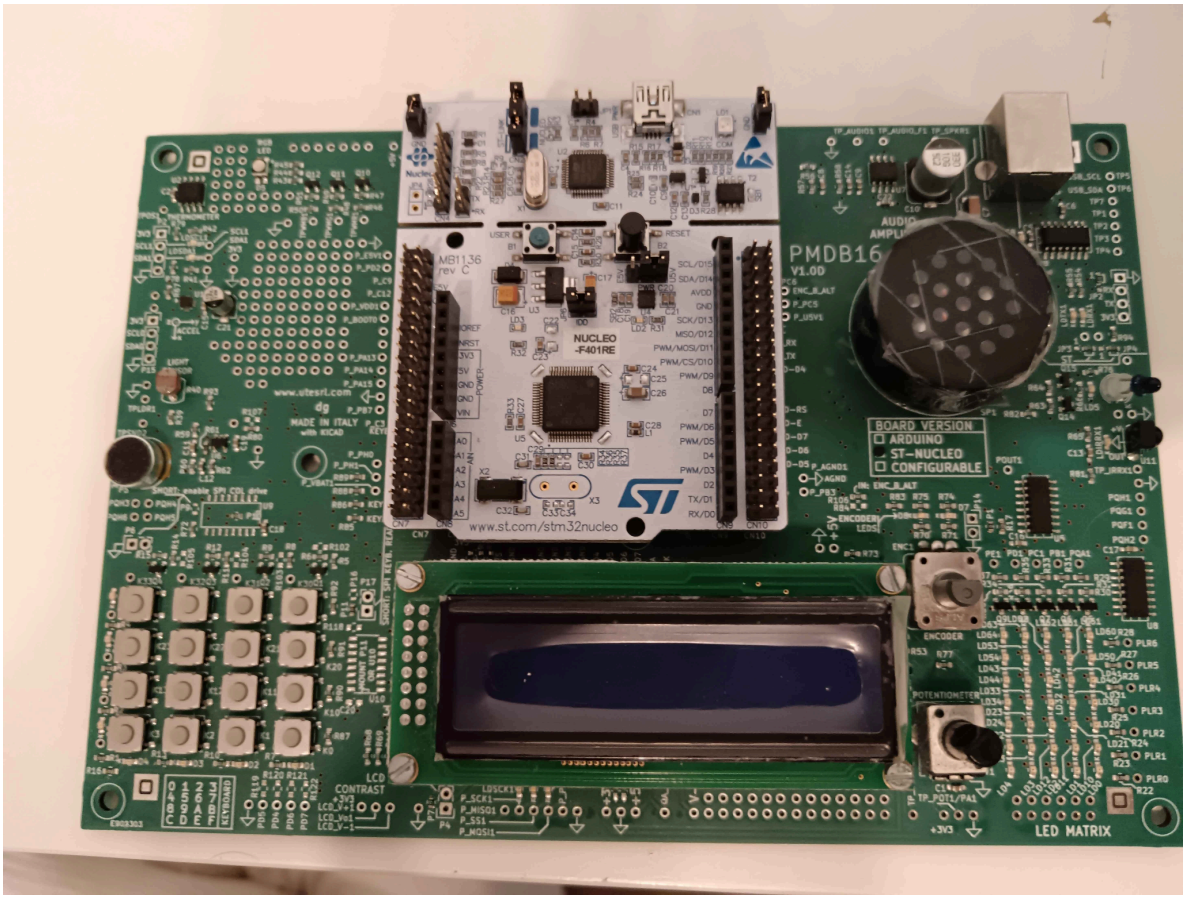
# Sensor system

## Homework Report




## HOMEWORK NUMBER 7

*2024 - 2025*

POLITECNICO
MILANO 1863

# Summary

# I. Members work

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

# II. Exercise 1

**General functionality:**
For this exercise, the goal is to retrieve data from the temperature sensor by using I²C. We also need to read all 11 bits within an interrupt routine. The acquisition is started with Timer 3 in an interruption routine. After that, the acquisition is done with the I²C communication working in DMA. These data acquired are sent to a terminal by using USART communication also working in DMA.

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

1. **PIN lookup**

The used peripherals with their corresponding PIN are displayed in the table below.

| **Peripheral** | **PIN** | **I/O** |
|---|---|---|
| I²C SDA | PB9 | I2C1_SDA |
| I²C SCL | PB8 | I2C1_SCl |

2. **Board configuration**

First we need to do the configuration of the I²C.
After the configuration of the I²C pins like the table above, we can go into the parameters of the I²C to modify some parameters.
We change the mode of the I²C from "Disable" to "I²C" to activate it and let all the other parameters in their default values.
To not block the MCU, we want I²C to work in DMA, so we activate the I²C event interrupt in the NVIC tab. Then, we configure the DMA request for I2C1_RX and I2C1_TX in normal mode and with the Data Width in Byte.

| | |
|---|---|
| Master Features | |
| I2C Speed Mode | Standard Mode |
| I2C Clock Speed (Hz) | 100000 |
| Slave Features | |
| Clock No Stretch Mode | Disabled |
| Primary Address Length selection | 7-bit |
| Dual Address Acknowledged | Disabled |
| Primary slave address | 0 |
| General Call address detection | Disabled |

*I²C general parameters*

Moreover, we want to send the data acquired to a terminal so we configured UART using DMA just like in project 6.

| | |
|---|---|
| Basic Parameters | |
| Baud Rate | 115200 Bits/s |
| Word Length | 8 Bits (including Parity) |
| Parity | None |
| Stop Bits | 1 |
| Advanced Parameters | |
| Data Direction | Receive and Transmit |
| Over Sampling | 16 Samples |

*USART general parameters*

We also need to retrieve and send the data at a rate of 1Hz (1 sec)
So we need to configure the timer TIM3, just like we did in project 6.

| | |
|---|---|
| Counter Settings | |
| Prescaler (PSC - 16 bits value) | 8399 |
| Counter Mode | Up |
| Counter Period (AutoReload Register - 16 bits v. | 9999 |
| Internal Clock Division (CKD) | No Division |
| auto-reload preload | Disable |
| Trigger Output (TRGO) Parameters | |
| Master/Slave Mode (MSM bit) | Disable (Trigger input effect not delayed) |
| Trigger Event Selection | Update Event |

*Timer 3 prescaler and period parameters*

### 3. Code implementation

First, we are going to initialize the I2C and the timer with theses 2 lines of code in the main function (not in the while loop) :

```
/** LM75 initialization -> set up pointer register to temperature sensor **/
if(HAL_I2C_Master_Transmit_DMA(&hi2c1, LM75_ADDR , &LM75_TEMP_ADDR, ONE_BYTE) != HAL_OK)
{
        Error_Handler();
}

/** Start the timer (1 sec) **/
if (HAL_TIM_Base_Start_IT(&htim3) != HAL_OK)
{
    Error_Handler();
}
```

The registers of the LM75 used in the I²C initialization are :

```
uint8_t LM75_ADDR = 0b10010000;  // Address left-shifted by one
uint8_t LM75_TEMP_ADDR = 0x00;   // Address of the pointer register
```

Then, after initializing the timer, we have the Timer callback outside of the main function, which is going to call a function to recover and send the data from the temperature sensor :

```
/**
 * @brief Timer Callback function,
 *        Calling the function to recover and send data
 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM3)
    {
       I2C_ReadSendTemp();         // Read and send temperature every second
    }
}
```

In the function, we use I²C to receive data that we store in an array of 2 bytes since I2C sends data byte per byte. The 2 bytes need to be stored in a unique uint16_t variable to get the real temperature.

**So, the first byte received is put in this variable and is shifted on the left. The second byte is stored after the first byte.**

*(For example : The first byte received is 0b11011001 and the second byte received is 0b00110111, so the uint16_t variable will be 0b11011001 00110111 or 0xd937)*

We can divide this variable by the value of $2^8$, so the value of a byte with all the bits at 1.

Here is the function :

```c
/**
 * @brief Recover data from the temperature sensor by using I²C
 *               Send the data to a terminal using USART
 */
void I2C_ReadSendTemp(void)
{
	char message[SIZE_STRING];
	int message_length = 0;
	uint8_t temperature_bytes[2];
	uint16_t raw_temperature;
	float temperature = 0;

	if(HAL_I2C_Master_Receive_DMA(&hi2c1, LM75_ADDR+1, temperature_bytes, TWO_BYTE)
					== HAL_OK)
	{
		// Cast the two bytes acquired in one variable
		raw_temperature = ((temperature_bytes[0] << EIGHT_BITS |
				temperature_bytes[1] )); // MSB and LSB assembling

	 /** Sign bit verification **/
		 /** If 16th bit is equal to 1 = Negative value **/
		if (raw_temperature & 0x8000)
		{
			raw_temperature |= 0x8000;  // Reverse all the bits except the 16th
			temperature = ((raw_temperature + 0x0001) / 256.0);
			message_length = snprintf(message, SIZE_STRING, "T: -%.3f\r\n",
					temperature);
		}

		 /** Positive value **/
		else
		{
			temperature = raw_temperature / 256.0;
			message_length = snprintf(message, SIZE_STRING, "T: %.3f\r\n",
					temperature);
		}
	}

	/** Error when receiving via I²C **/
	else
	{
		message_length = snprintf(message, SIZE_STRING, "ERROR : Can't read from
				LM75 !!!\r\n");
	}
	HAL_UART_Transmit_DMA(&huart2, (uint8_t *)message, message_length); // Send
temperature to terminal using UART
}
```

We do a verification of the 16th bit, because if its value is “1”, then the value is negative. After that, we need to modify the value by reversing all the other bits to get a regular value and by adding 1 for the first bit.

**Be Careful !!!** Like that, the timer doesn't work because the priority of the interruption need to be different than the I²C and the USART

**So, in the NVIC tab, we change the the preemption priority of the timer interrupt from "0" to "1"**

| | | | |
|---|---|---|---|
| TIM3 global interrupt | ☑ | 1 | 0 |
| I2C1 event interrupt | ☑ | 0 | 0 |
| I2C1 error interrupt | ☐ | 0 | 0 |
| USART2 global interrupt | ☑ | 0 | 0 |

**Moreover, we think that the compiler uses a 9 bit for the float and so the 11 bits of the temperature sensor are rounded with a ~ +1°C - 0,5°C, which creates the bug.** We didn't find any solutions for this problem.

## 4. Execution

Link to the video : https://youtu.be/y7vbAGlpmDo

# IV. Professors comments

| **EXERCISE 1 :** |
| --- |
| |